IBM Parallel System Support
Programs for AIX

Performance Monitoring Guide and Reference

Version 3 Release 1

SA22-7353-00

IBM Parallel System Support
Programs for AIX

Performance Monitoring Guide and Reference

Version 3 Release 1

SA22-7353-00

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page ix.

First Edition (October 1998)

This edition applies to Version 3 Release 1 of the IBM Parallel System Support Programs for AIX (PSSP) Licensed Program, program number 5765-D51, and to all subsequent releases and modifications until otherwise indicated in new editions.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

IBM welcomes your comments. A form for readers' comments may be provided at the back of this publication, or you may address your comments to the following address:

International Business Machines Corporation
Department 55JA, Mail Station P384
522 South Road
Poughkeepsie, NY 12601-5400
United States of America

FAX (United States & Canada): 1+914+432-9405
FAX (Other Countries):
Your International Access Code +1+914+432-9405

IBMLink (United States customers only): IBMUSM10(MHVRCFS)
IBM Mail Exchange: USIB6TC9 at IBMMAIL
Internet e-mail: mhvrcfs@us.ibm.com
World Wide Web: http://www.rs6000.ibm.com

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:

- Title and order number of this book
- Page number or topic related to your comment

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

Contents

Figures

Notices

Trademarks

Publicly Available Software

PSSP includes software that is publicly available:

expect	Programmed dialogue with interactive programs
Kerberos	Provides authentication of the execution of remote commands
NTP	Network Time Protocol
Perl	Practical Extraction and Report Language
SUP	Software Update Protocol
Tcl	Tool Command Language
TclX	Tool Command Language Extended
Tk	Tcl-based Tool Kit for X-windows

This book discusses the use of these products only as they apply specifically to the RS/6000 SP system. The distribution for these products includes the source code and associated documentation. (Kerberos does not ship source code.) **/usr/lpp/ssp/public** contains the compressed **tar** files of the publicly available software. (IBM has made minor modifications to the versions of Tcl and Tk used in the SP system to improve their security characteristics. Therefore, the IBM-supplied versions do not match exactly the versions you may build from the compressed **tar** files.) All copyright notices in the documentation must be respected. You can find version and distribution information for each of these products that are part of your selected install options in the **/usr/lpp/ssp/README/ssp.public.README** file.

About This Book

This book describes how to install, use, administer, and maintain The Performance Toolbox Parallel Extensions for AIX product. In addition, this book provides detailed syntax and parameter information for all commands and subroutines included with the PTPE programs. Problem determination information and a sample application are also included.

For a list of related books and information about accessing online information, see the bibliography in the back of the book.

This book applies to PSSP Version 3 Release 1. To find out what version of PSSP is running on your control workstation (node 0), enter the following:

```
splst_versions -t -n0
```

In response, the system displays something similar to:

```
0 PSSP-3.1
```

If the response indicates **PSSP-3.1**, this book applies to the version of PSSP that is running on your system.

To find out what version of PSSP is running on the nodes of your system, enter the following from your control workstation:

```
splst_versions -t -G
```

In response, the system displays something similar to:

```
1 PSSP-3.1
2 PSSP-3.1
7 PSSP-2.4
8 PSSP-2.2
```

If the response indicates **PSSP-3.1**, this book applies to the version of PSSP that is running on your system.

If you are running mixed levels of PSSP, be sure to maintain and refer to the appropriate documentation for whatever versions of PSSP you are running.

Who Should Use This Book

This book is intended for system administrators and those responsible for RS/6000 Scalable POWERparallel (SP) Systems performance.

Typographic Conventions

This book uses the following typographic conventions:

Typographic	Usage
Bold	• **Bold** words or characters represent system elements that you must use literally, such as commands, flags, and path names.
Italic	• *Italic* words or characters represent variable values that you must supply. • *Italics* are also used for book titles and for general emphasis in text.
`Constant width`	Examples and information that the system displays appear in `constant width` typeface.
[ ]	Brackets enclose optional items in format and syntax descriptions.
{ }	Braces enclose a list from which you must choose an item in format and syntax descriptions.
|	A vertical bar separates items in a list of choices. (In other words, it means "or.")
< >	Angle brackets (less-than and greater-than) enclose the name of a key on the keyboard. For example, <**Enter**> refers to the key on your terminal or workstation that is labeled with the word Enter.
...	An ellipsis indicates that you can repeat the preceding item one or more times.
<**Ctrl-***x*>	The notation <**Ctrl-***x*> indicates a control character sequence. For example, <**Ctrl-c**> means that you hold down the control key while pressing <**c**>.

Abbreviated Names

Use:	On:
Short Name	Full Name
PTPE API	Performance Toolbox Parallel Extensions for AIX Application Programming Interface: the shared C language programming library where the PTPE subroutines reside. These subroutines allow applications to control how SP performance statistics are collected and archived.
RSi	Remote Statistics interface: These allow an application to access statistics from remote nodes (or the local host) through a network interface.
SPMi	System Performance Measurement Interface: the shared C language programming library where Performance Toolbox subroutines reside. These allow an application to make available custom performance statistics about its own performance or that of some other system component. These subroutines also permit applications to access statistics on the local system without using the network interface.

Command and Subroutine Format

Commands and subroutines appear in this book in the following format:

Purpose	Provides the name of the command or subroutine and a brief description of its purpose.
Syntax	Includes a diagram that summarizes the use of the command or subroutine.
Parameters	Lists and describes the required or optional parameters, if any.

Description	Includes a complete description of the command or subroutine.
Return Values	Describes the results of the command or subroutine and lists the indications of success and failure.
Examples	Provides examples of ways in which the command or subroutine is typically used.
Related Information	Lists SP commands, functions, file formats, and special files that are employed by the command, that have a purpose which is related to that of the command. Also listed are related SP documents, other related documents, and miscellaneous information related to the command.

Performance Toolbox Parallel Extensions for AIX User's Guide

Chapter 1. Introducing Performance Toolbox Parallel Extensions for AIX

Performance Toolbox Parallel Extensions for AIX (PTPE) is a performance monitor for RS/6000 Scalable POWERparallel (SP) Systems. When installed on your SP, it allows easy access to performance information about both SP hardware and software (LPPs). This information is available as both run-time (current) and archived (historical) data that you can analyze, manipulate, print, and import to a database, should you so desire.

PTPE builds on the capabilities of Performance Toolbox for AIX, adding monitoring functions specific to the SP system. You can use PTPE to examine the current performance state of any node in your SP system. You decide what performance information to display, and view or print it from any node in the system. Before attempting to use PTPE, it is essential that you be familiar with Performance Toolbox and its user interfaces.

PTPE collects and archives performance statistics for each SP node. It calculates averages for common performance information for all SP nodes. All PTPE data is available for display to help you evaluate performance of the SP at both the node and system level. The Scalable POWERparallel Perspectives (SP Perspectives) graphical user interface (GUI) provides convenient access to the most frequently used PTPE setup and operational functions. A complete set of AIX commands is also provided.

An application programming interface is included to allow you to program for data extraction from PTPE archives through PTPE subroutines. Data compression and reduction is performed automatically on all archived performance data.

How Does PTPE Work?

Before performance data can be displayed, it must be sampled and collected at the individual nodes. These activities require access to statistics created at the operating system and application subsystem levels.

Once performance data is collected by the nodes, it can be gathered for display or archived locally on the nodes where it was generated.

If you want system-wide performance information at a glance, PTPE can use the gathered statistics to calculate averages.

Data Collection

After you install and start PTPE, a data sampler daemon runs on every node configured as a *reporter*. The daemon extracts performance-related information from the AIX operating system and application programs. This information is available for both display and archiving purposes.

Some nodes in the SP configuration execute other daemons as well, which perform data management duties for a group of reporter nodes. The *data manager* nodes' duties include averaging performance data for the reporting group and responding to performance data requests. Most SP nodes will run only one PTPE daemon, the data sampler, at any given time.

Data Archiving

It is often helpful to be able to track performance over an extended period of time. PTPE helps you do this by archiving the performance statistics of your choice. You can archive data on some nodes or all, and include as much or as little performance data as you want. A filter allows you to specify which performance information to enable for archival. When archiving many different statistics, you may find it easier to use the opposite approach and disable specific information for archival.

Data Summarization

What if you want to gauge the performance of all your SP nodes? Displaying individual performance statistics for more than 16 nodes can clutter your screen and be confusing, but PTPE gives you a quick and easy way to assess overall SP performance. It collects common performance data from as many nodes as you want and calculates averages so you can display system-wide statistics in a single window.

PTPE Components

PTPE consists of two major components, a run-time monitor and a performance data manager. The run-time monitor is your interface to the statistical information provided by PTPE data sampler daemons. The performance data manager is responsible for archiving and calculating historical data.

The Run-Time Monitor

Performance data for SP nodes is displayed through a run-time monitor, which uses IBM Performance Toolbox for AIX programs. For comprehensive information about these programs, refer to the *IBM Performance Toolbox for AIX: Guide and Reference*, SC23-2625.

You need not execute the PTPE run-time monitor on the same node for which you want performance data. The run-time monitor allows you to select and display any available performance information for any SP node. It provides two formats of instrumentation to view performance related information:

1. Recording Instruments

   These show performance data items over a period of time. These values can be plotted in:

   - Line graphs
   - Area graphs
   - Skyline graphs
   - Bar graphs

2. State Instruments

   These show only the current value of a performance data item and can be displayed as:

   - Bar graphs

- Pie charts
- Speedometer
- Indicator light

Any item of performance-related information can be displayed using any of these formats.

Optimal use of PTPE permits simultaneous display of performance information from a limited number of nodes. This optimal number varies, depending on your configuration and the load on the system running the run-time monitor. If you attempt to display performance data for too many nodes at once, you may find that the system's performance is adversely affected.

You may also find that you cannot concentrate a large number of data displays from many nodes.

You can reduce the overhead on your system, as well as the number of displays that you need to view, by monitoring the performance summaries prepared by the data managers instead of monitoring data from many nodes at once.

Use the run-time monitor to select the SP nodes, or groups of SP nodes, that you wish to monitor. The run-time monitor directly contacts the nodes that you select, requesting the specific performance data that you requested. The nodes respond by sending the requested information to the run-time monitor program at regular intervals. Each time the run-time monitor receives the information, the graphs displayed by the monitor are updated. You can adjust the update frequency to suit your needs.

Since data is transferred to the PTPE run-time monitor through a socket protocol, the maximum number of nodes that can be simultaneously monitored is limited to the number of available file descriptors. PTPE avoids the need to create socket connections to all nodes by providing performance data summaries on the nodes running performance data manager programs. Therefore, you need only monitor the summary information on the data manager nodes to assess the performance of the entire SP system.

The Performance Data Manager

PTPE must be capable of handling requests to manage all known performance information for all reporter nodes without placing an unacceptable burden on any single node. To accomplish this, information management is shared among the data manager nodes that run the PTPE performance data manager programs. This shared responsibility for information management is achieved through the use of a *monitoring hierarchy*.

The monitoring hierarchy organizes the SP into groups of reporter nodes according to your selection criteria. The criteria you select is important because it determines which nodes' performance data is summarized by the data manager nodes. A data manager node calculates summary statistics that include performance data only from the group of nodes it manages. You can specify groups manually or have them constructed automatically. The SP system can be logically divided into groups identified by any common feature, but groups should be created in such a way as to make their statistical averages meaningful. To name just three of the many possibilities, reporter nodes may be grouped by frame number, as they appear in

Figure 1 on page 6, by Ethernet subnet, or according to the kind of tasks they perform, such as interactive nodes and batch nodes.

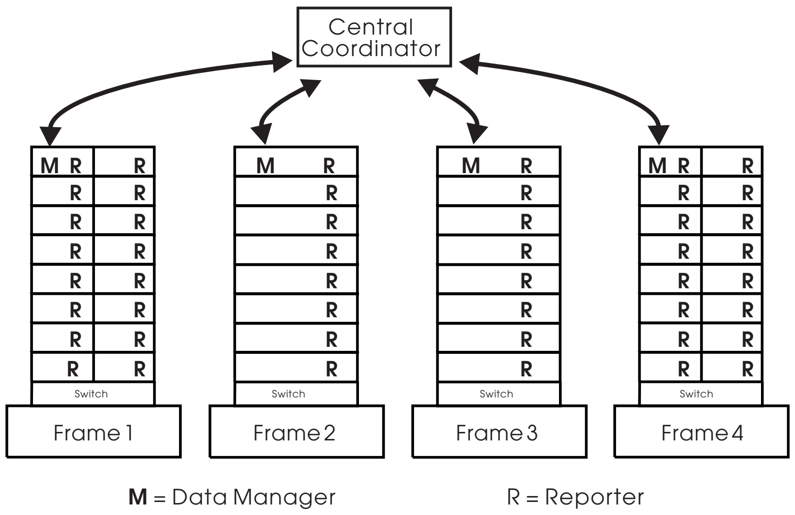


Figure 1. Monitoring Hierarchy by Frame

All nodes in a monitoring hierarchy must be from the same system partition. PTPE cannot support a monitoring hierarchy that contains nodes from multiple partitions. To monitor multiple system partitions, you must create a separate monitoring hierarchy in each partition. One PTPE session and one monitoring hierarchy must be established for each system partition you wish to monitor.

The Data Manager

In each reporting group in your monitoring hierarchy, one node is designated as the data manager node. This node runs the PTPE performance data manager programs. Like the sampler program, the performance data manager runs as daemon processes. It provides the following functions for PTPE:

- Calculates statistical averages for common performance information provided by the nodes that report to it in the monitoring hierarchy, and provides these averages as new performance statistics that can be monitored from the run-time monitor.
- Manages the performance data collection and archiving duties of the nodes in its reporting group. The data manager node instructs the reporter nodes when to start or stop collecting or archiving performance data.
- Routes requests for performance information to those nodes in its reporting group that can provide the requested data, and routes the reply back to the process making the request.

The PTPE performance data manager uses its own mechanism to identify the data to be summarized and archived. This mechanism is different from that used by the PTPE run-time monitor to select performance data for display, making it possible to archive and summarize performance data different from that currently displayed by the run-time monitor.

The number of data manager nodes in an SP configuration may vary, depending upon the tasks of the system and the method used to construct the reporting hierarchy. Although PTPE will allow every node to be designated as a data manager node, this strategy defeats the purpose of the monitoring hierarchy and is not recommended. The PTPE performance data manager not only handles application requests for performance data, it also prepares summary statistics for all SP nodes reporting to it. Therefore, if each node serves as a data manager node, the summary calculation capability of the performance data manager programs is lost. Efficient use of PTPE requires a balanced monitoring hierarchy.

Data manager nodes also help remove the scaling limitation inherent in run-time monitoring. By providing statistical averages of performance data for reporting groups, PTPE eliminates the need to display individual statistics from a larger set of nodes. The run-time monitor need only establish a connection with the data manager nodes to monitor these summaries, greatly reducing the number of socket connections required. Should a summary statistic appear abnormal, the run-time monitor can still establish connections to the reporter nodes that contribute to the summary, but this examination should only be done when investigating a performance problem. For routine SP performance monitoring, the summary information should be sufficient.

The Central Coordinator

Figure 1 on page 6 shows the data manager nodes communicating with a system designated as the central coordinator node. This node coordinates and administers the data manager nodes. Its responsibilities include:

- Calculate statistical averages of the performance summaries calculated by all data manager nodes in the monitoring hierarchy, yielding a system-wide performance summary that can be displayed with the run-time monitor.
- Manage the performance data collection and archiving duties of the data manager nodes, thereby effectively managing this effort for all nodes in the monitoring hierarchy. The central coordinator informs the data manager nodes when to instruct the nodes in their reporting groups to start or stop collection and archiving.
- Receive all requests from the PTPE application programming library, and route the requests to best fulfil them.

The Performance Monitoring Perspective

The performance monitoring graphical user interface included in SP Perspectives allows you to perform most PTPE functions through point-and-click operations. Use the performance monitoring perspective to:

- Create and modify the PTPE monitoring hierarchy
- Start and stop performance data collection
- Start and stop performance data archiving

“Using SP Perspectives” on page 43 provides an overview.

Chapter 2. Planning for PTPE

In order to integrate Performance Toolbox Parallel Extensions for AIX into your SP environment and use it to its fullest advantage, you should understand how it operates and what effect it can have on system performance.

Understanding the Monitoring Hierarchy

PTPE extends the capabilities of IBM Performance Toolbox for AIX in two ways:

1. It recognizes and handles performance statistics unique to the SP system
2. It circumvents the limitation inherent in simultaneously monitoring the performance of large numbers of nodes

PTPE distributes the management of performance information among a number of data manager nodes rather than giving total responsibility to a single node. Although one central coordinator node is designated when the monitoring hierarchy is created, the data manager nodes act as intermediaries, absorbing most of the administrative overhead and greatly reducing data transfer operations. The resulting central coordinator node workload is far less than that required by a single point of control for all nodes and all data management functions.

Distribution of management responsibility is the reason for PTPE's three-tiered monitoring hierarchy. At the bottom is a group of reporter nodes sharing some common characteristic. As Figure 2 shows, these nodes report to a data manager node that operates on the next higher tier. The data manager nodes, in turn, report to the central coordinator node at the top. The middle tier absorbs much of the data management burden that might otherwise overwhelm the central coordinator node.

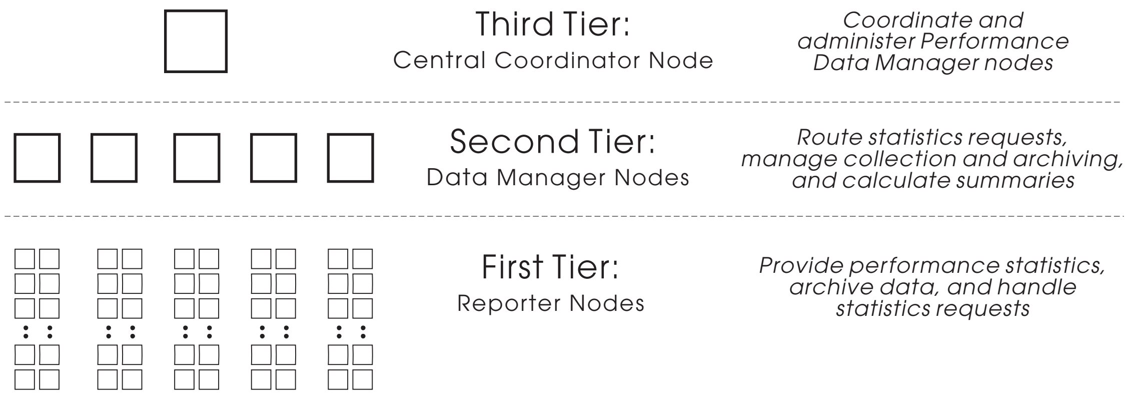

Third Tier:
Central Coordinator Node
Coordinate and administer Performance Data Manager nodes
Second Tier:
Data Manager Nodes
Route statistics requests, manage collection and archiving, and calculate summaries
First Tier:
Reporter Nodes
Provide performance statistics, archive data, and handle statistics requests

Figure 2. PTPE Uses a Three-tiered Hierarchy

Performance Considerations

Several considerations should guide your planning for the monitoring hierarchy:

1. **Which nodes in your system can accept an increase in network traffic?**

   The answer to this question will yield the initial candidates for the data manager and central coordinator nodes.

2. **Which reporter nodes can be logically grouped to yield the most meaningful performance summaries?**

   The answer to this question gives you an initial strategy for laying out the first tier in the monitoring hierarchy.

3. **How many nodes will belong to each reporting group?**

   The higher the number, the greater the traffic handled by the data manager. The lower the number, the more data manager nodes needed.

4. **Should any nodes not be monitored?**

   Some nodes may be dedicated to tasks that require all of their resources, and should be allowed to run without interference. Consider leaving nodes such as these out of the monitoring hierarchy.

5. **Which performance data should you monitor?**

   Choosing the correct set of data to monitor increases the effectiveness of the monitoring effort.

Although efficiency is a vital component in its design, PTPE still generates a certain amount of performance data traffic between reporter nodes and their data manager node. When the PTPE run-time monitor is set to monitor all available performance statistics, you might find performance degraded on the data manager node. When selecting a data manager node, consider nodes that are capable of accepting an increased network load or those where non-critical applications run.

Monitor Thoughtfully

Keep in mind that performance monitoring also makes demands on the processing resources of the very nodes being monitored, depriving other applications of those resources. Some of these applications may be more critical to end users than optimal performance of the system. Therefore, it is important to monitor performance information for the sake of improving system performance for all users rather than out of curiosity.

Remember too, that the efficiency of a monitoring effort depends on the relevance of the data gathered by the tool. If the statistic monitored during a session is not a true measure of the performance issue in question, then the session is wasted, regardless of the efficiency of the tool.

In order to provide the most complete monitoring function, the default PTPE setup collects and summarizes all available performance information on the nodes it monitors. (See “Performance Data for SP Components” on page 24 for a list of the SP-specific performance statistics PTPE collects.) The PTPE commands and API provide you with the capability to enable and disable collection of specific performance data. If you find that the default setup creates an unacceptable load on the monitored nodes, use these features to disable collection of any data that does not address the performance issue you want to examine. This will reduce the overhead of your monitoring activity. See “Controlling Data Collection and Summary” on page 34 for more information.

You should also decide which performance information to examine in a run-time fashion, and which to archive for later analysis. Monitoring a large amount of data from many nodes in the SP may not only be difficult, it may be impossible to display legibly on your terminal. Only performance information critical to the current state of the SP should be monitored in a run-time fashion. Consider using the PTPE commands and API to restrict collection to essential performance statistics and record the other information to the archive. “Archiving Performance Data” on page 35 tells you more.

Plan Your Hierarchy

The ideal monitoring hierarchy balances the number of reporting groups against the size of the reporting groups. A hierarchy with too few reporting groups that are large in size, such as SP System 1 in Figure 3, places a heavy load on the data manager nodes. At the opposite extreme, a hierarchy of too many reporting groups with too few reporters, like SP System 2 in the same figure, squanders data manager node resources on a load that could easily be handled by fewer nodes, and ties up the central coordinator node in unnecessary communication.

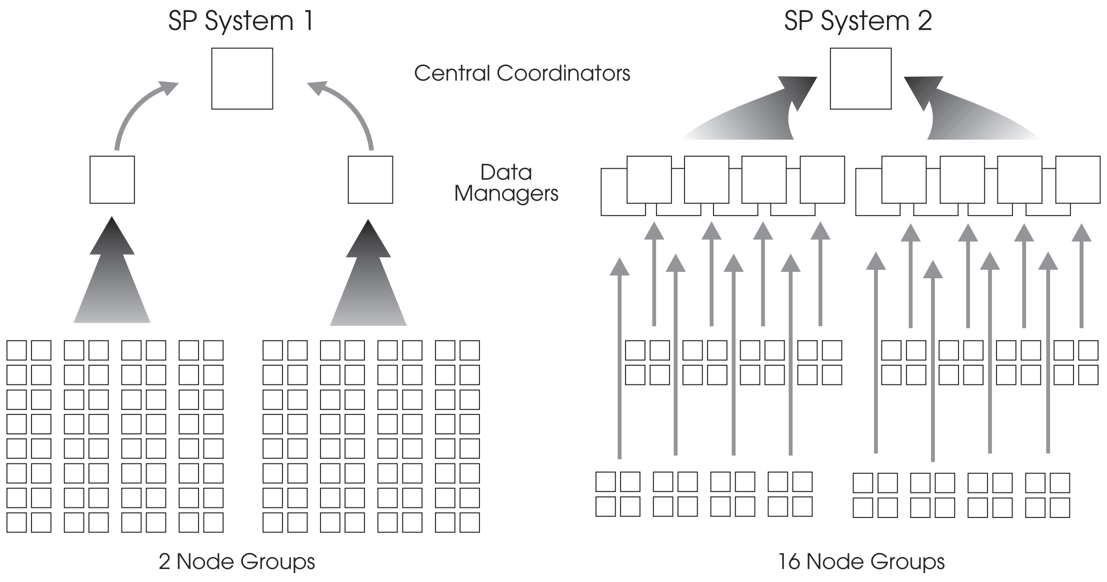


Figure 3. Large vs. Small Reporting Groups

To optimize the reporting groups in your monitoring hierarchy, follow these guidelines when configuring manually:

SP Systems with 16 reporter nodes or fewer:
A single reporting group should be sufficient. More than four reporting groups is excessive. A group with fewer than four reporter nodes wastes data manager node processing power.

SP Systems with more than 16 reporter nodes:
Create at least one reporting group. Use the defaults from **ptpehier** if possible. When altering groups manually, keep groups from exceeding 24 reporters.

It is also important to remember that transmitting performance data from one network to another can generate higher loads on nodes that are not directly involved with the performance monitoring effort, but are affected by network gateway activities. Sending performance information between networks also delays the reception of the performance information at the data manager node. Monitoring hierarchy groups should be confined within network boundaries.

Build Meaningful Groups

The monitoring hierarchy does more than distribute the processing overhead among data manager nodes. It also offers the opportunity to summarize performance information by creating statistical averages for reporting groups. By monitoring these performance data averages, you can determine the current performance status of the SP System without monitoring each node's statistics individually.

You may find that more meaningful averages can be calculated when the monitoring hierarchy is based on logical rather than physical boundaries. For example, you may have specific SP nodes designated for interactive processing, batch processing, and parallel processing. These categories may not necessarily coincide with network or frame boundaries. If your reporting groups are based upon physical boundaries, you will not be able to determine the average performance of your interactive, batch, and parallel nodes at a glance. However, if the interactive nodes comprise one group, the batch nodes another, and the parallel processing still another, each group would then produce meaningful performance summaries. If any logical group threatens to grow too large, simply split it into two smaller groups, preferably at a network or frame boundary.

Planning for System Growth

Whenever nodes are added to your SP System, they must be added to the monitoring hierarchy before PTPE can monitor their performance. Use the SP Perspectives (see page 43) or the **ptpehier** command (see “ptpehier” on page 99 for syntax) to update reporting groups.

If your hierarchy is based on one of the standard configurations (the SP frame or Ethernet network), you can have new reporter nodes added dynamically with SP Perspectives or the **ptpehier** command. If additional frames are added, these methods can automatically create new groups as well.

If your monitoring hierarchy is based on logical boundaries, it might be better to create new groups rather than add nodes to existing groups. For example, if you anticipate growth in the number of nodes dedicated to parallel processing, the creation of a new node group will help you avoid the performance problems that might come with the continual growth of a single parallel processing node group.

Whenever you add or remove nodes in your SP, you must update the monitoring hierarchy with SP Perspectives or **ptpehier**.

Specifying a Monitoring Hierarchy

Both SP Perspectives and the **ptpehier** command offer you the opportunity to accept standard monitoring hierarchies grouped by SP frames or Ethernet networks. If you've decided to group your nodes logically, or, if you do not wish to include all available nodes in the hierarchy, two methods are available for creating a customized monitoring hierarchy. You can use

1. The **ptpehier** command to configure groups through standard input
2. SP Perspectives to point and click on the GUI

You can also use either method to query the current hierarchy. See “Using SP Perspectives” on page 43 or “ptpehier” on page 99 for details and command syntax.

SDR Considerations

PTPE uses specialized data objects in the System Data Repository. These data objects must be present before you can use the data management features of PTPE, or its API. The **ptpeconf** command is provided to create these objects in the SDR. **ptpeconf** is run automatically as part of PTPE installation, but you can also run it manually (root authority is required) on the control workstation in the event that the SDR is damaged or the PTPE objects are destroyed. See “ptpeconf” on page 86 for more information and syntax.

The following SDR objects pertain to PTPE.

Class = SPDM

Contains general information about PTPE and its current activity. Also used by the PTPE API for establishing a programming session with PTPE. No more than one object of this class will exist in the SDR for each system partition.

Attribute Name	*Type*	*Description*	*Comments*
partition	S	Name of the system partition	
central_mgr	S	Name of central coordinator node	
active	I	Indicates whether performance data is currently being collected	0=no, otherwise indicates the number of nodes collecting data
archive	I	Indicates whether performance data is currently being archived	0=no, otherwise indicates the number of nodes archiving data
organization	I	Indicates the method used to construct monitoring hierarchy	
sampling_rate	I	Indicates the number of seconds between samplings of performance data on each node	0=collection off, otherwise specifies the interval in seconds
archiving_rate	I	Indicates the number of seconds between updates of the performance data archive on each node	0=archiving off. This value is always an even multiple of **sampling_rate**

Type S = string Type I = integer

Note: This class is system partition dependant.

Class = SPDM_NODES

Used to store the structure of the monitoring hierarchy. For each node in the monitoring hierarchy, one object of this class will exist.

Attribute Name	*Type*	*Description*
hostID	S	Contains the system's reliable network host name
node_number	I	Contains the SP node number assigned to this system
node_group	I	Identifies the node group to which this system belongs
reports_to	S	Contains the reliable network host name of the data manager node to which this node reports
num_reporters	I	Specifies the number of nodes that report to this node

Note: This class is system partition dependant

The perfmon User Group

Performance monitoring on the SP System demands certain processing resources. This, in turn, deprives other applications of these resources. Because performance monitoring inherently degrades the performance of the nodes being monitored, the responsibility for dedicating resources to such an effort should be entrusted to users who are capable of administering it effectively.

PTPE only permits users belonging to a special user group, called **perfmon**, to use its data management functions and API. This group is created through use of the **ptpegroup** command. **ptpegroup** runs automatically during installation, but you can also run it manually (root authority is required) if the user group is accidentally removed. See “ptpegroup” on page 97 for more information and command syntax. The **ptpegroup** command should be executed on the SP Control Workstation.

ptpegroup updates the **/etc/security/group** and **/etc/group** files on the control workstation. These files are part of a default file collection which is propagated to the SP nodes on an hourly basis using **supper**. Remember that the **perfmon** user group might not become valid on an SP node for up to an hour after it was created on the control workstation. See *IBM Parallel System Support Programs for AIX: Administration Guide* for information about using **supper** to propagate the change faster.

Upon completion of PTPE installation, the **perfmon** group contains only the **root** user. Any additional users assigned to the **perfmon** group must have authority to read and write to the System Data Repository. For PSSP version 2.2 and earlier, **root** is the only user with read/write access to the SDR. User's without SDR read/write access will be unable to perform PTPE operations. Therefore, they should not be made members of the **perfmon** user group.

Statistic Limit

Performance Toolbox Parallel Extensions for AIX currently supports a maximum of 50,000 performance statistics.

Chapter 3. Installing PTPE

Prerequisites

To install and run the Performance Toolbox Parallel Extensions for AIX, the following IBM licensed program products must also be installed:

1. AIX Version 4 Release 3.2
2. RS/6000 Cluster Technology
3. Performance Aide for AIX Version 4.1 (5696-899)

   This must be installed on all nodes where PTPE is to run (the nodes you intend to monitor), as well as on the control workstation
4. Performance Toolbox-Network for AIX Version 4.1 (5696-900)

   This must be installed on any nodes where you will display performance data (the nodes from which you intend to monitor)

PTPE Install Images

The following filesets in the PSSP installation media pertain to PTPE:

ptpe.docs	The reference material: • PDF file for this manual • Manual pages • HTML document files For additional information on this fileset, see “Accessing PSSP Documentation Online” on page 391.
ptpe.program	The PTPE programs. This software will not run unless RS/6000 Cluster Technology has been installed.
ssp.ptpegui	This image should be installed on any node on which you plan to run SP Perspectives.

PTPE Installation

Follow these steps to install the PTPE software. You must install this software on all nodes to be monitored, both those intended as data manager nodes and those intended as reporting group members, as well as on the control workstation. Chapter 2, “Planning for PTPE” on page 9 contains information about how to organize your monitoring hierarchy.

Installing PTPE on the Control Workstation

Install the Performance Toolbox Parallel Extensions for AIX software on the control workstation for running PTPE commands. The control workstation does not become part of the monitoring hierarchy.

Step 1. Select Install Images for Control Workstation

Decide whether to install all of the PTPE images or a subset of them:

1. **ptpe.program**
2. **ptpe.docs**

You must install the **ptpe.program** image on the control workstation.

Step 2. Install PTPE Software on Control Workstation

To install all of the PTPE images, use the following **installp** command:

```
installp -aXd install_device_name ptpe ssp.ptpegui
```

To install an individual PTPE image, use the following **installp** command:

```
installp -aXd install_device_name ptpe.image
```

where *ptpe.image* can be **ptpe.program** or **ptpe.docs**.

Note any messages that appear during installation. Some errors are easily fixed in later steps.

Step 3. Verify Application of PTPE Software on Control Workstation

Use the **lslpp** command to verify that the PTPE software has been applied. See the **lslpp** reference page if you need more information.

Step 4. Complete Installation on Control Workstation

In most cases, PTPE installation requires no additional processing. However, there may be situations in which the installation process can apply the software but cannot perform the processing necessary to configure the PTPE subsystem on the control workstation. This condition is indicated by display of the following message during the installation:

```
spdmdctrl: 2516-471 Cannot determine the number for this node.
```

This condition occurs when the control workstation is being installed for the first time, and the Parallel System Support Program (PSSP) command, **install_cw**, has not been executed. In this situation, the installation process cannot determine whether or not it is being executed on the control workstation, so it does not attempt to perform the configuration duties required on the control workstation.

To complete the installation, ensure that the **install_cw** command has been executed on the control workstation. After verifying that it has been executed (or after executing it), issue the following command to complete the installation process:

```
/usr/lpp/ptpe/bin/spdmdctrl -a
```

Installing PTPE on SP Nodes

Before you install the PTPE software on the SP nodes, you must first install the **ptpe.program** install image on the control workstation. Use **lslpp** to verify its complete installation on the control workstation before attempting to install PTPE on the SP nodes.

Step 1. Select Install Images for SP Nodes

As on the control workstation, you can install all the Performance Toolbox Parallel Extensions software from the installation media, or select specific install images for installation. Consideration of the software requirements and disk space should guide your decision. If disk space on the SP nodes is limited, you may find these suggestions helpful:

- The **ptpe.program** image must be installed on any SP node to be monitored
- The **ssp.ptpegui** image should be installed on those nodes where you plan to run SP Perspectives.
- The **ptpe.docs** fileset may be installed only on a group of documentation server nodes

Select the images that correspond to the functions you need, and ensure that the required software has been installed (or is being installed along with it).

Step 2. Install PTPE Software on SP Nodes

To install all of the PTPE images, use the following **installp** command:

`installp -aXd` *install_device_name* `ptpe ssp.ptpegui`

To install an individual PTPE image, use the following **installp** command:

`installp -aXd` *install_device_name ptpe.image*

where *ptpe.image* can be **ptpe.program** or **ptpe.docs**.

Note any messages that appear during installation. Some errors are easily fixed in later steps.

Step 3. Verify Application of Software on SP Nodes

Use the **lslpp** command to verify that the PTPE software has been applied. See the **lslpp** reference page if you need more information.

Step 4. Complete Installation on SP Nodes

In most cases, PTPE installation requires no additional processing. However, there may be situations in which the installation process can apply the software but cannot perform the processing necessary to configure the PTPE subsystem on the SP node. This condition is indicated by display of either of the following messages during the installation:

```
spdmdctrl: 2516-471 Cannot determine the number for this node.
spdmdctrl: 2516-475 A port number for the spdmd inetd subserver has
not been reserved by the control workstation.
```

This condition occurs when PTPE is installed on an SP node before being installed on the control workstation, or when the control workstation cannot perform the necessary configuration steps. To correct this condition:

1. Install **ptpe.program** (or complete its installation) on the control workstation
2. After verifying PTPE installation in the control workstation, execute the following command on the SP node:

   ```
   /usr/lpp/ptpe/bin/spdmdctrl -a
   ```

Chapter 4. Configuring PTPE

Prerequisites

Before you can use Performance Toolbox Parallel Extensions for AIX, you must create a monitoring hierarchy using either SP Perspectives or the **ptpehier** command. For an explanation of the monitoring hierarchy concept, see “Understanding the Monitoring Hierarchy” on page 9. Before the reporting hierarchy can be created, however, specific data object classes must exist in the System Data Repository (see “ptpeconf” on page 86), and the **perfmon** user group must exist on the system (see “ptpegroup” on page 97). These two tasks are automatically performed during PTPE installation, but can be repeated at any time.

In order to perform any operations affecting the monitoring hierarchy, whether through SP Perspectives or the command line, you must be a member of the **perfmon** user group, and you must have **perfmon** set as the primary group.

Before you can begin collecting performance information or recording this information to the archive, each node in the monitoring hierarchy must have enough disk space to store a statistic translation table. This table is necessary for PTPE to store and retrieve performance data in the archive. Because of the large number of statistics that PTPE supports, this table consumes approximately 13 MB of disk space. When collection begins, PTPE attempts to store this table in the **/var/adm/ptpe** directory. Ensure that the filesystem containing this directory on each node has at least 13 MB of space available. (Use the **df** command to display filesystem space.)

Because the **/var** filesystem is used by various logging utilities, IBM recommends that you set up a private filesystem for the **/var/adm/ptpe** directory on each node so that PTPE is guaranteed this space, and so that the PTPE archive and translation table do not consume space needed for other logging utilities. Making a separate filesystem also simplifies the effort necessary to grow this filesystem should the performance information archive files grow beyond the initial size.

Although you do not have to remove an existing hierarchy before replacing it, removing it will effectively lock out other users and prevent them from starting performance information collection and summarization on a hierarchy that is about to be replaced. The only other time you should remove the monitoring hierarchy is when you wish to disable PTPE for an extended period of time.

Selecting a Standard Monitoring Hierarchy

Follow these steps to configure PTPE using one of the standard monitoring hierarchies. If you prefer to configure a customized hierarchy, Chapter 2, “Planning for PTPE” on page 9 and “ptpehier” on page 99 can show you how.

Step 1. Configure the Monitoring Hierarchy

The structure of the reporting hierarchy impacts how summarized performance information is calculated. It also determines which systems are assigned the tasks of summarization and command delegation.

PTPE provides two options to construct the reporting hierarchy for efficient data transmission and a moderate level of management overhead on the SP System. Both options automatically appoint a PTPE central coordinator node.

Organization by Ethernet Local Area Subnetwork

By organizing the hierarchy according to Ethernet subLAN boundaries, the Performance Toolbox Parallel Extensions reduces the amount of network traffic that must be transmitted through network gateways, and avoid inflicting additional overhead to the gateway systems.

Using SP Perspectives: Click on the central coordinator node in the Hierarchy Pane, pull down the Actions menu, and select `Create Using Ethernet`.

If the central coordinator node has not yet been assigned, click on the node you wish to nominate in the Nodes Pane. Then, either click on the *Replace Central Coordinator* icon on the tool bar or pull down the Actions menu and select `Nodes → Assign Central Coordinator` from the actions displayed. The node appears at the top of the Hierarchy Pane.

Using the ptpehier Command: Enter the following command to configure a monitoring hierarchy organized by Ethernet local area subnetwork:

```
ptpehier -e
```

If you have not planned a customized monitoring hierarchy as discussed in Chapter 2, “Planning for PTPE” on page 9, organization by Ethernet local area subnetwork is recommended.

Organization by SP Node Frame

By organizing the hierarchy according to frame boundary, a minimal amount of cross-network traffic is generated, and summary performance information is prepared according to system boundaries that are easily recognizable by SP system users. The reporting group size is restricted to a maximum of 16 nodes to minimize the overhead on data manager nodes.

Using SP Perspectives: Click on the central coordinator node in the Hierarchy Pane, pull down the actions menu, and select `Create Using Frames`.

If the central coordinator node has not yet been assigned, click on the node you wish to nominate in the Nodes pane. Then, either click on the *Replace Central Coordinator* icon on the tool bar or pull down the Actions menu and select `Nodes → Assign Central Coordinator` from the actions displayed. The node appears at the top of the Hierarchy Pane.

Using the ptpehier Command: Enter the following command to configure a monitoring hierarchy organized by frame boundary:

```
ptpehier -f
```

Step 2. Initialize the Monitoring Hierarchy

Prepare the new monitoring hierarchy for performance data collection and summarization. Enter the following on the command line:

```
ptpectrl -i
```

Chapter 5. Using PTPE

PTPE comprises two major parts, a run-time monitor and a performance data manager. The run-time monitor is your interface to the statistical information provided by the PTPE data sampler daemons. The performance data manager is responsible for archiving and calculating summary data.

Using the Run-Time Monitor

The run-time monitor collects SP performance data and feeds it to the Performance Toolbox programming libraries and graphical user interfaces. For information on how to use the Performance Toolbox interfaces and libraries, consult the *IBM Performance Toolbox for AIX: Guide and Reference*.

Starting the SP Data Supplier

Whenever PTPE starts performance data collection, it automatically starts the programs on each node that provide SP-specific performance statistics. These same data programs are also available to Performance Toolbox for AIX, through the **ptpertm** command.

If the **/etc/perf/xmservd.res** file was present when PTPE was installed on the node, an entry was made in this file to start the **ptpertm** command whenever Performance Toolbox's **xmservd** daemon becomes active. This will provide Performance Toolbox with SP-specific performance information, which can be viewed from **xmperf** and **3dmon**.

If this file was not present when PTPE was installed, **ptpertm** will not be started automatically by **xmservd**, and the SP-specific performance information will not become available to Performance Toolbox. To make this information available, you may do one of two things:

1. Make sure your user ID has sufficient privilege to create files in the **/etc** directory, and create a **xmservd.res** file on the node you will monitor. Do this by copying the **xmservd.res** file from the Performance Toolbox for AIX's sample directory to the **/etc/perf** directory on that node:

   ```
   cp /usr/samples/perfagent/server/xmservd.res /etc/perf
   ```

   Once the file is created, edit the file and add the following line to the end of the file:

   ```
   supplier:       /usr/lpp/ptpe/bin/ptpertm -p
   ```

   Determine whether the **xmservd** daemon is currently running on the node by using the **ps** command. If the daemon is running, refresh the daemon by sending it a SIGHUP signal with the kill command:

   ```
   kill -1 <xmservd_process_id>
   ```

   Using this method will guarantee that the PTPE run-time monitor will start whenever **xmservd** is started. Since Performance Toolbox starts **xmservd** whenever an application uses its libraries, or one of its graphical user interfaces (GUIs) is started, this ensures that the SP performance data collected by the run-time monitor is always available.

2. Run the **ptpertm** command from the command line or shell script on the node you wish to monitor, placing the process in the background:

   ```
   ptpertm &
   ```

   Make a note of the process ID of the **ptpertm** process. Later, when you have completed your monitoring of the node, you can terminate the **ptpertm** process:

   ```
   kill <ptpertm_process_ID>
   ```

   After starting the **ptpertm** command, refresh the **xmservd** daemon running on the node:

   ```
   kill -1 <xmservd_process_ID>
   ```

   This method should be used only as a temporary measure. Using this method will not guarantee that every Performance Toolbox application will have access to the SP performance information.

Finding Out What Data is Available

You can use the **xmpeek -l** command to get a complete list of available performance statistics for a specific node.

Performance Data for SP Components

The SP-specific performance information accessible by the PTPE run-time monitor can be found under the context **DDS/IBM/PSSP.harmld**. This includes LoadLeveler, IBM Virtual Shared Disk, and CSS statistics.

LoadLeveler Performance Statistics

LL.STARTD.current_jobs
The number of active nodes managed by the **startd** daemon.

LL.STARTD.jobs_running
The number of jobs managed by the **startd** daemon that are in running state

LL.STARTD.jobs_pending
The number of jobs managed by **startd** that are in pending state

LL.STARTD.jobs_suspended
The number of jobs managed by **startd** that are in suspended state

LL.STARTD.total_jobs_rec
The total number of start job requests received from the startd

LL.STARTD.total_jobs_com
The total number of jobs started by **startd** that have been completed

LL.STARTD.total_jobs_rem
The total number of jobs that were canceled either by the user or LoadLeveler

LL.STARTD.total_jobs_vac
The total number of jobs that were terminated and returned to the queue to be rescheduled

LL.STARTD.total_jobs_rej
The total number of jobs that could not be run by **startd**

LL.STARTD.total_jobs_sus
The total number of jobs suspended by **startd**

LL.STARTD.total_conn
The total number of connections attempted by the **startd**

LL.STARTD.failed_conn
The number of connection attempts that failed

LL.STARTD.total_out_trans
The total number of outbound transactions attempted by **startd**

LL.STARTD.failed_out_trans
The number of outbound transactions that failed

LL.STARTD.total_in_trans
The total number of inbound transactions attempted by **startd**

LL.STARTD.failed_in_trans
The number of inbound transactions that failed

LL.SCHEDD.current_jobs
The number of jobs currently active for the **schedd** daemon

LL.SCHEDD.jobs_idle
The number of jobs currently active for the **schedd** daemon

LL.SCHEDD.jobs_pending
The number of jobs managed by the **schedd** daemon that are in pending state

LL.SCHEDD.jobs_starting
The number of jobs managed by the **schedd** daemon that are in starting state

LL.SCHEDD.jobs_running
The number of jobs managed by the **schedd** daemon that are in running stat

LL.SCHEDD.total_jobs_sub
The total number of jobs submitted to the **schedd** daemon

LL.SCHEDD.total_jobs_com
The total number of jobs submitted to the **schedd** daemon that completed

LL.SCHEDD.total_jobs_rem
The total number of jobs submitted to the **schedd** daemon that were canceled by the user or LoadLeveler

LL.SCHEDD.total_jobs_vac
The total number of jobs submitted to the **schedd** daemon that were terminated and returned to the queue to be rescheduled

LL.SCHEDD.total_jobs_rej
The total number of jobs submitted to the **schedd** daemon that could not be run

LL.SCHEDD.total_conn
The total number of connections attempted by the **schedd** daemon

LL.SCHEDD.failed_conn
The number of connection attempts that failed

LL.SCHEDD.total_out_trans
The total number of outbound transactions attempted by the **schedd** daemon

LL.SCHEDD.failed_out_trans
The number of outbound transactions that failed

LL.SCHEDD.total_in_trans
The total number of inbound transactions attempted by the **schedd** daemon

LL.SCHEDD.failed_in_trans
The number of inbound transactions that failed

IBM Virtual Shared Disk Performance Statistics

VSD.req_block_shortage
Requests queued waiting for a request block

VSD.pbuf_shortage
Requests queued waiting for a pbuf

VSD.cache_shortage
Requests queued waiting for a cache block

VSD.buddy_buf_shortage
Requests queued waiting for a buddy buffer

VSD.rej_requests
Rejected Requests

VSD.rej_responds
Rejected responses

VSD.Rej_no_buddy_buf
Rejected no buddy buffer

VSD.request_rework
Requests rework

VSD.timeout_error
timeouts

VSD.1_retry_count
Retries 1

VSD.2_retry_count
Retries 2

VSD.3_retry_count
Retries 3

VSD.4_retry_count
Retries 4

VSD.5_retry_count
Retries 5

VSD.6_retry_count
Retries 6

VSD.7_retry_count
Retries 7

VSD.8_retry_count
Retries 8

VSD.9_retry_count
Retries 9

VSD.avg_buddy_wait
Average buddy buffer wait_queue size

VSD.indirect_io
I/O is not performed directly from mbuf

VSD.comm_buf_shortage
Shortage on the Communication Buf pool

VSD.loc_req_rd
Local read

VSD.loc_req_wr
Local write

VSD.rem_req_rd
Remote read

VSD.rem_req_wr
Remote write

VSD.cli_req_rd
Physical read

VSD.cli_req_wr
Physical write

VSD.phy_req_rd
Physical read

VSD.phy_req_wr
Physical write

VSD.cache_hit
Cache Hits

VSD.bytes_rd
Total Byte Read

VSD.bytes_wr
Total Byte Write

CSS Performance Statistics

CSS.elapsed_time
Time in seconds since last reset

CSS.ipackets_msw
Packets received on interface(msw)

CSS.ipackets_lsw
Packets received on interface(lsw)

CSS.ibytes_msw
Total # of octets received(msw)

CSS.ibytes_lsw
Total # of octets received(lsw)

CSS.recvintr_msw
Number of receive interrupts(msw)

CSS.recvintr_lsw
Number of receive interrupts(lsw)

CSS.ierrors
Input errors on interface

CSS.opackets_msw
Packets sent on interface(msw)

CSS.opackets_lsw
Packets sent on interface(lsw)

CSS.obytes_msw
Total number of octets sent(msw)

CSS.obytes_lsw
Total number of octets sent(lsw)

CSS.oerrors
Output errors on interface

CSS.ipackets_drop
Number of packets not passed up

CSS.ibadpackets
# of bad packets received from adapter

CSS.stat_reserved
Reserved

CSS.windows_open
Active windows (IP, Service, MPCI)

Using the Performance Data Manager

The performance data manager adds two new capabilities to the Performance Toolbox base function: performance information summary and data retention.

Monitoring all vital performance information on even a small number of nodes in a run-time fashion can be a cumbersome task. For systems as large as the SP, this can become impossible using traditional monitoring tools.

The PTPE performance data manager is capable of monitoring a large number of nodes in a run-time fashion by providing summarized performance information to Performance Toolbox. Instead of monitoring individual performance statistics on a large number of nodes, you can monitor a small set of averaged statistics on a few nodes. Should any of these averaged statistics appear abnormal, the individual statistics used in their calculation can still be examined using Performance Toolbox interfaces. Through its commands and application programming interface, PTPE provides the capability to enable and disable specific performance statistics from the calculation.

Performance monitoring is often a cumbersome process, producing results of questionable value. Simplifying the task of monitoring performance information in a run-time fashion accomplishes little if no one is free to observe the information. Furthermore, data presented in a graphical format is often difficult to use as input to an application.

Using the PTPE performance data manager, you can archive performance statistics for later analysis. By retaining this information in a machine-readable format, you can make it available as input to calculations, complex analysis, reports, and even post-processing with such tools as the Performance Toolbox **azizo** command. Again, the PTPE commands and API allow you to enable and disable specific performance statistics for recording to the archive. You can also use the API to retrieve data from the archive. All archived data is retained after data collection has been shut down.

All PTPE performance data manager operations, such as the removal of archives, can be performed only by users who belong to the **perfmon** user group. This user group is created when PTPE is installed. Users can be added to this group using SMIT or the **chgrpmem** command. Before you can use performance data manager functions, however, **perfmon** must be your primary group.

Summarizing Performance Information

Summarized performance information simplifies the run-time monitoring effort on a large number of nodes. It is not intended as a replacement for the individual performance statistics provided by Performance Toolbox, but as a means of reducing the amount of data to be monitored in a run-time fashion when you want to determine the performance state of the SP as a whole.

These summarized performance statistics are the calculated averages of the same performance statistic gathered from each node in a group. For example, the PTPE performance data manager collects an individual performance statistic, such as *% of time CPU in user state*, for all members of a node group. Then it calculates the average for these statistics and presents the average as a new statistic to IBM Performance Toolbox for AIX.

The way in which these summarized statistics are calculated is determined by the monitoring hierarchy. The monitoring hierarchy dictates which nodes are members of the same reporting group, and which are designated as data manager nodes. The data manager nodes calculate the average of the individual data statistics from all members of their group, their own statistics included, and provide the summary statistic to IBM Performance Toolbox for AIXs.

As an example, consider a monitoring hierarchy for a 16-node system, shown in Figure 4 on page 30. Nodes 1 through 8 are members of one reporting group, and nodes 9 through 16 are members of a second group. Nodes 2 and 10 have been designated data manager nodes for their respective groups.

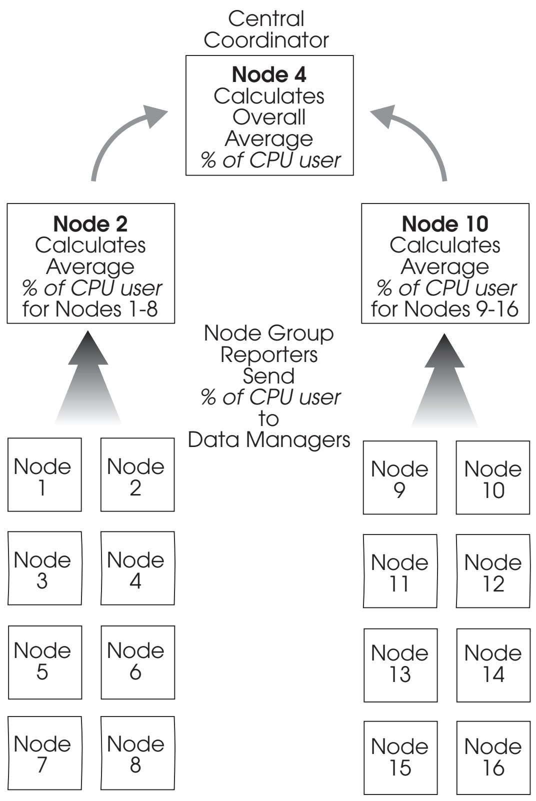

Central Coordinator
Node 4 Calculates Overall Average % of CPU user
Node 2 Calculates Average % of CPU user for Nodes 1-8
Node 10 Calculates Average % of CPU user for Nodes 9-16
Node Group Reporters Send % of CPU user to Data Managers
Node 1
Node 2
Node 3
Node 4
Node 5
Node 6
Node 7
Node 8
Node 9
Node 10
Node 11
Node 12
Node 13
Node 14
Node 15
Node 16

Figure 4. Monitoring Hierarchy of Two Reporting Groups

The summarized statistics for *% of time in CPU user state* would be available from nodes 2 and 10. The *% of time in CPU user state* on node 2 would be the calculated average of those values collected from nodes 1 through 8, while the same statistic on node 10 would be the average of those values collected from nodes 9 through 16.

The summarized statistic prepared by each data manager node is, in turn, sent to the monitoring hierarchy's central coordinator node, which prepares an overall system average of the statistic.

Locating Summary Information

Summarized performance statistics are available at the group level on the data manager nodes, and at the SP level on the central coordinator node. You can display them using the Performance Toolbox GUI, and capture them through its **Spmi** and **Rsi** libraries.

Understanding the structure of the monitoring hierarchy is essential to making effective use of summary statistics. Monitoring averages cannot help you to analyze system performance if you are uncertain which nodes' statistics have been calculated. Furthermore, you may have difficulty locating the summary statistics if you do not know the identity of the data manager nodes.

Since the organization of the monitoring hierarchy is crucial to the preparation of summary statistics, it should be created with care. PTPE can configure your SP System as a standard monitoring hierarchy based on either frame boundaries or Ethernet network, however, these hierarchies may yield average statistics that are less useful than those from a customized monitoring hierarchy. The summary performance information may be more meaningful if the hierarchy is organized into logical rather than physical groups. Refer to “Understanding the Monitoring Hierarchy” on page 9 for further information.

By default, PTPE collects **all available** performance statistics on the nodes and sends them to the performance data managers for summarization. This may produce more performance information than you want or need, and it might place too much overhead on your system. (See “Performance Considerations” on page 9 for a more detailed discussion.) The PTPE API (**libptpe.a**) contains subroutines you can use in your application programs to control the collection of performance statistics at the node level, as well as the transmission of statistics to performance data manager nodes. “Controlling Data Collection and Summary” on page 34 explains how to do this.

Collecting and Displaying Summary Data

This procedure includes examples of how to use PTPE on a system with two frames consisting of eight wide nodes each.

Step 1: Make Sure You're A Valid User

The Performance Data Manager can only be started by users in the **perfmon** user group. Ensure that your userid is a member of the **perfmon** user group. Also, be sure that you have executed **newgrp perfmon** to make **perfmon** your effective group.

Step 2: Familiarize Yourself With The Reporting Hierarchy

Skip this step if you already know how the monitoring hierarchy is organized.

You can view the current monitoring hierarchy using the **ptpehier** command with the **-p** flag. See “ptpehier” on page 99 for details.

Enter

```
ptpehier -p
```

A textual representation of your hierarchy is returned:

```
ptpehier: The current monitoring hierarchy structure is:
 spnode05.ibm.com
   spnode09.ibm.com
     spnode01.ibm.com
     spnode03.ibm.com
     spnode05.ibm.com
     spnode07.ibm.com
     spnode09.ibm.com
     spnode11.ibm.com
     spnode13.ibm.com
     spnode15.ibm.com
   spnode23.ibm.com
     spnode17.ibm.com
     spnode19.ibm.com
     spnode21.ibm.com
     spnode23.ibm.com
     spnode25.ibm.com
     spnode27.ibm.com
     spnode29.ibm.com
     spnode31.ibm.com
```

This displays the name of the central coordinator node at the top with data manager nodes indented below. Further indented below the data manager nodes are the names of the members of the reporting groups. In this example, spnode05 is the central coordinator node, while spnode09 and spnode23 are the data manager nodes.

All nodes in the monitoring hierarchy supply performance statistics when you initiate performance data collection. If the monitoring hierarchy includes nodes that you do not wish to monitor, you can create a customized hierarchy that includes only specified nodes by using SP Perspectives or the **ptpehier -i** command.

If, instead of displaying the hierarchy, the **ptpehier -p** returns the following message:

```
ptpehier: 2516-125 Cannot find the monitoring hierarchy.
```

This means that no monitoring hierarchy exists. You must create one, using either **ptpehier -e**, **ptpehier -f**, **ptpehier -i**, or SP Perspectives.

Step 3: Initialize Performance Data Summary

This step is not required every time the PTPE performance data manager is used, but it is required before the performance data manager is used for the first time after installation, and whenever the monitoring hierarchy structure is changed.

Execute performance data manager setup using the **ptpectrl** command: (See “ptpectrl” on page 88 for details.)

```
ptpectrl -i
```

The command returns the following messages:

```
ptpectrl: Beginning setup for performance information collection
ptpectrl: Reply from the Central Coordinator expected within 270 seconds. OK.
ptpectrl: Setup for collecting performance information succeeded.
```

Once the setup is complete, performance information collection and summarization can begin.

Step 4: Start Data Collection and Summary

Execute the **ptpectrl** command to start data collection and summarization at the data manager nodes: (See “ptpectrl” on page 88 for details.)

```
ptpectrl -c
```

The command returns:

```
ptpectrl: Starting collection of performance information.
ptpectrl: Reply from the Central Coordinator expected within 270 seconds.
```

This command may take several minutes to execute, especially on larger SP configurations. The time remaining will be updated every five seconds. When the central coordinator finally replies, the completed output (if everything was successful) will look like:

```
ptpectrl: Starting collection of performance information.
ptpectrl: Reply from the Central Coordinator expected within 235 seconds. OK.
ptpectrl: Performance information collection successfully started.
```

The command will then proceed to enable all performance statistics within the hierarchy. The progress of this effort will be reported back to the user. If all nodes within the hierarchy are not prepared to take the enablement request, the command will make additional attempts (up to 4 attempts will be made). Example output for this stage:

```
ptpectrl: Enabling statistics for performance information collection.
ptpectrl: Reply from the Central Coordinator expected within 270 seconds. OK.
ptpectrl: Some systems reported that they were not ready to accept the request.
          Pausing 15 seconds to retry command.
ptpectrl: Attempting the command again.
ptpectrl: Reply from the Central Coordinator expected within 270 seconds. OK.
ptpectrl: Enabling and restriction of statistics complete.
ptpectrl: Command completed.
```

Note: Aborting this command may leave the SP System in an unpredictable state from which recovery can be difficult.

Step 5: Monitor the Summarized Data

Once collection is started, the summarized performance information becomes available to the Performance Toolbox GUI and its **Spmi** and **Rsi** libraries. To examine these summarized statistics, log into a system where the Performance Toolbox Performance Manager package has been installed. Performance Toolbox allows access to this information. Refer to *IBM Performance Toolbox for AIX: Guide and Reference* for additional information.

If, for example, Performance Toolbox is installed on spnode01, execute the following command on that node:

```
xmperf
```

Use **xmperf** menu selections to create an instrument to monitor the summarized statistic for *% of time CPU in user state* from spnode09, and another instrument to monitor the same statistic from spnode23.

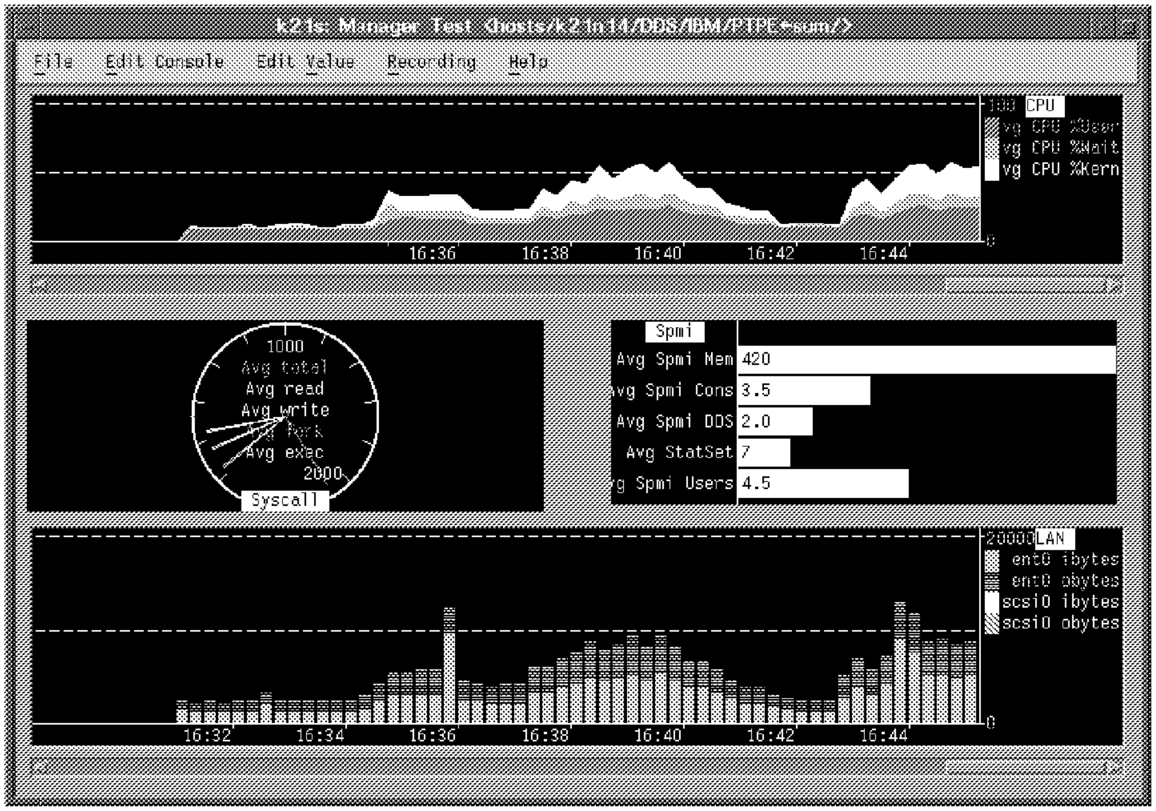


Figure 5. Performance Toolbox Performance Statistic Display

From the **xmperf** display, you can bring up additional instruments to view the individual node statistics that were calculated in the group's average. These may be of interest if one of the summary statistics arouses your suspicion.

Step 6: Stop Data Collection and Summarization

Performance monitoring creates an additional load on the nodes being monitored. When you have collected sufficient performance data, shut the monitoring process down using the **ptpectrl** command:

```
ptpectrl -s
```

The following messages are returned:

```
ptpectrl: Reply from the Central Coordinator expected within 270 seconds. OK.
ptpectrl: Performance information collection successfully stopped.
ptpectrl: Command completed.
```

Controlling Data Collection and Summary

By default, PTPE instructs all nodes in the monitoring hierarchy to make all performance information available to the data manager nodes for summary. This includes all performance information that can be detected by the **xmpeek -l** command. At regular intervals, this information is sent by each node to its managing node, which calculates averages for each statistic.

Normally, you will not need all of the performance information that can be provided for the SP. Instead, you will wish to monitor a core set of performance statistics, or you may have identified a list of specific information that is not to be monitored. PTPE provides two mechanisms for restricting collection to the statistics you want.

These can significantly reduce the load and network traffic generated by the monitoring programs.

The ptpectrl Command

Whenever this command is used to start collection or archiving of performance information, the command checks for a configuration file to determine what performance information should be enabled or disabled. This file, **ptpe.cf**, is not created by PTPE when the product is installed, but an example file is installed in **/usr/lpp/ptpe/samples/ptpe.cf**. For instructions on modifying this file, examine the sample files in “ptpe.cf Samples” on page 42, or refer to “Using the ptpe.cf File” on page 40.

The **ptpectrl** command initially searches the user's home directory for this configuration file. If the file is not located in that directory, the command next searches the **/etc/perf** directory. If the file is not found in either search, the command will enable all available performance information. Should the command find a configuration file, it will use the instructions within this file to identify what performance statistics should be collected, averaged, and recorded in the archive,

If collection or archiving is already active, the **ptpectrl** command provides the **-m** option to revise the list of enabled statistics.

For more information on the **ptpectrl** command and its options, refer to “ptpectrl” on page 88.

PTPE API Applications

The PTPE API provides interfaces to control statistic collection, averaging, and recording to the archive. The following subroutines are included in the PTPE C language programming library (**libptpe.a**) for this purpose:

The following subroutines are included in the PTPE C language programming library (**libptpe.a**) for this purpose:

PtpeColEnableStats	Identifies statistics to include in summary
PtpeColDisableStats	Identifies statistics to exclude from summary
PtpeQueryAvailStats	Lists statistics available for summary
PtpeColQueryStats	Identifies statistics currently included in summary

See Chapter 6, “Using the PTPE Application Programming Interface” on page 47

Archiving Performance Data

Some methods of performance analysis require historical performance information in order for trends to become apparent, and for hidden relationships between specific performance statistics to emerge. Run-time monitoring interfaces display only a limited history of performance statistics for the time period that the monitor was active. Furthermore, the history that the run-time monitoring tool can display is restricted to the performance information that was explicitly monitored at the time. These histories may be insufficient when a need arises for a performance statistic that was not monitored, or if the monitor was not active at a crucial time.

The PTPE performance data manager retains performance data in a machine-readable format. This data can be accumulated in the archive even while such monitoring interfaces as the Performance Toolbox **xmperf** and **3dmon** utilities, are inactive. The PTPE performance data manager can also retain data that is not being monitored by these applications, as well as data that is not selected for collection and summary. This means that you don't have to be monitoring a specific performance statistic in a run-time fashion in order to archive it.

IBM Performance Toolbox for AIX includes two commands, **a2ptx** and **azizo**, that can be useful in manipulating archived performance data. Refer to *IBM Performance Toolbox for AIX: Guide and Reference* for details.

“Controlling Performance Data” on page 68 contains more detailed information.

Archival Requirements

In order to archive performance data, performance summarization must be enabled in the monitoring hierarchy because the performance data manager uses the same daemon processes to archive performance data as it does to summarize it. It is not necessary to summarize the information that you wish to archive. (Only members of the **perfmon** user group may initiate performance data archival.)

Archive Access

Each node maintains its own performance data archive in a binary file, **/var/adm/ptpe/perflog**. The data manager nodes record their own performance data as well as the summary data they calculate for their reporting group.

PTPE gives you remote access to archive files through its C library (**libptpe.a**). You can also dump archive files in a text format using the **ptpedump** command. This tool, used in conjunction with the Performance Toolbox **a2ptx** command, allows you to create recording files that can be used by the **xmperf** and **azizo** applications.

While performance information summarization must be active in order to record performance information to the archive, neither performance summary nor performance archiving need be active to retrieve performance information from the archive. The only requirement is that the archive file **/var/adm/ptpe/perflog** must be present on the target node.

Note: The binary data in the archive is not directly accessible by the Performance Toolbox monitoring applications (**xmperf**, **3dmon**) or its programming libraries (**Spmi**, **Rsi**). Therefore, the information stored in the performance information archives cannot be displayed in a run-time fashion.

In order to display archived data through **xmperf**, or **3dmon**, it must first be converted to text format, then processed with the **a2ptx** tool, after which it can be loaded as a recording file.

Behavior Of The Performance Information Archive

Once archival is started, data is recorded in the **/var/adm/ptpe/perflog** file until one of the following events occurs:

1. All performance statistics are restricted from recording by an application using the PTPE programming library (**libptpe.a**). When all information is restricted, no information is written to the archive. However, the performance data manager continually checks whether any statistics have been selected for recording until archiving is shut off. This deprives the rest of the system of resources, therefore it should only be used to pause the archiving process.

2. The filesystem containing the **/var** directory has less than 5% of free space available. In this event, the performance data manager does not write to the archive. However, the performance data manager continually checks for the availability of free space, and resumes data recording when more than 5% of the total filesystem space becomes free. Information that would have been written to the performance archive during the interim is lost.
3. Archiving is shut down by the user.

The **/var/adm/perflog** file is created when performance data archiving is first started. Subsequent archiving sessions append more data to the file. In order to erase the **/var/adm/perflog** file, you must execute the **ptpectrl -e** command.

Archive Control

By default, the PTPE performance data manager records all available performance statistics for each node in that node's archive. This may may result in the archival of more information than you care to retain:

- The more information being retained, the greater the load on the system to record the information
- The more information, the more disk space consumed

The **ptpectrl** command provides a statistics configuration file that you can use to control the recording of specific performance statistics to the archive. The PTPE API library provides a set of control interfaces that allow applications the same function:

PtpeArchEnableStats	Identifies statistics that are to be recorded to the performance information archive
PtpeArchDisableStats	Identifies statistics that are not to be recorded to the performance information archive.
PtpeQueryAvailStats	Lists statistics available to be archived.
PtpeArchQueryStats	Identifies statistics currently being recorded to the performance information archive.

The PTPE API library contains interfaces that allow you to control the archival of specific performance statistics. The PTPE API library does not provide an interface to remove information from the performance archive once it has been recorded. Once a statistic is recorded, it remains in the archive until the archive is cleared.

When a statistic is disabled from the recording process, that statistic is not included in any subsequent update to the archive, however, previous recordings of that statistic remain.

Recording to the Archive

This procedure includes examples of how to use archiving on a system with two frames consisting of eight wide nodes each, the same as in the procedure for “Collecting and Displaying Summary Data” on page 31.

Step 1: Make Sure You're A Valid User

The PTPE performance data manager can only be started by users in the **perfmon** user group. Ensure that your userid is a member of the **perfmon** user group. Also, be sure that you have executed the following command to make **perfmon** your effective group.

```
newgrp perfmon
```

Step 2: Ensure that Data Collection and Summary is Active

You can determine the status of data collection from SP Perspectives, or using the **ptpectrl** command: (See “ptpectrl” on page 88 for details.)

```
ptpectrl -q
```

The following messages are returned:

```
ptpectrl: Performance information collection is active
ptpectrl: Performance information archiving is not active.
ptpectrl: Current performance information reporting frequency is 15 seconds.
ptpectrl: Current performance information recording frequency is 60 seconds
ptpectrl: Command completed.
```

If collection is not active, start it by using the procedure in “Step 4: Start Data Collection and Summary” on page 32. If collection is active continue to Step 3.

Step 3: Start Recording to Archive

Instruct members of the monitoring hierarchy to begin archiving performance data, using SP Perspectives or the **ptpectrl** command: (See “ptpectrl” on page 88 for details.)

```
ptpectrl -r
```

Like collection startup, archiving starts in two phases: start and enablement. The following messages are returned:

```
ptpectrl: Starting archiving of performance information.
ptpectrl: Reply from the Central Coordinator expected within 270 seconds. OK.
ptpectrl: Performance information archiving successfully started.
```

The command then attempts to enable all statistics for archiving, or enables and restricts variables for archiving (if the configuration file was found). The output from a successful enablement/disablement is:

```
ptpectrl: Enabling statistics for performance information collection.
ptpectrl: Reply from the Central Coordinator expected within 270 seconds. OK.
ptpectrl: Enabling and restriction of statistics complete.
ptpectrl: Command completed.
```

Step 4: Shut Down Archive Recording

Once you have all the performance information you wish to retain, stop the archiving process to conserve system resources. You can stop archiving using SP Perspectives, or the **ptpectrl** command: (See “ptpectrl” on page 88 for details.)

```
ptpectrl -t
```

The following messages are displayed:

```
ptpectrl: Reply from the Central Coordinator expected within 270 seconds. OK.
ptpectrl: Reply from the Central Coordinator expected within 270 seconds. OK.
ptpectrl: Performance information archiving successfully stopped.
ptpectrl: Command completed.
```

If you no longer need performance data summaries calculated, you can stop both the archiving and collection actions by issuing:

```
ptpectrl -s
```

Formatting a Text Version of the Archive

You can convert the data archive to a text format usable in reports, or as input to other AIX commands. The performance data manager provides two methods for obtaining a text version of the archive:

1. The **ptpedump** command instructs one or more nodes in the monitoring hierarchy to generate a text file, containing the performance information found in the performance data archive. The file is written in the **/var/adm/ptpe** directory, to a file named **perflog.txt**<*today*>, where <*today*> is the date the command was executed in month-day-year format, for example **perflog.txt100195** for October 1, 1995.

   Since this command places the file in **/var/adm/ptpe**, there must be enough room in the **/var** directory on the nodes to accept the output.

2. The **spdm_dump** command executes only on the local node, and writes a text formatted version of the performance archive for the local node to standard output.

Method 1 - ptpedump

These commands can be used to create a text file containing the archived data for two nodes in a reporting hierarchy, spnode09.ibm.com and spnode13.ibm.com. Before creating the text file, enter the following on the Control Workstation to ensure that there is enough space to accept the text file in the **/var** filesystem.

```
dsh -w spnode09.ibm.com,spnode13.ibm.com df /var
```

A display similar to the following appears:

```
spnode09.ibm.com: Filesystem    512-blocks      Free %Used    Iused %Iused Mounted on
spnode09.ibm.com: /dev/hd9var         8192      4195   40%      442    44% /var
spnode13.ibm.com: Filesystem    512-blocks      Free %Used    Iused %Iused Mounted on
spnode13.ibm.com: /dev/hd9var         8192      5336   35%      443    44% /var
```

Now create the text file by entering:

```
ptpedump -n spnode09.ibm.com spnode13.ibm.com
```

The following message is displayed:

```
ptpedump: command has completed.
```

The **/var/adm/ptpe/perflog.txt100195** file is created on both the spnode09.ibm.com and spnode13.ibm.com nodes.

Method 2 - spdm_dump

The following uses pipes to generate a report from performance information archive of the *PagSp/%totalfree* statistic for the spnode09.ibm.com node. This command must be executed on spnode09.ibm.com.

```
spdm_dump | grep PagSp/%totalfree | awk '{printf("%s %s %s %s %s: %s\n", $1, $2, $3, $4, $5, $10)}'
```

Selecting Performance Statistics to Archive

By default, PTPE instructs all nodes in the monitoring hierarchy to make all performance information available for archival. Therefore, all performance information that can be detected by the **xmpeek -l** command can be archived.

If you have no need to archive all performance statistics you can limit archiving to a smaller set of statistics to reduce the processing load and the amount of disk space required to store the archive using the **ptpectrl** command or the PTPE API. “Controlling Performance Data Archival” on page 75 describes how.

Obtaining Specific Performance Data from the Archive

The preferred method for gaining access to data in the performance archive is through the PTPE C programming library (**libptpe.a**). This method does not require additional disk space for a text dump of the performance information archive, nor does it require you to log into individual nodes to pipe the archive through other commands. However, it does require that you write an application for access to specific performance information in the archive.

Refer to “Obtaining Performance Data” on page 77 for more information.

Using the ptpe.cf File

This file indicates which performance statistics should be collected by the data manager and central coordiantor nodes, and used to calculate summary statistics. It also specifies which statistics should be written into a node's performance information archive file when archiving is active. The format of the file permits four possible instructions for the treatment of a statistic:

1. Collect but do not archive
2. Archive but do not collect
3. Collect and archive
4. Ignore

For samples of the **ptpe.cf** file, refer to “ptpe.cf Samples” on page 42.

The **ptpe.cf** is read by the **ptpectrl** command whenever performance information collection or performance information archiving is started (whenever the **-a**, **-c**, and **-r** options are used). The file is also read by the **ptpectrl** command whenever the **-m** option is issued.

This file is ignored by the **ptpectrl** command whenever the **-n** option is used. A replacement for this file is read when the **-l** option of the **ptpectrl** command is invoked. Refer to “ptpectrl” on page 88 for explanations of these flags and further information.

The PTPE programming library does not make use of this file or its format.

How PTPE Uses ptpe.cf

The **ptpectrl** command searches for a file named **ptpe.cf** whenever any of the aforementioned conditions exist in the user's $HOME directory. If the file is not located in that directory, the command searches for the file in the **/etc/perf** directory. If the file cannot be located in either directory, all available performance information is collected by the data manager and the central coordinator nodes, and it is also written to the performance information archive files on each node.

If a **ptpe.cf** file (or replacement) is located, any statistics that are not listed in the file are not collected by the data manager nodes and central coordinator nor are they written to the performance information archive files.

If the statistics configuration file instructs the data manager node and the central coordinator not to collect a statistic for averaging, the averaged statistics that are located in the PTPE_sum/ contexts on those nodes are not updated. However, the contexts for these statistics WILL STILL EXIST. The user must bear this fact in mind. Tools such as **xmperf** permit you to bring up instruments to display these statistics, but since the data manager nodes and central coordinator were instructed to ignore this statistic, the instrument is not updated, and displays only a constant value for that statistic.

Specifying Statistic Entries in ptpe.cf

The PTPE statistics configuration file follows a specific format. Entries that do not conform to the format rules will be ignored.

Any statistics that are to be collected, archived, or both, must have entries in the statistics configuration file. Statistic entries must begin in the first column of a line. Leading white space is not permitted. Statistic entries use the following format:

statistic_name:collection_flag,archiving_flag

Where:

- *statistic_name* is the name of the statistic, using the IBM Performance Toolbox for AIX Spmi naming format.
- *collection_flag* indicates if the statistic will be collected by the data manager and central coordinator nodes. This field may have a value of 0 (zero = do not collect) or 1 (one = collect).
- *archiving_flag* Indicates if the statistic will be written into a node's performance information archive file. This field may have a value of 0 (zero = do not archive) or 1 (one = archive).

Blank space is not permitted within a statistic entry, and comments are not permitted at the end of the entry. The values of the collection_flag and the archiving_flag can be set to different values.

Note that the specification of the collection_flag and the archiving_flag are optional. If these flags are not provided, the statistic is collected by the data manager and the central coordinator nodes, and is written to each node's archiving file.

Using Wildcards

A wildcard, denoted by the character * (asterisk), may be placed within the *statistic_name* field of a statistic entry. This will permit the entry to match all known statistics of a specific name pattern on a node.

Only one wildcard is permitted per statistic entry.

The *collection_flag* and *archiving_flag* fields cannot be wildcards.

Writing Comments

Any line within the statistics configuration file beginning with the character # (pound sign) is recognized as a comment.

Leading blank space is not permitted on comment lines. Comments are also not permitted on statistic entry lines.

Blank Lines

Blank lines are permitted within the statistic configuration file. However, blank space is not permitted in statistic entries.

ptpe.cf Samples

These lines illustrate how to use the **ptpe.cf** file. They can be found in **/usr/lpp/ptpe/samples/ptpe.cf**.

Selecting Statistics to Archive

The following lines instruct all nodes to archive specific performance statistics, and instruct the data manager and the central coordinator nodes to collect these statistics for averaging. Although performance statistics not specified in the configuration file are still listed, zero values are archived for them.

```
Mem/Virt/pagein
Mem/Virt/pageout
Mem/Kmem/mbuf/failures
```

The following lines use full notation to instruct all nodes to archive specific performance statistics, and instruct the data manager and the central coordinator nodes to collect these statistics for averaging.

```
LAN/tok0/bytesout:1,1
LAN/tok0/bytesin:1,1
LAN/tok0/framesout:1,1
LAN/tok0/framesin:1,1
```

Collecting without Archiving

These entries instruct the data manager and the central coordinator nodes to collect statistics, but the nodes are not to write them to the performance information archive files.

```
Proc/runque:1,0
Proc/pswitch:1,0
```

Archiving without Collecting

These entries instruct all nodes to write these statistics to their performance information archive files, but also instruct the data manager and central coordinator nodes not to collect these statistics for averaging.

```
IP/NetIF/tr0/ipacket:0,1
IP/NetIF/tr0/opacket:0,1
```

Excluding Statistics

This entry restricts a specific statistic from both collection and archiving. This might be used when a statistic is being disabled for a specific run of PTPE, but will be reactivated in a later run.

```
Disk/hdisk1/busy:0,0
```

Using Wildcards

This example shows how a wildcard could be used to identify statistics for all available paging devices.

```
PagSp/*/size:1,0
PagSp/*/%free:1,1
```

The following line shows how a wildcard could be used to identify any statistic ending with a specific field. Because this entry has not provided any collection or archiving flags, any statistics matching this pattern will be collected by data manager and central coordinator nodes, and is also written to the performance information archive files on all nodes.

```
*/rcverrors
```

All CPU related statistics can be identified using this wildcard format.

```
CPU/*:0,1
```

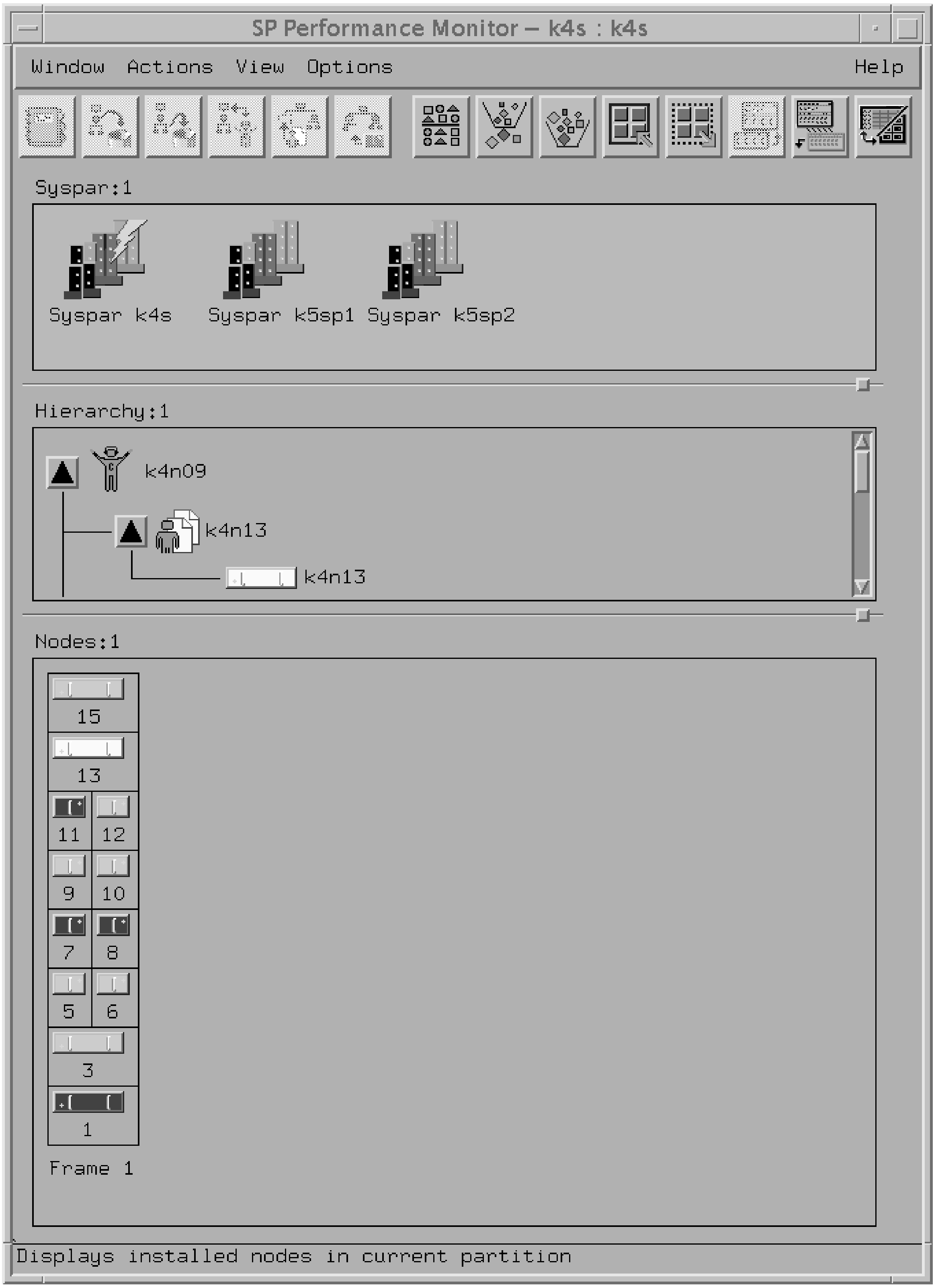
Using SP Perspectives

This section provides an overview of the Performance Monitoring Perspective. For general information regarding all of SP Perspectives, see the *IBM Parallel System Support Programs for AIX Administration Guide*.

You can use the Performance Monitoring Perspective to perform most of the PTPE command set functions through point and click operations. Use the **perspectives** command to bring up the SP Perspectives Launch Pad, then double-click on the SP Performance Monitor icon. You can also start the Performance Monitoring Perspective directly, without initializing the SP Perspectives Launch Pad, by entering

```
spperfmon
```

When starting the Performance Monitoring Perspective, a window similar to the following one appears:



Displays installed nodes in current partition

Figure 6. The Performance Monitoring Perspective GUI

The default Performance Monitoring Perspective window uses three panes to display SP system information:

1. *Hierarchy pane*: This shows the current monitoring hierarchy, displaying the central coordinator at the top, with data manager nodes below and reporter nodes at the bottom. By default, the monitoring hierarchy stored in the SDR is displayed when this perspective is initialized.

   This workspace is your monitoring hierarchy editor. The current pane, selected by clicking in the mouse anywhere in a pane area, appears lighter in color than the other panes.

2. *Syspar pane*: This shows how the SP is partitioned. The current system partition (distinguished by a lightening bolt in the icon) is the one that displays its objects in the Hierarchy and Nodes panes. If other partitions are defined for the SP, you can use this pane to select them.

3. *Nodes pane*: This shows the nodes in the current SP system partition, organized by frame in the default display, but you can sort and filter them to suit your purposes.

Like other SP Perspectives, this one has a *Menu Bar*, a *Tool Bar*, and, below the *panes*, an *Information Area*, where a line of text is displayed that describes Tool Bar icon functions and Menu Bar actions, depending on where you point the mouse.

For specific information regarding the icons appearing in the panes, the menu bar, or the tool bar, refer to the online information for this Perspective.

Chapter 6. Using the PTPE Application Programming Interface

The Performance Toolbox Parallel Extensions Application Programming Interface library (PTPE API), located in **libptpe.a**, is the preferred means for access to statistical information from the performance information archive. The library provides subroutines that can retrieve specific performance information from the archives maintained by one or more specified nodes without requiring the application to make direct connections to these nodes. The PTPE API includes a set of subroutines that permit your program to control data collection, summarization, and archival on each node in the monitoring hierarchy. With these subroutines, your application can process performance statistics relevant to current and future monitoring efforts, and ignore irrelevant information. This can significantly reduce the operational overhead of performance monitoring, as well as reduce the amount of network traffic generated by PTPE and the amount of disk space necessary to contain the performance data archives.

The PTPE API supports C language programs. To use the API, your program should include the <**spdm.h**> header file, which provides all the necessary data types and functional prototypes, as well as a definition of all the error codes used by the API. The application must link with shared library **libptpe.a** during compilation by using the **-lptpe** compiler directive.

When executing PTPE API control functions, **libptpe.a** establishes a socket connection with the central coordinator node of the monitoring hierarchy. Once the connection is established:

1. The library transmits the control command to the central coordinator node, along with the list of nodes that are expected to carry out the command
2. The central coordinator node determines which nodes must be alerted by examining the monitoring hierarchy and finding the data manager nodes responsible for the nodes specified in the API command
3. The central coordinator node forwards the command to these data manager nodes, which, in turn, forward it to the target nodes
4. The target nodes perform the operation and report their results to the data manager nodes
5. The data manager nodes collect all results and pass them to the central coordinator node
6. The central coordinator node assembles all responses, and gives the complete results to the PTPE API library through the socket

With this strategy, the application need only establish a network connection to the central coordinator node, instead of establishing connections to all nodes involved in the API request.

The PTPE API contains subroutines for:

Controlling API Sessions	Establishes and ends API session
Handling Data Types	Manipulates host and statistical data structures

Controlling Performance Data	Controls collection and archival of performance data
Obtaining Performance Data	Deliver performance data to application program

An API session is required only when an application is requesting performance data or indicating what data should be collected or archived through the API. From the application's point of view, a session is the time between the **PtpeOpenSession** and **PtpeCloseSession** calls.

API requests for performance data that are made by an application that does not currently have an open session will fail.

The examples in this chapter do not contain sufficient code for execution, nor are they intended as a tutorial. They are included only to illustrate how the PTPE subroutines are called.

Controlling Sessions

An PTPE API session is established by an application whenever the application needs to use or control performance information. When establishing an API session, the application locks critical information in the System Data Repository (SDR). In this way it ensures that no other applications can alter the monitoring hierarchy structure or change the current status of performance information collection or archiving, which could cause the application to fail or obtain invalid results. API sessions can only be established by applications with the appropriate permissions. Refer to “The perfmon User Group” on page 14 for SDR information.

Because a session cannot be established without locking information in the System Data Repository, only one session can be active in the entire system partition at any time. Therefore, an application should only obtain a session when one is required, and it should release the session when it is no longer needed.

The following subroutines are provided for API session control:

Subroutine Name	**Page**	**Purpose**
PtpeOpenSession	277	Establishes a session, permitting the caller access to archived performance information and configure statistics for collection or archiving.
PtpeCloseSession	176	Releases a session, relinquishing the application's access to archived performance information and configuration of statistics.

Termination Signals

All PTPE API applications should set up a handler for termination signals. The purpose of this handler is to close the PTPE API session if the process is prematurely terminated by the user or the system. If such a handler is not set up, the application can leave the PTPE status information in an incorrect state if it is interrupted or killed, and possibly prevent other applications or PTPE commands from executing properly.

At a minimum, a handler should be set for the SIGHUP, SIGINT, SIGQUIT, and SIGTERM signals. These are the most common signals sent by users to terminate a process. We recommend adding the SIGABRT, SIGBUS, SIGSEGV, SIGSYS, SIGXCPU, and SIGXFSZ to the set of signals to be handled by the routine, to allow for possible errors in the application that would cause termination of the application by the operating system.

The following code segment demonstrates how and application could establish a session with the **PtpeOpenSession** subroutine, and close it with the **PtpeCloseSession** subroutine. This example also demonstrates how a signal handler could be used to relinquish the PTPE API session when a termination signal is received.

```
#include <stdio.h>                          /* Basic I/O Capabilities */
#include <spdm.h>                           /* PTPE data types        */
#include <sys/signal.h>                     /* Signals supported      */

session_ptr_t       sblock;

/*
 * Signal handling function - close the PTPE session before exiting.
 */
void
sighdl(int sigval)
{
    char                        *signame;
    extern session_ptr_t        sblock;

    psignal(sigval, signame);
    printf("Terminating because signal %s was received\n", signame);
    free(signame);
    if (sblock != (session_ptr_t) NULL) {
            if (PtpeCloseSession(sblock) != PTPE_SUCCESS) {
                    printf("Cannot close session\n");
            }
    }
    exit(sigval);
}

/*
 * Mainline code
 */
main(int argc, char **argv)
{
    int                        rc;
    extern session_ptr_t        sblock;
    extern void                sighdl();
    sblock = (session_ptr_t) NULL;

    /* Set up signal handling function */
    signal(SIGHUP, sighdl);
    signal(SIGINT, sighdl);
    signal(SIGQUIT, sighdl);
    signal(SIGTERM, sighdl);
    signal(SIGABRT, sighdl);
    signal(SIGBUS, sighdl);
    signal(SIGSEGV, sighdl);
    signal(SIGSYS, sighdl);
    signal(SIGXCPU, sighdl);
    signal(SIGXFSZ, sighdl);

    /* establish session to run the PTPE command */
    rc = PtpeOpenSession(&sblock);
    switch (rc) {
        case PTPE_SUCCESS:    break;
        case PTPE_RONLY_SESS: printf("Read-only session. ");
                              printf("Cannot issue control APIs\n");
                              break;
        case PTPE_AUTH:       printf("Not authorized.\n");
                              return(1);
        case PTPE_LOCKED:     printf("Another appl running.\n");
                              return(1);
        default:              printf("Error getting session.\n");
                              return(1);
}
                           :
                           :
/* perform PTPE functions */
                           :
                           :
/* PTPE functions done - release session */
rc = PtpeCloseSession(&sblock);
switch (rc) {
    case PTPE_SUCCESS:    break;
    case PTPE_NO_SESSION: printf("No session to close!\n");
                          return(1);
    default:              printf("Error closing session.\n");
                          return(1);
}
/* session now dropped */
                           :
                           :
return(0);
}
```

Regular and Read-Only Sessions

When a regular session is established, **libptpe.a** gains locks on the PTPE data objects in the System Data Repository. This implies that the shared library establishes a session between itself and the SDR. While the session is held, the locks on the data objects are also held, preventing other utilities from updating them (such as the **ptpehier** and **ptpectrl** commands), and also preventing other PTPE API applications from establishing a session.

Regular sessions should be held as long as the application requires the monitoring hierarchy structure to remain consistent, and also as long as the application needs to exercise sole control over performance information collection and archiving. However, when coding the application, remember that other PTPE API applications on this or any other node in the monitoring hierarchy will not be able to obtain a session until the session is released by this application.

Under certain conditions, the **PtpeOpenSession** routine will return a code of PTPE_RONLY_SESS, indicating that a read-only session was established by the routine. This condition is likely to occur when an application, running on a node in one SP system partition, sets its environment to reference another SP system partition and then attempts to open a PTPE session. When a read-only session is established, the holder of the session does not lock the PTPE data objects in the System Data Repository, but instead has read-only access to the information. With such a session, PTPE cannot guarantee that the monitoring hierarchy will remain in a consistent state, or that another application will not issue controls upon the hierarchy. Therefore, the application holding a read-only session can only issue a limited set of queries.

Sessions should always be explicitly terminated with the **PtpeCloseSession** subroutine before the application exits.

Handling Data Types

The PTPE API provides four data types to contain system information and performance statistic information:

1. Hosts
2. Statistics
3. Host lists
4. Statistics lists

The term *host* is used whenever the API is dealing with system information, and the term *statistic* is used when the API is dealing with performance statistic information. **A host has the following properties:**

Name	The name by which the host is known to the network.
Result Code	Indicates how successfully the host performed a previous API request.

Statistics have a different set of properties:

Name	The absolute path name by which the statistic is known to Performance Toolbox.
Type	Indicates whether the statistic is an integer, or floating point value.
Value	The statistical value.
Timestamp	Indicates when the value was observed.
Result Code	Indicates how successfully a previous API subroutine handled the statistic.

Hosts and statistics are organized into groups of host lists and statistics lists. These lists also have unique properties:

Anchor Point Indicates where the list resides in memory.

Current Entry Pointer Indicates which member of the list will be processed by the next data manipulation subroutine, and can be set to reference various list members by other data manipulation routines.

Many PTPE subroutines perform operations on host lists containing one or more hosts. Some of these hosts must also have statistics lists assigned to them whenever they use a subroutine that enables, disables, or retrieves lists of statistics.

These four data types have the following relationships:

- Statistics are added to statistic lists
- Hosts are added to host lists
- Statistic lists are assigned to hosts, once a host is assigned to a host list
- Information control or access functions are performed on host lists

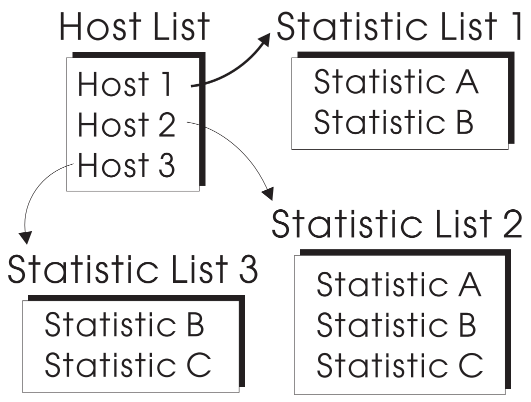


Figure 7. Data Type Relationships

To ensure that all data types are used properly, the PTPE API provides a series of data handling subroutines that are designed to manipulate the data structures that contain statistic and host information. Your application should use these interfaces when manipulating the data types. Do not manipulate the data types directly.

Subroutine Name	**Page**	**Purpose**
PtpeAddHostToList	114	Adds a new host to a host list.
PtpeAddStatToList	116	Adds a new statistic to the statistic list.
PtpeAssignStatsToHost	170	Associates a statistic list with a specific host in the host list.
PtpeDelHostFromList	225	Removes a host from a host list.
PtpeDelStatFromList	227	Removes a statistic from a statistic list.
PtpeEmptyHostList		Clears out a host list.
PtpeEmptyStatList	231	Clears out a statistic list.

Subroutine Name	Page	Purpose
PtpeFindHost	233	Checks if a certain host is specified in a host list, and prepares the host list to accept input for that host.
PtpeFindStat	235	Checks if a certain statistic is specified in a statistic list, and prepares the statistic list to accept input for that statistic.
PtpeFirstHost	237	Prepares host list for operations on the first host specified in the host list.
PtpeFirstStat	239	Prepares the statistic list of the currently referenced host for operations on the first statistic specified in the statistics list
PtpeFreeHostList	241	Deallocates a host list and its anchor point.
PtpeFreeStatList	243	Deallocates a statistics list and its anchor point.
PtpeGetHost	245	Retrieves the name of the currently referenced host in the host list.
PtpeGetHostResult	247	Determines the success or failure of a host to the previously executed API subroutine.
PtpeGetHostStatList	250	Retrieves the pointer to the currently referenced host's statistic list, so this list may be used in subsequent **PtpeAddStatToList**, **PtpeDelStatFromList**, **PtpeFindStat**, **PtpeAssignStatsToHost**, and **PtpeFreeStatList** calls.
PtpeGetStatName	252	Retrieves the name of the currently referenced statistic in the currently referenced host's statistic list.
PtpeGetStatResult	254	Determines if the previously executed API subroutine was successful in handling the specified statistic.
PtpeGetStatTime	256	Retrieves the timestamp value for the currently referenced statistic in the host's statistic list. Timestamp is returned in a struct tm format.
PtpeGetStatType	258	Retrieves the data type of the currently referenced statistic in the currently referenced host's statistic list.

Subroutine Name	Page	Purpose
PtpeGetStatValueFloat	261	Retrieves the value of the currently referenced statistic in the currently referenced host's statistic list. This subroutine can only be used on floating point statistics.
PtpeGetStatValueLong	263	Retrieves the value of the currently referenced statistic in the currently referenced host's statistic list. This subroutine can only be used on long integer statistics.
PtpeInitHostList	265	Allocates an empty host list for the caller. This list identifies the hosts involved with subsequent API calls, and associates a list of statistics to each host in the list.
PtpeInitStatList	267	Allocates a statistic list for the caller. This list identifies the statistics that are to be used in conjunction with one or more hosts in the host list during subsequent API calls.
PtpeIsLastHost	269	Checks if there are any more hosts in the host list after the currently indicated host.
PtpeLastStat	271	Checks if there are any more statistics in the statistic list after the currently referenced statistic
PtpeNextHost	273	Prepares host list for operations on the subsequent host specified in the host list.
PtpeNextStat	275	Prepares statistic list of the currently referenced host for operations on the next statistic in the statistics list
PtpeRemoveStatsFromHost	298	Disassociates the specified host's statistic list in the host list from that host.
PtpeSetStatTime	300	Sets the timestamp value for the currently referenced statistic in the host's statistic list.
PtpeStatIsFloat	303	Determines if the currently referenced statistic is of the data type PTPE_FLOAT
PtpeStatIsLong	305	Determines if the currently referenced statistic is of the data type PTPE_LONG.

Many of the subroutines that set system or statistic values within these data types, or retrieve system or statistic information from these data types, deal with the current item in the respective list. Each list contains an anchor, which indicates the location of the list in memory, as well as a pointer to the current item in the list.

Assignment and retrieval operations use the current item pointer to locate the member of the list that is the target of the operation. Whenever an item is inserted into either a statistics list of a host list, the current item pointer is updated to reference the newly added item. Whenever items are removed from either list, the current pointer is reset to the beginning member of the list. If either list type contains an empty list, the current item pointer references nothing.

To set or retrieve information from list items other than the latest added item or the start of the list, the PTPE API provides navigation routines to change the current pointer setting for each list type.

- PtpeFirstHost
- PtpeFirstStat
- PtpeNextHost
- PtpeNextStat
- PtpeFindHost
- PtpeFindStat

Two subroutines are also provided to determine whether the current item pointer is referencing the last item in the list:

- PtpeIsLastHost
- PtpeIsLastStat

Your applications must ensure that the host lists and statistics lists it uses have properly positioned their current item pointers before using subroutines that set or retrieve information in these lists. If care is not taken to ensure this, erroneous results can easily occur.

Manipulating Host Lists

Before nodes can be added to a host list, a host list must first be created with the **PtpeInitHostList** subroutine. Once the host list is created, you may add nodes to it using the **PtpeAddHostToList** subroutine, by specifying the name of the node. It is important that the name you specify for the node be identical to the name identifying it in the monitoring hierarchy. For example, if the monitoring hierarchy recognizes a node by its fully qualified host name (such as spnode07.ibm.com), this host name must be used when adding the node to the host list. If the host is also known by another name (for example, a system is known both as spnode20 and batch3), the name used must be the name by which the monitoring hierarchy recognizes it. The names and formats must match exactly, or the API will conclude that the requested node does not exist when an API command is executed on the host list (the error will not be reported when the host is added to the list).

On those occasions when an operation is not to be performed on all the nodes contained in a host list, the systems that are not to receive the command can be removed from the host list with the **PtpeDelHostFromList** routine. The entire contents of a host list can be cleared with the **PtpeEmptyHostList** routine, for occasions when an entirely different set of nodes will be targeted for a PTPE API command.

When all functions on nodes in the monitoring hierarchy are complete, the host list should be freed using the **PtpeFreeHostList** routine. This returns the memory reserved for the host list memory anchor back to the system.

Adding Nodes to a Host List

The following code segment demonstrates how three nodes can be added to a host list, which will be used in later PTPE API operations. It also shows one of the nodes being removed from the host list to prevent later PTPE API commands from executing on that system.

```
#include <stdio.h>            /* basic I/O capability      */
#include <spdm.h>             /* PTPE data types           */

static char *sysnames[] = {
   "spnode04.ibm.com",
   "spnode12.ibm.com",
   "spnode23.ibm.com"
};
static int num_hosts = 3;

main(int argc, char **argv)
{
  session_ptr_t    sblock;    /* PTPE session control info */
  host_list_t      targets;   /* Where commands will run   */
  int              rc;        /* Ftn call return code      */
  int              i;         /* Loop counter              */
                 :
                 :
  rc = PtpeOpenSession(&sblock);

                 :
                 :
  /*
   * Set up a host list for future PTPE API commands. Hosts
   * not in this list will not receive the API command, even
   * if they're in the hierarchy.
   */
  targets=(host_list_t)NULL;
  rc = PtpeInitHostList(&targets);
  if (rc != PTPE_SUCCESS) {
    printf("Cannot get a host list.\n");
    rc = PtpeCloseSession(&sblock);
    return(1);
}
/* add hosts to list */
for (i = 0 ; i < num_hosts ; i++) {
  rc = PtpeAddHostToList(sysname[i], targets);
  if (rc != PTPE_SUCCESS) {
    printf("Cannot add %s to host list\n", sysname[i]);
    rc = PtpeFreeHostList(&targets);
    if (rc != PTPE_SUCCESS) {
      printf("Error freeing host list\n");
    }
    rc = PtpeCloseSession(&sblock);
    return(1);
  }
}
```

```
                  :
                  :
/* Perform PTPE API operations on all three systems */
                  :
                  :
/*
 * "spnode12.ibm.com" should not be included in any of
 * following API subroutines.  Remove it from the host list.
 */
rc = PtpeDelHostFromList(sysnames[1], targets);
switch (rc) {
  case PTPE_SUCCESS:        break;
  case PTPE_HOST_NOT_FOUND: printf("%s not in host list\n",
                                    sysnames[1]);
                            rc = PtpeFreeHostList(&targets);
                            rc = PtpeCloseSession(&sblock);
                            return(1);
  default:                  printf("Can't remove %s\n",
                                    sysnames[1]);
                            rc = PtpeFreeHostList(&targets);
                            rc = PtpeCloseSession(&sblock);
                            return(1);
}
                :
                :
/* Perform remaining PTPE API operations */
                :
                :
 rc = PtpeFreeHostList(&targets);
 if (rc != PTPE_SUCCESS) {
   printf("Error freeing host list\n");
}
 rc = PtpeCloseSession(&sblock);
                :
                :
 return(0);
}
```

Obtaining a List of Available Nodes

To eliminate the need for all PTPE API applications to know all the nodes in the monitoring hierarchy, the PTPE API provides the **PtpeQueryAvailHosts.** subroutine. This subroutine accepts an initialized host list as an argument, and seeds this host list with all the nodes known in the monitoring hierarchy. The **PtpeDelHostFromList** subroutine can then be used to remove nodes from the list that will not be used in subsequent API commands. Unlike other data manipulation subroutines, **PtpeQueryAvailHosts** requires a PTPE API session to be active, in order to retrieve the current monitoring hierarchy structure.

```
#include <stdio.h>            /* basic I/O capability      */
#include <spdm.h>             /* PTPE data types           */

main(int argc, char **argv)
{
   session_ptr_t    sblock;    /* PTPE session control info */
   host_list_t      targets;   /* Where commands will run   */
   int              rc;        /* Ftn call return code      */
   int              i;         /* Loop counter              */
                   :
                   :
   rc = PtpeOpenSession(&sblock);

                   :
                   :
 /*
  * Set up a host list for future PTPE API commands. Hosts
  * not in this list will not receive the API command, even
  * if they're in the hierarchy.
  */
 targets=(host_list_t)NULL;
 rc = PtpeInitHostList(&targets);
 if (rc != PTPE_SUCCESS) {
   printf("Cannot get a host list.\n");
   rc = PtpeCloseSession(&sblock);
   return(1);
 }
 /*
  * Get list of all available systems.  Remove "spnode13"
  * from the list, since it won't be involved in future API
  * subroutines.
  */
  rc = PtpeQueryAvailHosts(sblock, &targets);
  if (rc != PTPE_SUCCESS) {
    printf("Cannot get list of systems.\n");
    rc = PtpeFreeHostList(&targets);
    rc = PtpeCloseSession(&sblock);
    return(1);
  }
  rc = PtpeDelHostFromList("spnode13.ibm.com", targets);
                  :
                  :
  rc = PtpeFreeHostList(&targets);
  rc = PtpeCloseSession(&sblock);
  return(0);
}
```

Once a host list is constructed and any necessary statistics lists are assigned to the nodes in the host list (see “Navigating the Host List” on page 59), PTPE API operations can be performed on these systems. When an API command has been completed, each node will record its results, such as “completed,” “error,” “couldn't find,” etc., in its entry in the host list. To determine how a node responded to an API command, the host list must be scanned, and the results checked in each host list entry.

Navigating the Host List

The PTPE API provides subroutines for navigating the host list, and for checking the results of the previously executed API command. The following code segment demonstrates how an application would determine a node's response to an API request. For the sake of brevity, the error handling code has been omitted.

```
                  :
                  :
host_list_t targets;        /* Where commands will run     */
char hostname[PTPE_NMLN];   /* Name of system in host list */
int  result;                /* How system did in API call  */
                  :
                  :
  /* check results of API command */
  rc = PtpeFirstHost(targets);
  for(;;) {
    /* get name of host and its result code */
    rc = PtpeGetHost(hostname, targets);
    rc = PtpeGetHostResult(targets, &result);
    printf("%s result was %d\n", hostname, result);
                  :
                  :
    /* perform some processing based on the results */
                  :
                  :
    /* are we at the last host in the list? */
    rc = PtpeIsLastHost(targets);
    if (rc == PTPE_TRUE) {
      break;
    }
    /* more hosts to go - move onto next one */
    rc = PtpeNextHost(targets);
  }
                  :
                  :
```

Manipulating Statistics Lists

Before statistics can be added to a statistics list, one must first be created with the **PtpeInitStatList** subroutine. Statistics may then be added to the statistics list through the **PtpeAddStatToList** routine, by specifying the name of the statistic. The names used are full Performance Toolbox statistical context names, relative to the Top context on a system (see the discussion on “System Performance Measurement Interface Overview for Programming” in the *IBM Performance Toolbox for AIX: Guide and Reference* for the format of statistic names). The full context path does not include the *hosts/hostname* prefix. If the statistic name is not specified in the proper format, the API will conclude that the statistic does not exist on a system, but only after an API command is executed using this statistic. (The error is not reported when the statistic is added to a statistics list.)

Removing statistics from a statistics list differs, depending on whether or not the statistics list was assigned to a host in a host list. Statistics can be easily removed from unassigned statistics lists with the **PtpeDelStatFromList** subroutine. However, removing statistics from a list that has been assigned to a host is a bit more involved.

When statistics lists are assigned to a host list through the **PtpeAssignStatsToHost** subroutine, the PTPE API library prepares a copy of the statistics list, and assigns the copy to the host. This allows the application to maintain a master statistics list, and assign the same master list to a series of hosts in a host list. Since the API copies a statistics list when it is assigned, any later modifications to the master statistics list are not reflected in the assigned statistics list. To make changes to a statistics list once it has been assigned to a host, the statistics list must be removed from the host with the **PtpeRemoveStatsFromHost** routine (which discards the statistics list), and a modified statistics list must be assigned as the new statistics list to the host.

The entire contents of an unassigned statistics list can be cleared with the **PtpeEmptyStatList** subroutine, for occasions when an entirely different set of statistics is to be assigned to hosts for future API commands. When an unassigned statistics list is no longer needed, it should be freed with the **PtpeFreeStatList** subroutine. This returns the memory reserved for the host list memory anchor back to the system. Statistics lists that are assigned to hosts in a host list are freed when the host list is freed with the **PtpeFreeHostList** subroutine.

Adding to the Statistics List

The following code segment demonstrates how statistics can be added to a statistics list, which will be assigned at a later point in the code to a host entry in an existing host list. It also shows one of the statistics being removed from the unassigned statistics list.

```
#include <stdio.h>            /* basic I/O capability      */
#include <spdm.h>             /* PTPE data types           */

static char *statnames[] = {
  "PagSp/%totalfree",
  "NetIF/tr0/ipacket",
  "NetIF/tr0/opacket",
  "CPU/cpu0/user"
};
static int num_stats = 4;

main(int argc, char **argv)
{
  session_ptr_t    sblock;    /* PTPE session control info */
  host_list_t      targets;   /* Where commands will run   */
  stat_list_t      slist;     /* Statistics used by API    */
  int              rc;        /* Ftn call return code      */
  int              i;         /* Loop counter              */
                  :
                  :
  rc = PtpeOpenSession(&sblock);

  rc = PtpeInitHostList(&targets);
                  :
                  :
  /* create statistics list */
  slist=(stat_list_t)NULL;
  rc = PtpeIniStatList(&slist);
  if (rc != PTPE_SUCCESS) {
     printf("Cannot set up statistics list.\n");
     rc = PtpeFreeHostList(&targets);
```

```
      rc = PtpeCloseSession(&sblock);
      return(1);
   }
   /* add statistics to our own working list */
   for (i = 0 ; i < num_stats ; i++) {
     rc = PtpeAddStatToList(statnames[i], slist);
     if (rc != PTPE_SUCCESS) {
       printf("Cannot add %s to stat list\n", statnames[i]);
       rc = PtpeFreeStatList(&slist);
       if (rc != PTPE_SUCCESS) {
          printf("Error freeing statistics list\n");
     }
     rc = PtpeFreeHostList(&targets);
     rc = PtpeCloseSession(&sblock);
     return(1);
   }
 }
                   :
                   :
   /*
    * "PagSp/%totalfree" should not be assigned to any of
    * the hosts that would follow.  Remove it from the stats
    * list.
    */
    rc = PtpeDelStatFromList(statnames[0], slist);
    switch (rc) {
       case PTPE_SUCCESS:        break;
       case PTPE_STAT_NOT_FOUND: printf("%s not in stat list\n",
                                         statnames[0]);
                                 rc = PtpeFreeStatList(slist;);
                                 rc = PtpeFreeHostList(&targets);
                                 rc = PtpeCloseSession(&sblock);
                                 return(1);
       default:                  printf("Can't remove %s\n",
                                          statnames[0]);
                                 rc = PtpeFreeStatList(&slist);
                                 rc = PtpeFreeHostList(&targets);
                                 rc = PtpeCloseSession(&sblock);
                                 return(1);
  }
                         :
                         :
  /* Perform remaining PTPE API operations */
                         :
                         :
   rc = PtpeFreeStatList(&slist);
   if (rc != PTPE_SUCCESS) {
      printf("Error freeing statistics list\n");
   }
   rc = PtpeFreeHostList(&targets);
   rc = PtpeCloseSession(&sblock);
                          :
                          :
   return(0);
 }
```

Assigning Statistics Lists to Hosts

Most API control subroutines perform operations on one or more statistics on one or more nodes in the monitoring hierarchy. In order to carry out these operations, the statistics list must be assigned to the node where the API control operation is to be carried out. Statistics lists are assigned to a node using the **PtpeAssignStatsToHost** subroutine. If a change to a statistics list that has been assigned to a host must be made, the statistics list must be removed from the node's entry in the host list by the **PtpeRemoveStatsFromHost** subroutine, and a revised statistics list is then reassigned to the host.

The following code segment demonstrates how statistics lists can be assigned to several nodes in an existing host list. For the sake of brevity, some of the error handling code has been omitted.

```
#include <stdio.h>            /* basic I/O capability      */
#include <spdm.h>             /* PTPE data types           */

static char *sysnames[] = {
  "spnode04.ibm.com",
  "spnode12.ibm.com",
  "spnode23.ibm.com"
};
static char *statnames[] = {
  "PagSp/%totalfree",
  "NetIF/tr0/ipacket",
  "NetIF/tr0/opacket",
  "CPU/cpu0/user"
};
static int num_stats = 4;
static int num_hosts = 3;

main(int argc, char **argv)
{
  session_ptr_t    sblock;    /* PTPE session control info */
  host_list_t      targets;   /* Where commands will run   */
  stat_list_t      slist;     /* Statistics used by API    */
  int              rc;        /* Ftn call return code      */
  int              i;         /* Loop counter              */
                  :
                  :
  /*
   * Create statistics list for "spnode04" operation
   */
  rc = PtpeEmptyStatList(&slist);
  if (rc != PTPE_SUCCESS) {
    printf("Cannot empty statistics list.\n");
    rc = PtpeFreeStatList(&slist);
    rc = PtpeFreeHostList(&targets);
    rc = PtpeCloseSession(&sblock);
    return(1);
  }
  rc = PtpeAddStatToList(statname[0], slist);
  if (rc != PTPE_SUCCESS) {
    printf("Cannot add %s to statistics list\n", statname[0]);
    rc = PtpeFreeStatList(&slist);
    rc = PtpeFreeHostList(&targets);
    rc = PtpeCloseSession(&sblock);
    return(1);
  }
   /*
    * Add statistics list to "spnode04".  Need to set the
    * host list to "spnode04"'s position first.
    */
  rc = PtpeFindHost(sysnames[0], targets);
  if (rc == PTPE_SUCCESS) {
     rc = PtpeAssignStatsToHost(slist, targets);
     if (rc != PTPE_SUCCESS) {
       printf("Cannot assign stats to %s\n", sysname[0]);
       rc = PtpeFreeStatList(&slist);
       rc = PtpeFreeHostList(&targets);
       rc = PtpeCloseSession(&sblock);
       return(1);
  }
 }
                 :
                 :
  /*
   * Add remaining statistics to lists for "spnode12" and
   * "spnode23". Notice that these additions take place in
   * the unassigned list, not the list assigned to "spnode04"
   */
  for (i = 1 ; i < num_stats ; i++) {
    rc = PtpeAddStatToList(statname[i], slist);
  }
  for (i = 1 ; i < num_hosts ; i++) {
    rc = PtpeFindHost(sysname[i], targets);
    rc = PtpeAssignStatsToHost(slist, targets);
  }
                :
                :
  /*
   * For some reason, we need to change the statistics that
   * "spnode12" will use in the next API command.  Need to
   * build a "correct" statistics list, remove the statistics
   * list currently assigned to "spnode12", and assign the
   * the correct list.
   */
  rc = PtpeEmptyStatList(&slist);
  rc = PtpeAddStatToList(statname[0], slist);
  rc = PtpeFindHost(sysname[1], targets);
  rc = PtpeRemoveStatsFromHost(targets);
  if (rc == PTPE_SUCCESS) {
    rc = PtpeAssignStatsToHost(slist, targets);
  }
                :
                :
}
```

When statistics lists have been assigned to nodes in a host list, the PTPE API represents them internally as a linked list of linked lists. At the conclusion of the previous code segment, the PTPE API would construct a list in memory that can be conceptually represented as shown in Figure 8 on page 64.

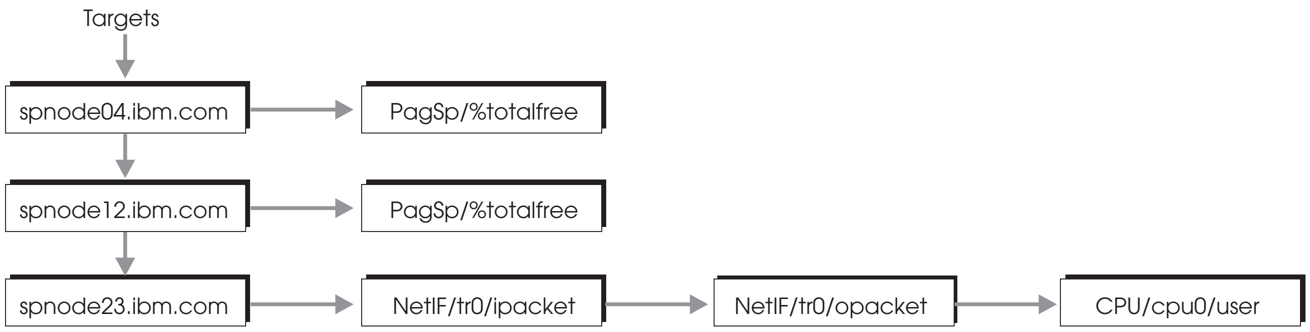


Figure 8. Statistical Lists Linked in Memory

When an API command is performed using the *targets* host list:

- The command is executed on the spnode04.ibm.com system, using only the *PagSp/%totalfree* statistic. No other statistics that might be available on spnode04.ibm.com would be involved in the command.
- The command is executed on the spnode12.ibm.com system, using only the *PagSp/%totalfree* statistic. No other statistics that might be available on spnode12.ibm.com would be involved in the command.
- The command is executed on the spnode23.ibm.com system, using the *NetIF/tr0/ipacket*, *NetIF/tr0/opacket*, and the *CPU/cpu0/user* statistics. No other statistics that might be available on spnode23.ibm.com would be involved in the command.
- The command is not executed on any other systems that might exist in the monitoring hierarchy.

Querying Statistics Lists

To avoid giving the PTPE API applications the responsibility of knowing all the performance statistics that are available on any particular node, the PTPE API provides the convenience subroutine **PtpeQueryAvailStats.** This subroutine accepts a host list as an argument, and sends a command to the Central Coordinator Node to find all the performance statistics that are available on the systems specified in the host list. The statistics lists can then be copied into an unassigned statistics list from a host list entry by using the **PtpeGetHostStatList** subroutine. The **PtpeDelStatFromList** routine can then be used to remove statistics from the list that will not be used in subsequent API commands. Unlike other data manipulation subroutines, **PtpeQueryAvailStats** requires a PTPE API session to be active, in order to retrieve the current monitoring hierarchy structure.

```
#include <stdio.h>            /* basic I/O capability      */
#include <spdm.h>             /* PTPE data types           */

main{int argc, char **argv)
{
  session_ptr_t    sblock;    /* PTPE session control info */
  host_list_t      targets;   /* Where commands will run   */
  host_list_t      reply;     /* Where answer is placed    */ |
  stat_list_t      slist;     /* Statistics used by API    */
  int              rc;        /* Ftn call return code      */
  int              i;         /* Loop counter              */
                 :
                 :
  rc = PtpeOpenSession(&sblock);
  targets = (host_list_t)NULL;
  rc = PtpeInitHostList(&targets);
  rc = PtpeQueryAvailHosts(sblock, &targets);
  reply = (host_list_t) NULL;
                  :
                  :
  /*
   * Get list of all available statistics from the nodes in
   * the host list.  The statistics list will be found in the
   * "reply" host list, which should contain entries for all
   * systems listed in the "targets" list.
   */
  rc = PtpeQueryAvailStats(sblock, targets, &reply);
  if (rc != PTPE_SUCCESS) {
    printf("Cannot get statistics lists from hosts.\n");
    rc = PtpeFreeHostList(&targets);
    rc = PtpeCloseSession(&sblock);
    return(1);
  }
                  :
                  :
  /*
   * Get the statistics list from "spnode12".
   */
  slist = (stat_list_t)NULL;
  rc = PtpeFindHost("spnode12.ibm.com", reply);
  rc = PtpeGetHostStatList(slist, reply);
  if (rc != PTPE_SUCCESS) {
    printf("Cannot get statistics list from spnode12\n");
    rc = PtpeFreeStatList(&slist);
    rc = PtpeFreeHostList(&reply);
    rc = PtpeFreeHostList(&targets);
    rc = PtpeCloseSession(&sblock);
    return(1);
  }
                  :
                  :
  rc = PtpeFreeStatList(&slist);
  rc = PtpeFreeHostList(&reply);
  rc = PtpeFreeHostList(&targets);
  rc = PtpeCloseSession(&sblock);
  return(0);
}
```

Statistic Timestamps

For specific API commands that use statistics lists, such as requests to retrieve statistics from the performance information archive, a timestamp must also be specified for each statistic in a statistics list. This is necessary for the API subroutines to locate the statistic in the system's archive. The timestamp indicates time of the desired observance of the performance statistic.

The PTPE API permits an application to specify a timestamp for a performance statistic in one of three ways:

1. By indicating a date and time. The PTPE API will attempt to find the recording of the statistic that most closely matches the specified time of day without

exceeding it. To specify a timestamp in this manner, an application would use the **PtpeSetStatTime** parameter PTPE_MATCH.

2. By requesting the earliest recording of the performance statistic. To specify this option, the PTPE_EARLIEST parameter is passed to the **PtpeSetStatTime** subroutine.
3. By requesting the last recording of a performance statistic. To specify this option, the PTPE_LATEST parameter is passed to the **PtpeSetStatTime** subroutine.

As with other statistics list manipulation routines, the **PtpeSetStatTime** function is performed only on an unassigned statistics list. Timestamps cannot be set on assigned statistics lists; if the timestamps on an assigned statistics list are incorrect or require modification, the statistic list assigned to a host must be unassigned, and a new statistics list with the proper timestamp information assigned to the host.

The following code segment demonstrates how timestamps can be set in an unassigned statistics list.

```
               :
               :
stat_list_t slist;          /* Statistics list under const.*/
struct tm   tstmp;          /* Timestamp to use            */
int         rc;             /* Ftn call return code        */
               :
               :
/*
 * Use latest observance of the following statistic in
 * the next API operation.
 */
rc = PtpeFindStat("PagSp/%totalfree", slist);
rc = PtpeSetStatTime(PTPE_LATEST, NULL, slist);
if (rc != PTPE_SUCCESS) {
   printf("Cannot set time stamp for PagSp/%%totalfree\n");
}
/*
 * Use observation for 2:30pm on the 1st of October, 1995
 * for the following statistic in the next API operation
 */
rc = PtpeFindStat("CPU/cpu0/user", slist);
memset((void *) &tstmp, 0, sizeof(struct tm));
tstmp.tm_hour = 14;   tstmp.tm_min = 30;
tstmp.tm_mday =  1;   tstmp.rm_mon =  9;
tstmp.tm_year = 95;
rc = PtpeSetStatTime(PTPE_MATCH, &tstmp, slist);
switch (rc) {
   case PTPE_SUCCESS:     break;
   case PTPE_TIME_APPROX: printf("PtpeSetStatTime had to ");
                         printf("calculate an approximate ");
                         printf("timestamp\n");
   default:              printf("Cannot set time stamp for");
                         printf(" CPU/cpu0/user\n");
}
/*
 * Use earliest observance of the following statistic in
 * the next API operation.
 */
```

```
rc = PtpeFindStat("NetIF/tr0/ipacket", slist);
rc = PtpeSetStatTime(PTPE_EARLIEST, NULL, slist);
if (rc != PTPE_SUCCESS) {
   printf("Cannot set time stamp for NetIF/tr0/ipacket\n");
}
                 :
                 :
```

Determining the Success of a Statistics List Operation

Once a host list is constructed, and any necessary statistics lists are assigned to the hosts in the host list, PTPE API operations can be performed on these systems. When an API command has been completed, the results of the operation on the statistics -- such as “found,” “couldn't find,” etc. -- are placed in the statistics' list entries. To determine how the operation on a statistic was completed on a host, the statistics list must be retrieved from the system's host list entry, the list must be scanned, and the results checked in each statistics list entry.

The following code segment demonstrates how an application would determine how statistics were handled by an API request. For the sake of brevity, the error handling code has been omitted.

```
                 :
                 :
host_list_t targets;        /* Where commands were run     */
stat_list_t slist;          /* Copy of stats list of host  */
struct tm   tstmp;          /* Time statistics was observed*/
char  hostname[PTPE_NMLN];  /* Name of system in host list */
char  statname[PTPE_STNL];  /* Name of statistic in list   */
int   result;               /* How system/stat did in call */
long  ldata;                /* When stat is a "long"       */
float fdata;                /* When stat is a "float"      */
int   rc;                   /* Ftn call return code        */
                :
                :
/*
* Check results of an API request to get statistics
*/
rc = PtpeFirstHost(targets);
for(;;) {
  rc = PtpeGetHost(hostname, targets);
  rc = PtpeGetHostResult(targets, &result);
  printf("%s result was %d\n", hostname, result);
   /*
    * Get statistics, their results, and their values
    */
  rc = PtpeEmptyStatList(&slist);
  rc = PtpeGetHostStatList(slist, targets);
  rc = PtpeFirstStat(slist);
  for (;;) {
    rc = PtpeGetStatName(statname, slist);
    rc = PtpeGetStatResult(slist, &result);
    if (result == PTPE_STAT_NOT_FOUND) {
       printf("\t%s not found on %s\n", statname, hostname);
    }
    else {
      /* get value and timestamp from the statistic */
```

```
         rc = PtpeGetStatTime(&tstmp, slist);
         rc = PtpeStatIsLong(slist);
         if (rc == PTPE_TRUE) {
            rc = PtpeGetStatValueLong(&ldata, slist);
            printf("\t%s value was %d", statname, ldata);
            printf(" at %s\n", asctime(&tstmp));
      }

      else {

      rc = PtpeGetStatValueFloat(&fdata, slist);
      printf("\t%s value was %10.2f\n", statname, fdata);
      printf(" at %s\n", asctime(&tstmp));
   }

   }
   /* Are we at last statistic in list? */
   rc = PtpeIsLastStat(slist);
   if (rc == PTPE_TRUE) {
     break;
   }
   rc = PtpeNextStat(slist);
}
/* are we last host in the host list? */
rc = PtpeIsLastHost(targets);
if (rc == PTPE_TRUE) {
   break;
}
/* more hosts to go - move onto next one */
rc = PtpeNextHost(targets);
}
               :
               :
```

Controlling Performance Data

Subroutines in this category allow an application to control which performance information items are collected or archived, or whether any performance information is collected or archived at all. All subroutines in this category require an established session, for two reasons:

1. The session protects the monitoring hierarchy from alteration. The monitoring hierarchy dictates how performance information is collected.
2. The session ensures the user has adequate permissions. Performance information collection and archiving requires systems to dedicate resources to the monitoring task, and only privileged users should be allowed to dedicate such resources.

None of the subroutines in this category, with the exception of **PtpeColQueryState** and **PtpeArchQueryState**, can be issued from a read-only PTPE API session.

The following subroutines are provided for performance information control:

Subroutine Name	Page	Purpose
PtpeArchDisableAllStats	118	Instructs one or more nodes to cease archiving any statistics that it was archiving.
PtpeArchDisableStats	123	Instructs one or more specified systems to cease archiving of specified statistics, which are listed by the caller.
PtpeArchEnableAllStats	128	Instructs one or more specified systems to archive all available statistics.
PtpeArchEnableStats	133	Instructs one or more specified systems to archive additional statistics, which are specified by the caller.
PtpeArchQueryState	144	Reports on the overall state of performance information archiving in the system.
PtpeArchStartAllHosts	152	Starts the performance data archiving. Same function as the **ptpectrl -r** command.
PtpeArchStartHosts	156	Starts performance data archiving on a subset of nodes. Similar in function as the **ptpectrl -r** command, with the exception that the nodes must be specified by the caller.
PtpeArchStopAllHosts	161	Stops the performance data archiving. Same function as the ptpectrl -t command.
PtpeArchStopHosts	165	Stops performance information archiving on a subset of the nodes. Similar in function to the **ptpectrl -t** command, with the exception that the nodes must be specified by the caller.
PtpeChangeHostRates	172	Sets the current data collection rate or the current data recording rate used by the monitoring hierarchy.
PtpeColDisableAllStats	178	Instructs one or more nodes to cease collection of any statistics it was collecting.
PtpeColDisableStats	183	Instructs one or more specified systems to cease collection of specific statistics, which are listed by the caller.
PtpeColEnableAllStats	188	Instructs one or more specified systems to collect all available statistics.

Subroutine Name	Page	Purpose
PtpeColEnableStats	193	Instructs one or more specified systems to collect additional statistics, which are specified by the caller.
PtpeColQueryAvailStats	206	Retrieves a list of all statistics that one or more nodes can possibly provide for collection or archiving.
PtpeColQueryState	203	Reports on the overall state of performance information collection in the system.
PtpeColSetup	211	Performs setup work to permit performance data collection to begin. Provides the same function as the **ptpectrl -i** command.
PtpeColStart	215	Starts the performance data collection. Same function as the **ptpectrl -c** command.
PtpeColStop	220	Stops the performance data collection. Same function as the **ptpectrl -s** command.
PtpeQueryAllHostStatus	280	Obtains the current status of performance information collection and archiving from all systems in the monitoring hierarchy. Differs from **PtpeColQueryState** and **PtpeArchQueryState**, which give overall status.
PtpeQueryAvailHosts	284	Obtains a list of the hosts available for performance data collection and archiving functions.
PtpeQueryHostRates	291	Obtains the current data collection rate and the current data recording rate used by the monitoring hierarchy.
PtpeQueryHostStatus	293	Obtains the current status of performance information collection and archiving from one or more specified systems. Differs from **PtpeColQueryState** and **PtpeArchQueryState**, which give overall status.

All subroutines that control how performance information is collected and archived, as well as all those that retrieve performance information from nodes in the monitoring hierarchy, accept a host list and a pointer to an uninitialized host list as arguments. These parameters are used as follows:

- The first argument specifies the *targets* of the subroutine. These are the nodes on which the application wishes to execute the subroutine. In many cases, this list also contains assigned statistics lists that specify what statistics to use in the command.

- The second argument will contain a *reply* list for use after the API subroutine has executed. The subroutine constructs this list to report the results of the command. This is the list the application should scan to determine the success or failure of the API request, and to retrieve any statistics values returned by the subroutine. This host list pointer should be explicitly set to NULL before issuing the subroutine.

The only exceptions to this format are the **PtpeColSetup**, **PtpeColStart**, and **PtpeColStop** routines, which are treated as special cases.

Controlling Data Collection

The control subroutines are further divided into two categories: collection control and archiving control. By making this distinction, the API allows you to construct separate host lists so you can include and exclude different sets of statistics for archival than those specified for collection and summary.

The three routines provided by the PTPE API for collection setup, startup, and shutdown (**PtpeColSetup**, **PtpeColStart**, and **PtpeColStop**) differ in format from the majority of the API control routines. These routines do not accept a host list specifying the nodes where the command will execute. Instead, they execute on all systems in the monitoring hierarchy by default. This makes it impossible for an application to start or stop performance information collection and summarization on a subset of the nodes in the monitoring hierarchy; all nodes perform the setup, startup, or shutdown command. If you do not intend to start data collection on all nodes in the monitoring hierarchy, you can remove those nodes from the hierarchy using SP Perspectives or the **ptpehier -i** command.

These three routines are also the only ones that return two host lists as replies:

1. A list of data manager nodes that failed the command
2. A list of reporter nodes that failed the command

Either list can be scanned using the host list scanning subroutines mentioned in “Manipulating Host Lists” on page 55.

For example, the following code segment illustrates how an application would use these return lists to determine if the setup for performance information collection succeeded.

```
#include <stdio.h>            /* basic I/O capability      */
#include <spdm.h>             /* PTPE data types           */

main(int argc, char **argv)
{
  session_ptr_t    sblock;    /* PTPE session control info */
  host_list_t      mgrs;      /* Mgr Nodes that failed cmd */
  host_list_t      nodes;     /* Reg Nodes that failed cmd */
  int              rc;        /* Ftn call return code      */
  int              result;    /* How host responded to cmd */
  char             host[PTPE_NMLN]; /* Name of failing sys */
                  :
                  :
rc = PtpeOpenSession(&sblock);
mgrs = nodes = (host_list_t) NULL;
                  :
                  :
```

```
  /*
   * Setup for collection in the monitoring hierarchy
   */
  rc = PtpeColSetup(sblock, &mgrs,; &nodes);
  switch (rc) {
    case PTPE_SUCCESS:      break;
    case PTPE_SUCCESS_BADR:
      printf("Success, but cannot read reply message\n");
      break;
    case PTPE_API_FAILED_BADR:
      printf("Failure, but cannot read reply to determine ");
      printf("what systems failed the command\n");
      /* clean up lists, drop session, and exit */
    case PTPE_API_FAILED:
      rc = PtpeFirstHost(mgrs);
      if (rc == PTPE_SUCCESS) { /* in case no mgrs failed  */
        printf("These managers failed - command was not ");
        printf("sent to their reporting nodes:\n");
        for (;;) {
          bzero(host, PTPE_NLMN);
          rc = PtpeGetHost(host, mgrs);
          rc = PtpeGetHost(mgrs, &result);
          printf("\t %s responded with code %d\n", host,
                 result);
          rc = PtpeIsLastHost(mgrs);
          if (rc == PTPE_TRUE) {
             break;
        }
        rc = PtpeNextHost(mgrs);
      }
    }
    rc = PtpeFirstHost(nodes);
    if (rc == PTPE_SUCCESS) { /* in case no nodes failed */
      printf("These nodes failed to carry out the ");
      printf("setup command:\n");
      for (;;) {
        bzero(host, PTPE_NLMN);
        rc = PtpeGetHost(host, nodes);
        rc = PtpeGetHost(nodes, &result);
        printf("\t %s responded with code %d\n", host,
                result);
        rc = PtpeIsLastHost(nodes);
        if (rc == PTPE_TRUE) {
           break;
        }
        rc = PtpeNextHost(nodes);
     }
   }
   /* clean up lists, drop session, and exit */
 }
 rc = PtpeFreeHostList(&mgrs);
 rc = PtpeFreeHostList(&nodes);
                 :
                 :
}
```

The collection control subroutines are provided to start performance information collection and summarization on all systems, stop collection and summarization on all systems, and to prepare (setup) the monitoring hierarchy for performance information collection and summary. The setup function only needs to be performed in the following conditions:

1. When the monitoring hierarchy structure has been changed
2. When new performance information has been made available on one or more systems in the monitoring hierarchy.

An example of the first condition is when a system administrator updates the monitoring hierarchy with the **ptpehier** command. An example of the second condition is when a new LPP supported by the run-time monitor Monitor is installed on several nodes in the monitoring hierarchy. Unless one of these conditions exists, an application need not perform the setup function.

Enabling and Disabling Data for Summary

The remaining collection control subroutines permit an application to specify performance information that should be either enabled or disabled for summary by the data manager node. By indicating which performance information should and should not be made available for summary, an application can save computational and network resources that would otherwise be consumed to forward and summarize unneeded information. If a managing system is the target of one of these routines, the summary information that it would normally send forward to the central coordinator node can also be selectively enabled or restricted. Convenience routines are provided to enable or disable all information from specific systems.

The following code segment demonstrates how these interfaces might be used to start performance information collection and summarization, and include only a specific set of performance information on one of the systems in the monitoring hierarchy. Note that all remaining systems still have all their available performance information included for collection and summarization in this example.

```
#include <stdio.h>             /* basic I/O capability      */
#include <spdm.h>              /* PTPE data types           */

static char *sysnames[] = {
  "spnode04.ibm.com",
  "spnode12.ibm.com",
  "spnode23.ibm.com"
};
static char *statnames[] = {
  "PagSp/%totalfree",
  "NetIF/tr0/ipacket",
  "NetIF/tr0/opacket",
  "CPU/cpu0/user"
};
static int num_stats = 4;
static int num_hosts = 3;

main(int argc, char **argv)
{
  session_ptr_t    sblock;    /* PTPE session control info */
  host_list_t      targets;   /* Systems to restrict info  */
  host_list_t      mgrs;      /* Mgr Nodes that failed cmd */
  host_list_t      nodes;     /* Reg Nodes that failed cmd */
  stat_list_t      slist;     /* Statistics to enable      */
  int              rc;        /* Ftn call return code      */
  int             result;    /* How host responded to cmd */
  char            host[PTPE_NMLN]; /* Name of failing sys */
  char            stat[PTPE_STNL]; /* Stat from API call  */
                :
                :
  rc = PtpeOpenSession(&sblock);
  mgrs = nodes = (host_list_t) NULL;
                :
                :
/*
 * Set up host list where statistics will be enabled, and
 * the statistics list containing the statistics to be
 * enabled. "spnode12.ibm.com" will be the system used in
 * this test.
 */
rc = PtpeInitHostList(&targets);
rc = PtpeAddHostToList(sysnames[1], targets);
rc = PtpeInitStatList(&slist);
rc = PtpeAddStatToList(statnames[1], slist);
rc = PtpeAddStatToList(statnames[3], slist);
/*
 * Start collection. This example assumes success - for
 * an example of how to test for success, see previous
 * code segments in this document.
 * NOTICE - this routine merely instructs all systems
 * in the hierarchy to begin collecting data, but until
 * another routine tells them what data is to be collected,
 * nothing is being collected!
 */
rc = PtpeColStart(sblock, &mgrs,; &nodes);
(void) PtpeFreeHostList(&nodes);
/*
 * Disable all statistics currently enabled for collection
 * on the targeted system.  It is easier from a programming
 * point of view to disable all and enable a few, instead
 * of finding the list of all statistics to disable.
 */
rc = PtpeColDisableAllStats(sblock, targets, &nodes);
/* do appropriate error checking - see earlier examples */
                :
                :
/*
 * Assign the list of statistics to enable to the targeted
 * host list, and enable only those statistics we want.
 */
(void) PtpeFreeHostList(&nodes);
rc = PtpeFindHost(sysnames[1], targets);
rc = PtpeAssignStatsToHost(slist, targets);
rc = PtpeColEnableStats(sblock, targets, &reply);
switch (rc) {
  case PTPE_SUCCESS:
    break;
  case PTPE_SUCCESS_BADR:
   printf("Statistics seem to be enabled, but cannot ");
   printf("read confirmation from central coordinator node\n");
   break;
  case PTPE_API_FAILED:
   printf("Statistics enablement failed on nodes:\n");
  /* determine systems that failed the command */
  rc = PtpeFirstHost(reply);
  for (;;) {
    bzero(host, PTPE_NMLN);
    rc = PtpeGetHost(host, reply);
    rc = PtpeGetHostResult(reply, &result);
    printf("\t %s failed with code %d\n", host, result);
    /* check results of statistics for this system */
    rc = PtpeEmptyStatList(&slist);
    rc = PtpeGetHostStatList(slist, reply);
    rc = PtpeFirstStat(slist);
    for (;;) {
      bzero(host, PTPE_STNL);
      rc = PtpeGetStat(stat, slist);
      rc = PtpeGetStatResult(slist, &result);
      printf("\t\t Statistic %s result code was %d\n",
             stat, result);
      rc = PtpeIsLastStat(slist);
      if (rc == PTPE_TRUE) {
        break;
      }
      rc = PtpeNextStat(slist);
   }
   /* go onto next host in list, if any */
   rc = PtpeIsLastHost(reply);
   if (rc == PTPE_TRUE) {
     break;
   }
   rc = PtpeNextHost(reply);
  }
  break;
 case PTPE_API_FAILED_BADR:
  printf("Enablement failed, and cannot read the reply ");
  printf("from the central coordinator node to debug the cause\n");
  break;
 }
               :
               :
}
```

Controlling Performance Data Archival

The following archiving control interfaces allow the application to start recording performance information on one or more systems in the monitoring hierarchy.

PtpeArchEnableStats	Identifies statistics available for archiving
PtpeArchDisableStats	Identifies statistics restricted from archiving
PtpeColQueryAvailStats	Lists the performance statistics available for archiving
PtpeArchQueryStats	Identifies statistics currently included in archival

Unlike the control routines for performance information collection and summarization, these control routines permit the application to start or stop recording of performance information on part or all of the monitoring hierarchy. This permits an application to start collection for run-time monitoring on all nodes in the monitoring hierarchy, while recording performance information only on those systems that will require more detailed analysis at a later time. However, to start archiving on **any system** in the monitoring hierarchy, performance information collection must be active on **all systems** in the hierarchy.

If any node in the monitoring hierarchy is currently recording performance information to its archive, the general status of archiving in the hierarchy is said to be active. In other words, the **ptpectrl -q** command will report that archiving is active even if only one node out of all the systems in the monitoring hierarchy is recording its performance information. When a global indication is insufficient, the **PtpeQueryAllHostStatus** routine is provided as a means to determine exactly which nodes are collecting and archiving in the monitoring hierarchy, and which are not.

Additional subroutines are provided to enable for archival all statistics or only specific statistics, as well as to disable from archival all statistics or only specific statistics. A subroutine is also included for determining which statistics are currently enabled for recording.

The archiving control commands accept host lists that specify different hosts or different statistics (or both) from those provided to any collection control routines. This gives an application the capability to record a different set of performance information from that being passed to the managing node for summary. The API also gives the application the capability to record a different set of performance statistics on different systems in the hierarchy.

The following code segment extends the example portrayed in the previous code segment. In this segment, performance information archiving is started on two systems, one of which was used in the previous example. On that system, different sets of statistics are enabled for archiving and for collection.

```
             :
             :
/*
 * Set up host list where statistics will be enabled.
 */
rc = PtpeEmptyHostList(&targets);
rc = PtpeAddHostToList(sysnames[1], targets);
rc = PtpeAddHostToList(sysnames[2], targets);
/*
 * Start archiving on these systems alone.  Because
 * "spnode04" was not made a member of the target list,
 * the archiving command will not reach it.  For the sake
 * of brevity, the complete error check will be bypassed
 * in this example.
 * NOTICE - this routine merely tells the systems to start
 * archiving data, but until another routine tells them
 * what data is to be archived, nothing is being recorded!
 */
rc = PtpeArchStartHosts(sblock, targets, &reply);
if (rc != PTPE_SUCCESS)  {
  printf("Cannot start archiving!\n");
  /* free up lists, drop session, and exit */
}
(void) PtpeFreeHostList(&reply);
/*
 * In the previous example, NetIF/tr0/ipacket and
 * CPU/cpu0/user were enabled for collection on "spnode12".
 * In this example, NetIF/tr0/ipacket and NetIF/tr0/opacket
 * will be enabled for archiving, even through "opacket" is
 * not enabled for collection.
 */
rc = PtpeEmptyStatList(&slist);
rc = PtpeAddStatToList(statnames[1], slist);
rc = PtpeAddStatToList(statnames[2], slist);
rc = PtpeFindHost(sysnames[1], targets);
rc = PtpeAssignStatsToHost(slist, targets);
/*
 * On "spnode23", PagSp/%totalfree and CPU/cpu0/user will
 * be enabled for archiving.
 */
rc = PtpeEmptyStatList(&slist);
rc = PtpeAddStatToList(statnames[1], slist);
rc = PtpeAddStatToList(statnames[2], slist);
rc = PtpeFindHost(sysnames[1], targets);
rc = PtpeAssignStatsToHost(slist, targets);
/*
 * Enable the statistics for archiving.  After this call,
 * the systems should begin recording this information (but
 * ONLY this information).  Again, the full error detection
 * and handling has been omitted from the example for
 * brevity.
 */
rc = PtpeArchEnableStats(sblock, targets, &reply);
if (rc != PTPE_SUCCESS) {
   printf("Cannot enable statistics for archiving!\n");
   /* free up lists, drop session, and exit */
}
(void) PtpeFreeHostList(&reply);
/*
 * The statistics are now being archived on "spnode12" and
 * "spnode23".
 */
                :
                :
```

Obtaining Performance Data

Routines of this category permit an application to use performance information currently being collected by nodes in the monitoring hierarchy, as well as information residing in a node's performance information archive.

Subroutine Name	Page	Purpose
PtpeArchGetStats	138	Retrieves the values for a list of specified statistics for one or more hosts, which have been specified by the caller.
PtpeArchQueryStats	147	Retrieves the list of statistics that one or more specified hosts are currently archiving.
PtpeColGetStats	198	Retrieves the current non-archived values for a list of statistics for one or more hosts, which have been specified by the caller.
PtpeColQueryStats	206	Retrieves the list of statistics that one or more specified hosts are currently collecting.

In order for the PTPE API to obtain performance information on any of the systems in the monitoring hierarchy, the application must construct a host list that contains all the systems where performance information access is desired. Each node entry in the host list must also have a statistics list assigned to it. The statistics list contains the list of performance information to be obtained from that specific system.

The PTPE API provides subroutines to obtain the most recent performance information from the systems in the monitoring hierarchy. Subroutines are also provided to obtain recorded performance information from the archives maintained by each system.

Use the **PtpeColGetStats** subroutine to obtain the most recently available performance information from systems in the monitoring hierarchy. This subroutine is similar in concept to the data retrieval functions in the Performance Toolbox RSi programming library, which retrieve the most recent value for performance statistics from targeted systems. These two libraries use different means to achieve the same end, and each has advantages and disadvantages.

The **PtpeColGetStats** routine requires less network resources from the system that is executing the application: it establishes a single network connection to the central coordinator node to obtain this information from any system in the monitoring hierarchy. In contrast, an RSi application establishes a connection to each system it retrieves data from through the RSi library.

The RSi library provides more recent information: its routines obtain performance information from the Performance Toolbox shared memory and transmit it directly to the RSi application. The PTPE API, on the other hand, must pass through the system's managing node and the central coordinator node before it is relayed to the PTPE API application.

Either interface can be used, and one interface does not preclude the use of the other. When the priority obtaining the most recent performance information possible from a system, use the RSi interfaces. When it is more important to obtain performance information from a large number of systems with as little network overhead as possible, use the PTPE API interface, specifically, the **PtpeColGetStats** subroutine.

PtpeArchStats is the PTPE API subroutine that obtains performance information recorded in the performance information archives. Depending on the setting of the timestamps for each statistic in a system's statistics list, the earliest observation in the archive can be returned, the latest can be returned, or the observation closest to a specified date and time can be returned (see “Statistic Timestamps” on page 65).

Each statistic, whether returned by **PtpeColGetStats** or **PtpeArchGetStats**, has a timestamp included as part of the return value. However, this timestamp should be **ignored** when the statistic is retrieved using **PtpeColGetStats**. **PtpeColGetStats** sets the timestamp field to a -1 value, which would translate to a time prior to the standard UNIX timing epoch if it were to be accidentally used as a timestamp.

In the following code segment, statistics are retrieved from three nodes in the monitoring hierarchy using the PTPE API. In this example, the most recently available performance information is obtained by using **PtpeColGetStats**. For the sake of brevity, all possible error conditions are not addressed in this example.

```
#include <stdio.h>            /* basic I/O capability      */
#include <spdm.h>             /* PTPE data types           */

main(int argc, char **argv)
{
  session_ptr_t    sblock;    /* PTPE session control info */
  host_list_t      targets;   /* Where commands will run   */
  host_list_t      reply;     /* Where answer is placed    */ |
  stat_list_t      slist;     /* Statistics used by API    */
  int              rc;        /* Ftn call return code      */
  int              result;    /* How host/stat did in cmd  */
  char             hostname[PTPE_NMLN];
  char             statname[PTPE_STNL];
                  :
                  :
  rc = PtpeOpenSession(&sblock);
  rc = PtpeInitHostList(&targets);
  rc = PtpeInitStatList(&slist);
                  :
                  :
  /*
   * Set up host lists and assign statistics lists to each
   * host.  Since we'll be using PtpeColGetStats, we don't
   * need to set the timestamps in each statistic.
   */
                  :
                  :
  /*
   * Get the most recent (as close to run-time as possible)
   * values for the requested statistics from these systems.
   */
  reply = (host_list_t) NULL;
  rc = PtpeColGetStats(sblock, targets, &reply);
  switch (rc) {
    case PTPE_SUCCESS:
      break;
    case PTPE_SUCCESS_BADR:
      printf("Routine succeded, but couldn't get back the ");
      printf("reply -- treating as an error.\n");
      /* deallocate lists and exit */
    case PTPE_LIMITED:
      printf("Not all of the systems carried out the cmd\n");
      /* will have to check result codes for each system */
    case PTPE_LIMITED_BADR:
      printf("Not all of the systems carried out the cmd\n");
      printf("Couldn't read the reply - treating as error\n");
      /* deallocate lists and exit */
    case PTPE_API_FAILED:
      printf("All systems failed to execute command\n");
      /* deallocate lists and exit */
    case PTPE_API_FAILED_BADR:
      printf("All systems failed to execute command\n");
      printf("Couldn't get reply to determine the reasons\n");
     /* deallocate lists and exit */
  }
  /*
   * Show results and statistics from each host.
   */
  rc = PtpeFirstHost(reply);
  for (;;) {
    rc = PtpeGetHost(Hostname, reply);
    rc = PtpeGetHostResult(reply, &result);
    if (rc == PTPE_SUCCESS) {
      rc = PtpeEmptyStatList(&slist);
      rc = PtpeGetHostStatList(slist, reply);
      rc = PtpeFirstStat(&slist);
      for (;;) {
        /* get statistics names, results, and values */
                  :
                  :
      }
  }
  else {
    printf("System %s failed PtpeColGetStats with an ");
    printf("error code of %d\n", result);
  }
  rc = PtpeIsLastHost(reply);
  if (rc == PTPE_TRUE) {
    break;
  }
  rc = PtpeNextHost(reply);
 }
 /*
  * Free up the "reply" list so the same pointer can be used
  * by subsequent API commands.
  */
 rc = PtpeFreeHostList(&reply);
               :
               :
```

The only changes required for this code section to retrieve performance information from the archive are:

1. Set the timestamps on the statistics in the statistics list, and
2. Replace the **PtpeColGetStats** with the **PtpeArchGetStats** subroutine.

Some General Cautions

The PTPE API avoids writing to memory that has not been allocated by ensuring that pointers it receives are either initialized or cleared out before use. If the subroutine you are using expects this condition (see Chapter 8, “The PTPE API Subroutines” on page 113) and finds that local variables have not been set to zero values, the API subroutine may return a **PTPE_INV_PTR** code.

Chapter 8, “The PTPE API Subroutines” on page 113, states whether subroutines expect parameters to contain NULL or non-NULL values. Please follow these guidelines, and initialize your local variables accordingly.

Under no circumstances should applications attempt to use the pointers in the host list and statistics list data structures directly. The API uses and sets these pointers internally, and these pointers can appear to change unexpectedly if they are manipulated. Subroutines are provided to insulate the application from this behavior in the pointers, so please use the subroutines instead of using the pointers directly.

Compiling Source Code

To utilize the PTPE API, the application should include the <**spdm.h**> header file in the source code. The source code should also be linked with the PTPE API library during compilation by specifying `-lptpe` on the compile command line:

```
$ cc -lptpe -o ptpeappl ptpeappl.c
```

The PTPE API is implemented as a shared library. Should the library change for service reasons, an API application need not recompile to use the most recent library.

Sample PTPE Application

A sample application program that uses the PTPE API subroutines is located in **usr/lpp/ptpe/samples**. See Appendix B, “PTPE Sample Application Program” on page 369.

Performance Toolbox Parallel Extensions for AIX Command and Programming Reference

Chapter 7. PTPE Commands

Use these commands to perform control functions for Performance Toolbox Parallel Extensions for AIX.

ptpeconf

Purpose

ptpeconf creates data classes in the System Data Repository for use by the Performance Toolbox Parallel Extensions.

Syntax

ptpeconf [**–h**] [**–t**]

Parameters

–h Displays the syntax of this command.

–t Tests only for the presence of the data classes required by the Performance Toolbox Parallel Extensions. The results of the test are reported to standard output. The data classes are not created if they do not exist.

Description

The **ptptconf** command tests for the presence of specific data classes in the System Data Repository required by the Performance Toolbox Parallel Extensions. If these data classes are not present, the command will create these classes unless the -t option has been specified.

ptpeconf is executed as part of the Performance Toolbox Parallel Extensions installation procedure. It is provided as a command in case the data classes need to be recreated in the System Data Repository, or created in another system partition.

Restrictions

You must be logged in as **root** to execute this command.

Results

The data classes created by **ptptconf** are specific to the system partition in which the command was executed. To create these data classes in another system partition, **ptptconf** must be executed in that partition. These data objects must exist in the system partitions where the Performance Toolbox Parallel Extensions product will execute.

Examples

1. To check if the data classes required by the Performance Toolbox Parallel Extensions exist for this system partition in the System Data Repository, enter:

   ```
   /usr/lpp/ptpe/bin/ptpeconf -t
   ```

   The results of the test will be reported to standard output.

2. To create the necessary data classes in the System Data Repository for this system partition, enter:

   ```
   /usr/lpp/ptpe/bin/ptpeconf
   ```

Files

- **/usr/lpp/ptpe/bin/ptpeconf** Contains the **ptptconf** command
- **/etc/groups** Contains group identifiers
- **/etc/security/groups** Contains group identifiers
- **/usr/lib/nls/msg/*/ptpe.cat** Message catalog for the PTPE product
- **/usr/lib/libptpe.a** The Performance Toolbox Parallel Extensions shared library

Prerequisite Information

This command is part of the Performance Toolbox Parallel Extensions for AIX (PTPE) feature of the IBM Parallel System Support Programs for AIX licensed program product.

Related Information

"ptpectrl" on page 88
"ptpegroup" on page 97
"ptpehier" on page 99

ptpectrl

Purpose

ptpectrl controls the collection, summarization and archival of performance information.

Syntax

ptpectrl {**–i | [–a | –c**] [**–f** *report*,*record*] [–l *filename* **| –n**] [**–r**] **|** [**–m** [**–l** *file*] [**–n**]] **| –v** [**–l** *file*] **|** [**–s | –t | –e | –q | –h**]}

Parameters

–a Initialize and start collection.

Performs the actions of the **ptpectrl -i** and **ptpectrl -c** command sequences.

–c Begin Data Collection.

Begins collection of performance related statistics for the entire system. Also begins preparation of summary performance statistics, if that feature is installed.

–e Erase archive contents.

Erases the contents of any performance information archive files that may exist on the systems within the monitoring hierarchy. This option will not execute when performance information collection and summarization is active.

–f Frequency control.

Alters the frequency with which performance data is collected for averaging into summary statistics (*report*) and the rate at which performance data is written to the archive (*record*). (*report*) and (*record*) are specified in seconds. The values are governed by the following rules:

- Either value may be zero, indicating that its corresponding interval will not be changed by the command.
- If report is non-zero, it must be an integer value between 5 to 300, inclusive. If the value does not meet this criteria, the command will alter the value to meet it.
- If record is non-zero, it must be an integer value between 30 to 900, inclusive. If the value does not meet this criteria, the command will alter the value to meet it.
- When both values are non-zero, record must be evenly dividable by report. If the value does not meet this criteria, the command will alter the value to meet it.

Both values must be supplied, separated by a comma. No white space is permitted between the values.

–h Command Usage Help.

Displays the syntax for the command, and a terse description of its usage.

–i Initialize Data Collection.

Performs setup tasks to prepare the monitoring hierarchy for performance information collection and summarization. This option should be used after modifying the structure of a monitoring hierarchy, before starting collection and summarization on the monitoring hierarchy for the first time.

–l *filename* Instructs the command to use filename in place of the **ptpe.cf** statistics configuration file.

–m Enables and excludes statistics for collection and archiving while collection or archiving is active. This option is not used with the **–a**, **–c**, or **–r** options.

The **–m** option searches the user's home directory for the **ptpe.cf** statistics configuration file. If the file is not located in that directory, the command next searches the **/etc/perf** directory for the file. Once the file is located, the command uses its contents to determine which statistics should be made available for data collection and recording to the performance data archive. An alternate file can be specified with the **–l** option.

If the command cannot locate the statistics configuration file **ptpe.cf** and no alternate file is specified with the **–l** option, or if the **–n** option is also specified, all performance information is made available for collection and archiving.

–n Instructs the command to ignore the **ptpe.cf** statistics configuration file. All performance statistics available in the reporting hierarchy will be enabled for aggregation and recording to the performance information archive.

–q Report Status.

Reports the current status of performance information collection and archiving within the monitoring hierarchy.

–r Begin Data Recording.

Instructs all nodes in the monitoring hierarchy to record their performance related statistics to the Performance Data Archive. Performance information collection and summarization must be active, or the **-a** or **-c** options must first be specified, in order to issue this option.

–s Stop Data Collection.

Shuts down performance data collection throughout the system, as well as summary statistic preparation and data archival if those options are active.

–t End Data Recording.

Instructs all nodes in the monitoring hierarchy to cease recording of performance statistics in the Performance Data Archive. Collection and summarization remain active.

–v Tests the contents of a statistics configuration file. If the **-l** option is not used, the test is performed on the **ptpe.cf** file.

Description

The **ptpectrl** command controls activity with the Performance Toolbox Parallel Extensions monitoring hierarchy. Through this command, performance information collection and archival can be started or ended on all systems making up the PTPE monitoring hierarchy.

Before the **ptpectrl** command can be used, specific data object classes must exist in the System Data Repository (see “ptpeconf” on page 86), and the **perfmon** user group must exist on all systems within the monitoring hierarchy (see “ptpegroup” on page 97). A monitoring hierarchy must also have been established (see the **ptpehier** command). The user must have **perfmon** set as the user's effective group to issue the **ptpectrl** command.

Restrictions

This command is expected to be issued from a node in the system partition that will be monitored. When issued from a node outside the partition to be monitored, the command will be unable to issue control commands to the monitoring hierarchy. In such cases, only the **–q**, **–v**, and **–l** options are available.

This command can only be executed by members of the **perfmon** group.

Results

ptpectrl retrieves the current monitoring hierarchy configuration from the System Data Repository, and establishes a socket connection with the system identified as the central coordinator Node for the hierarchy. **ptpectrl** then encodes the specified command, along with the structure of the monitoring hierarchy, into a message and transmits the message to the central coordinator Node. The central coordinator Node decodes the instructions and the monitoring hierarchy from the message, identifies the systems identified as data manager nodes for the reporting groups within the hierarchy, and establishes socket connections with these systems. The central coordinator Node then forwards the message from **ptpectrl** to these data manager nodes. The data manager nodes decode the instructions and the monitoring hierarchy from the command message, determine which systems report to it in the hierarchy, and establish socket connections with these systems. The data manager nodes then send specific instructions to the systems within its reporting group to carry out the request made by the **ptpectrl** command.

Each system reports its success or failure to its data manager node. The data manager node, in turn, reports each system's results to the central coordinator Node, which prepares a report containing all results of the command and passes it along to the **ptpectrl** command. Once the results of the command have been reported, the systems drop their sockets connections with their data manager nodes.

ptpectrl parses the reply from the central coordinator and reports overall success or failure of the command, based on the percentage of systems that failed to carry out the command. If less than half of the systems in the monitoring hierarchy fail the request, **ptpectrl** will report that the overall command succeeded; if half or more failed, the command will report that the overall command failed. Unless all systems failed to carry out the command requested, **ptpectrl** will report the names of the systems that failed to carry out the command.

A statistics configuration file is used by the **ptpectrl** command when performance information collection and summarization is started, and when performance information archiving is started. The configuration file tells the command which performance statistics to make available for collection or archiving. The semantics of the file permit a set of statistics to be enabled for collection only, for archiving only, or for both. The syntax of the file permits the use of wildcards in the entries. Further details can be found in the **ptpe.cf** file format documentation. Through the use of a statistics configuration file, **ptpectrl** can restrict collection or archiving of certain statistics, thereby conserving system resources and reducing network traffic.

When performance information collection and summarization is begun by **ptpectrl**, or when performance information archiving is begun by the command, the command searches for a statistics configuration file named **ptpe**.cf in the user's home directory. If the file cannot be found in the user's home directory, **ptpectrl** searches the **/etc/perf** directory for the file. If the file cannot be found in either directory, or if the **–n** flag was specified, all statistics available in the reporting hierarchy are enabled for performance information collection or archiving. The user may specify an alternate to the **ptpe.cf** file by using the **–l** option.

The statistics configuration file can be altered after collection or archiving has been started in the monitoring hierarchy. To alert the hierarchy to these changes, issue the **–m** option when the changes to the configuration file are complete. This option resends the configuration file to the monitoring hierarchy, replacing the file used at collection or archive setup.

In the event that all statistics are to be made available for collection or archiving after a configuration file has been used, use the **–m** and **–n** options together. This enables all available statistics in the monitoring hierarchy for collection and recording to archive.

The performance information archive is stored in the **/var** filesystem on each system. Should the amount of free space on this filesystem fall below 5 percent, the system will not add performance information to the archive. If the amount of free space should rise again above 5 percent, recording will resume; information that would have been recorded in the interim is lost.

Examples

1. To prepare a new or revised monitoring hierarchy for performance information collection and summarization, enter:

   ```
   ptpectrl -i
   ```

2. To prepare a new or revised monitoring hierarchy, and to begin performance information collection in the same command, enter:

   ```
   ptpectrl -a
   ```

3. To begin performance information collection, summarization, and archival, enter:

   ```
   ptpectrl -c -r
   ```

4. To change the list of statistics currently being collected and archived in the monitoring hierarchy after collection or archiving has been started, enter:

   ```
   ptpectrl -m -l revised_ptpe.cf
   ```

5. To shut down performance information collection and summarization, enter:

```
ptpectrl -s
```

This command will also shut down performance information recording of that option was previously started with **ptpectrl -r**.

6. To determine the current status of performance information collection and archiving, enter:

   ```
   ptpectrl -q
   ```

   Status is reported to standard output.

7. To verify the structure of the statistics configuration file **ptpe.cf**, located in the user's home directory, enter:

   ```
   ptpectrl -v
   ```

   Any structural errors in the configuration file are reported in the output.

Files

$HOME/ptpe.cf Statistics configuration file
/usr/sbin/ptpectrl Contains the ptpectrl command
/usr/lpp/ptpe/bin/ptpedelete Contains the instructions to delete performance information archives on all systems in the monitoring hierarchy
/usr/group Contains group identifiers
/usr/security/group Contains group identifiers
/var/adm/ptpe/perflog Contains the archive of performance information for this system
/var/adm/ptpe/perftab Performance statistics archive table
/etc/services Contains the port assignments for service daemons
/usr/lib/nls/msg/*/ptpe.cat Message catalog for the PTPE product
/usr/lib/libptpe.a The Performance Toolbox Parallel Extensions shared library

Prerequisite Information

This command is part of the Performance Toolbox Parallel Extensions for AIX (PTPE) feature of the IBM Parallel System Support Programs for AIX licensed program product.

Related Information

"Using the ptpe.cf File" on page 40
"ptpeconf" on page 86
"ptpegroup" on page 97
"ptpehier" on page 99
"ptpedump" on page 93
Appendix A, "PTPE Files" on page 365

ptpedump

Purpose

ptpedump prepares text formatted versions of the Performance Toolbox Parallel Extensions information archive.

Syntax

ptpedump [**–h**] | [[**–c** | **–s**] [**–n** *node* [*node*...]]]

Parameters

–c Dumps the archive files in comma-separated format. The resulting dump can be used as input to a database application.

–h Help. Displays the syntax of the command.

–n*node [...]* Target nodes. Causes the performance information on only the named node(s) to be dumped to a text file. Node names are separated by spaces.

–s Dumps the archive files in spreadsheet format. The dump is formatted in 132-column output, which may result in the creation of many files. These files can be used as input to the IBM Performance Toolbox for AIX **a2ptx** command.

Description

The **ptpedump** command causes one or more systems in the monitoring hierarchy to dump the entire contents of that system's performance information archive to a file in text format. This text formatted file can then be imported into a database, or used as input to the IBM Performance Toolbox for AIX **a2ptx** command. **a2ptx** generates a recording file that can be played back graphically by the **xmperf** command. The archive dump can also be input to a spreadsheet application or or be used to create a historical report of system performance. By default, **ptpedump** instructs all systems to dump their performance information archives.

Before the **ptpedump** command can be used, specific data object classes must exist in the System Data Repository (see “ptpeconf” on page 86) and the **perfmon** user group must exist on all systems within the monitoring hierarchy (see “ptpegroup” on page 97). A monitoring hierarchy must also have been established (see “ptpehier” on page 99). The user must have **perfmon** set as the user's effective group to issue the **ptpedump** command.

Restrictions

This command can only be executed by members of the **perfmon** group.

Results

ptpedump creates a text formatted version of the **/var/adm/ptpe/perflog** file on one or more nodes in the monitoring hierarchy, which will also reside in the **/var** filesystem. Before issuing the command, the user should make sure that there is enough free space on the **/var** filesystem on the targeted systems to accept this file. If not, the **spdm_dump** command can be run locally on each system to dump a text version of the archive file to standard output which can be redirected to a filesystem with enough space to accept the output.

The formatted text file will be named **/var/adm/ptpe/perflog.txt**<*date*>, where <*date*> is a numeric representation of the date when the file was created, in MMDDYYYY format. For example, the filename **perflog.txt100195** indicates that the file was created on the first of October, 1995.

When the **–c** and **–s** options are not specified, the command dumps the archive contents in the default format. A single heading line will appear at the top of the file, followed by one or more text records in the following format:

```
Timestamp   Length   Data Type   Statistic   Value
```

where *Timestamp* is the time when the statistics was recorded in YY/MM/DD HH:MM:SS format, *Length* is the length of the statistic name, *Data Type* is a single character that represents the data type of the value, *Statistic* is the full context name of the performance statistic as named by Performance Toolbox, and *Value* is the value of the statistic at the time it was recorded. The maximum length of an output record is 180 characters, depending on the length of the statistic name. The data type identifier is one of the following values:

	Performance Toolbox
Code	**Data Type**
i	SiInt
i	SiInt
I	SiUInt or SiUnsign
l	SiLong
L	SiULong
s	SiShort
S	SiUShort
c	SiChar
a	SiAddr
f	SiFloat
d	SiDouble

When the **–c** option is specified, the command dumps the contents of the archive in comma-separated format. The files uses the same naming convention as the default format (**/var/adm/ptpe/perflog.txt**<*date*>). A single heading line, specifying the name of the node, appears at the top of the file, followed by one or more text records in the following format:

```
Time="Timestamp",   Statistic=Value
```

where *Timestamp* is the time when the statistics was recorded in YY/MM/DD HH:MM:SS format, *Statistic* is the full-context name of the performance statistic as named by Performance Toolbox, and *Value* is the value of the statistic at the time it was recorded.

When the **–s** option is specified, the command dumps the contents of the archive in spreadsheet format. The output is formatted for use with 132-column output devices, using the statistic names along the horizontal axis and the recording times along the vertical axis. Because statistic names can be quite long, it is possible for a single spreadsheet to contain only a few statistics. In such cases, the command generates multiple files until all statistics in the archive on a node are dumped to a file. The file name uses the same convention as the default and comma-separated formats (**/var/adm/ptpe/perflog.txt**<*date*>) along with a part number suffix, should the output be contained in multiple files. For example, the complete spreadsheet output for October 1, 1995, consisting of three files, would be created as:

/var/adm/ptpe/perflog.txt100195_1
/var/adm/ptpe/perflog.txt100195_2
/var/adm/ptpe/perflog.txt100195_3

Two headings lines appear at the top of each spreadsheet file. The first line indicates the name of the node for which the report was created. The second line lists a timestamp heading, along with the names of the statistics contained in the report:

```
hostname:     Node
Timestamp     Statistic A     Statistic B     ...
```

where *Node* is the name of the node from which this report was generated, and the *Statistics* are the names of the performance statistics included in the report. One or more records follow the heading, containing a time value in YY/MM/DD HH:MM:SS format, along with any values observed for the statistics. If no values were found for a statistic at a given timestamp, a hyphen is placed in the column for that statistic for that timestamp value. If **ptpedump** is not provided a list of systems, the command retrieves the monitoring hierarchy from the System Data Repository. It then invokes a distributed shell to execute the **spdm_dump** command on all systems in the monitoring hierarchy (see “spdm_dump” on page 109). **spdm_dump** is executed locally on each node, and the output from **spdm_dump** is routed to **/var/adm/ptpe/perflog.txt**<*date*> file on that system.

Statistics names appearing as **????** indicate that the **spdm_dump** utility could not determine the correct IBM Performance Toolbox for AIX statistic name for this data. This should only occur when the performance statistic translation table file, **/var/adm/ptpe/pertab**, is damaged or erased. In such cases it is not possible to retrieve performance statistics names.

ptpedump only creates text formatted files on the same systems where the archive files reside. To dump the archive to a remote file, the **spdm_dump** command should be used locally, redirecting its output to a network file system file.

ptpedump does not check for sufficient space in the **/var/adm/ptpe** filesystem before dumping archive files. No error is reported if all files on a node cannot be created using the **–s** option.

Examples

1. To dump the performance archives on all systems in the monitoring hierarchy to a text formatted file, enter:

   ```
   ptpedump
   ```

2. To dump the archive files on all nodes in comma-separated format for possible use as database input, enter:

   ```
   ptpedump -c
   ```

3. To dump the archive files on nodes `spnode09.ibm.com` and `spnode18.ibm.com`, enter:

   ```
   ptpedump -n spnode09.ibm.com spnode18.ibm.com
   ```

4. To dump the archive files on node `spnode14.ibm.com` in spreadsheet format for use as input to the **a2ptx** command, enter:

   ```
   ptpedump -s -n spnode14.ibm.com
   ```

Files

- **/usr/lpp/ptpe/bin/ptpedump** Contains the ptpedump command
- **/usr/lpp/ptpe/bin/spdm_dump** Contains the spdm_dump command
- **/var/adm/ptpe/perflog** Contains the archived performance
- **/var/adm/ptpe/perftab** Contains the performance statistics archive table information for the local system
- **/var/adm/ptpe/perflog.txt**<*date*> The text formatted contents of the performance information archive on the local system
- **/etc/group** Contains group information
- **/etc/security/group** Contains group information
- **/usr/lib/nls/msg/*/ptpe.cat** Message catalog for the PTPE product
- **/usr/lib/libptpe.a** The Performance Toolbox Parallel Extensions shared library

Prerequisite Information

This command is part of the Performance Toolbox Parallel Extensions for AIX (PTPE) feature of the IBM Parallel System Support Programs for AIX licensed program product.

Related Information

- "ptpeconf" on page 86
- "ptpegroup" on page 97
- "ptpehier" on page 99
- "spdm_dump" on page 109
- "The a2ptx Recording Generator" in the *IBM Performance Toolbox for AIX: Guide and Reference*

ptpegroup

Purpose

ptpegroup creates the user group required by PTPE.

Syntax

ptpegroup [**–h**]

Parameters

–h Help. Displays a usage message to standard output.

Description

The **ptpegroup** command creates the **perfmon** user group on the system from which it is executed. The root user is set as the initial member and administrator of the user group. Only members of this group may utilize PTPE functions, and only when these users have set the **perfmon** user group as their effective group.

ptpegroup executes as part of the Performance Toolbox Parallel Extensions installation procedure. It is provided as a command in case the **perfmon** user group should need to be recreated on one or more systems.

ptpegroup does not force the revised **/etc/security/group** and **/etc/group** files to be propagated to the other nodes. Therefore, this command should be run from the control workstation, which propagates these files to the nodes on an hourly basis. **ptpegroup** can also be executed on another node which maintain the **/etc/group** and **/etc/security/group** as part of a file collection.

If **ptpeconf** is executed on the control workstation, the remaining SP nodes may not become aware of the **perfmon** group for up to an hour after the command was executed. If faster response time is desired, use the supper command to force update of the **user.admin** file collection.

Restrictions

This command can only be executed by members of the **perfmon** group.

Results

Once the **perfmon** group has been created, users may be assigned to the group, and group administration can be reassigned if desired. It is recommended that new users be created and assigned to the **perfmon** group, instead of adding existing users to this group. Users added to this group should have sufficient privilege to gain read-write access to the System Data Repository. Users may be added to the group, and administration of the group can be altered, using the System Management Interface Tool (SMIT) or the **chgrpmem** command.

Examples

1. To create the **perfmon** user group on the control workstation, log onto the control workstation and enter:

```
/usr/lpp/ptpe/bin/ptpegroup
```

Files

- **/usr/lpp/ptpe/bin/ptpegroup** Contains the **ptpegroup** command
- **/etc/group** Contains group information
- **/etc/security/group** Contains group information
- **/usr/lib/nls/msg/*/ptpe.cat** Message catalog for the PTPE product

Prerequisite Information

This command is part of the Performance Toolbox Parallel Extensions for AIX (PTPE) feature of the IBM Parallel System Support Programs for AIX licensed program product.

Related Information

- "ptpeconf" on page 86
- "ptpectrl" on page 88
- "ptpehier" on page 99
- "spdm_dump" on page 109
- The **chgrpmem** command in *IBM AIX for RISC System/6000 Commands Reference*
- The **supper** command in *IBM Parallel System Support Programs for AIX: Command and Technical Reference*
- The System Management Interface Tool in *IBM Parallel System Support Programs for AIX: Administration Guide*
- "Managing File Collections" in *IBM Parallel System Support Programs for AIX: Administration Guide*
- "The System Data Repository" in *IBM Parallel System Support Programs for AIX: Administration Guide*

ptpehier

Purpose

ptpehier creates or removes Performance Toolbox Parallel Extensions monitoring hierarchy.

Syntax

ptpehier {[**–e**[**–c** *host*] **| –f**[**–c** *host*] **| –i –c** *host*] [**–p**] | [**–p**] | [**–d**] | [**–h**]}

Parameters

–c Central Coordinator.

Instructs the **ptpehier** command to assign the central coordinator Node responsibilities to the system specified by "host". For the **-e** and **-f** options, the system specified by *host* must exist in the Node class of the System Data Repository, and must be a fully specified host name address. For the **-i** option, the system specified by "host" must appear in the input stream.

The **-c** flag is optional when for the **-e** or **-f** are specified; these standard hierarchies automatically appoint a central coordinator node if none is specified.

–d Delete Current Monitoring Hierarchy.

Only a limited set of PTPE functions will be available, and it will be impossible to start performance information collection and summarization without the monitoring hierarchy.

–e Create Monitoring Hierarchy Automatically by Ethernet.

Creates a monitoring hierarchy automatically, using Ethernet local area subnetworks as the basis for constructing the hierarchy. All systems that are members of a common Ethernet local area subnetwork will be assigned to the same node group in the monitoring hierarchy. Data manager nodes will be nominated from the node groups on an arbitrary basis. If the **-c** option is not specified, the central coordinator Node will also be assigned arbitrarily.

–f Create Monitoring Hierarchy Automatically by Frame.

Creates a monitoring hierarchy automatically, using SP node frames as the basis for constructing the hierarchy. All systems that are members of the same SP node frame will be assigned to the same node group in the monitoring hierarchy. Data manager nodes will be nominated from the node groups on an arbitrary basis. If the **-c** option is not specified, the central coordinator Node will also be assigned arbitrarily.

–h Help.

Displays a usage message to standard output.

–i Create Monitoring Hierarchy by Standard Input.

Creates a monitoring hierarchy according to the user's specification, which is supplied via standard input. When this option is specified, **-c** must also be specified to name the central coordinator Node.

The user provides a specification of the node groups in the monitoring hierarchy through standard input. Standard input takes the following format:

```
{
name of first host in the node group
name of second host in the node group
name of third host in the node group
                 :
                 :
name of last host in the node group
}
{
specification of the next group's contents
}
                 :
additional group specifications
                 :
Ctrl-D
```

The character '{' is used to signal the start of a node group, and '}' is used to signal the end of a node group. Each node group must contain at least one host. The first host named as a member of the node group is assigned as the data manager node of the group. The user cannot specify a host as both a managing node for a node group and as the central coordinator Node for the monitoring hierarchy. Input is terminated by the Ctrl-D character.

–p Print monitoring hierarchy.

Displays a text representation of the monitoring hierarchy to standard output. If this option is combined with the **-e**, **-f**, or **-i** options, the hierarchy is displayed after it has been created.

Description

The **ptpehier** command constructs, displays, and removes the monitoring hierarchy used by PTPE. A monitoring hierarchy cannot be modified, replaced, or removed while performance information collection and summarization is active. Before the **ptpehier** command can be used, specific data object classes must exist in the System Data Repository (see “ptpeconf” on page 86), and the **perfmon** user group must exist on the system (see “ptpegroup” on page 97). The user must be a member of the **perfmon** user group, and must have **perfmon** set as the user's effective group to issue the **ptpehier** command.

Before using the Performance Toolbox Parallel Extensions product for the first time, you must create a monitoring hierarchy. (For an explanation of the monitoring hierarchy concept, see “Understanding the Monitoring Hierarchy” on page 9). For initial use, using a hierarchy based on Ethernet local area subnetwork is recommended. You can construct this hierarchy using the **ptpehier -e** command. However, other options are provided to permit more experienced users to restructure the hierarchy or to construct a customized hierarchy.

The structure of the monitoring hierarchy impacts how summarized performance information is calculated. It also determines which systems are assigned the tasks of summarization and command delegation. **ptpehier** provides two different options that configure a standard monitoring hierarchy for efficient data transmission and a moderate level of management overhead on the SP platform:

1. Organization by Ethernet local area subnetwork. By configuring the hierarchy along Ethernet subLAN boundaries, the Performance Toolbox Parallel Extensions reduces the amount of network traffic that must be transmitted

through network gateways, and avoid inflicting additional overhead to the gateway systems.

2. Organization by SP node frame. By configuring the hierarchy along frame boundaries, a minimal amount of cross-network traffic is generated, and summary performance information is prepared according to system boundaries that are easily recognizable by SP system users. Node groups are also limited to a maximum of 16 systems, limiting the overhead on data manager nodes.

ptpehier also permits you to configure a customized monitoring hierarchy. Because the structure of the monitoring hierarchy has a great impact on the function and computational expense of the Performance Toolbox Parallel Extensions, users should plan the optimal monitoring hierarchy structure before constructing a customized hierarchy with this command. Consult the (planning section of our user manual) for suggestions on planning a customized hierarchy.

A monitoring hierarchy does not need to be removed before a new hierarchy can replace it. However, removing a hierarchy would prevent other users from starting performance information collection and summarization on a hierarchy that is about to be replaced. Removal of a monitoring hierarchy should only be done when the hierarchy is about to be replaced to lock out other users, or when the Performance Toolbox Parallel Extensions function is to be disabled for an extended period of time.

Restrictions

This command is expected to be issued from a node in the system partition for which the monitoring hierarchy is, or will be, built. When issued from a node outside the partition, the command will be unable to modify the structure of the monitoring hierarchy. In such cases, only the **–h** and **–p** options are available.

This command can only be executed by members of the **perfmon** group.

Results

If **–p** is specified, the output uses indentation to illustrate the levels of the hierarchy. All systems at the next indentation level report to the system at the previous indentation level. The output resembles the following format.

```
central coordinator Node Host Name
  Data Manager Node Host Name
     Node Group Member Host Name
     Node Group Member Host Name
     Node Group Member Host Name
          :
          :
  Data Manager Node Host Name
     Node Group Member Host Name
     Node Group Member Host Name
     Node Group Member Host Name
          :
          :
```

Examples

1. To display the current monitoring hierarchy, enter:

   ```
   ptpehier -p
   ```

2. To create a monitoring hierarchy according to SP node frame, and display the resulting hierarchy upon completion, enter:

   ```
   ptpehier -f -p
   ```

3. To create a monitoring hierarchy according to Ethernet local area subnetwork, and to assign the central coordinator Node responsibilities to the system known as spnode09.ibm.com, enter:

   ```
   ptpehier -e -c spnode09.ibm.com
   ```

   spnode09.ibm.com must exist as a *reliable_hostname* in the Node Class for one of the SP nodes in the System Data Repository.

4. To create a hierarchy that organizes systems into the following hierarchical structure:

   ```
                   spnode09.ibm.com
                              |
                ---------------------------------
               |                                |
        spnode03.ibm.com                spnode21.ibm.com
               |                                |
      --------------------             --------------------
     |                    |             |                    |

   spnode01.ibm.com spnode03.ibm.com  spnode17.ibm.com spnode19.ibm.com
   spnode05.ibm.com spnode07.ibm.com  spnode21.ibm.com spnode23.ibm.com
   spnode09.ibm.com spnode10.ibm.com  spnode25.ibm.com spnode27.ibm.com
   spnode11.ibm.com spnode12.ibm.com  spnode29.ibm.com spnode31.ibm.com
   spnode13.ibm.com spnode14.ibm.com
   spnode15.ibm.com spnode16.ibm.com
   ```

   Create a file to contain the node group specifications. The file content would appear as:

```
{
spnode03.ibm.com
spnode01.ibm.com
spnode05.ibm.com
spnode07.ibm.com
spnode09.ibm.com
spnode10.ibm.com
spnode11.ibm.com
spnode12.ibm.com
spnode13.ibm.com
spnode14.ibm.com
spnode15.ibm.com
spnode16.ibm.com
}
{
spnode21.ibm.com
spnode17.ibm.com
spnode19.ibm.com
spnode23.ibm.com
spnode25.ibm.com
spnode27.ibm.com
spnode29.ibm.com
spnode31.ibm.com
}
```

Use this file as input to the following **ptpehier** command:

```
ptpehier -i -c spnode09.ibm.com < node_group_spec_file
```

Files

/usr/lpp/ptpe/bin/ptpehier Contains the **ptpehier** command
/etc/group Contains group information
/etc/security/group Contains group information
/usr/lib/nls/msg/*/ptpe.cat Message catalog for the PTPE product
/usr/lib/libptpe.a The Performance Toolbox Parallel Extensions shared library

Prerequisite Information

This command is part of the Performance Toolbox Parallel Extensions for AIX (PTPE) feature of the IBM Parallel System Support Programs for AIX licensed program product.

Related Information

"ptpeconf" on page 86
"ptpectrl" on page 88
"ptpedump" on page 93
"ptpegroup" on page 97
"Understanding the Monitoring Hierarchy" on page 9
"The System Data Repository" in *IBM Parallel System Support Programs for AIX: Administration Guide*

ptpertm

Purpose

ptpertm starts the SP performance data supplier daemon on a node, making the performance data available to IBM Performance Toolbox for AIX.

Syntax

ptpertm [**–p**]

Parameters

-p Informs the command to direct its output to the error log instead of a standard output device. This option is used when running the command from the IBM Performance Toolbox for AIX **xmservd** daemon.

Description

The **ptpertm** command starts the SP performance data supplier program on the local node, and instructs the program to export all available performance data to IBM Performance Toolbox for AIX. This performance information becomes available as performance statistics from the **Spmi** programming library of Performance Toolbox.

This command is provided so that standalone Performance Toolbox sessions can obtain SP-specific performance information from a given node without starting performance information collection throughout the entire SP system.

Description

When **ptpertm** executes, the command invokes the SP Resource Monitor program, **harmld**. This program determines what specific SP hardware and software programs are available, and begins providing performance data for them to IBM Performance Toolbox for AIX. The new information is accessible to **Spmi** and **Rsi** applications, as well as the Performance Toolbox graphical interfaces.

ptpertm only starts providing performance data for the node on which the command is executed. It is provided so that Performance Toolbox can access performance statistics for a specific node without starting data collection throughout the entire monitoring hierarchy. If SP-specific performance data is required for all nodes, start performance data collection using the **ptpectrl** command.

The command can be configured to start whenever the **xmservd** daemon becomes active for a node. To do so, the following entry should be added to the end of the node's **xmservd.res** file:

```
supplier:      /usr/lpp/ptpe/bin/ptpertm.p
```

The **–p** option causes **ptpertm** to direct any error output to the error log instead of to the terminal. This is necessary, since **xmservd** will invoke the command as a daemon, thereby removing any connection to a terminal. If the **xmservd** command is active at the time the **xmservd.res** file is modified, refresh the daemon by issuing a **kill -1** against the **xmservd** daemon. For more information about the **xmservd** daemon and its resolution file, refer to the *IBM Performance Toolbox for AIX: Guide and Reference*.

Although **ptpertm** may be invoked from a command line or a shell script, it is not the preferred method for issuing the command. Running **ptpertm** from the command line should only be done temporarily.

Examples

1. To obtain SP-specific performance information for a node without starting performance data collection on the entire SP, modify the **xmrservd.res** file on the node to include a data supplier entry for the **ptpertm** command:

   ```
   supplier:    /usr/lpp/ptpe/bin/ptpertm -p
   ```

   Test the **xmservd** to determine whether it is active using **ps** command. If the daemon is active, note its process ID. Instruct the daemon to refresh itself by sending a SIGINT with the **kill** command:

   ```
   kill -1 xmservd_process_ID
   ```

   This refreshes the **xmservd** daemon, causing it to reexamine its resolution file. The daemon will discover the new data supplier entry for **ptpertm** and start the command as a daemon. Once it is started, **ptpertm** makes all SP-specific performance data for this node available to Performance Toolbox as new statistics. contains the **spdm_dump** command

Files

/etc/perf/xmservd.res The customized **xmservd** resolution file
/usr/lpp/perfagent/xmservd.res The default **xmservd** resolution file

Prerequisite Information

This command is part of the Performance Toolbox Parallel Extensions for AIX (PTPE) feature of the IBM Parallel System Support Programs for AIX licensed program product.

Related Information

"Step 4: Start Data Collection and Summary" on page 32
"xmservd" in the *IBM Performance Toolbox for AIX: Guide and Reference*

spdmdctrl

Purpose

spdmdctrl starts, stops, adds, and deletes the PTPE subsystem from a node.

Syntax

spdmdctrl {**–a** | **–c** | **–d** | **–h** | **–k** | **–s** }

Parameters

–a Add the PTPE subsystem to the node.

The PTPE **spdmd** daemon is registered as an **inetd** subserver with the System Resource Controller on the node. The **spdmd** daemon is also registered as a service in the **/etc/services** file, and registered with the **inetd** master daemon in the **/etc/inetd.conf** file. The PTPE subsystem is then enabled through the System Resource Controller.

–c Clean up after the subsystem.

All PTPE registrations with the node's System Resource Controller and the **inetd** master daemon are removed. This flag is similar to the **-d** flag, and provided to support the **syspar_ctrl** command. The PTPE subsystem must be stopped with the the **-k** flag before this flag can be used.

–d Delete the subsystem from the node.

All PTPE registrations with the node's System Resource Controller and the **inetd** master daemon are removed. This flag is similar to the **-c** flag.

–h Display usage information for this command.

–k Stop the PTPE subsystem.

The PTPE subsystem is disabled using the System Resource Controller, which disables the PTPE **inetd** subserver daemon **spdmd**. Any subsequent attempts to issue PTPE API instructions to the node will fail. The PTPE subsystem remains registered as a service, and also remains registered with the **inetd** master daemon on the node.

–s Start the PTPE subsystem.

The PTPE subsystem is enabled using the System Resource Controller, which enables the PTPE **inetd** subserver daemon **spdmd**.

Description

spdmdctrl performs partition-sensitive configuration and removal duties for PTPE. This command is provided to support the **syspar_ctrl** command, which configures and removes partition sensitive subsystems. It is also provided to configure and remove PTPE during the normal installation and deinstallation process.

spdmdctrl configures or removes the PTPE subsystem on the node where it is executed. If the command is executed on the control workstation, extra processing is performed when the PTPE subsystem is added for the first time. From the control workstation, the command will reserve a port for exclusive use by the PTPE daemon **spdmd**. The command also creates the **SPDM** and **SPDM_NODES**

classes in the System Data Repository if they do not already exist, using the **ptpeconf** command. Finally, the command issues the **ptpegroup** command to create the **perfmon** user group.

If **spdmdctrl** cannot determine whether it is executing on the control workstation, the command will report a failure and exit. This condition may occur during the installation of PTPE if the **install_cw** command has not been executed. In such cases, the PTPE software will be installed, but the subsystem will not be configured. The subsystem can be configured at a later time, by using either the **syspar_ctrl** command, or by issuing this command again on the control workstation and on all nodes.

Restrictions

You must be logged in as **root** to execute this command.

Results

When adding the PTPE subsystem, all partition sensitive configuration will be performed, and the PTPE **inetd** subserver daemon, **spdmd**, will be enabled for operation.

When stopping the PTPE subsystem, all configuration information and registrations remain, but the PTPE **inetd** subserver daemon, **spdmd**, is disabled.

When removing or cleaning the PTPE subsystem, all registrations for PTPE on the node are removed. However, the information stored in the System Data Repository, including the port reserved for use by PTPE, remains.

If the **spdmdctrl** command encounters an error, it exits with a non-zero value. In all successful conditions, the command exits with a zero value.

Examples

1. To add the PTPE subsystem to the node, enter:
   ```
   /usr/lpp/ptpe/bin/spdmdctrl -a
   ```
2. To start the PTPE subsystem on the node, enter:
   ```
   /usr/lpp/ptpe/bin/spdmdctrl -s
   ```
3. To stop the PTPE subsystem on the node, enter:
   ```
   /usr/lpp/ptpe/bin/spdmdctrl -k
   ```
4. To delete the PTPE subsystem from the node, enter:
   ```
   /usr/lpp/ptpe/bin/spdmdctrl -d
   ```

Files

/usr/lpp/ptpe/bin/spdmdctrl Contains the **spdmdctrl** command
/etc/services Contains group services
/etc/inetd.conf Contains the **inetd** configuration file

Prerequisite Information

This command is part of the Performance Toolbox Parallel Extensions for AIX (PTPE) feature of the IBM Parallel System Support Programs for AIX licensed program product.

Related Information

“ptpeconf” on page 86
“ptpegroup” on page 97

spdm_dump

Purpose

spdm_dump dumps the contents of the local performance information archive to standard output.

Syntax

spdm_dump [**–c**]

Parameters

–c Formats the text dump of the archive files in comma-separated format. The resulting dump can be used as input to a database application.

Description

The **spdm_dump** command reads the contents of the local system's performance information archive, **/var/adm/ptpe/perflog**, and dumps the contents of the archive to either standard output in full content format (the default), or comma-separated format (using the **–c** option). Comma-separated format has the statistic name, timestamp, and value. Full content format adds name length and data type identifier to those elements provide by comma-separated format. The output of this command can be redirected to a file or piped to commands that accept text formatted input for further processing.

Restrictions

This command can only be executed by members of the **perfmon** group.

Results

For each performance statistic entry in the **/var/adm/ptpe/perflog** file, a text record is generated and written to standard output.

When the **–c** option is not specified, the command dumps the archive contents in the full-content (default) format. A single heading line will appear at the head of the output, followed by one or more text records in the following format:

```
Timestamp   Length   Data Type   Statistic   Value
```

where:

- *Timestamp* is the time when the statistic was recorded in MMDDYYYY HH:MM:SS format
- *Length* is the length of the statistic name
- *Data Type* is a single character that represents the data type of the value
- *Statistic* is the full context name of the performance statistic as named by Performance Toolbox
- *Value* is the value of the statistic at the time it was recorded

The maximum length of an output record is 180 characters, depending on the length of the statistic name. The data type identifier is one of the following values:

Performance Toolbox

Code	**Data Type**
i	SiInt
i	SiInt
I	SiUInt or SiUnsign
l	SiLong
L	SiULong
s	SiShort
S	SiUShort
c	SiChar
a	SiAddr
f	SiFloat
d	SiDouble

When the **–c** option is specified, the command dumps the contents of the archive in comma-separated format. The file uses the same naming convention as the default format (**/var/adm/ptpe/perflog.txt**<*date*>). A single heading line, specifying the name of the node, appears at the top of the file, followed by one or more text records in the following format:

```
Time="Timestamp",   Statistic=Value
```

where:

- *Timestamp* is the time when the statistics was recorded in YY/MM/DD HH:MM:SS format
- *Statistic* is the full-context name of the performance statistic as named by Performance Toolbox
- *Value* is the value of the statistic at the time it was recorded.

Statistics names appearing as **????** indicate that the **spdm_dump** utility could not determine the correct IBM Performance Toolbox for AIX statistic name for this data. This should only occur when the performance statistic translation table file, **/var/adm/ptpe/pertab**, is damaged or erased. In such cases it is not possible to retrieve performance statistics names.

Examples

1. To obtain a history of all observances of the performance statistic *PagSp/%totalfree* for the local system in its performance information archive, enter:

```
spdm_dump | grep PagSp/%totalfree | awk '{printf("%s %s %s %s %s: %s\n", $1, $2, $3, $4, $5, $10)}'
```

   The output of **spdm**_dump is filtered through **grep** to obtain only those observances of the *PagSp/%totalfree* variable. The records that pass the **grep** filter are then further stripped to only display the time the statistic was observed in stime() format ($1 through $5 of the **awk** command), a colon (:), and the value of the variable at that time.

2. To dump the entire contents of the node's archive file in comma-separated format and input it to a database application, enter:

   ```
   spdm_dump -c | dbin_prog
   ```

Files

/usr/lpp/ptpe/bin/spdm_dump contains the **spdm_dump** command
/var/adm/ptpe/perflog Contains the archived performance information for the local system
/var/adm/ptpe/perftab Contains the performance statistics archive table

Prerequisite Information

This command is part of the Performance Toolbox Parallel Extensions for AIX (PTPE) feature of the IBM Parallel System Support Programs for AIX licensed program product.

Related Information

“ptpeconf” on page 86
“ptpectrl” on page 88
“ptpedump” on page 93
“ptpegroup” on page 97
“Archiving Performance Data” on page 35
“Monitoring Statistics with xmperf” in *IBM Performance Toolbox for AIX: Guide and Reference*

Chapter 8. The PTPE API Subroutines

Use these subroutines to write application programs that use Performance Toolbox Parallel Extensions for AIX.

PtpeAddHostToList

Purpose

Creates an entry for a specified system in an existing host list.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeAddHostToList(hostname, hanchor)
char         *hostname;
host_list_t   hanchor;
```

Parameters

hostname
: Points to a NULL-terminated character string of up to PTPE_NMLN characters in length, which contains the network name of the system to be added to the host list.

hanchor
: The anchor point of an existing host list.

Description

PtpeAddHostToList allocates an entry to contain host information for the system specified by *hostname*. This entry is inserted at the beginning of the host list, and the internal pointers in the host list anchor are updated to point to the new entry as the “current” entry in the host list.

hostname should be specified in the same manner as the system is known to the Performance Toolbox for AIX monitoring hierarchy. If the system is defined in the monitoring hierarchy by its fully qualified network name, such as spnode05.ibm.com, the same format should be used when adding the system to the host list. If a system is known by more than one name to the network, the name that should be used by this subroutine is the name by which the system is known to the monitoring hierarchy. If the name used by this subroutine is not the same name, in the same format as used by the monitoring hierarchy, subsequent API subroutines will not be able to locate the system in the monitoring hierarchy, causing failures in later API subroutines. No verification is performed on the system name format, nor is the system confirmed to exist, at the time the system is added to the host list.

The host list indicated by *hanchor* may be an empty or non-empty list.

Return Codes

Upon successful completion, this subroutine returns the value of PTPE_SUCCESS. If an error has occurred, the return code is set to one of the following values to indicate the cause of the error:

PTPE_INV_HOSTLIST	*hanchor* has a NULL value.
PTPE_INV_HOSTNAME	*hostname* has a NULL value, or points to a NULL character string.
PTPE_NO_MEMORY	Could not allocate a new entry for the host list.
PTPE_BAD_LOC_PTR	The memory pointers in *hanchor* have been corrupted.

Examples

```
#include <spdm.h>

host_list_t    hanchor;
char           newhost[PTPE_NMLN];
int            rc;

strcpy(newhost, "spnode05.ibm.com");
rc = PtpeAddHostToList(newhost, hanchor);
if (rc != PTPE_SUCCESS) {
/* handle error */
}
```

Related Information

- “PtpeInitHostList” on page 265
- “PtpeDelHostFromList” on page 225
- “PtpeFindHost” on page 233

PtpeAddStatToList

Purpose

Creates an entry for a specified statistic in an existing statistics list.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeAddStatToList(statname, sanchor)
char          *statname;
stat_list_t   sanchor;
```

Parameters

statname
Points to a NULL-terminated character string of up to PTPE_STNL characters in length, which contains the full context path of the statistic to be added to the statistics list. This name is specified in the same format as a fully specified statistic name in the Performance Toolbox **Spmi** library, relative to the top context.

sanchor
The anchor point of an existing statistics list.

Description

PtpeAddStatToList allocates an entry to contain statistic information for the statistic specified by *statname*. This entry is inserted at the beginning of the statistics list, and the internal pointers in the statistics list anchor are updated to point to the new entry as the "current" entry in the statistics list.

statname specifies the full context path of the statistic being entered into the list. This name is provided in **Spmi** format, relative to the top context. The path name does not include the **/host/**<*hostname*> prefix used by the Performance Toolbox **Rsi** programming library. No verification is performed on the format of the statistic name, nor does the API confirm that such a statistic exists, at the time the statistic is added to the statistics list.

The statistics list indicated by *sanchor* may be an empty or non-empty list. If the statistics list was previously assigned to a system by the **PtpeAssignStatsToHost** subroutine, the addition made by this subroutine occurs in the "unassigned" statistics list *sanchor* only; the copy of the statistics list that was previously assigned to a system remains unchanged by this subroutine.

Return Codes

Upon successful completion, this subroutine returns the value of PTPE_SUCCESS. If an error has occurred, the return code is set to one of the following values to indicate the cause of the error:

PTPE_INV_STATLIST	*sanchor* has a NULL value.
PTPE_INV_STATNAME	*hostname* has a NULL value, or points to a NULL character string.
PTPE_NO_MEMORY	Could not allocate a new entry for the statistics list.
PTPE_BAD_LOC_PTR	The memory pointers in *sanchor* have been corrupted.

Examples

```
#include <spdm.h>

stat_list_t     sanchor;
char            newstat[PTPE_STNL];
int             rc;

strcpy(newstat, "Mem/Virt/%free");
rc = PtpeAddStatToList(newstat, sanchor);
if (rc != PTPE_SUCCESS) {
/* handle error */
}
```

Related Information

- “PtpeInitStatList” on page 267
- “PtpeDelStatFromList” on page 227
- “PtpeFindStat” on page 235
- “PtpeAssignStatsToHost” on page 170
- “PtpeGetHostStatList” on page 250

  The discussion of statistical contexts in the Performance Toolbox 1.2 and 2.1 for AIX Guide and Reference.

PtpeArchDisableAllStats

Purpose

Makes all statistics unavailable for archiving on one or more systems in the Performance Toolbox Parallel Extensions monitoring hierarchy.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeArchDisableAllStats(sblock, targets,
                            reply)
session_ptr_t   sblock;
host_list_t     targets;
host_list_t     *reply;
```

Parameters

sblock
The address of a session control block. This information block is created and initialized by the **PtpeOpenSession** subroutine.

targets
The anchor of a host list that contains at least one entry. All systems in this list should not have statistics lists assigned to them.

reply
A pointer to a host_list_t data type, which should be set to a NULL value before invoking this subroutine. Upon successful completion of the subroutine, an anchor point for a host list will be placed at this location. This list will contain entries for each system in the *targets* list. Each entry will contain a result code to indicate whether or not the system was successful in carrying out the request.

Description

PtpeArchDisableAllStats instructs one or more systems in the Performance Toolbox Parallel Extensions to make all the statistics that the system can possibly supply unavailable for recording to the performance information archive. The subroutine is similar to the **PtpeArchEnableStats** subroutine, except the subroutine deactivates all statistics instead of a set of statistics that the application selects.

The application must first establish a PTPE API session by using the **PtpeOpenSession** subroutine. The application must also construct a host list for the command, containing an entry for each system that the application needs to instruct. This list is then provided in the *targets* parameter.

When **PtpeArchDisableAllStats** is executed, the subroutine relays the command, along with a list of the system involved, to the monitoring hierarchy's central coordinator node. The central coordinator node determines which nodes are data managers for the systems in the *targets* list, and forwards the command to only those data manager nodes that manage systems in the host list. The data manager nodes then relay the command to those systems in their reporting groups that have been listed in the *targets* list.

Each system in the *targets* list makes all statistics known to that system unavailable for recording to the performance information archive. The system then indicates its success or failure in the effort to its data manager node.

Data manager nodes construct partial host lists, with an entry for each system in its reporting group that was targeted for the command. The system's success or failure *reply* is placed in the partial host list. This list is then relayed to the central coordinator node, who constructs a complete host list. The complete host list is transmitted back to the **PtpeArchDisableAllStats** subroutine, which provides it to the application in the *reply* parameter.

PtpeArchEnableAllStats will provide an indication of the overall success or failure of the query in the return code:

PTPE_SUCCESS	All systems in the *targets* list successfully responded to the command.
PTPE_LIMITED	Some systems in the *targets* list were unable to respond successfully to the command.
PTPE_API_FAILED	All systems in the *targets* list were unable to respond successfully to the command.

To determine which systems were not successful, the reply list should be scanned, and the results for each system in the list checked using the **PtpeGetHostResult** subroutine. The result code will provide an indication of the error for the system.

Making statistics available or unavailable for archiving has no impact upon which statistics are collected for summarization. This subroutine cannot be issued from a PTPE read-only session.

Return Codes

Upon successful completion, a return code of PTPE_SUCCESS is returned to the caller. If an error occurred, one of the following return codes is provided:

PTPE_SUCCESS_BADR	The central coordinator node has indicated that all systems in the *targets* list responded successfully to the command, but an error occurred in reading the *reply* list response from the central coordintor Node.
PTPE_LIMITED	Some of the systems in the *targets* list were unable to reply to the command. *reply* contains the list of all systems involved in the command; the application can determine which systems failed by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_LIMITED_BADR	Some systems in the *targets* list were unable to respond to the command, and an error occurred in reading the *reply* list response from the central coordinator node.

PTPE_API_FAILED	The central coordinator node has indicated that all systems in the *targets* list failed to respond to the command. *reply* contains all systems involved in the command, along with the reasons for the failure on these systems. The application can determine the cause of the error by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_API_FAILED_BADR	The central coordinator node has indicated that all systems failed to respond to the command, and an error occurred in reading the *reply* list from the central coordinator node.
PTPE_STATE	Performance information collection and summarization is not currently active in the monitoring hierarchy.
PTPE_NO_CONNECT	Could not establish a network connection to the central coordinator node. *reply* list not modified.
PTPE_BAD_SEND	Could not successfully transmit the command to the central coordinator node. *reply* is not modified.
PTPE_BAD_RECEIVE	Error in receiving the reply to the command from the central coordinator node. It is impossible to determine if the query was successful. *reply* is not modified.
PTPE_TIMEOUT	A reply to the command was not received from the central coordinator node in the time allowed. *reply* is not modified.
PTPE_INV_HIERARCHY	The monitoring hierarchy contains a node that is not a member of the currently active system partition. The hierarchy must be repaired by removing any nodes that are not members of the active system partition before this subroutine can be used.
PTPE_INV_HOSTLIST	*targets* either contains a NULL value, or does not anchor a valid host list.
PTPE_INV_PTR	*sblock* has a NULL value, or *reply* is not NULL.
PTPE_EMPTY	*targets* is an empty host list.
PTPE_HOST_NOT_FOUND	One or more systems listed in *targets* could not be found in the monitoring hierarchy. *reply* contains the list of systems that could not be found in the hierarchy.
PTPE_NO_SESSION	A PTPE API session was not previously established by this application (see “PtpeOpenSession” on page 277).
PTPE_NO_HIERARCHY	A monitoring hierarchy does not exist. This subroutine cannot function without a monitoring hierarchy (see “ptpehier” on page 99).
PTPE_NO_MEMORY	Could not allocate enough memory to initiate the query. *reply* is not modified.
PTPE_MEMORY	An internal error occurred.

PTPE_BAD_LOC_PTR	The internal pointers in the *targets* anchor have been corrupted.
PTPE_SDR	An unexpected error occurred while obtaining information from the PTPE data classes in the System Data Repository. These data classes might not exist, or the System Data Repository might be experiencing difficulties.
PTPE_RONLY_SESS	The application attempted to issue this subroutine from a PTPE read-only session. A regular session must be established to issue this instruction. Ensure that the application is executing in the proper SP system partition.

Examples

```
#include <spdm.h>

session_ptr_t   sblock;
host_list_t     targets, reply;
int             rc;

/* set up session and "targets" earlier */
reply = (host_list_t) NULL;
rc = PtpeArchDisableAllStats(sblock, targets,
                             &reply);
if (rc != PTPE_SUCCESS) {
  printf("Not all systems succeded.\n");
}
/* loop through "reply" and check results */
/* on each system with PtpeGetHostResult  */
```

Related Information

- “PtpeOpenSession” on page 277
- “PtpeArchStartHosts” on page 156
- “PtpeArchStartAllHosts” on page 152
- “PtpeArchEnableStats” on page 133
- “PtpeArchEnableAllStats” on page 128
- “PtpeArchDisableStats” on page 123
- “PtpeQueryAvailStats” on page 286
- “PtpeArchQueryStats” on page 147
- “PtpeInitHostList” on page 265
- “PtpeAddHostToList” on page 114
- “PtpeFirstHost” on page 237
- “PtpeNextHost” on page 273
- “PtpeFindHost” on page 233
- “PtpeFreeHostList” on page 241
- “PtpeGetHost” on page 245

- “PtpeGetHostResult” on page 247
- “PtpeInitStatList” on page 267
- “PtpeAddStatToList” on page 116
- “PtpeFirstStat” on page 239
- “PtpeNextStat” on page 275
- “PtpeFindStat” on page 235
- “PtpeFreeStatList” on page 243
- “PtpeAssignStatsToHost” on page 170
- “PtpeGetHostStatList” on page 250
- “PtpeGetStatName” on page 252
- “PtpeGetStatResult” on page 254

PtpeArchDisableStats

Purpose

Makes one or more statistics unavailable for archiving on one or more systems in the Performance Toolbox Parallel Extensions monitoring hierarchy.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeArchDisableStats(sblock, targets, reply)
session_ptr_t   sblock;
host_list_t     targets;
host_list_t     *reply;
```

Parameters

sblock
: The address of a session control block. This information block is created and initialized by the **PtpeOpenSession** subroutine.

targets
: The anchor of a host list that contains at least one entry. All systems in this list should have statistics lists assigned to them (see “PtpeAssignStatsToHost” on page 170).

reply
: A pointer to a host_list_t data type, which should be set to a NULL value before invoking this subroutine. Upon successful completion of the subroutine, an anchor point for a host list will be placed at this location. This list will contain entries for each system in the *targets* list. Each entry will contain a result code to indicate whether or not the system was successful in carrying out the request. Each entry will also have a statistics list assigned to it, containing an entry for each statistic passed to the system, with a result code to indicate if the statistic was successfully activated.

Description

PtpeArchDisableStats instructs one or more systems in the Performance Toolbox Parallel Extensions to make a set of statistics unavailable for recording to the performance information archive.

The application must first establish a PTPE API session by using the **PtpeOpenSession** subroutine. The application must also construct a host list for the command, containing an entry for each system that the application needs to query. This list is then provided in the *targets* parameter. Each entry in the *targets* list should also have a statistics list assigned to it (see “PtpeAssignStatsToHost” on page 170), containing the list of statistics to be made unavailable.

When **PtpeArchDisableStats** is executed, the subroutine relays the command, along with the host list of systems involved, to the monitoring hierarchy's central

coordinator node. The central coordinator node determines which nodes are data managers for the systems in the *targets* list, and forwards the command to only those data manager nodes that manage systems in the host list. The data manager nodes then relay the command to those systems in their reporting groups that have been listed in the *targets* list.

Each system in the *targets* list extracts the list of statistics for their respective entry from the command message, and searches for these statistics. Any of the statistics in this list that also exist on the system are made unavailable for recording to the archive. The PTPE daemons on the system create a *reply* statistics list, containing all statistics passed to this system in the command. The daemons will set the result codes in each statistics list entry to one of the following values:

PTPE_SUCCESS	The statistic was made unavailable for archiving.
PTPE_STAT_NOT_FOUND	The statistic could not be found on the system.

The list of statistics is then relayed to the data manager node.

Data manager nodes construct partial host lists, with an entry for each system in its reporting group that was targeted for the command. The statistics lists received from each node are assigned to their respective host list entries in the data manager node's partial host list. This list is then relayed to the central coordinator node, who constructs a complete host list. The complete host list is transmitted back to the **PtpeArchDisableStats** subroutine, which provides it to the application in the *reply* parameter.

PtpeArchDisableStats will provide an indication of the overall success or failure of the query in the return code:

PTPE_SUCCESS	All systems in the *targets* list successfully responded to the command.
PTPE_LIMITED	Some systems in the *targets* list were unable to respond successfully to the command.
PTPE_API_FAILED	All systems in the *targets* list were unable to respond successfully to the command.

To determine which systems were not successful, the reply list should be scanned, and the results for each system in the list checked using the **PtpeGetHostResult** subroutine. The result code will provide an indication of the error for the system. To check for failures caused by missing statistics, extract the statistics list for each entry by using the **PtpeGetHostStatList** subroutine, and scan the list using the statistics list scanning subroutines.

Making statistics available or unavailable for archiving has no impact upon which statistics are collected for summarization. This subroutine cannot be issued from a PTPE read-only session.

Return Codes

Upon successful completion, a return code of PTPE_SUCCESS is returned to the caller. If an error occurred, one of the following return codes is provided:

PTPE_SUCCESS_BADR	The central coordinator node has indicated that all systems in the *targets* list responded successfully to the command, but an error occurred in reading the *reply* list response from the central coordinator Node.
PTPE_LIMITED	Some of the systems in the *targets* list were unable to reply to the command. *reply* contains the list of all systems involved in the command; the application can determine which systems failed by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_LIMITED_BADR	Some systems in the *targets* list were unable to respond to the command, and an error occurred in reading the *reply* list response from the central coordinator node.
PTPE_API_FAILED	The central coordinator node has indicated that all systems in the *targets* list failed to respond to the command. *reply* contains all systems involved in the command, along with the reasons for the failure on these systems. The application can determine the cause of the error by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_API_FAILED_BADR	The central coordinator node has indicated that all systems failed to respond to the command, and an error occurred in reading the *reply* list from the central coordinator node.
PTPE_STATE	Performance information collection and summarization is not currently active in the monitoring hierarchy.
PTPE_NO_CONNECT	Could not establish a network connection to the central coordinator node. *reply* list not modified.
PTPE_BAD_SEND	Could not successfully transmit the command to the central coordinator node. *reply* is not modified.
PTPE_BAD_RECEIVE	Error in receiving the reply to the command from the central coordinator node. It is impossible to determine if the query was successful. *reply* is not modified.
PTPE_RONLY_SESS	The application attempted to issue this subroutine from a PTPE read-only session. A regular session must be established to issue this instruction. Ensure that the application is executing in the proper SP system partition.
PTPE_TIMEOUT	A reply to the command was not received from the central coordinator node in the time allowed. reply is not modified.
PTPE_INV_HIERARCHY	The monitoring hierarchy contains a node that is not a member of the currently active system partition. The hierarchy must be repaired by removing any nodes that are not members of the active system partition before this subroutine can be used.

PTPE_INV_HOSTLIST	*targets* either contains a NULL value, or does not anchor a valid host list.
PTPE_INV_PTR	*sblock* has a NULL value, or *reply* is not NULL.
PTPE_EMPTY	*targets* is an empty host list.
PTPE_HOST_NOT_FOUND	One or more systems listed in *targets* could not be found in the monitoring hierarchy. *reply* contains the list of systems that could not be found in the hierarchy.
PTPE_INV_STATLIST	One or more systems in the *targets* list does not have a valid statistics list assigned to it.
PTPE_NO_SESSION	A PTPE API session was not previously established by this application (see “PtpeOpenSession” on page 277).
PTPE_NO_HIERARCHY	A monitoring hierarchy does not exist. This subroutine cannot function without a monitoring hierarchy (see “ptpehier” on page 99).
PTPE_NO_MEMORY	Could not allocate enough memory to initiate the query. *reply* is not modified.
PTPE_MEMORY	An internal error occurred.
PTPE_BAD_LOC_PTR	The internal pointers in the *targets* anchor have been corrupted.
PTPE_SDR	An unexpected error occurred while obtaining information from the PTPE data classes in the System Data Repository. These data classes might not exist, or the System Data Repository might be experiencing difficulties.

Examples

```
#include <spdm.h>

session_ptr_t   sblock;
host_list_t     targets, reply;
int             rc;

/* set up session and "targets" earlier */
reply = (host_list_t) NULL;
rc = PtpeArchDisableStats(sblock, targets,
                          &reply);
if (rc != PTPE_SUCCESS) {
  printf("Not all systems succeeded.\n");
}
/* loop through "reply" and check results */
/* on each system with PtpeGetHostResult  */
```

Related Information

- “PtpeOpenSession” on page 277
- “PtpeArchStartHosts” on page 156
- “PtpeArchStartAllHosts” on page 152
- “PtpeArchEnableStats” on page 133

- “PtpeArchEnableAllStats” on page 128
- “PtpeArchDisableStats” on page 123
- “PtpeArchDisableAllStats” on page 118
- “PtpeQueryAvailStats” on page 286
- “PtpeArchQueryStats” on page 147
- “PtpeInitHostList” on page 265
- “PtpeAddHostToList” on page 114
- “PtpeFirstHost” on page 237
- “PtpeNextHost” on page 273
- “PtpeFindHost” on page 233
- “PtpeFreeHostList” on page 241
- “PtpeGetHost” on page 245
- “PtpeGetHostResult” on page 247
- “PtpeInitStatList” on page 267
- “PtpeAddStatToList” on page 116
- “PtpeFirstStat” on page 239
- “PtpeNextStat” on page 275
- “PtpeFindStat” on page 235
- “PtpeFreeStatList” on page 243
- “PtpeAssignStatsToHost” on page 170
- “PtpeGetHostStatList” on page 250
- “PtpeGetStatName” on page 252
- “PtpeGetStatResult” on page 254

PtpeArchEnableAllStats

Purpose

Makes all statistics available for archiving on one or more systems in the Performance Toolbox Parallel Extensions monitoring hierarchy.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeArchEnableAllStats(sblock, targets, reply)
session_ptr_t   sblock;
host_list_t     targets;
host_list_t     *reply;
```

Parameters

sblock
: The address of a session control block. This information block is created and initialized by the **PtpeOpenSession** subroutine.

targets
: The anchor of a host list that contains at least one entry. All systems in this list should not have statistics lists assigned to them.

reply
: A pointer to a host_list_t data type, which should be set to a NULL value before invoking this subroutine. Upon successful completion of the subroutine, an anchor point for a host list will be placed at this location. This list will contain entries for each system in the *targets* list. Each entry will contain a result code to indicate whether or not the system was successful in carrying out the request.

Description

PtpeArchEnableAllStats instructs one or more systems in the Performance Toolbox Parallel Extensions to make all the statistics that the system can possibly supply available for recording to the performance information archive. The subroutine is similar to the **PtpeArchEnableStats** subroutine, except the subroutine activates all statistics instead of a set of statistics that the application selects.

The application must first establish a PTPE API session by using the **PtpeOpenSession** subroutine. The application must also construct a host list for the command, containing an entry for each system that the application needs to instruct. This list is then provided in the *targets* parameter.

When **PtpeArchEnableAllStats** is executed, the subroutine relays the command, along with the host list of systems involved, to the monitoring hierarchy's central coordinator node. The central coordinator node determines which nodes are data managers for the systems in the *targets* list, and forwards the command to only those data manager nodes that manage systems in the host list. The data manager

nodes then relay the command to those systems in their reporting groups that have been listed in the *targets* list.

Each system in the *targets* list makes all statistics known to that system available for recording to the performance information archive. The system then indicates its success or failure in the effort to its data manager node.

Data manager nodes construct partial host lists, with an entry for each system in its reporting group that was targeted for the command. The system's success or failure *reply* is placed in the partial host list. This list is then relayed to the central coordinator node, who constructs a complete host list. The complete host list is transmitted back to the **PtpeArchEnableAllStats** subroutine, which provides it to the application in the *reply* parameter.

PtpeArchEnableAllStats will provide an indication of the overall success or failure of the query in the return code:

PTPE_SUCCESS	All systems in the *targets* list successfully responded to the command.
PTPE_LIMITED	Some systems in the *targets* list were unable to respond successfully to the command.
PTPE_API_FAILED	All systems in the *targets* list were unable to respond successfully to the command.

To determine which systems were not successful, the *reply* list should be scanned, and the results for each system in the list checked using the **PtpeGetHostResult** subroutine. The result code will provide an indication of the error for the system.

Making statistics available or unavailable for archiving has no impact upon which statistics are collected for summarization. This subroutine cannot be issued from a PTPE read-only session.

Return Codes

Upon successful completion, a return code of PTPE_SUCCESS is returned to the caller. If an error occurred, one of the following return codes is provided:

PTPE_SUCCESS_BADR	The central coordinator node has indicated that all systems in the *targets* list responded successfully to the command, but an error occurred in reading the *reply* list response from the central coordinator Node.
PTPE_LIMITED	Some of the systems in the *targets* list were unable to reply to the command. *reply* contains the list of all systems involved in the command; the application can determine which systems failed by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_LIMITED_BADR	Some systems in the *targets* list were unable to respond to the command, and an error occurred in reading the *reply* list response from the central coordinator node.

PTPE_API_FAILED	The central coordinator node has indicated that all systems in the *targets* list failed to respond to the command. *reply* contains all systems involved in the command, along with the reasons for the failure on these systems. The application can determine the cause of the error by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_API_FAILED_BADR	The central coordinator node has indicated that all systems failed to respond to the command, and an error occurred in reading the *reply* list from the central coordinator node.
PTPE_STATE	Performance information collection and summarization is not currently active in the monitoring hierarchy.
PTPE_NO_CONNECT	Could not establish a network connection to the central coordinator node. *reply* list not modified.
PTPE_BAD_SEND	Could not successfully transmit the command to the central coordinator node. *reply* is not modified.
PTPE_BAD_RECEIVE	Error in receiving the reply to the command from the central coordinator node. It is impossible to determine if the query was successful. *reply* is not modified.
PTPE_RONLY_SESS	The application attempted to issue this subroutine from a PTPE read-only session. A regular session must be established to issue this instruction. Ensure that the application is executing in the proper SP system partition.
PTPE_TIMEOUT	A reply to the command was not received from the central coordinator node in the time allowed. reply is not modified.
PTPE_INV_HIERARCHY	The monitoring hierarchy contains a node that is not a member of the currently active system partition. The hierarchy must be repaired by removing any nodes that are not members of the active system partition before this subroutine can be used.
PTPE_INV_HOSTLIST	*targets* either contains a NULL value, or does not anchor a valid host list.
PTPE_INV_PTR	*sblock* has a NULL value, or *reply* is not NULL.
PTPE_EMPTY	*targets* is an empty host list.
PTPE_HOST_NOT_FOUND	One or more systems listed in *targets* could not be found in the monitoring hierarchy. *reply* contains the list of systems that could not be found in the hierarchy.
PTPE_NO_SESSION	A PTPE API session was not previously established by this application (see “PtpeOpenSession” on page 277)
PTPE_NO_HIERARCHY	A monitoring hierarchy does not exist. This subroutine cannot function without a monitoring hierarchy (see “ptpehier” on page 99)

PTPE_NO_MEMORY	Could not allocate enough memory to initiate the query. *reply* is not modified.
PTPE_MEMORY	An internal error occurred.
PTPE_BAD_LOC_PTR	The internal pointers in the *targets* anchor have been corrupted.
PTPE_SDR	An unexpected error occurred while obtaining information from the PTPE data classes in the System Data Repository. These data classes might not exist, or the System Data Repository might be experiencing difficulties.

Examples

```
#include <spdm.h>

session_ptr_t   sblock;
host_list_t     targets, reply;
int             rc;

/* set up session and "targets" earlier */
reply = (host_list_t) NULL;
rc = PtpeArchEnableAllStats(sblock, targets,
                            &reply);
if (rc != PTPE_SUCCESS) {
  printf("Not all systems succeded.\n");
}
/* loop through "reply" and check results */
/* on each system with PtpeGetHostResult  */
```

Related Information

- “PtpeOpenSession” on page 277
- “PtpeArchStartHosts” on page 156
- “PtpeArchStartAllHosts” on page 152
- “PtpeArchEnableStats” on page 133
- “PtpeArchEnableAllStats” on page 128
- “PtpeArchDisableStats” on page 123
- “PtpeArchDisableAllStats” on page 118
- “PtpeQueryAvailStats” on page 286
- “PtpeArchQueryStats” on page 147
- “PtpeInitHostList” on page 265
- “PtpeAddHostToList” on page 114
- “PtpeFirstHost” on page 237
- “PtpeNextHost” on page 273
- “PtpeFindHost” on page 233
- “PtpeFreeHostList” on page 241
- “PtpeGetHost” on page 245

- “PtpeGetHostResult” on page 247
- “PtpeInitStatList” on page 267
- “PtpeAddStatToList” on page 116
- “PtpeFirstStat” on page 239
- “PtpeNextStat” on page 275
- “PtpeFindStat” on page 235
- “PtpeFreeStatList” on page 243
- “PtpeAssignStatsToHost” on page 170
- “PtpeGetHostStatList” on page 250
- “PtpeGetStatName” on page 252
- “PtpeGetStatResult” on page 254

PtpeArchEnableStats

Purpose

Makes one or more statistics available for archiving on one or more systems in the Performance Toolbox Parallel Extensions monitoring hierarchy.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeArchEnableStats(sblock, targets, reply)
session_ptr_t   sblock;
host_list_t     targets;
host_list_t     *reply;
```

Parameters

sblock
The address of a session control block. This information block is created and initialized by the **PtpeOpenSession** subroutine.

targets
The anchor of a host list that contains at least one entry. All systems in this list should have statistics lists assigned to them (see “PtpeAssignStatsToHost” on page 170).

reply
A pointer to a host_list_t data type, which should be set to a NULL value before invoking this subroutine. Upon successful completion of the subroutine, an anchor point for a host list will be placed at this location. This list will contain entries for each system in the *targets* list. Each entry will contain a result code to indicate whether or not the system was successful in carrying out the request. Each entry will also have a statistics list assigned to it, containing an entry for each statistic passed to the system, with a result code to indicate if the statistic was successfully activated.

Description

PtpeArchEnableStats instructs one or more systems in the Performance Toolbox Parallel Extensions to make a set of statistics available for recording to the performance information archive.

The application must first establish a PTPE API session by using the **PtpeOpenSession** subroutine. The application must also construct a host list for the command, containing an entry for each system that the application needs to query. This list is then provided in the *targets* parameter. Each entry in the *targets* list should also have a statistics list assigned to it (see “PtpeAssignStatsToHost” on page 170), containing the list of statistics to be made available.

When **PtpeArchEnableStats** is executed, the subroutine relays the command, along with the host list of systems involved, to the monitoring hierarchy's central

coordinator node. The central coordinator node determines which nodes are data managers for the systems in the *targets* list, and forwards the command to only those data manager nodes that manage systems in the host list. The data manager nodes then relay the command to those systems in their reporting groups that have been listed in the *targets* list.

Each system in the *targets* list extracts the list of statistics for their respective entry from the command message, and searches for these statistics. Any of the statistics in this list that also exist on the system are made available for recording to the archive. The PTPE daemons on the system create a *reply* statistics list, containing all statistics passed to this system in the command. The daemons will set the result codes in each statistics list entry to one of the following values:

PTPE_SUCCESS	The statistic was made available for archiving.
PTPE_STAT_NOT_FOUND	The statistic could not be found on the system.

The list of statistics is then relayed to the data manager node.

Data manager nodes construct partial host lists, with an entry for each system in its reporting group that was targeted for the command. The statistics lists received from each node are assigned to their respective host list entries in the data manager node's partial host list. This list is then relayed to the central coordinator node, who constructs a complete host list. The complete host list is transmitted back to the **PtpeArchEnableStats** subroutine, which provides it to the application in the *reply* parameter.

PtpeArchEnableStats will provide an indication of the overall success or failure of the query in the return code:

PTPE_SUCCESS	All systems in the *targets* list successfully responded to the command.
PTPE_LIMITED	Some systems in the *targets* list were unable to respond successfully to the command.
PTPE_API_FAILED	All systems in the *targets* list were unable to respond successfully to the command.

To determine which systems were not successful, the reply list should be scanned, and the results for each system in the list checked using the **PtpeGetHostResult** subroutine. The result code will provide an indication of the error for the system. To check for failures caused by missing statistics, extract the statistics list for each entry by using the **PtpeGetHostStatList** subroutine, and scan the list using the statistics list scanning subroutines.

Making statistics available or unavailable for archiving has no impact upon which statistics are collected for summarization. This subroutine cannot be issued from a PTPE read-only session.

Return Codes

Upon successful completion, a return code of PTPE_SUCCESS is returned to the caller. If an error occurred, one of the following return codes is provided:

PTPE_SUCCESS_BADR	The central coordinator node has indicated that all systems in the *targets* list responded successfully to the command, but an error occurred in reading the *reply* list response from the central coordinator Node.
PTPE_LIMITED	Some of the systems in the *targets* list were unable to reply to the command. *reply* contains the list of all systems involved in the command; the application can determine which systems failed by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_LIMITED_BADR	Some systems in the *targets* list were unable to respond to the command, and an error occurred in reading the *reply* list response from the central coordinator node.
PTPE_API_FAILED	The central coordinator node has indicated that all systems in the *targets* list failed to respond to the command. *reply* contains all systems involved in the command, along with the reasons for the failure on these systems. The application can determine the cause of the error by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_API_FAILED_BADR	The central coordinator node has indicated that all systems failed to respond to the command, and an error occurred in reading the *reply* list from the central coordinator node.
PTPE_STATE	Performance information collection and summarization is not currently active in the monitoring hierarchy.
PTPE_NO_CONNECT	Could not establish a network connection to the central coordinator node. *reply* list not modified.
PTPE_BAD_SEND	Could not successfully transmit the command to the central coordinator node. *reply* is not modified.
PTPE_BAD_RECEIVE	Error in receiving the reply to the command from the central coordinator node. It is impossible to determine if the query was successful. *reply* is not modified.
PTPE_RONLY_SESS	The application attempted to issue this subroutine from a PTPE read-only session. A regular session must be established to issue this instruction. Ensure that the application is executing in the proper SP system partition.
PTPE_TIMEOUT	A reply to the command was not received from the central coordinator node in the time allowed. reply is not modified.
PTPE_INV_HIERARCHY	The monitoring hierarchy contains a node that is not a member of the currently active system partition. The hierarchy must be repaired by removing any nodes that are not members of the active system partition before this subroutine can be used.

PTPE_INV_HOSTLIST	*targets* either contains a NULL value, or does not anchor a valid host list.
PTPE_INV_PTR	*sblock* has a NULL value, or *reply* is not NULL.
PTPE_EMPTY	*targets* is an empty host list.
PTPE_HOST_NOT_FOUND	One or more systems listed in *targets* could not be found in the monitoring hierarchy. *reply* contains the list of systems that could not be found in the hierarchy.
PTPE_INV_STATLIST	One or more systems in the *targets* list does not have a valid statistics list assigned to it.
PTPE_NO_SESSION	A PTPE API session was not previously established by this application (see "PtpeOpenSession" on page 277).
PTPE_NO_HIERARCHY	A monitoring hierarchy does not exist. This subroutine cannot function without a monitoring hierarchy (see "ptpehier" on page 99).
PTPE_NO_MEMORY	Could not allocate enough memory to initiate the query. *reply* is not modified.
PTPE_MEMORY	An internal error occurred.
PTPE_BAD_LOC_PTR	The internal pointers in the *targets* anchor have been corrupted.
PTPE_SDR	An unexpected error occurred while obtaining information from the PTPE data classes in the System Data Repository. These data classes might not exist, or the System Data Repository might be experiencing difficulties.

Examples

```
#include <spdm.h>

session_ptr_t   sblock;
host_list_t     targets, reply;
int             rc;

/* set up session and "targets" earlier */
reply = (host_list_t) NULL;
rc = PtpeArchEnableStats(sblock, targets,
                         &reply);
if (rc != PTPE_SUCCESS) {
  printf("Not all systems succeeded.\n");
}
/* loop through "reply" and check results */
/* on each system with PtpeGetHostResult  */
```

Related Information

- "PtpeOpenSession" on page 277
- "PtpeColStart" on page 215
- "PtpeArchStartHosts" on page 156
- "PtpeArchStartAllHosts" on page 152

- “PtpeArchEnableStats” on page 133
- “PtpeArchEnableAllStats” on page 128
- “PtpeArchDisableStats” on page 123
- “PtpeArchDisableAllStats” on page 118
- “PtpeQueryAvailStats” on page 286
- “PtpeArchQueryStats” on page 147
- “PtpeInitHostList” on page 265
- “PtpeAddHostToList” on page 114
- “PtpeFirstHost” on page 237
- “PtpeNextHost” on page 273
- “PtpeFindHost” on page 233
- “PtpeFreeHostList” on page 241
- “PtpeGetHost” on page 245
- “PtpeGetHostResult” on page 247
- “PtpeInitStatList” on page 267
- “PtpeAddStatToList” on page 116
- “PtpeFirstStat” on page 239
- “PtpeNextStat” on page 275
- “PtpeFindStat” on page 235
- “PtpeFreeStatList” on page 243
- “PtpeAssignStatsToHost” on page 170
- “PtpeGetHostStatList” on page 250
- “PtpeGetStatName” on page 252
- “PtpeGetStatResult” on page 254

PtpeArchGetStats

Purpose

Retrieves recorded values for a set of statistics on one or more systems in the Performance Toolbox Parallel Extensions monitoring hierarchy.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeArchGetStats(sblock, targets, reply)
session_ptr_t   sblock;
host_list_t     targets;
host_list_t     *reply;
```

Parameters

sblock
: The address of a session control block. This information block is created and initialized by the **PtpeOpenSession** subroutine.

targets
: The anchor of a host list that contains at least one entry. All systems in this list should have statistics lists assigned to them (see “PtpeAssignStatsToHost” on page 170).

reply
: A pointer to a host_list_t data type, which should be set to a NULL value before invoking this subroutine. Upon successful completion of the subroutine, an anchor point for a host list will be placed at this location. This list will contain entries for each system in the *targets* list. Each entry will contain a result code to indicate whether or not the system was successful in carrying out the request. Each entry will also have a statistics list assigned to it, containing an entry for each statistic passed to the system, either containing the statistics current value or a result code indicating why the statistic could not be retrieved.

Description

PtpeArchGetStats retrieves a set of statistics from the performance information archives on one or more systems in the Performance Toolbox Parallel Extension monitoring hierarchy.

The application must first establish a PTPE API session by using the **PtpeOpenSession** subroutine. The application must also construct a host list for the command, containing an entry for each system that the application needs to query. This list is then provided in the *targets* parameter

Each entry in the *targets* list should also have a statistics list assigned to it (see “PtpeAssignStatsToHost” on page 170), containing the list of statistics to be retrieved. Each statistic entry should have its timestamp set with the

PtpeSetStatTime subroutine, to indicate which observation for the statistic should be obtained from the performance information archive.

This subroutine does not require performance information collection or archiving to be currently active in the monitoring hierarchy. The only requirement is that a performance information archive file exist on all systems in the *targets* list.

When **PtpeArchGetStats** is executed, the subroutine relays the command, along with the host list of systems involved, to the monitoring hierarchy's central coordinator node. The central coordinator node determines which nodes are managers for the systems in the *targets* list, and forwards the command to only those data manager nodes that manage systems in the host list. The data manager nodes then relay the command to those systems in their reporting groups that have been listed in the *targets* list.

Each system in the *targets* list extracts the list of statistics for their respective entry from the command message, and searches for these statistics in the archive. The method used to locate a statistic in the archive depends on the setting of the statistic's timestamp:

PTPE_EARLIEST	The first observance for the statistic is found, and the value of the statistic in that observance is retrieved.
PTPE_LATEST	The last observance of the statistic is found, and the value of the statistic in that observance is retrieved.
PTPE_MATCH	The date and time supplied by the application is used to locate the observance in the archive. The observance closest to the data and time specified by the application (without exceeding the data and time specified by the application) is found, and the value of the statistic in that observance is retrieved The daemons will set the value field for each statistic that was located, and will also set the result codes in each statistics list entry to one of the following values:
PTPE_SUCCESS	The statistic was located in the archive.
PTPE_TIME_APPROX	An exact match for the data and time could not be found in the archive, so the closest entry was substituted in its place.
PTPE_STAT_NOT_FOUND	The statistic could not be found in the archive. The list of statistics is then relayed to the data manager node.

Data manager nodes construct partial host lists, with an entry for each system in its reporting group that was targeted for the command. The statistics lists received from each node are assigned to their respective host list entries in the data manager node's partial host list. This list is then relayed to the central coordinator node, who constructs a complete host list. The complete host list is transmitted back to the **PtpeArchGetStats** subroutine, which provides it to the application in the *reply* parameter.

PtpeArchGetStats will provide an indication of the overall success or failure of the query in the return code:

PTPE_SUCCESS	All systems in the *targets* list successfully responded to the query.
PTPE_LIMITED	Some systems in the *targets* list were unable to respond successfully to the query.
PTPE_API_FAILED	All systems in the *targets* list were unable to respond successfully to the query.

To determine which systems were not successful, the reply list should be scanned, and the results for each system in the list checked using the **PtpeGetHostResult** subroutine. The result code will provide an indication of the error for the system. To check for failures caused by missing statistics, extract the statistics list for each entry by using the **PtpeGetHostStatList** subroutine, and scan the list using the statistics list scanning subroutines.

Return Codes

Upon successful completion, a return code of PTPE_SUCCESS is returned to the caller. If an error occurred, one of the following return codes is provided:

PTPE_SUCCESS_BADR	The central coordinator node has indicated that all systems in the *targets* list responded successfully to the query, but an error occurred in reading the *reply* list response from the central coordinator Node.
PTPE_LIMITED	Some of the systems in the *targets* list were unable to reply to the query. The list *reply* points to contains all systems involved in the query; the application can determine which systems failed by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_RONLY_SESS	The application attempted to issue this subroutine from a PTPE read-only session. A regular session must be established to issue this instruction. Ensure that the application is executing in the proper SP system partition.
PTPE_LIMITED_BADR	Some systems in the *targets* list were unable to respond to the query, and an error occurred in reading the *reply* list response from the central coordinator node.
PTPE_API_FAILED	The central coordinator node has indicated that all systems in the *targets* list failed to respond to the query. *reply* contains all systems involved in the query, along with the reasons for the failure on these systems. The application can determine the cause of the error by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.

PTPE_API_FAILED_BADR	The central coordinator node has indicated that all systems failed to respond to the query, and an error occurred in reading the *reply* list from the central coordinator node.
PTPE_NO_CONNECT	Could not establish a network connection to the central coordinator node. *reply* list not modified.
PTPE_BAD_SEND	Could not successfully transmit the command to the central coordinator node. *reply* is not modified.
PTPE_BAD_RECEIVE	Error in receiving the reply to the command from the central coordinator node. It is impossible to determine if the query was successful. *reply* is not modified.
PTPE_TIMEOUT	A reply to the command was not received from the central coordinator node in the time allowed. reply is not modified.
PTPE_INV_HIERARCHY	The monitoring hierarchy contains a node that is not a member of the currently active system partition. The hierarchy must be repaired by removing any nodes that are not members of the active system partition before this subroutine can be used.
PTPE_INV_HOSTLIST	*targets* either contains a NULL value, or does not anchor a valid host list.
PTPE_INV_PTR	*sblock* has a NULL value, or *reply* is not NULL.
PTPE_EMPTY	*targets* is an empty host list.
PTPE_HOST_NOT_FOUND	One or more systems listed in *targets* could not be found in the monitoring hierarchy. *reply* contains the list of systems that could not be found in the hierarchy.
PTPE_INV_STATLIST	One or more systems in the *targets* list does not have a valid statistics list assigned to it.
PTPE_NO_SESSION	A PTPE API session was not previously established by this application (see "PtpeOpenSession" on page 277)
PTPE_NO_HIERARCHY	A monitoring hierarchy does not exist. This subroutine cannot function without a monitoring hierarchy (see "ptpehier" on page 99)
PTPE_NO_MEMORY	Could not allocate enough memory to initiate the query. *reply* is not modified.
PTPE_MEMORY	An internal error occurred.
PTPE_BAD_LOC_PTR	The internal pointers in the *targets* anchor have been corrupted.
PTPE_SDR	An unexpected error occurred while obtaining information from the PTPE data classes in the System Data Repository. These data classes might not exist, or the System Data Repository might be experiencing difficulties.

Examples

```
#include <spdm.h>

session_ptr_t   sblock;
host_list_t     targets, reply;
int             rc;

/* set up session and "targets" earlier */
reply = (host_list_t) NULL;
rc = PtpeArchGetStats(sblock, targets,
                      &reply);
if (rc != PTPE_SUCCESS ) {
  printf("Not all system succeeded.\n");
}
/* loop through "reply" and check results */
/* on each system with PtpeGetHostResult  */
```

Related Information

- “PtpeOpenSession” on page 277
- “PtpeArchStartHosts” on page 156
- “PtpeArchStartAllHosts” on page 152
- “PtpeArchEnableStats” on page 133
- “PtpeArchDisableStats” on page 123
- “PtpeArchDisableAllStats” on page 118
- “PtpeQueryAvailStats” on page 286
- “PtpeArchQueryStats” on page 147
- “PtpeInitHostList” on page 265
- “PtpeAddHostToList” on page 114
- “PtpeFirstHost” on page 237
- “PtpeNextHost” on page 273
- “PtpeFindHost” on page 233
- “PtpeFreeHostList” on page 241
- “PtpeGetHost” on page 245
- “PtpeGetHostResult” on page 247
- “PtpeInitStatList” on page 267
- “PtpeAddStatToList” on page 116
- “PtpeFirstStat” on page 239
- “PtpeNextStat” on page 275
- “PtpeFindStat” on page 235
- “PtpeFreeStatList” on page 243
- “PtpeAssignStatsToHost” on page 170
- “PtpeGetHostStatList” on page 250
- “PtpeGetStatName” on page 252

- “PtpeGetStatResult” on page 254

PtpeArchQueryState

Purpose

Determines the current status of performance information archiving in the Performance Toolbox Parallel Extensions monitoring hierarchy.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeArchQueryState(sblock)
session_ptr_t   sblock;
```

Parameters

sblock
: The address of a session control block. This information block is created and initialized by the **PtpeOpenSession** subroutine.

Description

PtpeArchQueryStates reports to the calling application whether or not performance information archiving is active in the monitoring hierarchy.

The application must first establish a PTPE API session by using the **PtpeOpenSession** subroutine. The inquiry does not use a host list.

The status of collection and summarization is reflected in the return code, which will be one of three status values:

PTPE_ARCH_ACTIVE	Performance information collection and summarization is currently active on at least one system in the monitoring hierarchy.
PTPE_ARCH_OFF	Performance information collection and summarization is not active on any system in the monitoring hierarchy.

On rare occasions, such as when the **ptpectrl -r** command or the **PtpeArchStartHosts** subroutine are interrupted, some systems may not have been able to fully abort the archive startup attempt. This status is indicated by the following return code:

PTPE_COL_ERROR	Cannot determine status of performance information collection and summarization.

While this status persists, other PTPE commands and API subroutine may not function properly. When this status is returned by this subroutine, the application should execute the **PtpeArchStopHosts** or **PtpeColStop** subroutines in an attempt to clear this error state.

Return Codes

Upon successful completion, a return code of PTPE_ARCH_ACTIVE or PTPE_ARCH_OFF is returned to the caller. If an error occurred, one of the following return codes is provided:

PTPE_INV_PTR	*sblock* has a NULL value.
PTPE_NO_SESSION	A PTPE API session was not previously established by this application (see “PtpeOpenSession” on page 277).
PTPE_NO_MEMORY	Could not allocate enough memory to initiate the query.
PTPE_MEMORY	An internal error occurred.
PTPE_NO_LOCK_OBJ	The PTPE data classes do not exist in the System Data Repository (see “ptpeconf” on page 86).
PTPE_SDR	An unexpected error occurred while obtaining information from the PTPE data classes in the System Data Repository. These data classes might not exist, or the System Data Repository might be experiencing difficulties.

Examples

```
#include <spdm.h>

session_ptr_t   sblock;
host_list_t     mgrs, nodes;
int             rc;

/* set up session */
rc = PtpeArchQueryState(sblock);
switch (rc) {
case PTPE_ARCH_ACTIVE:
     printf("Archiving is active\n");
     break;
case PTPE_ARCH_OFF:
     printf("Archiving is inactive\n");
     break;
case PTPE_COL_ERROR:
     printf("Cannot determine status -");
     printf("attempting to reset status.\n");
     mgrs = (host_list_t) NULL;
     nodes = (host_list_t) NULL;
     rc = PtpeColStop(sblock, &mgrs, &nodes);
     /* check results from PtpeColStop */
     break;
default:
/* handle other error */
}
```

Related Information

- “PtpeOpenSession” on page 277
- “PtpeArchStartHosts” on page 156
- “PtpeArchStartAllHosts” on page 152
- “PtpeArchStopHosts” on page 165
- “PtpeArchStopAllHosts” on page 161
- “PtpeColStart” on page 215
- “PtpeColStop” on page 220
- “ptpectrl” on page 88

PtpeArchQueryStats

Purpose

Retrieves the list of statistics being recorded to the performance information archive on one or more systems in the monitoring hierarchy.

Library

libptpe.a

Syntax

```
#include <spdm.h>
int PtpeArchQueryStats(sblock, targets, reply)
session_ptr_t   sblock;
host_list_t     targets;
host_list_t     *reply;
```

Parameters

sblock
: The address of a session control block. This information block is created and initialized by the **PtpeOpenSession** subroutine.

targets
: The anchor of a host list that contains at least one entry. This list should not contain any systems that have statistics assigned to them.

reply
: A pointer to a host_list_t data type, which should be set to a NULL value before invoking this subroutine. Upon successful completion of the subroutine, an anchor point for a host list will be placed at this location. This list will contain entries for each system in the *targets* list. A statistics list will be assigned to those systems that are recording performance information, and will contain an entry for each statistic being archived on that system.

Description

PtpeArchQueryStats asks the Performance Toolbox Parallel Extensions to provide the list of statistics being recorded to the performance information archive by one or more systems in the monitoring hierarchy.

The application must first establish a PTPE API session by using the **PtpeOpenSession** subroutine. The application must also construct a host list for the query, containing an entry for each system that the application needs to query. This list is then provided in the *targets* parameter.

When **PtpeArchQueryStats** is executed, the subroutine relays the query command, along with the host list of systems to query, to the monitoring hierarchy's central coordinator node. The central coordinator node determines which nodes are data managers for the systems in the *targets* list, and forwards the query to only those data manager nodes that manage systems in the query list. The data manager nodes then relay the query to those systems in their reporting groups that have been listed in the targets list.

Each system in the *targets* list constructs a statistics list, containing one entry for each statistics that system is currently making available to Performance Toolbox Parallel Extensions for archiving. On systems that also double as data manager nodes, this list will also include any summary statistics that the node prepares. This list is then relayed back to the data manager node.

Data manager nodes construct partial host lists, with an entry for each system in its reporting group that was targeted for the query. The statistics lists received from each node are assigned to their respective host list entries in the data manager node's partial host list. This list is then relayed to the central coordinator node, who constructs a complete host list. The complete host list is transmitted back to the **PtpeArchQueryStats** subroutine, which provides it to the application in the *reply* parameter.

PtpeArchQueryStats will provide an indication of the overall success or failure of the query in the return code:

PTPE_SUCCESS	All systems in the *targets* list successfully responded to the query.
PTPE_LIMITED	Some systems in the *targets* list were unable to respond successfully to the query.
PTPE_API_FAILED	All systems in the *targets* list were unable to respond successfully to the query.

To determine which systems were not successful, the reply list should be scanned, and the results for each system in the list checked using the **PtpeGetHostResult** subroutine. The result code will provide an indication of the error for the system. To determine what statistics are being archived on those systems that successfully responded to the query, extract the statistics list for each entry by using the **PtpeGetHostStatList** subroutine, and scan the list using the statistics list scanning subroutines.

Return Codes

Upon successful completion, a return code of PTPE_SUCCESS is returned to the caller. If an error occurred, one of the following return codes is provided:

PTPE_SUCCESS_BADR	The central coordinator node has indicated that all systems in the *targets* list responded successfully to the query, but an error occurred in reading the *reply* list response from the central coordinator Node. The application should treat this as an error, since the contents of the *reply* list cannot be guaranteed to be accurate.
PTPE_LIMITED	Some of the systems in the *targets* list were unable to reply to the query. The list *reply* points to contains all systems involved in the query; the application can determine which systems failed by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_LIMITED_BADR	Some systems in the *targets* list were unable to respond to the query, and an error occurred in reading the *reply* list response from the central coordinator node. The application should treat this as an error,

	since the contents of the *reply* list cannot be guaranteed to be accurate.
PTPE_API_FAILED	The central coordinator node has indicated that all systems in the *targets* list failed to respond to the query. *reply* contains all systems involved in the query, along with the reasons for the failure on these systems. The application can determine the cause of the error by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_API_FAILED_BADR	The central coordinator node has indicated that all systems failed to respond to the query, and an error occurred in reading the *reply* list from the central coordinator node. the application should treat this as an error, since the contents of the *reply* list cannot be guaranteed.
PTPE_STATE	Performance information collection and summarization is not currently active, or archiving is not currently active, in the monitoring hierarchy.
PTPE_NO_CONNECT	Could not establish a network connection to the central coordinator. *reply* list not modified.
PTPE_BAD_SEND	Could not successfully transmit the query to the central coordinator node. *reply* is not modified.
PTPE_BAD_RECEIVE	Error in receiving the reply to the query from the central coordinator node. It is impossible to determine if the query was successful. *reply* is not modified.
PTPE_TIMEOUT	A reply to the query was not received from the central coordinator node in the time allowed. reply is not modified.
PTPE_INV_HIERARCHY	The monitoring hierarchy contains a node that is not a member of the currently active system partition. The hierarchy must be repaired by removing any nodes that are not members of the active system partition before this subroutine can be used.
PTPE_INV_HOSTLIST	*targets* either contains a NULL value, or does not anchor a valid host list.
PTPE_INV_PTR	*sblock* has a NULL value, or *reply* is not NULL.
PTPE_EMPTY	*targets* is an empty host list.
PTPE_HOST_NOT_FOUND	One or more systems listed in *targets* could not be found in the monitoring hierarchy. *reply* contains the list of systems that could not be found in the hierarchy.
PTPE_NO_SESSION	A PTPE API session was not previously established by this application (see “PtpeOpenSession” on page 277)
PTPE_NO_HIERARCHY	A monitoring hierarchy does not exist. This subroutine cannot function without a monitoring hierarchy (see “ptpehier” on page 99)

PTPE_NO_MEMORY	Could not allocate enough memory to initiate the query. *reply* is not modified.
PTPE_MEMORY	An internal error occurred.
PTPE_BAD_LOC_PTR	The internal pointers in the *targets* anchor have been corrupted.
PTPE_SDR	An unexpected error occurred while obtaining information from the PTPE data classes in the System Data Repository. These data classes might not exist, or the System Data Repository might be experiencing difficulties.

Examples

```
#include <spdm.h>

session_ptr_t   sblock;
host_list_t     targets, reply;
int             rc;

/* set up session and "targets" earlier */
reply = (host_list_t) NULL;
rc = PtpeArchQueryStats(sblock,targets,&reply);
if (rc != PTPE_SUCCESS) {
      printf("Not all systems succeeded.\n");
}
/* loop through "reply" and check results */
/* on each system with PtpeGetHostResult  */
```

Related Information

- “PtpeOpenSession” on page 277
- “PtpeColStart” on page 215
- “PtpeArchStartHosts” on page 156
- “PtpeArchStartAllHosts” on page 152
- “PtpeArchEnableStats” on page 133
- “PtpeArchEnableAllStats” on page 128
- “PtpeArchDisableStats” on page 123
- “PtpeArchDisableAllStats” on page 118
- “PtpeInitHostList” on page 265
- “PtpeAddHostToList” on page 114
- “PtpeFirstHost” on page 237
- “PtpeNextHost” on page 273
- “PtpeFindHost” on page 233
- “PtpeFreeHostList” on page 241
- “PtpeGetHost” on page 245
- “PtpeGetHostResult” on page 247
- “PtpeInitStatList” on page 267

- “PtpeAddStatToList” on page 116
- “PtpeFirstStat” on page 239
- “PtpeNextStat” on page 275
- “PtpeFindStat” on page 235
- “PtpeFreeStatList” on page 243
- “PtpeAssignStatsToHost” on page 170
- “PtpeGetHostStatList” on page 250
- “PtpeGetStatName” on page 252
- “PtpeGetStatResult” on page 254

PtpeArchStartAllHosts

Purpose

Starts recording of performance information on all systems in the Performance Toolbox Parallel Extensions monitoring hierarchy.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeArchStartAllHosts(sblock, reply)
session_ptr_t   sblock;
host_list_t     *reply;
```

Parameters

sblock
The address of a session control block. This information block is created and initialized by the **PtpeOpenSession** subroutine.

reply
A pointer to a host_list_t data type, which should be set to a NULL value before invoking this subroutine. Upon successful completion of the subroutine, an anchor point for a host list will be placed at this location. This list will contain entries for each system in the *targets* list. Each entry will contain a result code to indicate whether or not the system was successful in carrying out the request.

Description

PtpeArchStartAllHosts instructs all systems in the Performance Toolbox Parallel Extensions to start recording of performance information to the archive. Statistics that have been previously selected for recoding by a **PtpeArchEnableStats** or **PtpeArchEnableAllStats** subroutine will be recorded to the performance information archive.

The application must first establish a PTPE API session by using the **PtpeOpenSession** subroutine. Performance information collection and summarization must also be active in the monitoring hierarchy (see "PtpeColQueryStats" on page 206 or "ptpectrl" on page 88). The command is similar to the **PtpeArchStartHosts** subroutine, except that the command is automatically relayed to all systems in the monitoring hierarchy.

When **PtpeArchStartAllHosts** is executed, the subroutine relays the command to the monitoring hierarchy's central coordinator node. The central coordinator Node forwards the command to all data manager nodes in the monitoring hierarchy, which in turn relay the command to all systems in their reporting groups.

The Performance Toolbox Parallel Extensions daemons on each system will begin recording any performance information that has been marked for archiving. Initially, no statistics are marked for recoding to the archive; the application must specify which statistics are to be recorded by issuing a call to the **PtpeArchEnableStats** or the **PtpeArchEnableAllStats** subroutine, either before or after the call to **PtpeArchStartHosts.** If a daemon cannot begin recording information to its archive, it will report an error back to the managing system.

Data manager nodes construct partial host lists, with an entry for each system in its reporting group. The system's success or failure reply is placed in the partial host list. This list is then relayed to the central coordinator node, who constructs a complete host list. The complete host list is transmitted back to the **PtpeArchStartAllHosts** subroutine, which provides it to the application in the *reply* parameter.

PtpeArchStartAllHosts will provide an indication of the overall success or failure of the command in the return code:

PTPE_SUCCESS	All systems in the monitoring hierarchy successfully responded to the command.
PTPE_LIMITED	Some systems in the monitoring hierarchy were unable to respond successfully to the command.
PTPE_API_FAILED	All systems in the monitoring hierarchy were unable to respond successfully to the command.

To determine which systems were not successful, the reply list should be scanned, and the results for each system in the list checked using the **PtpeGetHostResult** subroutine. The result code will provide an indication of the error for the system. This subroutine cannot be issued from a PTPE read-only session.

Return Codes

Upon successful completion, a return code of PTPE_SUCCESS is returned to the caller. If an error occurred, one of the following return codes is provided:

PTPE_SUCCESS_BADR	The central coordinator node has indicated that all systems in the monitoring hierarchy responded successfully to the command, but an error occurred in reading the *reply* list response from the central coordinator node.
PTPE_LIMITED	Some of the systems in the monitoring hierarchy were unable to reply to the command. *reply* contains the list of all systems involved in the command; the application can determine which systems failed by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_LIMITED_BADR	Some systems in the monitoring hierarchy were unable to respond to the command, and an error occurred in reading the *reply* list response from the central coordinator node.
PTPE_API_FAILED	The central coordinator node has indicated that all systems in the monitoring hierarchy failed to respond to the command. *reply* contains all systems involved in the command, along with the reasons for the failure on these systems. The application can determine the cause of the error by performing the

	PtpeGetHostResult subroutine on all systems in the *reply* list.
PTPE_API_FAILED_BADR	The central coordinator node has indicated that all systems failed to respond to the command, and an error occurred in reading the *reply* list from the central coordinator node.
PTPE_STATE	Performance information collection and summarization is not currently active in the monitoring hierarchy.
PTPE_NO_CONNECT	Could not establish a network connection to the central coordinator node. *reply* list not modified.
PTPE_BAD_SEND	Could not successfully transmit the command to the central coordinator node. *reply* is not modified.
PTPE_BAD_RECEIVE	Error in receiving the reply to the command from the central coordinator node. It is impossible to determine if the query was successful. *reply* is not modified.
PTPE_RONLY_SESS	The application attempted to issue this subroutine from a PTPE read-only session. A regular session must be established to issue this instruction. Ensure that the application is executing in the proper SP system partition.
PTPE_TIMEOUT	A reply to the command was not received from the central coordinator node in the time allowed. reply is not modified.
PTPE_INV_HIERARCHY	The monitoring hierarchy contains a node that is not a member of the currently active system partition. The hierarchy must be repaired by removing any nodes that are not members of the active system partition before this subroutine can be used.
PTPE_INV_PTR	*sblock* has a NULL value, or *reply* is not NULL.
PTPE_NO_SESSION	A PTPE API session was not previously established by this application (see “PtpeOpenSession” on page 277).
PTPE_NO_HIERARCHY	A monitoring hierarchy does not exist. This subroutine cannot function without a monitoring hierarchy (see “ptpehier” on page 99).
PTPE_NO_MEMORY	Could not allocate enough memory to initiate the query. *reply* is not modified.
PTPE_MEMORY	An internal error occurred.
PTPE_SDR	An unexpected error occurred while obtaining information from the PTPE data classes in the System Data Repository. These data classes might not exist, or the System Data Repository might be experiencing difficulties.

Examples

```
#include <spdm.h>

session_ptr_t   sblock;
host_list_t     targets, reply;
int             rc;

/* set up session and "targets" earlier */
reply = (host_list_t) NULL;
rc = PtpeArchStartAllHosts(sblock, &reply);
switch (rc) {
case PTPE_STATE:
     printf("Have to start collection first\n");
     break;
case PTPE_SUCCESS:
     break;
default:
     /* loop through "reply" & check results */
     /* on each system w/ PtpeGetHostResult  */
```

Related Information

- "PtpeOpenSession" on page 277
- "PtpeArchStartHosts" on page 156
- "PtpeArchStopHosts" on page 165
- "PtpeArchStopAllHosts" on page 161
- "PtpeArchEnableStats" on page 133
- "PtpeArchEnableAllStats" on page 128
- "PtpeArchDisableStats" on page 123
- "PtpeQueryAvailStats" on page 286
- "PtpeArchQueryStats" on page 147
- "PtpeInitHostList" on page 265
- "PtpeAddHostToList" on page 114
- "PtpeFirstHost" on page 237
- "PtpeNextHost" on page 273
- "PtpeFindHost" on page 233
- "PtpeFreeHostList" on page 241
- "PtpeGetHost" on page 245
- "PtpeGetHostResult" on page 247

PtpeArchStartHosts

Purpose

Starts recording of performance information on one or more systems in the Performance Toolbox Parallel Extensions monitoring hierarchy.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeArchStartHosts(sblock, targets, reply)
session_ptr_t   sblock;
host_list_t     targets;
host_list_t     *reply;
```

Parameters

sblock
: The address of a session control block. This information block is created and initialized by the **PtpeOpenSession** subroutine.

targets
: The anchor of a host list that contains at least one entry. All systems in this list should not have statistics lists assigned to them.

reply
: A pointer to a host_list_t data type, which should be set to a NULL value before invoking this subroutine. Upon successful completion of the subroutine, an anchor point for a host list will be placed at this location. This list will contain entries for each system in the *targets* list. Each entry will contain a result code to indicate whether or not the system was successful in carrying out the request.

Description

PtpeArchStartHosts instructs one or more systems in the Performance Toolbox Parallel Extensions to start recording of performance information to the archive. Statistics that have been previously selected for recoding by a **PtpeArchEnableStats** or **PtpeArchEnableAllStats** subroutine will be recorded to the performance information archive.

The application must first establish a PTPE API session by using the **PtpeOpenSession** subroutine. Performance information collection and summarization must also be active in the monitoring hierarchy (see "PtpeColStart" on page 215 or "ptpectrl" on page 88). The application must also construct a host list for the command, containing an entry for each system that the application will instruct to begin recording. This list is provided in the *targets* parameter.

When **PtpeArchStartHosts** is executed, the subroutine relays the command, along with the host list of systems involved, to the monitoring hierarchy's central coordinator node. The central coordinator node determines which nodes are data managers for the systems in the *targets* list, and forwards the command to only those data manager nodes that manage systems in the host list. The data manager nodes then relay the command to those systems in their reporting groups that have been listed in the *targets* list.

The Performance Toolbox Parallel Extensions daemons on the systems in the *targets* list will begin recording any performance information that has been marked for archiving. Initially, no statistics are marked for recoding to the archive; the application must specify which statistics are to be recorded by issuing a call to the **PtpeArchEnableStats** or the **PtpeArchEnableAllStats** subroutine, either before or after the call to **PtpeArchStartHosts.** If a daemon cannot begin recording information to its archive, it will report an error back to the managing system.

Data manager nodes construct partial host lists, with an entry for each system in its reporting group that was targeted for the command. The system's success or failure *reply* is placed in the partial host list. This list is then relayed to the central coordinator node, who constructs a complete host list. The complete host list is transmitted back to the **PtpeArchStartHosts** subroutine, which provides it to the application in the *reply* parameter.

PtpeArchStartHosts will provide an indication of the overall success or failure of the command in the return code:

PTPE_SUCCESS	All systems in the *targets* list successfully responded to the command.
PTPE_LIMITED	Some systems in the *targets* list were unable to respond successfully to the command.
PTPE_API_FAILED	All systems in the *targets* list were unable to respond successfully to the command.

To determine which systems were not successful, the reply list should be scanned, and the results for each system in the list checked using the **PtpeGetHostResult** subroutine. The result code will provide an indication of the error for the system. This subroutine cannot be issued from a PTPE read-only session.

Return Codes

Upon successful completion, a return code of PTPE_SUCCESS is returned to the caller. If an error occurred, one of the following return codes is provided:

PTPE_SUCCESS_BADR	The central coordinator node has indicated that all systems in the *targets* list responded successfully to the command, but an error occurred in reading the *reply* list response from the central coordinator Node.
PTPE_LIMITED	Some of the systems in the *targets* list were unable to reply to the command. *reply* contains the list of all systems involved in the command; the application can determine which systems failed by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_LIMITED_BADR	Some systems in the *targets* list were unable to respond to the command, and an error occurred in reading the *reply* list response from the central coordinator node.

PTPE_API_FAILED	The central coordinator node has indicated that all systems in the *targets* list failed to respond to the command. *reply* contains all systems involved in the command, along with the reasons for the failure on these systems. The application can determine the cause of the error by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_API_FAILED_BADR	The central coordinator node has indicated that all systems failed to respond to the command, and an error occurred in reading the *reply* list from the central coordinator node.
PTPE_STATE	Performance information collection and summarization is not currently active in the monitoring hierarchy.
PTPE_NO_CONNECT	Could not establish a network connection to the central coordinator node. *reply* list not modified.
PTPE_BAD_SEND	Could not successfully transmit the command to the central coordinator node. *reply* is not modified.
PTPE_BAD_RECEIVE	Error in receiving the reply to the command from the central coordinator node. It is impossible to determine if the query was successful. *reply* is not modified.
PTPE_RONLY_SESS	The application attempted to issue this subroutine from a PTPE read-only session. A regular session must be established to issue this instruction. Ensure that the application is executing in the proper SP system partition.
PTPE_TIMEOUT	A reply to the command was not received from the central coordinator node in the time allowed. reply is not modified.
PTPE_INV_HIERARCHY	The monitoring hierarchy contains a node that is not a member of the currently active system partition. The hierarchy must be repaired by removing any nodes that are not members of the active system partition before this subroutine can be used.
PTPE_INV_HOSTLIST	*targets* either contains a NULL value, or does not anchor a valid host list.
PTPE_INV_PTR	*sblock* has a NULL value, or *reply* is not NULL.
PTPE_EMPTY	*targets* is an empty host list.
PTPE_HOST_NOT_FOUND	One or more systems listed in *targets* could not be found in the monitoring hierarchy. *reply* contains the list of systems that could not be found in the hierarchy.
PTPE_NO_SESSION	A PTPE API session was not previously established by this application (see “PtpeOpenSession” on page 277)
PTPE_NO_HIERARCHY	A monitoring hierarchy does not exist. This subroutine cannot function without a monitoring hierarchy (see “ptpehier” on page 99).

PTPE_NO_MEMORY	Could not allocate enough memory to initiate the query. *reply* is not modified.
PTPE_MEMORY	An internal error occurred.
PTPE_BAD_LOC_PTR	The internal pointers in the *targets* anchor have been corrupted.
PTPE_SDR	An unexpected error occurred while obtaining information from the PTPE data classes in the System Data Repository. These data classes might not exist, or the System Data Repository might be experiencing difficulties.

Examples

```
#include <spdm.h>

session_ptr_t   sblock;
host_list_t     targets, reply;
int             rc;

/* set up session and "targets" earlier */
reply = (host_list_t) NULL;
rc = PtpeArchStartHosts(sblock, targets,
                        &reply);
switch (rc) {
case PTPE_STATE:
   printf("Have to start collection first\n");
   break;
case PTPE_SUCCESS:
   break;
default:
   /* loop through "reply" & check results */
   /* on each system w/ PtpeGetHostResult  */
```

Related Information

- “PtpeOpenSession” on page 277
- “PtpeArchStartAllHosts” on page 152
- “PtpeArchStopHosts” on page 165
- “PtpeArchStopAllHosts” on page 161
- “PtpeArchEnableStats” on page 133
- “PtpeArchEnableAllStats” on page 128
- “PtpeArchDisableStats” on page 123
- “PtpeQueryAvailStats” on page 286
- “PtpeArchQueryStats” on page 147
- “PtpeInitHostList” on page 265
- “PtpeAddHostToList” on page 114
- “PtpeFirstHost” on page 237
- “PtpeNextHost” on page 273
- “PtpeFindHost” on page 233

- “PtpeFreeHostList” on page 241
- “PtpeGetHost” on page 245
- “PtpeGetHostResult” on page 247

PtpeArchStopAllHosts

Purpose

Ends recording of performance information on all systems in the Performance Toolbox Parallel Extensions monitoring hierarchy.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeArchStopAllHosts(sblock, reply)
session_ptr_t   sblock;
host_list_t     *reply;
```

Parameters

sblock
The address of a session control block. This information block is created and initialized by the **PtpeOpenSession** subroutine.

reply
A pointer to a host_list_t data type, which should be set to a NULL value before invoking this subroutine. Upon successful completion of the subroutine, an anchor point for a host list will be placed at this location. This list will contain entries for each system in the *targets* list. Each entry will contain a result code to indicate whether or not the system was successful in carrying out the request.

Description

PtpeArchStopAllHosts instructs all systems in the Performance Toolbox Parallel Extensions to end recording of performance information to the archive. Recording to the performance information archive is halted, but statistics previously selected for recording by a **PtpeArchEnableStats** or **PtpeArchEnableAllStats** remain selected.

The application must first establish a PTPE API session by using the **PtpeOpenSession** subroutine. Performance information collection and summarization must also be active in the monitoring hierarchy (see “PtpeColStart” on page 215 or “ptpectrl” on page 88). This command is similar to the **PtpeArchStopHosts** subroutine, except that the command is automatically relayed to all systems in the monitoring hierarchy.

When **PtpeArchStopAllHosts** is executed, the subroutine relays the command to the monitoring hierarchy's central coordinator node. The central coordinator node forwards the command to all data manager nodes in the monitoring hierarchy, which in turn relay the command to all systems in their reporting groups.

The Performance Toolbox Parallel Extensions daemons on each system will cease recording any performance information that has been marked for archiving. While this will stop any further updates to the performance information archive for this system, it will not clear any statistics selected by a previous **PtpeArchEnableStats** or **PtpeArchEnableAllStats** subroutine. This permits an application to pause recording of performance information, then resume later without requiring the application to re-select the statistics to be recorded. The system reports its success or failure in ending performance information recording to its managing system.

Data manager nodes construct partial host lists, with an entry for each system in its reporting group. The system's success or failure reply is placed in the partial host list. This list is then relayed to the central coordinator node, who constructs a complete host list. The complete host list is transmitted back to the **PtpeArchStopAllHosts** subroutine, which provides it to the application in the *reply* parameter.

PtpeArchStopAllHosts will provide an indication of the overall success or failure of the command in the return code:

PTPE_SUCCESS	All systems in the monitoring hierarchy successfully responded to the command.
PTPE_LIMITED	Some systems in the monitoring hierarchy were unable to respond successfully to the command.
PTPE_API_FAILED	All systems in the monitoring hierarchy were unable to respond successfully to the command.

To determine which systems were not successful, the reply list should be scanned, and the results for each system in the list checked using the **PtpeGetHostResult** subroutine. The result code will provide an indication of the error for the system. This subroutine cannot be issued from a PTPE read-only session.

Return Codes

Upon successful completion, a return code of PTPE_SUCCESS is returned to the caller. If an error occurred, one of the following return codes is provided:

PTPE_SUCCESS_BADR	The central coordinator node has indicated that all systems in the monitoring hierarchy responded successfully to the command, but an error occurred in reading the *reply* list response from the central coordinator node.
PTPE_LIMITED	Some of the systems in the monitoring hierarchy were unable to reply to the command. *reply* contains the list of all systems involved in the command; the application can determine which systems failed by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_LIMITED_BADR	Some systems in the monitoring hierarchy were unable to respond to the command, and an error occurred in reading the *reply* list response from the central coordinator node.
PTPE_API_FAILED	The central coordinator node has indicated that all systems in the monitoring hierarchy failed to respond to the command. *reply* contains all systems involved in the command, along with the reasons for the failure on these systems. The application can determine the

	cause of the error by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_API_FAILED_BADR	The central coordinator node has indicated that all systems failed to respond to the command, and an error occurred in reading the *reply* list from the central coordinator node.
PTPE_STATE	Performance information collection and summarization is not currently active in the monitoring hierarchy.
PTPE_NO_CONNECT	Could not establish a network connection to the central coordinator node. *reply* list not modified.
PTPE_BAD_SEND	Could not successfully transmit the command to the central coordinator node. *reply* is not modified.
PTPE_BAD_RECEIVE	Error in receiving the reply to the command from the central coordinator node. It is impossible to determine if the query was successful. *reply* is not modified.
PTPE_RONLY_SESS	The application attempted to issue this subroutine from a PTPE read-only session. A regular session must be established to issue this instruction. Ensure that the application is executing in the proper SP system partition.
PTPE_TIMEOUT	A reply to the command was not received from the central coordinator node in the time allowed. reply is not modified.
PTPE_INV_HIERARCHY	The monitoring hierarchy contains a node that is not a member of the currently active system partition. The hierarchy must be repaired by removing any nodes that are not members of the active system partition before this subroutine can be used.
PTPE_INV_PTR	*sblock* has a NULL value, or *reply* is not NULL.
PTPE_NO_SESSION	A PTPE API session was not previously established by this application (see “PtpeOpenSession” on page 277).
PTPE_NO_HIERARCHY	A monitoring hierarchy does not exist. This subroutine cannot function without a monitoring hierarchy (see “ptpehier” on page 99)
PTPE_NO_MEMORY	Could not allocate enough memory to initiate the query. *reply* is not modified.
PTPE_MEMORY	An internal error occurred.
PTPE_SDR	An unexpected error occurred while obtaining information from the PTPE data classes in the System Data Repository. These data classes might not exist, or the System Data Repository might be experiencing difficulties.

Examples

```
#include <spdm.h>

session_ptr_t   sblock;
host_list_t     targets, reply;
int             rc;

/* set up session and "targets" earlier */
reply = (host_list_t) NULL;
rc = PtpeArchStopAllHosts(sblock, &reply);
switch (rc) {
case PTPE_STATE:
   printf("Collection or archiving not ");
   printf("currently active.\n");
   break;
case PTPE_SUCCESS:
   break;
default:
   /* loop through "reply" & check results */
   /* on each system w/ PtpeGetHostResult  */
```

Related Information

- “PtpeOpenSession” on page 277
- “PtpeArchStartHosts” on page 156
- “PtpeArchStartAllHosts” on page 152
- “PtpeArchStopAllHosts” on page 161
- “PtpeArchEnableStats” on page 133
- “PtpeArchEnableAllStats” on page 128
- “PtpeArchDisableStats” on page 123
- “PtpeQueryAvailStats” on page 286
- “PtpeArchQueryStats” on page 147
- “PtpeInitHostList” on page 265
- “PtpeAddHostToList” on page 114
- “PtpeFirstHost” on page 237
- “PtpeNextHost” on page 273
- “PtpeFindHost” on page 233
- “PtpeFreeHostList” on page 241
- “PtpeGetHost” on page 245
- “PtpeGetHostResult” on page 247

PtpeArchStopHosts

Purpose

Ends recording of performance information on one or more systems in the Performance Toolbox Parallel Extensions monitoring hierarchy.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeArchStopHosts(sblock, targets, reply)
session_ptr_t   sblock;
host_list_t     targets;
host_list_t     *reply;
```

Parameters

sblock
The address of a session control block. This information block is created and initialized by the **PtpeOpenSession** subroutine.

targets
The anchor of a host list that contains at least one entry. All systems in this list should not have statistics lists assigned to them.

reply
A pointer to a host_list_t data type, which should be set to a NULL value before invoking this subroutine. Upon successful completion of the subroutine, an anchor point for a host list will be placed at this location. This list will contain entries for each system in the *targets* list. Each entry will contain a result code to indicate whether or not the system was successful in carrying out the request.

Description

PtpeArchStopHosts instructs one or more systems in the Performance Toolbox Parallel Extensions to end recording of performance information to the archive. Recording to the performance information archive is halted, but statistics previously selected for recording by a **PtpeArchEnableStats** or **PtpeArchEnableAllStats** remain selected.

The application must first establish a PTPE API session by using the **PtpeOpenSession** subroutine. Performance information collection and summarization must also be active in the monitoring hierarchy (see “PtpeColStart” on page 215 or “ptpectrl” on page 88). The application must also construct a host list for the command, containing an entry for each system that the application will instruct to cease recording. This list is provided in the *targets* parameter.

When **PtpeArchStopHosts** is executed, the subroutine relays the command, along with the host list of systems involved, to the monitoring hierarchy's central coordinator node. The central coordinator node determines which nodes are data managers for the systems in the *targets* list, and forwards the command to only those data manager nodes that manage systems in the host list. The data manager nodes then relay the command to those systems in their reporting groups that have been listed in the *targets* list.

The Performance Toolbox Parallel Extensions daemons on the systems in the *targets* list will cease recording any performance information that has been marked for archiving. While this will stop any further updates to the performance information archive for this system, it will not clear any statistics selected by a previous **PtpeArchEnableStats** or **PtpeArchEnableAllStats** subroutine. This permits an application to pause recording of performance information, then resume later without requiring the application to re-select the statistics to be recorded. The system reports its success or failure in ending performance information recording to its data manager node.

Data manager nodes construct partial host lists, with an entry for each system in its reporting group that was targeted for the command. The system's success or failure *reply* is placed in the partial host list. This list is then relayed to the central coordinator node, who constructs a complete host list. The complete host list is transmitted back to the **PtpeArchStopHosts** subroutine, which provides it to the application in the *reply* parameter.

PtpeArchStopHosts will provide an indication of the overall success or failure of the command in the return code:

PTPE_SUCCESS	All systems in the *targets* list successfully responded to the command.
PTPE_LIMITED	Some systems in the *targets* list were unable to respond successfully to the command.
PTPE_API_FAILED	All systems in the *targets* list were unable to respond successfully to the command.

To determine which systems were not successful, the reply list should be scanned, and the results for each system in the list checked using the **PtpeGetHostResult** subroutine. The result code will provide an indication of the error for the system. This subroutine cannot be issued from a PTPE read-only session.

Return Codes

Upon successful completion, a return code of PTPE_SUCCESS is returned to the caller. If an error occurred, one of the following return codes is provided:

PTPE_SUCCESS_BADR	The central coordinator node has indicated that all systems in the *targets* list responded successfully to the command, but an error occurred in reading the *reply* list response from the central coordinator Node.
PTPE_LIMITED	Some of the systems in the *targets* list were unable to reply to the command. *reply* contains the list of all systems involved in the command; the application can determine which systems failed by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.

PTPE_LIMITED_BADR	Some systems in the *targets* list were unable to respond to the command, and an error occurred in reading the *reply* list response from the central coordinator node.
PTPE_API_FAILED	The central coordinator node has indicated that all systems in the *targets* list failed to respond to the command. *reply* contains all systems involved in the command, along with the reasons for the failure on these systems. The application can determine the cause of the error by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_API_FAILED_BADR	The central coordinator node has indicated that all systems failed to respond to the command, and an error occurred in reading the *reply* list from the central coordinator node.
PTPE_STATE	Performance information collection and summarization is not currently active, or performance information archiving is not currently active in the monitoring heirarchy.
PTPE_NO_CONNECT	Could not establish a network connection to the central coordinator node. *reply* list not modified.
PTPE_BAD_SEND	Could not successfully transmit the command to the central coordinator node. *reply* is not modified.
PTPE_BAD_RECEIVE	Error in receiving the reply to the command from the central coordinator node. It is impossible to determine if the query was successful. *reply* is not modified.
PTPE_RONLY_SESS	The application attempted to issue this subroutine from a PTPE read-only session. A regular session must be established to issue this instruction. Ensure that the application is executing in the proper SP system partition.
PTPE_TIMEOUT	A reply to the command was not received from the central coordinator node in the time allowed. reply is not modified.
PTPE_INV_HIERARCHY	The monitoring hierarchy contains a node that is not a member of the currently active system partition. The hierarchy must be repaired by removing any nodes that are not members of the active system partition before this subroutine can be used.
PTPE_INV_HOSTLIST	*targets* either contains a NULL value, or does not anchor a valid host list.
PTPE_INV_PTR	*sblock* has a NULL value, or *reply* is not NULL.
PTPE_EMPTY	*targets* is an empty host list.
PTPE_HOST_NOT_FOUND	One or more systems listed in *targets* could not be found in the monitoring hierarchy. *reply* contains the list of systems that could not be found in the hierarchy.

PTPE_NO_SESSION	A PTPE API session was not previously established by this application (see “PtpeOpenSession” on page 277)
PTPE_NO_HIERARCHY	A monitoring hierarchy does not exist. This subroutine cannot function without a monitoring hierarchy (see “ptpehier” on page 99).
PTPE_NO_MEMORY	Could not allocate enough memory to initiate the query. *reply* is not modified.
PTPE_MEMORY	An internal error occurred.
PTPE_BAD_LOC_PTR	The internal pointers in the *targets* anchor have been corrupted.
PTPE_SDR	An unexpected error occurred while obtaining information from the PTPE data classes in the System Data Repository. These data classes might not exist, or the System Data Repository might be experiencing difficulties.

Examples

```
#include <spdm.h>

session_ptr_t   sblock;
host_list_t     targets, reply;
int             rc;

/* set up session and "targets" earlier */
reply = (host_list_t) NULL;
rc = PtpeArchStopHosts(sblock, targets,
                       &reply);
switch (rc) {
case PTPE_STATE:
     printf("Collection or archiving not ");
     printf("active\n");
     break;
case PTPE_SUCCESS:
     break;
default:
     /* loop through "reply" & check results */
     /* on each system w/ PtpeGetHostResult  */
```

Related Information

- “PtpeOpenSession” on page 277
- “PtpeArchStartHosts” on page 156
- “PtpeArchStartAllHosts” on page 152
- “PtpeArchStopAllHosts” on page 161
- “PtpeArchEnableStats” on page 133
- “PtpeArchEnableAllStats” on page 128
- “PtpeArchDisableStats” on page 123
- “PtpeQueryAvailStats” on page 286

- “PtpeArchQueryStats” on page 147
- “PtpeInitHostList” on page 265
- “PtpeAddHostToList” on page 114
- “PtpeFirstHost” on page 237
- “PtpeNextHost” on page 273
- “PtpeFindHost” on page 233
- “PtpeFreeHostList” on page 241
- “PtpeGetHost” on page 245
- “PtpeGetHostResult” on page 247

PtpeAssignStatsToHost

Purpose

Assigns an existing statistics list to the currently referenced host in a host list.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeAssignStatsToHost(sanchor, hanchor)
stat_list_t     sanchor;
host_list_t     hanchor;
```

Parameters

sanchor
The anchor point of an existing non-empty statistics list.

hanchor
The anchor point of a non-empty host list.

Description

PtpeAssignStatsToHost associates the list of statistics contained in the *sanchor* statistics list with the “current” entry in the host list anchored at *hanchor*. In subsequent API subroutines, this list of statistics will be used during the operation on the system whose information is contained in the “current” entry in *hanchor*. This list of statistics will continue to be associated with the system until the statistics list is removed with the **PtpeRemoveStatsFromHost** subroutine.

An application builds a statistics list that contains the list of statistics that the program wishes to retrieve from one or more systems in the Performance Toolbox Parallel Extensions monitoring hierarchy. In order to retrieve these statistics, a host list must be constructed to contain the list of systems involved in the data retrieval process. But to inform the PTPE API subroutines what specific performance statistics need to be retrieved from which systems, the application needs to assign the list of statistics to the systems in the monitoring hierarchy using the **PtpeAssignStatsToHost** subroutine.

A copy of the *sanchor* statistics list is made by this subroutine, and the copy is assigned to the “current” entry in the *hanchor* list. This permits the application to assign the same set of statistics to multiple systems in the *hanchor* host list. Also, since a copy is made and the copy is assigned only to the “current” *hanchor* entry, an application can modify the statistics list and assign a different set of statistics to other systems in the *hanchor* list. By assigning differing statistics lists to different entries in *hanchor*, an API subroutine such as **PtpeArchGetStats** can retrieve different statistics from multiple systems in the same call.

An error results if an uninitialized or empty host list is provided to this subroutine, or if an uninitialized statistics list is used.

Return Codes

Upon successful completion, this subroutine returns the value of PTPE_SUCCESS. If an error has occurred, the return code is set to one of the following values to indicate the cause of the error:

PTPE_INV_STATLIST	*sanchor* has a NULL value.
PTPE_INV_HOSTLIST	*hanchor* has a NULL value.
PTPE_EMPTY	Either *hanchor* or *sanchor* references an empty list.
PTPE_NO_MEMORY	The API subroutine could not allocate enough memory to make a copy of the *sanchor* list.
PTPE_LIST_END	An internal error occurred - the end of the statistics list was prematurely encountered.
PTPE_BAD_LOC_PTR	The memory pointers in *sanchor* have been corrupted.
PTPE_INV_STATNAME	An entry in the *sanchor* statistics list has been corrupted.
PTPE_MEMORY	An internal memory usage error occurred.

Examples

```
#include <spdm.h>

host_list_t   hanchor;
stat_list_t   sanchor;
int           rc;

rc = PtpeFindHost("spnode05.ibm.com",
                  hanchor);
rc = PtpeAssignStatsToHost(sanchor, hanchor);
if (rc != PTPE_SUCCESS) {
       /* handle error condition */
}
```

Related Information

- "PtpeInitHostList" on page 265
- "PtpeInitStatList" on page 267
- "PtpeRemoveStatsFromHost" on page 298
- "PtpeFirstHost" on page 237
- "PtpeNextHost" on page 273
- "PtpeFindHost" on page 233
- "PtpeAddHostToList" on page 114
- "PtpeDelHostFromList" on page 225

PtpeChangeHostRates

Purpose

Sets the current time intervals used by the monitoring hierarchy for sampling and recording of performance information.

Library

libptpe.a

Syntax

```
#include <spdm.h>
int PtpeChangeHostRates(sblock, srate, arate osrate, oarate)
session_ptr_t   sblock;
int             *srate;
int             *arate;
int             *osrate;
int             *oarate;
```

Parameters

sblock
The address of a session control block. This information block is created and initialized by the **PtpeOpenSession** subroutine.

srate
Contains the new time interval between samplings of performance data for the monitoring hierarchy. This value is expected to be in seconds. A value of 0 (zero) in this parameter instructs the subroutine to retain the current sampling rate. *arate* and *srate* cannot both be 0.

arate
Contains the new time interval between recordings of performance data to the archive. This value is expected to be in seconds, and should be an even multiple of the *srate* parameter, if one is provided. A value of 0 (zero) in this parameter instructs the subroutine to retain the current archiving rate. *arate* and *srate* cannot both be 0.

osrate
A pointer to a memory location where the subroutine stores the previous performance data sampling rate.

oarate
A pointer to a memory location where the subroutine stores the previous performance data archiving rate.

Description

PtpeChangeHostRates sets the current time intervals between performance information samples and recordings in the monitoring hierarchy, after collection and archiving have been started. Both intervals are expressed in units of seconds.

Use this routine to set either one or both of these intervals. When specifying both intervals, the recording interval should always be an even multiple of the sampling rate. If it is not, the subroutine will truncate the recording rate to the last whole multiple of the sampling interval. When setting only one of the intervals, the other should be specified as a value of 0 (zero).

The performance data sampling and recording intervals previously used by the reporting hierarchy are returned at the locations indicated for the *psrate* and *oarate* parameters.

All systems in the performance monitoring hierarchy use the same sampling and recording rates.

This subroutine cannot be used in a PTPE read-only session.

Return Codes

Upon successful completion, this subroutine returns the value of PTPE_SUCCESS. If an error has occurred, the return code is set to one of the following values to indicate the cause of the error:

PTPE_TIME_APPROX	An exact match for the data and time could not be found in the archive, so the closest entry was substituted in its place.
PTPE_INV_HIERARCHY	The monitoring hierarchy contains a node that is not a member of the currently active system partition. The hierarchy must be repaired by removing any nodes that are not members of the active system partition before this subroutine can be used.
PTPE_INV_PTR	*sblock* has a NULL value, or *reply* is not NULL.
PTPE_NO_SESSION	A PTPE API session was not previously established by this application (see “PtpeOpenSession” on page 277)
PTPE_NO_HIERARCHY	A monitoring hierarchy does not exist. This subroutine cannot function without a monitoring hierarchy (see “ptpehier” on page 99)
PTPE_NO_MEMORY	Could not allocate enough memory to initiate the query. *reply* is not modified.
PTPE_MEMORY	An internal error occurred.
PTPE_RONLY_SESS	The application attempted to issue this subroutine from a PTPE read-only session. A regular session must be established to issue this instruction. Ensure that the application is executing in the proper SP system partition.
PTPE_TIMEOUT	A reply to the command was not received from the central coordinator node in the time allowed. reply is not modified.
PTPE_NO_LOCK_OBJ	The PTPE data classes do not exist in the System Data Repository (see “ptpeconf” on page 86).
PTPE_SDR	An unexpected error occurred while obtaining information from the PTPE data classes in the System Data Repository. These data classes might not exist,

	or the System Data Repository might be experiencing difficulties.
PTPE_STATE	Performance information collection and summarization is not currently active in the monitoring hierarchy.
PTPE_NO_CONNECT	Could not establish a network connection to the central coordinator. *reply* list not modified.
PTPE_BAD_RECEIVE	Error in receiving the reply to the query from the central coordinator node. It is impossible to determine if the query was successful. *reply* is not modified.
PTPE_LIMITED	Some of the systems in the *targets* list were unable to reply to the query. The list *reply* points to contains all systems involved in the query; the application can determine which systems failed by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_LIMITED_BADR	Some systems in the *targets* list were unable to respond to the query, and an error occurred in reading the *reply* list response from the central coordinator node. The application should treat this as an error, since the contents of the *reply* list cannot be guaranteed to be accurate.
PTPE_API_FAILED	The central coordinator node has indicated that all systems in the *targets* list failed to respond to the query. *reply* contains all systems involved in the query, along with the reasons for the failure on these systems. The application can determine the cause of the error by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_API_FAILED_BADR	The central coordinator node has indicated that all systems failed to respond to the query, and an error occurred in reading the *reply* list from the central coordinator node. the application should treat this as an error, since the contents of the *reply* list cannot be guaranteed.

Examples

```
#include <spdm.h>

session_ptr_t   sblock;
int             srate, osrate;
int             arate, oarate;
int             rc;

/* set up PTPE session and start collection earlier in the code */

   /* set new data sampling and archiving rates */
   srate=30;         /*seconds*/
   arate=srate*4;
   rc PtpeChangeHostRates(sblock,srate,arate,&osrate,&oarate);

   switch (rc) {
     case PTPE_SUCCESS:
        printf("Sampling rate changed to %d seconds\n",srate);
        printf("Archiving rate changed to %d seconds\n",arate);
        printf("Could not obtain sampling and archiving rates\n");
        break;
     case PTPE_TIME_APPROX:
        printf("Time values set, but archiving rate adjusted\n");
        printf("Use PtpeQueryHostRates to get new rates\n");
        break;
     default:
        printf("Could not set sampling and archiving rates\");
}
```

Related Information

- “PtpeArchStartHosts” on page 156
- “PtpeArchStartAllHosts” on page 152
- “PtpeQueryHostRates” on page 291
- “PtpeColStart” on page 215

PtpeCloseSession

Purpose

Relinquishes a session with the Performance Toolbox Parallel Extensions.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeCloseSession(sblock)
session_ptr_t   *sblock;
```

Parameters

sblock
: A pointer to a session_ptr_t data type, which points to a session control information block previously established by a **PtpeOpenSession** subroutine. Upon successful completion of the subroutine, the session control block is freed.

Description

PtpeCloseSession relinquishes a session between the application and the Performance Toolbox Parallel Extensions. By closing the session, the application releases control over performance information collection, summarization, and archiving in the PTPE monitoring hierarchy. The application also releases the ability to obtain current or archived performance information from the systems in the monitoring hierarchy.

When a session is established, PTPE grants sole control of the monitoring hierarchy to the application for the duration of the session. This means that other commands, such as **ptpehier** or **ptpectrl**, cannot modify the hierarchy structure, or change the current status of performance information collection and archiving. Other PTPE API applications are also restricted from controlling performance information collection and archiving while the session is held. If multiple PTPE API applications must execute in parallel, the applications should coordinate their use of sessions, ensuring that one application releases the session for the other application(s) to acquire. Applications must recognize that no other application will be able to acquire a session while a session is held by another application, and should be coded to handle this eventuality.

Releasing a session will not return the Performance Toolbox Parallel Extensions monitoring hierarchy to the status it had prior to the **PtpeOpenSession** call. Any actions taken by the application while the session was active remain in effect, until reset or modified by another application. For example, if an application marked specific statistics for archiving, these statistics remain marked after the session is release with **PtpeCloseSession.**

A session cannot be established if another application holds a session, or if the application attempts to acquire a session while either the **ptpectrl** or **ptpehier** commands are executing. A session also cannot be established if the **ptpeconf** command was never executed to create the necessary PTPE data classes in the System Data Repository.

Return Codes

Upon successful completion, a return code of PTPE_SUCCESS is returned to the caller. If an error occurred, one of the following return codes is provided:

PTPE_NO_SESSION	The application did not previously hold an active session.
PTPE_INV_PTR	*sblock* has a NULL value.
PTPE_MEMORY	An internal error occurred.
PTPE_SDR	An unexpected error occurred while obtaining information from the PTPE data classes in the System Data Repository. These data classes might not exist, or the System Data Repository might be experiencing difficulties.

Examples

```
#include <spdm.h>

session_ptr_t   sblock;
int             rc;

rc = PtpeOpenSession(&sblock);
if (rc != PTPE_SUCCESS) {
/* handle error condition */
}
```

Related Information

- “PtpeOpenSession” on page 277
- “ptpeconf” on page 86
- “ptpegroup” on page 97
- The AIX **newgrp** command

PtpeColDisableAllStats

Purpose

Makes all statistics unavailable for collection and summarization on one or more systems in the Performance Toolbox Parallel Extensions monitoring hierarchy.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeColEnableAllStats(sblock, targets, reply)
session_ptr_t   sblock;
host_list_t     targets;
host_list_t     *reply;
```

Parameters

sblock
: The address of a session control block. This information block is created and initialized by the **PtpeOpenSession** subroutine.

targets
: The anchor of a host list that contains at least one entry. All systems in this list should not have statistics lists assigned to them.

reply
: A pointer to a host_list_t data type, which should be set to a NULL value before invoking this subroutine. Upon successful completion of the subroutine, an anchor point for a host list will be placed at this location. This list will contain entries for each system in the *targets* list. Each entry will contain a result code to indicate whether or not the system was successful in carrying out the request.

Description

PtpeColEnableAllStats instructs one or more systems in the Performance Toolbox Parallel Extensions to make all the statistics that the system can possibly supply available for performance information collection and summarization. The subroutine is similar to the **PtpeColEnableStats** subroutine, except the subroutine activates all statistics instead of a set of statistics that the application selects. The application must first establish a PTPE API session by using the **PtpeOpenSession** subroutine. The application must also construct a host list for the command, containing an entry for each system that the application needs to query. This list is then provided in the *targets* parameter.

When **PtpeColEnableAllStats** is executed, the subroutine relays the command, along with the host list of systems involved, to the monitoring hierarchy's central coordinator node. The central coordinator node determines which nodes are data managers for the systems in the *targets* list, and forwards the command to only those data manager nodes that manage systems in the host list. The data manager

nodes then relay the command to those systems in their reporting groups that have been listed in the *targets* list.

Each system in the *targets* list makes all statistics known to that system available for performance information collection and summarization. The system then indicates its success or failure in the effort to its data manager node.

Data manager nodes construct partial host lists, with an entry for each system in its reporting group that was targeted for the command. The system's success or failure *reply* is placed in the partial host list. This list is then relayed to the central coordinator node, who constructs a complete host list. The complete host list is transmitted back to the **PtpeColDisableAllStats** subroutine, which provides it to the application in the *reply* parameter.

PtpeColEnableAllStats will provide an indication of the overall success or failure of the query in the return code:

PTPE_SUCCESS	All systems in the *targets* list successfully responded to the command.
PTPE_LIMITED	Some systems in the *targets* list were unable to respond successfully to the command.
PTPE_API_FAILED	All systems in the *targets* list were unable to respond successfully to the command.

To determine which systems were not successful, the reply list should be scanned, and the results for each system in the list checked using the **PtpeGetHostResult** subroutine. The result code will provide an indication of the error for the system.

Making statistics available or unavailable for collection and summarization has no impact upon which statistics are recorded to the performance information archive. This subroutine cannot be issued from a PTPE read-only session.

Return Codes

Upon successful completion, a return code of PTPE_SUCCESS is returned to the caller. If an error occurred, one of the following return codes is provided:

PTPE_SUCCESS_BADR	The central coordinator node has indicated that all systems in the *targets* list responded successfully to the command, but an error occurred in reading the *reply* list response from the central coordinator Node.
PTPE_LIMITED	Some of the systems in the *targets* list were unable to reply to the command. *reply* contains the list of all systems involved in the command; the application can determine which systems failed by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_LIMITED_BADR	Some systems in the *targets* list were unable to respond to the command, and an error occurred in reading the *reply* list response from the central coordinator node.

PTPE_API_FAILED	The central coordinator node has indicated that all systems in the *targets* list failed to respond to the command. *reply* contains all systems involved in the command, along with the reasons for the failure on these systems. The application can determine the cause of the error by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_API_FAILED_BADR	The central coordinator node has indicated that all systems failed to respond to the command, and an error occurred in reading the *reply* list from the central coordinator node.
PTPE_STATE	Performance information collection and summarization is not currently active in the monitoring hierarchy.
PTPE_NO_CONNECT	Could not establish a network connection to the central coordinator node. *reply* list not modified.
PTPE_BAD_SEND	Could not successfully transmit the command to the central coordinator node. *reply* is not modified.
PTPE_BAD_RECEIVE	Error in receiving the reply to the command from the central coordinator node. It is impossible to determine if the query was successful. *reply* is not modified.
PTPE_RONLY_SESS	The application attempted to issue this subroutine from a PTPE read-only session. A regular session must be established to issue this instruction. Ensure that the application is executing in the proper SP system partition.
PTPE_TIMEOUT	A reply to the command was not received from the central coordinator node in the time allowed. reply is not modified.
PTPE_INV_HIERARCHY	The monitoring hierarchy contains a node that is not a member of the currently active system partition. The hierarchy must be repaired by removing any nodes that are not members of the active system partition before this subroutine can be used.
PTPE_INV_HOSTLIST	*targets* either contains a NULL value, or does not anchor a valid host list.
PTPE_INV_PTR	*sblock* has a NULL value, or *reply* is not NULL.
PTPE_EMPTY	*targets* is an empty host list.
PTPE_HOST_NOT_FOUND	One or more systems listed in *targets* could not be found in the monitoring hierarchy. *reply* contains the list of systems that could not be found in the hierarchy.
PTPE_NO_SESSION	A PTPE API session was not previously established by this application (see “PtpeOpenSession” on page 277).
PTPE_NO_HIERARCHY	A monitoring hierarchy does not exist. This subroutine cannot function without a monitoring hierarchy (see “ptpehier” on page 99).

PTPE_NO_MEMORY	Could not allocate enough memory to initiate the query. *reply* is not modified.
PTPE_MEMORY	An internal error occurred.
PTPE_BAD_LOC_PTR	The internal pointers in the *targets* anchor have been corrupted.
PTPE_SDR	An unexpected error occurred while obtaining information from the PTPE data classes in the System Data Repository. These data classes might not exist, or the System Data Repository might be experiencing difficulties.

Examples

```
#include <spdm.h>

session_ptr_t   sblock;
host_list_t     targets, reply;
int             rc;

/* set up session and "targets" earlier */
reply = (host_list_t) NULL;
rc = PtpeColDisableAllStats(sblock, targets,
                            &reply);
if (rc != PTPE_SUCCESS) {
      printf("Not all systems succeeded.\n");
}
/* loop through "reply" and check results */
/* on each system with PtpeGetHostResult  */
```

Related Information

- “PtpeOpenSession” on page 277
- “PtpeColEnableStats” on page 193
- “PtpeColEnableAllStats” on page 188
- “PtpeColDisableStats” on page 183
- “PtpeColDisableAllStats” on page 178
- “PtpeQueryAvailStats” on page 286
- “PtpeColQueryStats” on page 206
- “PtpeInitHostList” on page 265
- “PtpeAddHostToList” on page 114
- “PtpeFirstHost” on page 237
- “PtpeNextHost” on page 273
- “PtpeFindHost” on page 233
- “PtpeFreeHostList” on page 241
- “PtpeGetHost” on page 245
- “PtpeGetHostResult” on page 247
- “PtpeInitStatList” on page 267

- “PtpeAddStatToList” on page 116
- “PtpeFirstStat” on page 239
- “PtpeNextStat” on page 275
- “PtpeFindStat” on page 235
- “PtpeFreeStatList” on page 243
- “PtpeAssignStatsToHost” on page 170
- “PtpeGetHostStatList” on page 250
- “PtpeGetStatName” on page 252
- “PtpeGetStatResult” on page 254

PtpeColDisableStats

Purpose

Makes one or more statistics unavailable for collection and summarization on one or more systems in the Performance Toolbox Parallel Extensions monitoring hierarchy.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeColDisableStats(sblock, targets, reply)
session_ptr_t   sblock;
host_list_t     targets;
host_list_t     *reply;
```

Parameters

sblock
The address of a session control block. This information block is created and initialized by the **PtpeOpenSession** subroutine.

targets
The anchor of a host list that contains at least one entry. All systems in this list should have statistics lists assigned to them (see “PtpeAssignStatsToHost” on page 170).

reply
A pointer to a host_list_t data type, which should be set to a NULL value before invoking this subroutine. Upon successful completion of the subroutine, an anchor point for a host list will be placed at this location. This list will contain entries for each system in the *targets* list. Each entry will contain a result code to indicate whether or not the system was successful in carrying out the request. Each entry will also have a statistics list assigned to it, containing an entry for each statistic passed to the system, with a result code to indicate if the statistic was successfully deactivated.

Description

PtpeColEnableStats instructs one or more systems in the Performance Toolbox Parallel Extensions to make a set of statistics unavailable for performance information collection and summarization. The application must first establish a PTPE API session by using the **PtpeOpenSession** subroutine. The application must also construct a host list for the command, containing an entry for each system that the application needs to control. This list is then provided in the *targets* parameter. Each entry in the *targets* list should also have a statistics list assigned to it (see “PtpeAssignStatsToHost” on page 170), containing the list of statistics to be made unavailable.

When **PtpeColDisableStats** is executed, the subroutine relays the command, along with the host list of systems involved, to the monitoring hierarchy's central

coordinator node. The central coordinator node determines which nodes are data managers for the systems in the *targets* list, and forwards the command to only those data manager nodes that manage systems in the host list. The data manager nodes then relay the command to those systems in their reporting groups that have been listed in the *targets* list.

Each system in the *targets* list extracts the list of statistics for their respective entry from the command message, and searches for these statistics. Any of the statistics in this list that also exist on the system are made unavailable for collection. The PTPE daemons on the system create a *reply* statistics list, containing all statistics passed to this system in the command. The daemons will set the result codes in each statistics list entry to one of the following values:

PTPE_SUCCESS	The statistic was made unavailable for collection.
PTPE_STAT_NOT_FOUND	The statistic could not be found on the system.

The list of statistics is then relayed to the data manager node.

Data manager nodes construct partial host lists, with an entry for each system in its reporting group that was targeted for the command. The statistics lists received from each node are assigned to their respective host list entries in the data manager node's partial host list. This list is then relayed to the central coordinator node, who constructs a complete host list. The complete host list is transmitted back to the **PtpeColDisableStats** subroutine, which provides it to the application in the *reply* parameter.

PtpeColDisableStats will provide an indication of the overall success or failure of the query in the return code:

PTPE_SUCCESS	All systems in the *targets* list successfully responded to the command.
PTPE_LIMITED	Some systems in the *targets* list were unable to respond successfully to the command.
PTPE_API_FAILED	All systems in the *targets* list were unable to respond successfully to the command.

To determine which systems were not successful, the reply list should be scanned, and the results for each system in the list checked using the **PtpeGetHostResult** subroutine. The result code will provide an indication of the error for the system. To check for failures caused by missing statistics, extract the statistics list for each entry by using the **PtpeGetHostStatList** subroutine, and scan the list using the statistics list scanning subroutines.

Making statistics available or unavailable for collection and summarization has no impact upon which statistics are recorded to the performance information archive. This subroutine cannot be issued from a PTPE read-only session.

Return Codes

Upon successful completion, a return code of PTPE_SUCCESS is returned to the caller. If an error occurred, one of the following return codes is provided:

PTPE_SUCCESS_BADR	The central coordinator node has indicated that all systems in the *targets* list responded successfully to the command, but an error occurred in reading the *reply* list response from the central coordinator Node.
PTPE_LIMITED	Some of the systems in the *targets* list were unable to reply to the command. *reply* contains the list of all systems involved in the command; the application can determine which systems failed by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_LIMITED_BADR	Some systems in the *targets* list were unable to respond to the command, and an error occurred in reading the *reply* list response from the central coordinator node.
PTPE_API_FAILED	The central coordinator node has indicated that all systems in the *targets* list failed to respond to the command. *reply* contains all systems involved in the command, along with the reasons for the failure on these systems. The application can determine the cause of the error by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_API_FAILED_BADR	The central coordinator node has indicated that all systems failed to respond to the command, and an error occurred in reading the *reply* list from the central coordinator node.
PTPE_STATE	Performance information collection and summarization is not currently active in the monitoring hierarchy.
PTPE_NO_CONNECT	Could not establish a network connection to the central coordinator node. *reply* list not modified.
PTPE_BAD_SEND	Could not successfully transmit the command to the central coordinator node. *reply* is not modified.
PTPE_BAD_RECEIVE	Error in receiving the reply to the command from the central coordinator node. It is impossible to determine if the query was successful. *reply* is not modified.
PTPE_RONLY_SESS	The application attempted to issue this subroutine from a PTPE read-only session. A regular session must be established to issue this instruction. Ensure that the application is executing in the proper SP system partition.
PTPE_TIMEOUT	A reply to the command was not received from the central coordinator node in the time allowed. reply is not modified.
PTPE_INV_HIERARCHY	The monitoring hierarchy contains a node that is not a member of the currently active system partition. The hierarchy must be repaired by removing any nodes that are not members of the active system partition before this subroutine can be used.

PTPE_INV_HOSTLIST	*targets* either contains a NULL value, or does not anchor a valid host list.
PTPE_INV_PTR	*sblock* has a NULL value, or *reply* is not NULL.
PTPE_EMPTY	*targets* is an empty host list.
PTPE_HOST_NOT_FOUND	One or more systems listed in *targets* could not be found in the monitoring hierarchy. *reply* contains the list of systems that could not be found in the hierarchy.
PTPE_INV_STATLIST	One or more systems in the *targets* list does not have a valid statistics list assigned to it.
PTPE_NO_SESSION	A PTPE API session was not previously established by this application (see “PtpeOpenSession” on page 277).
PTPE_NO_HIERARCHY	A monitoring hierarchy does not exist. This subroutine cannot function without a monitoring hierarchy (see “ptpehier” on page 99)
PTPE_NO_MEMORY	Could not allocate enough memory to initiate the query. *reply* is not modified.
PTPE_MEMORY	An internal error occurred.
PTPE_BAD_LOC_PTR	The internal pointers in the *targets* anchor have been corrupted.
PTPE_SDR	An unexpected error occurred while obtaining information from the PTPE data classes in the System Data Repository. These data classes might not exist, or the System Data Repository might be experiencing difficulties.

Examples

```
#include <spdm.h>

session_ptr_t   sblock;
host_list_t     targets, reply;
int             rc;

/* set up session and "targets" earlier */
reply = (host_list_t) NULL;
rc = PtpeColDisableStats(sblock, targets,
                         &reply);
if (rc != PTPE_SUCCESS) {
     printf("Not all systems succeeded.\n");
}
/* loop through "reply" and check results */
/* on each system with PtpeGetHostResult  */
```

Related Information

- “PtpeOpenSession” on page 277
- “PtpeColDisableStats” on page 183
- “PtpeColEnableStats” on page 193
- “PtpeColEnableAllStats” on page 188

- “PtpeColDisableAllStats” on page 178
- “PtpeQueryAvailStats” on page 286
- “PtpeColQueryStats” on page 206
- “PtpeInitHostList” on page 265
- “PtpeAddHostToList” on page 114
- “PtpeFirstHost” on page 237
- “PtpeNextHost” on page 273
- “PtpeFindHost” on page 233
- “PtpeFreeHostList” on page 241
- “PtpeGetHost” on page 245
- “PtpeGetHostResult” on page 247
- “PtpeInitStatList” on page 267
- “PtpeAddStatToList” on page 116
- “PtpeFirstStat” on page 239
- “PtpeNextStat” on page 275
- “PtpeFindStat” on page 235
- “PtpeFreeStatList” on page 243
- “PtpeAssignStatsToHost” on page 170
- “PtpeGetHostStatList” on page 250
- “PtpeGetStatName” on page 252
- “PtpeGetStatResult” on page 254

PtpeColEnableAllStats

Purpose

Makes all statistics available for collection and summarization on one or more systems in the Performance Toolbox Parallel Extensions monitoring hierarchy.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeColEnableAllStats(sblock, targets, reply)
session_ptr_t   sblock;
host_list_t     targets;
host_list_t     *reply;
```

Parameters

sblock
The address of a session control block. This information block is created and initialized by the **PtpeOpenSession** subroutine.

targets
The anchor of a host list that contains at least one entry. All systems in this list should not have statistics lists assigned to them.

reply
A pointer to a host_list_t data type, which should be set to a NULL value before invoking this subroutine. Upon successful completion of the subroutine, an anchor point for a host list will be placed at this location. This list will contain entries for each system in the *targets* list. Each entry will contain a result code to indicate whether or not the system was successful in carrying out the request.

Description

PtpeColEnableAllStats instructs one or more systems in the Performance Toolbox Parallel Extensions to make all the statistics that the system can possibly supply available for performance information collection and summarization. The subroutine is similar to the **PtpeColEnableStats** subroutine, except the subroutine activates all statistics instead of a set of statistics that the application selects.

The application must first establish a PTPE API session by using the **PtpeOpenSession** subroutine. The application must also construct a host list for the command, containing an entry for each system that the application needs to query. This list is then provided in the *targets* parameter.

When **PtpeColEnableAllStats** is executed, the subroutine relays the command, along with the host list of systems involved, to the monitoring hierarchy's central coordinator node. The central coordinator node determines which nodes are data managers for the systems in the *targets* list, and forwards the command to only those data manager nodes that manage systems in the host list. The data manager

nodes then relay the command to those systems in their reporting groups that have been listed in the *targets* list.

Each system in the *targets* list makes all statistics known to that system available for performance information collection and summarization. The system then indicates its success or failure in the effort to its data manager node.

Data manager nodes construct partial host lists, with an entry for each system in its reporting group that was targeted for the command. The system's success or failure *reply* is placed in the partial host list. This list is then relayed to the central coordinator node, who constructs a complete host list. The complete host list is transmitted back to the **PtpeColEnableAllStats** subroutine, which provides it to the application in the *reply* parameter.

PtpeColEnableAllStats will provide an indication of the overall success or failure of the query in the return code:

PTPE_SUCCESS	All systems in the *targets* list successfully responded to the command.
PTPE_LIMITED	Some systems in the *targets* list were unable to respond successfully to the command.
PTPE_API_FAILED	All systems in the *targets* list were unable to respond successfully to the command.

To determine which systems were not successful, the reply list should be scanned, and the results for each system in the list checked using the **PtpeGetHostResult** subroutine. The result code will provide an indication of the error for the system.

Making statistics available or unavailable for collection and summarization has no impact upon which statistics are recorded to the performance information archive. This subroutine cannot be issued from a PTPE read-only session.

Return Codes

Upon successful completion, a return code of PTPE_SUCCESS is returned to the caller. If an error occurred, one of the following return codes is provided:

PTPE_SUCCESS_BADR	The central coordinator node has indicated that all systems in the *targets* list responded successfully to the command, but an error occurred in reading the *reply* list response from the central coordinator Node.
PTPE_LIMITED	Some of the systems in the *targets* list were unable to reply to the command. *reply* contains the list of all systems involved in the command; the application can determine which systems failed by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_LIMITED_BADR	Some systems in the *targets* list were unable to respond to the command, and an error occurred in reading the *reply* list response from the central coordinator node.

PTPE_API_FAILED	The central coordinator node has indicated that all systems in the *targets* list failed to respond to the command. *reply* contains all systems involved in the command, along with the reasons for the failure on these systems. The application can determine the cause of the error by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_API_FAILED_BADR	The central coordinator node has indicated that all systems failed to respond to the command, and an error occurred in reading the *reply* list from the central coordinator node.
PTPE_STATE	Performance information collection and summarization is not currently active in the monitoring hierarchy.
PTPE_NO_CONNECT	Could not establish a network connection to the central coordinator node. *reply* list not modified.
PTPE_BAD_SEND	Could not successfully transmit the command to the central coordinator node. *reply* is not modified.
PTPE_BAD_RECEIVE	Error in receiving the reply to the command from the central coordinator node. It is impossible to determine if the query was successful. *reply* is not modified.
PTPE_RONLY_SESS	The application attempted to issue this subroutine from a PTPE read-only session. A regular session must be established to issue this instruction. Ensure that the application is executing in the proper SP system partition.
PTPE_TIMEOUT	A reply to the command was not received from the central coordinator node in the time allowed. reply is not modified.
PTPE_INV_HIERARCHY	The monitoring hierarchy contains a node that is not a member of the currently active system partition. The hierarchy must be repaired by removing any nodes that are not members of the active system partition before this subroutine can be used.
PTPE_INV_HOSTLIST	*targets* either contains a NULL value, or does not anchor a valid host list.
PTPE_INV_PTR	*sblock* has a NULL value, or *reply* is not NULL.
PTPE_EMPTY	*targets* is an empty host list.
PTPE_HOST_NOT_FOUND	One or more systems listed in *targets* could not be found in the monitoring hierarchy. *reply* contains the list of systems that could not be found in the hierarchy.
PTPE_NO_SESSION	A PTPE API session was not previously established by this application (see “PtpeOpenSession” on page 277).
PTPE_NO_HIERARCHY	A monitoring hierarchy does not exist. This subroutine cannot function without a monitoring hierarchy (see “ptpehier” on page 99).

PTPE_NO_MEMORY	Could not allocate enough memory to initiate the query. *reply* is not modified.
PTPE_MEMORY	An internal error occurred.
PTPE_BAD_LOC_PTR	The internal pointers in the *targets* anchor have been corrupted.
PTPE_SDR	An unexpected error occurred while obtaining information from the PTPE data classes in the System Data Repository. These data classes might not exist, or the System Data Repository might be experiencing difficulties.

Examples

```
#include <spdm.h>

session_ptr_t   sblock;
host_list_t     targets, reply;
int             rc;

/* set up session and "targets" earlier */
reply = (host_list_t) NULL;
rc = PtpeColEnableAllStats(sblock, targets,
                           &reply);
if (rc != PTPE_SUCCESS) {
    printf("Not all systems succeded.\n");
}
/* loop through "reply" and check results */
/* on each system with PtpeGetHostResult  */
```

Related Information

- “PtpeOpenSession” on page 277
- “PtpeColEnableStats” on page 193
- “PtpeColDisableStats” on page 183
- “PtpeColDisableAllStats” on page 178
- “PtpeQueryAvailStats” on page 286
- “PtpeColQueryStats” on page 206
- “PtpeInitHostList” on page 265
- “PtpeAddHostToList” on page 114
- “PtpeFirstHost” on page 237
- “PtpeNextHost” on page 273
- “PtpeFindHost” on page 233
- “PtpeFreeHostList” on page 241
- “PtpeGetHost” on page 245
- “PtpeGetHostResult” on page 247
- “PtpeInitStatList” on page 267
- “PtpeAddStatToList” on page 116

- “PtpeFirstStat” on page 239
- “PtpeNextStat” on page 275
- “PtpeFindStat” on page 235
- “PtpeFreeStatList” on page 243
- “PtpeAssignStatsToHost” on page 170
- “PtpeGetHostStatList” on page 250
- “PtpeGetStatName” on page 252
- “PtpeGetStatResult” on page 254

PtpeColEnableStats

Purpose

Makes one or more statistics available for collection and summarization on one or more systems in the Performance Toolbox Parallel Extensions monitoring hierarchy.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeColEnableStats(sblock, targets, reply)
session_ptr_t   sblock;
host_list_t     targets;
host_list_t     *reply;
```

Parameters

sblock
: The address of a session control block. This information block is created and initialized by the **PtpeOpenSession** subroutine.

targets
: The anchor of a host list that contains at least one entry. All systems in this list should have statistics lists assigned to them (see "PtpeAssignStatsToHost" on page 170).

reply
: A pointer to a host_list_t data type, which should be set to a NULL value before invoking this subroutine. Upon successful completion of the subroutine, an anchor point for a host list will be placed at this location. This list will contain entries for each system in the *targets* list. Each entry will contain a result code to indicate whether or not the system was successful in carrying out the request. Each entry will also have a statistics list assigned to it, containing an entry for each statistic passed to the system, with a result code to indicate if the statistic was successfully activated.

Description

PtpeColEnableStats instructs one or more systems in the Performance Toolbox Parallel Extensions to make a set of statistics available for performance information collection and summarization.

The application must first establish a PTPE API session by using the **PtpeOpenSession** subroutine. The application must also construct a host list for the command, containing an entry for each system that the application needs to query. This list is then provided in the *targets* parameter. Each entry in the *targets* list should also have a statistics list assigned to it (see "PtpeAssignStatsToHost" on page 170) containing the list of statistics to be made available.

When **PtpeColEnableStats** is executed, the subroutine relays the command, along with the host list of systems involved, to the monitoring hierarchy's central

coordinator node. The central coordinator node determines which nodes are data managers for the systems in the *targets* list, and forwards the command to only those data manager nodes that manage systems in the host list. The data manager nodes then relay the command to those systems in their reporting groups that have been listed in the *targets* list.

Each system in the *targets* list extracts the list of statistics for their respective entry from the command message, and searches for these statistics. Any of the statistics in this list that also exist on the system are made available for collection. The PTPE daemons on the system create a *reply* statistics list, containing all statistics passed to this system in the command. The daemons will set the result codes in each statistics list entry to one of the following values:

PTPE_SUCCESS	The statistic was made available for collection.
PTPE_STAT_NOT_FOUND	The statistic could not be found on the system.

The list of statistics is then relayed to the data manager node.

Data manager nodes construct partial host lists, with an entry for each system in its reporting group that was targeted for the command. The statistics lists received from each node are assigned to their respective host list entries in the data manager node's partial host list. This list is then relayed to the central coordinator node, who constructs a complete host list. The complete host list is transmitted back to the **PtpeColEnableStats** subroutine, which provides it to the application in the *reply* parameter.

PtpeColEnableStats will provide an indication of the overall success or failure of the query in the return code:

PTPE_SUCCESS	All systems in the *targets* list successfully responded to the command.
PTPE_LIMITED	Some systems in the *targets* list were unable to respond successfully to the command.
PTPE_API_FAILED	All systems in the *targets* list were unable to respond successfully to the command.

To determine which systems were not successful, the reply list should be scanned, and the results for each system in the list checked using the **PtpeGetHostResult** subroutine. The result code will provide an indication of the error for the system. To check for failures caused by missing statistics, extract the statistics list for each entry by using the **PtpeGetHostStatList** subroutine, and scan the list using the statistics list scanning subroutines.

Making statistics available or unavailable for collection and summarization has no impact upon which statistics are recorded to the performance information archive. This subroutine cannot be issued from a PTPE read-only session.

Return Codes

Upon successful completion, a return code of PTPE_SUCCESS is returned to the caller. If an error occurred, one of the following return codes is provided:

PTPE_SUCCESS_BADR	The central coordinator node has indicated that all systems in the *targets* list responded successfully to the command, but an error occurred in reading the *reply* list response from the central coordinator Node.
PTPE_LIMITED	Some of the systems in the *targets* list were unable to reply to the command. *reply* contains the list of all systems involved in the command; the application can determine which systems failed by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_LIMITED_BADR	Some systems in the *targets* list were unable to respond to the command, and an error occurred in reading the *reply* list response from the central coordinator node.
PTPE_API_FAILED	The central coordinator node has indicated that all systems in the *targets* list failed to respond to the command. The *reply* contains all systems involved in the command, along with the reasons for the failure on these systems. The application can determine the cause of the error by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_API_FAILED_BADR	The central coordinator node has indicated that all systems failed to respond to the command, and an error occurred in reading the *reply* list from the central coordinator node.
PTPE_STATE	Performance information collection and summarization is not currently active in the monitoring hierarchy.
PTPE_NO_CONNECT	Could not establish a network connection to the central coordinator node. *reply* list not modified.
PTPE_BAD_SEND	Could not successfully transmit the command to the central coordinator node. *reply* is not modified.
PTPE_BAD_RECEIVE	Error in receiving the reply to the command from the central coordinator node. It is impossible to determine if the query was successful. *reply* is not modified.
PTPE_RONLY_SESS	The application attempted to issue this subroutine from a PTPE read-only session. A regular session must be established to issue this instruction. Ensure that the application is executing in the proper SP system partition.
PTPE_TIMEOUT	A reply to the command was not received from the central coordinator node in the time allowed. reply is not modified.
PTPE_INV_HIERARCHY	The monitoring hierarchy contains a node that is not a member of the currently active system partition. The hierarchy must be repaired by removing any nodes that are not members of the active system partition before this subroutine can be used.

PTPE_INV_HOSTLIST	*targets* either contains a NULL value, or does not anchor a valid host list.
PTPE_INV_PTR	*sblock* has a NULL value, or *reply* is not NULL.
PTPE_EMPTY	*targets* is an empty host list.
PTPE_HOST_NOT_FOUND	One or more systems listed in *targets* could not be found in the monitoring hierarchy. *reply* contains the list of systems that could not be found in the hierarchy.
PTPE_INV_STATLIST	One or more systems in the *targets* list does not have a valid statistics list assigned to it.
PTPE_NO_SESSION	A PTPE API session was not previously established by this application (see "PtpeOpenSession" on page 277).
PTPE_NO_HIERARCHY	A monitoring hierarchy does not exist. This subroutine cannot function without a monitoring hierarchy (see "ptpehier" on page 99).
PTPE_NO_MEMORY	Could not allocate enough memory to initiate the query. *reply* is not modified.
PTPE_MEMORY	An internal error occurred.
PTPE_BAD_LOC_PTR	The internal pointers in the *targets* anchor have been corrupted.
PTPE_SDR	An unexpected error occurred while obtaining information from the PTPE data classes in the System Data Repository. These data classes might not exist, or the System Data Repository might be experiencing difficulties.

Examples

```
#include <spdm.h>

session_ptr_t   sblock;
host_list_t     targets, reply;
int             rc;

/* set up session and "targets" earlier */
reply = (host_list_t) NULL;
rc = PtpeColEnableStats(sblock, targets,
                        &reply);
if (rc != PTPE_SUCCESS) {
    printf("Not all systems succeeded.\n");
}
/* loop through "reply" and check results */
/* on each system with PtpeGetHostResult  */
```

Related Information

- "PtpeOpenSession" on page 277
- "PtpeColStart" on page 215
- "PtpeColEnableStats" on page 193
- "PtpeColDisableStats" on page 183

- “PtpeColDisableAllStats” on page 178
- “PtpeQueryAvailStats” on page 286
- “PtpeColQueryStats” on page 206
- “PtpeInitHostList” on page 265
- “PtpeAddHostToList” on page 114
- “PtpeFirstHost” on page 237
- “PtpeNextHost” on page 273
- “PtpeFindHost” on page 233
- “PtpeFreeHostList” on page 241
- “PtpeGetHost” on page 245
- “PtpeGetHostResult” on page 247
- “PtpeInitStatList” on page 267
- “PtpeAddStatToList” on page 116
- “PtpeFirstStat” on page 239
- “PtpeNextStat” on page 275
- “PtpeFindStat” on page 235
- “PtpeFreeStatList” on page 243
- “PtpeAssignStatsToHost” on page 170
- “PtpeGetHostStatList” on page 250
- “PtpeGetStatName” on page 252
- “PtpeGetStatResult” on page 254

PtpeColGetStats

Purpose

Retrieves the current (non-archived) value of a set of statistics on one or more systems in the Performance Toolbox Parallel Extensions monitoring hierarchy.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeColGetStats(sblock, targets, reply)
session_ptr_t   sblock;
host_list_t     targets;
host_list_t     *reply;
```

Parameters

sblock
The address of a session control block. This information block is created and initialized by the **PtpeOpenSession** subroutine.

targets
The anchor of a host list that contains at least one entry. All systems in this list should have statistics lists assigned to them (see “PtpeAssignStatsToHost” on page 170).

reply
A pointer to a host_list_t data type, which should be set to a NULL value before invoking this subroutine. Upon successful completion of the subroutine, an anchor point for a host list will be placed at this location. This list will contain entries for each system in the *targets* list. Each entry will contain a result code to indicate whether or not the system was successful in carrying out the request. Each entry will also have a statistics list assigned to it, containing an entry for each statistic passed to the system, either containing the statistics current value or a result code indicating why the statistic could not be retrieved.

Description

PtpeColGetStats retrieves a set of statistics from one or more systems in the Performance Toolbox Parallel Extension monitoring hierarchy. This function duplicates the function available in the Performance Toolbox for AIX **Spmi** and **Rsi** libraries.

The application must first establish a PTPE API session by using the **PtpeOpenSession** subroutine. The application must also construct a host list for the command, containing an entry for each system that the application needs to query. This list is then provided in the *targets* parameter. Each entry in the *targets* list should also have a statistics list assigned to it (see “PtpeAssignStatsToHost” on page 170), containing the list of statistics to be retrieved.

When **PtpeColGetStats** is executed, the subroutine relays the command, along with the host list of systems involved, to the monitoring hierarchy's central coordinator node. The central coordinator node determines which nodes are managers for the systems in the *targets* list, and forwards the command to only those data manager nodes that manage systems in the host list. The data manager nodes then relay the command to those systems in their reporting groups that have been listed in the *targets* list.

Each system in the *targets* list extracts the list of statistics for their respective entry from the command message, and searches for these statistics. The current values for any statistics in this list, that also exist n the system, are retrieved. The PTPE daemons on the system create a *reply* statistics list, containing all statistics passed to this system in the command. The daemons will set the value field for each statistic that was located, and will also set the result codes in each statistics list entry to one of the following values:

PTPE_SUCCESS	The statistic was located on the system.
PTPE_STAT_NOT_FOUND	The statistic could not be found on the system.

The list of statistics is then relayed to the data manager node.

Data manager nodes construct partial host lists, with an entry for each system in its reporting group that was targeted for the command. The statistics lists received from each node are assigned to their respective host list entries in the data manager node's partial host list. This list is then relayed to the central coordinator node, who constructs a complete host list. The complete host list is transmitted back to the **PtpeColGetStats** subroutine, which provides it to the application in the *reply* parameter.

PtpeColGetStats will provide an indication of the overall success or failure of the query in the return code:

PTPE_SUCCESS	All systems in the *targets* list successfully responded to the query.
PTPE_LIMITED	Some systems in the *targets* list were unable to respond successfully to the query.
PTPE_API_FAILED	All systems in the *targets* list were unable to respond successfully to the query. To determine which systems were not successful, the reply list should be scanned, and the results for each system in the list checked using the **PtpeGetHostResult** subroutine. The result code will provide an indication of the error for the system. To check for failures caused by missing statistics, extract the statistics list for each entry by using the **PtpeGetHostStatList** subroutine, and scan the list using the statistics list scanning subroutines.

Return Codes

Upon successful completion, a return code of PTPE_SUCCESS is returned to the caller. If an error occurred, one of the following return codes is provided:

PTPE_SUCCESS_BADR	The central coordinator node has indicated that all systems in the *targets* list responded successfully to the query, but an error occurred in reading the *reply* list response from the central coordinator Node.
PTPE_LIMITED	Some of the systems in the *targets* list were unable to reply to the query. The list *reply* points to contains all systems involved in the query; the application can determine which systems failed by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_RONLY_SESS	The application attempted to issue this subroutine from a PTPE read-only session. A regular session must be established to issue this instruction. Ensure that the application is executing in the proper SP system partition.
PTPE_LIMITED_BADR	Some systems in the *targets* list were unable to respond to the query, and an error occurred in reading the *reply* list response from the central coordinator node.
PTPE_API_FAILED	The central coordinator node has indicated that all systems in the *targets* list failed to respond to the query. *reply* contains all systems involved in the query, along with the reasons for the failure on these systems. The application can determine the cause of the error by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_API_FAILED_BADR	The central coordinator node has indicated that all systems failed to respond to the query, and an error occurred in reading the *reply* list from the central coordinator node.
PTPE_STATE	Performance information collection and summarization is not currently active in the monitoring hierarchy.
PTPE_NO_CONNECT	Could not establish a network connection to the central coordinator node. *reply* list not modified.
PTPE_BAD_SEND	Could not successfully transmit the command to the central coordinator node. *reply* is not modified.
PTPE_BAD_RECEIVE	Error in receiving the reply to the command from the central coordinator node. It is impossible to determine if the query was successful. *reply* is not modified.
PTPE_TIMEOUT	A reply to the command was not received from the central coordinator node in the time allowed. reply is not modified.
PTPE_INV_HIERARCHY	The monitoring hierarchy contains a node that is not a member of the currently active system partition. The hierarchy must be repaired by removing any nodes that are not members of the active system partition before this subroutine can be used.

PTPE_INV_HOSTLIST	*targets* either contains a NULL value, or does not anchor a valid host list.
PTPE_INV_PTR	*sblock* has a NULL value, or *reply* is not NULL.
PTPE_EMPTY	*targets* is an empty host list.
PTPE_HOST_NOT_FOUND	One or more systems listed in *targets* could not be found in the monitoring hierarchy. *reply* contains the list of systems that could not be found in the hierarchy.
PTPE_INV_STATLIST	One or more systems in the *targets* list does not have a valid statistics list assigned to it.
PTPE_NO_SESSION	A PTPE API session was not previously established by this application (see “PtpeOpenSession” on page 277)
PTPE_NO_HIERARCHY	A monitoring hierarchy does not exist. This subroutine cannot function without a monitoring hierarchy (see “ptpehier” on page 99)
PTPE_NO_MEMORY	Could not allocate enough memory to initiate the query. *reply* is not modified.
PTPE_MEMORY	An internal error occurred.
PTPE_BAD_LOC_PTR	The internal pointers in the *targets* anchor have been corrupted.
PTPE_SDR	An unexpected error occurred while obtaining information from the PTPE data classes in the System Data Repository. These data classes might not exist, or the System Data Repository might be experiencing difficulties.

Examples

```
#include <spdm.h>

session_ptr_t   sblock;
host_list_t     targets, reply;
int             rc;

/* set up session and "targets" earlier */
reply = (host_list_t) NULL;
rc = PtpeColGetStats(sblock, targets,
                     &reply);
if (rc != PTPE_SUCCESS) {
    printf("Not all systems succeeded.\n");
}
/* loop through "reply" and check results */
/* on each system with PtpeGetHostResult  */
```

Related Information

- “PtpeOpenSession” on page 277
- “PtpeColStart” on page 215
- “PtpeArchEnableStats” on page 133
- “PtpeArchEnableAllStats” on page 128

- “PtpeColDisableStats” on page 183
- “PtpeColDisableAllStats” on page 178
- “PtpeQueryAvailStats” on page 286
- “PtpeColQueryStats” on page 206
- “PtpeInitHostList” on page 265
- “PtpeAddHostToList” on page 114
- “PtpeFirstHost” on page 237
- “PtpeNextHost” on page 273
- “PtpeFindHost” on page 233
- “PtpeFreeHostList” on page 241
- “PtpeGetHost” on page 245
- “PtpeGetHostResult” on page 247
- “PtpeInitStatList” on page 267
- “PtpeAddStatToList” on page 116
- “PtpeFirstStat” on page 239
- “PtpeNextStat” on page 275
- “PtpeFindStat” on page 235
- “PtpeFreeStatList” on page 243
- “PtpeAssignStatsToHost” on page 170
- “PtpeGetHostStatList” on page 250
- “PtpeGetStatName” on page 252
- “PtpeGetStatResult” on page 254

PtpeColQueryState

Purpose

Determines the current status of performance information collection and summarization in the Performance Toolbox Parallel Extensions monitoring hierarchy.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeColQueryState(sblock)
session_ptr_t   sblock;
```

Parameters

sblock
: The address of a session control block. This information block is created and initialized by the **PtpeOpenSession** subroutine.

Description

PtpeColQueryStates reports to the calling application whether or not performance information collection and summarization is active in the monitoring hierarchy.

The application must first establish a PTPE API session by using the **PtpeOpenSession** subroutine. The inquiry does not use a host list.

The status of collection and summarization is reflected in the return code, which will be one of three status values:

PTPE_COL_ACTIVE	Performance information collection and summarization is currently active on all systems in the monitoring hierarchy.
PTPE_COL_OFF	Performance information collection and summarization is not active on any system in the monitoring hierarchy. On rare occasions, such as when the **ptpectrl -c** command or the **PtpeColStart** subroutine are interrupted, some systems may be attempting to collect and summarize performance information while most of the systems are not. This status is indicated by the following return code:
PTPE_COL_ERROR	Cannot determine status of performance information collection and summarization.

While this status persists, other PTPE commands and API subroutine may not function properly. When this status is returned by this subroutine, the application should execute the **PtpeColStop** subroutine in an attempt to clear this error state.

Return Codes

Upon successful completion, a return code of PTPE_COL_ACTIVE or PTPE_COL_OFF is returned to the caller. If an error occurred, one of the following return codes is provided:

PTPE_INV_PTR	*sblock* has a NULL value.
PTPE_NO_SESSION	A PTPE API session was not previously established by this application (see "PtpeOpenSession" on page 277).
PTPE_NO_MEMORY	Could not allocate enough memory to initiate the query.
PTPE_MEMORY	An internal error occurred.
PTPE_NO_LOCK_OBJ	The PTPE data classes do not exist in the System Data Repository (see "ptpeconf" on page 86).
PTPE_SDR	An unexpected error occurred while obtaining information from the PTPE data classes in the System Data Repository. These data classes might not exist, or the System Data Repository might be experiencing difficulties.

Examples

```
#include <spdm.h>

session_ptr_t   sblock;
host_list_t     mgrs, nodes;
int             rc;

/* set up session */
rc = PtpeColQueryState(sblock);
switch (rc) {
case PTPE_COL_ACTIVE:
     printf("Collection is active\n");
     break;
case PTPE_COL_OFF:
     printf("Collection is inactive\n");
     break;
case PTPE_COL_ERROR:
     printf("Cannot determine status -");
     printf("attempting to reset status.\n");
     mgrs = (host_list_t) NULL;
     nodes = (host_list_t) NULL;
     rc = PtpeColStop(sblock, &mgrs, &nodes);
     /* check results from PtpeColStop */
     break;
default:
     /*handle other error */
}
```

Related Information

- “PtpeOpenSession” on page 277
- “PtpeColStart” on page 215
- “PtpeColStop” on page 220
- “ptpectrl” on page 88

PtpeColQueryStats

Purpose

Retrieves the list of statistics being collected and summarized on one or more systems in the monitoring hierarchy.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeColQueryStats(sblock, targets, reply)
session_ptr_t   sblock;
host_list_t     targets;
host_list_t     *reply;
```

Parameters

sblock
The address of a session control block. This information block is created and initialized by the **PtpeOpenSession** subroutine.

targets
The anchor of a host list that contains at least one entry. This list should not contain any systems that have statistics assigned to them.

reply
A pointer to a host_list_t data type, which should be set to a NULL value before invoking this subroutine. Upon successful completion of the subroutine, an anchor point for a host list will be placed at this location. This list will contain entries for each system in the *targets* list. A statistics list will be assigned to those systems that are collecting and aggregating performance information, and will contain an entry for each statistic being collected on that system.

Description

PtpeColQueryStats asks the Performance Toolbox Parallel Extensions to provide the list of statistics being collected and summarized by one or more systems in the monitoring hierarchy.

The application must first establish a PTPE API session by using the **PtpeOpenSession** subroutine. The application must also construct a host list for the query, containing an entry for each system that the application needs to query. This list is then provided in the *targets* parameter.

When **PtpeColQueryStats** is executed, the subroutine relays the query command, along with the host list of systems to query, to the monitoring hierarchy's central coordinator node. The central coordinator node determines which nodes are data managers for the systems in the *targets* list, and forwards the query to only those data manager nodes that manage systems in the query list. The data manager nodes then relay the query to those systems in their reporting groups that have been listed in the targets list.

Each system in the *targets* list constructs a statistics list, containing one entry for each statistics that system is currently making available to Performance Toolbox Parallel Extensions for collection and summarization. On systems that also double as managing systems, this list will also include any summary statistics that the node prepares. This list is then relayed back to the data manager node.

Data manager nodes construct partial host lists, with an entry for each system in its reporting group that was targeted for the query. The statistics lists received from each node are assigned to their respective host list entries in the data manager node's partial host list. This list is then relayed to the central coordinator node, who constructs a complete host list. The complete host list is transmitted back to the **PtpeColQueryStats** subroutine, which provides it to the application in the *reply* parameter.

PtpeColQueryStats will provide an indication of the overall success or failure of the query in the return code:

PTPE_SUCCESS	All systems in the *targets* list successfully responded to the query.
PTPE_LIMITED	Some systems in the *targets* list were unable to respond successfully to the query.
PTPE_API_FAILED	All systems in the *targets* list were unable to respond successfully to the query. To determine which systems were not successful, the reply list should be scanned, and the results for each system in the list checked using the **PtpeGetHostResult** subroutine. The result code will provide an indication of the error for the system. To determine what statistics are being collected for those systems that successfully responded to the query, extract the statistics list for each entry by using the **PtpeGetHostStatList** subroutine, and scan the list using the statistics list scanning subroutines.

This subroutine cannot be issued from a PTPE read-only session.

Return Codes

Upon successful completion, a return code of PTPE_SUCCESS is returned to the caller. If an error occurred, one of the following return codes is provided:

PTPE_SUCCESS_BADR	The central coordinator node has indicated that all systems in the *targets* list responded successfully to the query, but an error occurred in reading the *reply* list response from the central coordinator Node. The application should treat this as an error, since the contents of the *reply* list cannot be guaranteed to be accurate.
PTPE_LIMITED	Some of the systems in the *targets* list were unable to reply to the query. The list *reply* points to contains all systems involved in the query; the application can determine which systems failed by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.

PTPE_LIMITED_BADR	Some systems in the *targets* list were unable to respond to the query, and an error occurred in reading the *reply* list response from the central coordinator node. The application should treat this as an error, since the contents of the *reply* list cannot be guaranteed to be accurate.
PTPE_API_FAILED	The central coordinator node has indicated that all systems in the *targets* list failed to respond to the query. *reply* contains all systems involved in the query, along with the reasons for the failure on these systems. The application can determine the cause of the error by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_API_FAILED_BADR	The central coordinator node has indicated that all systems failed to respond to the query, and an error occurred in reading the *reply* list from the central coordinator node. the application should treat this as an error, since the contents of the *reply* list cannot be guaranteed.
PTPE_STATE	Performance information collection and summarization is not currently active in the monitoring hierarchy.
PTPE_NO_CONNECT	Could not establish a network connection to the central coordinator. *reply* list not modified.
PTPE_BAD_SEND	Could not successfully transmit the query to the central coordinator node. *reply* is not modified.
PTPE_BAD_RECEIVE	Error in receiving the reply to the query from the central coordinator node. It is impossible to determine if the query was successful. *reply* is not modified.
PTPE_RONLY_SESS	The application attempted to issue this subroutine from a PTPE read-only session. A regular session must be established to issue this instruction. Ensure that the application is executing in the proper SP system partition.
PTPE_TIMEOUT	A reply to the query was not received from the central coordinator node in the time allowed. reply is not modified.
PTPE_INV_HIERARCHY	The monitoring hierarchy contains a node that is not a member of the currently active system partition. The hierarchy must be repaired by removing any nodes that are not members of the active system partition before this subroutine can be used.
PTPE_INV_HOSTLIST	*targets* either contains a NULL value, or does not anchor a valid host list.
PTPE_INV_PTR	*sblock* has a NULL value, or *reply* is not NULL.
PTPE_EMPTY	*targets* is an empty host list.
PTPE_HOST_NOT_FOUND	One or more systems listed in *targets* could not be found in the monitoring hierarchy. *reply* contains the list of systems that could not be found in the hierarchy.

PTPE_NO_SESSION	A PTPE API session was not previously established by this application (see “PtpeOpenSession” on page 277).
PTPE_NO_HIERARCHY	A monitoring hierarchy does not exist. This subroutine cannot function without a monitoring hierarchy (see “ptpehier” on page 99).
PTPE_NO_MEMORY	Could not allocate enough memory to initiate the query. *reply* is not modified.
PTPE_MEMORY	An internal error occurred.
PTPE_BAD_LOC_PTR	The internal pointers in the *targets* anchor have been corrupted.
PTPE_SDR	An unexpected error occurred while obtaining information from the PTPE data classes in the System Data Repository. These data classes might not exist, or the System Data Repository might be experiencing difficulties.

Examples

```
#include <spdm.h>

session_ptr_t   sblock;
host_list_t     targets, reply;
int             rc;

/* set up session and "targets" earlier */
reply = (host_list_t) NULL;
rc = PtpeColQueryStats(sblock, targets, &reply);
if (rc != PTPE_SUCCESS) {
      printf("Not all systems succeded.\n");
}
/* loop through "reply" and check results */
/* on each system with PtpeGetHostResult  */
```

Related Information

- “PtpeOpenSession” on page 277
- “PtpeColStart” on page 215
- “PtpeColEnableStats” on page 193
- “PtpeArchEnableStats” on page 133
- “PtpeColDisableStats” on page 183
- “PtpeInitHostList” on page 265
- “PtpeAddHostToList” on page 114
- “PtpeFirstHost” on page 237
- “PtpeNextHost” on page 273
- “PtpeFindHost” on page 233
- “PtpeFreeHostList” on page 241
- “PtpeGetHost” on page 245

- “PtpeGetHostResult” on page 247
- “PtpeInitStatList” on page 267
- “PtpeAddStatToList” on page 116
- “PtpeFirstStat” on page 239
- “PtpeNextStat” on page 275
- “PtpeFindStat” on page 235
- “PtpeFreeStatList” on page 243
- “PtpeAssignStatsToHost” on page 170
- “PtpeGetHostStatList” on page 250
- “PtpeGetStatName” on page 252
- “PtpeGetStatResult” on page 254

PtpeColSetup

Purpose

Instructs all systems in the Performance Toolbox Parallel Extensions monitoring hierarchy to set up for performance information collection and summarization.

Library

libptpe.a

Syntax

```
#include <spdm.h>
int PtpeColSetup(sblock, mgrs, nodes)
session_ptr_t    sblock;
host_list_t      *mgrs;
host_list_t      *nodes;
```

Parameters

sblock
: The address of a session control block. This information block is created and initialized by the **PtpeOpenSession** subroutine.

mgrs
: A pointer to a host_list_t data type, which should be set to a NULL value before invoking this subroutine. Upon completion of the subroutine, an anchor point for a host list may be placed at this location, containing the list of any managing nodes that failed to carry out the setup command.

nodes
: A pointer to a host_list_t data type, which should be set to a NULL value before invoking this subroutine. Upon completion of the subroutine, an anchor point for a host list may be placed at this location, containing the list of any reporter nodes that failed to carry out the setup command.

Description

PtpeColSetup instructs all systems in the Performance Toolbox Parallel Extensions monitoring hierarchy to set up for performance information collection and summarization. This subroutine should be executed by an application when the monitoring hierarchy structure has been modified, before using the hierarchy for the first time. This subroutine should also be used whenever new performance information becomes available on systems in the monitoring hierarchy (such as when a new LPP has been installed).

During collection setup, all systems in the hierarchy report all performance information available on these systems to the managing systems and the PTPE central coordinator node. The central coordinator node prepares a complete list of all statistics that can be provided by any system in the monitoring hierarchy, and uses this list whenever performance information collection and summarization is begun to set up collection structures in the data manager nodes.

Should the monitoring hierarchy structure change, either for adjustments in the existing structure or because new systems were added, this subroutine should be performed. This subroutine should also be used when new performance information be made available in an existing monitoring hierarchy.

Should more than half of the systems in the Performance Toolbox Parallel Extensions monitoring hierarchy successfully complete the setup duties, this subroutine will report a successful completion of the setup. Should less than half of the systems successfully complete the setup duty, this subroutine will report a failure. Any systems that failed the setup are provided in the lists anchored at *mgrs* and *nodes*. The *mgrs* list will indicate the data manager nodes that could not complete the setup tasks, which may have prevented the systems that reported to these nodes from also completing the task. *nodes* will contain the list of any other systems that could not carry out the setup tasks. Entries in each list will have their respective result codes set to indicate the reason for failure.

PtpeColSetup requires that a PTPE API session was previously established with the **PtpeOpenSession** subroutine.

This subroutine performs the same function as the **ptpectrl -i** command. This subroutine cannot be issued from a PTPE read-only session.

Return Codes

Upon successful completion, this subroutine returns one of two values:

PTPE_RONLY_SESS	The application attempted to issue this subroutine from a PTPE read-only session. A regular session must be established to issue this instruction. Ensure that the application is executing in the proper SP system partition.
PTPE_SUCCESS	At least half of the systems in the monitoring hierarchy successfully completed the setup duties. Any systems that did not successfully carry out the setup duties are provided in the *mgrs* and *nodes*. lists.
PTPE_SUCCESS_BADR	At least half of the systems in the monitoring hierarchy successfully completed the setup duties. An error occurred while reading the list of any failing managing and reporter nodes from the central coordinator node.

If an error has occurred, the return code is set to one of the following values to indicate the cause of the error:

PTPE_API_FAILED	The central coordinator node has indicated that the setup function has failed (at least half of the systems in the monitoring hierarchy could not successfully complete the setup function). Any data manager nodes systems that could not carry out the setup function are listed in the *mgrs* list; any other systems that failed the setup function are listed in the *nodes* list.
PTPE_API_FAILED_BADR	The central coordinator node has indicated that the setup function has failed (at least half of the systems in the monitoring hierarchy could not successfully complete the setup function). The subroutine could not obtain the list of the systems that failed the setup function, so *mgrs* and *nodes* remains unchanged.

PTPE_COL_ACTIVE	Performance information collection and summarization is currently active in the monitoring hierarchy. The setup function cannot be performed while collection and summarization are active. *mgrs* and *nodes* are not modified.
PTPE_COL_ERROR	Performance information collection and summarization is current in an indeterminate state. The setup function cannot be performed while this state persists. The application should attempt to clear this condition by issuing a **PtpeColStop** call. *mgrs* and *nodes* are not modified.
PTPE_NO_CONNECT	Could not establish a network connection to the central coordinator node. *mgrs* and *nodes* are not modified.
PTPE_BAD_SEND	Could not successfully transmit the setup command to the central coordinator node. *mgrs* and *nodes* are not modified.
PTPE_BAD_RECEIVE	Error in receiving the reply to the setup command from the central coordinator node. It is impossible to determine if the setup function succeeded or failed; the subroutine treats it as a failure. *mgrs* and *nodes* are not modified.
PTPE_TIMEOUT	A reply to the setup command was not received from the central coordinator node in the time allowed. *mgrs* and *nodes* are not modified.
PTPE_CMGR_NOEXEC	Could contact the central coordinator node, but could not execute the collection daemon **spdmcold** on the *mgrs* and *nodes* are not modified.
PTPE_CMGR_CONTACT	The collection daemon **spdmcold** was successfully started on the central coordinator node, but the communication link with the daemon was lost. *mgrs* and *nodes* are not modified.
PTPE_INV_HIERARCHY	The monitoring hierarchy contains a node that is not a member of the currently active system partition. The hierarchy must be repaired by removing any nodes that are not members of the active system partition before this subroutine can be used.
PTPE_INV_PTR	data manager nodes, nodes, or *sblock* has a NULL value.
PTPE_NO_SESSION	A PTPE API session was not previously established by this application (see “PtpeOpenSession” on page 277).
PTPE_LOCKED	Another PTPE API application has exclusive use of the monitoring hierarchy.
PTPE_NO_LOCK_OBJ	The PTPE data classes do not exist in the System Data Repository (see “ptpeconf” on page 86).

PTPE_NO_HIERARCHY	A monitoring hierarchy does not exist. This subroutine cannot function without a monitoring hierarchy (see “ptpehier” on page 99).
PTPE_NO_MEMORY	Could not allocate enough memory to initiate the setup function. *mgrs* and *nodes* are not modified.
PTPE_MEMORY	An internal error occurred.
PTPE_SDR	An unexpected error occurred while obtaining information from the PTPE data classes in the System Data Repository. These data classes might not exist, of the System Data Repository might be experiencing difficulties.

Examples

```
#include <spdm.h>

session_ptr_t   sblock;
host_list_t     mgrs, nodes;
int             rc;

rc = PtpeOpenSession(&sblock);
if (rc != PTPE_SUCCESS) {
   printf("Cannot open session\n");
   return(0);
}
mgrs = (host_list_t) NULL;
nodes = (host_list_t) NULL;
rc = PtpeColSetup(sblock, &mgrs, &nodes);
if (rc != PTPE_SUCCESS) {
     /* scan mgrs and show which systems */
     /* failed                           */
}
```

Related Information

- “PtpeOpenSession” on page 277
- “PtpeFirstHost” on page 237
- “PtpeNextHost” on page 273
- “PtpeFindHost” on page 233
- “PtpeFreeHostList” on page 241
- “PtpeGetHost” on page 245
- “PtpeGetHostResult” on page 247
- “PtpeColStart” on page 215
- “PtpeColStop” on page 220

PtpeColStart

Purpose

PtpeColStart instructs all systems in the Performance Toolbox Parallel Extensions monitoring hierarchy to start the daemon programs needed for performance data collection and summarization.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeColStart(sblock, mgrs, nodes)
session_ptr_t   sblock;
host_list_t     *mgrs;
host_list_t     *nodes;
```

Parameters

sblock
The address of a session control block. This information block is created and initialized by the **PtpeOpenSession** subroutine.

mgrs
A pointer to a host_list_t data type, which should be set to a NULL value before invoking this subroutine. Upon completion of the subroutine, an anchor point for a host list may be placed at this location, containing the list of any managing nodes that failed to carry out the start command.

nodes
A pointer to a host_list_t data type, which should be set to a NULL value before invoking this subroutine. Upon completion of the subroutine, an anchor point for a host list may be placed at this location, containing the list of any reporter nodes that failed to carry out the start command.

Description

PtpeColStart instructs all systems in the Performance Toolbox Parallel Extensions monitoring hierarchy to begin performance data collection and summarization. This subroutine requires a monitoring hierarchy to be in place (see "ptpehier" on page 99), and that the hierarchy have been previously set up since the last time the hierarchy was modified (see "ptpectrl" on page 88 and "PtpeColSetup" on page 211).

During performance collection startup, the central coordinator Node obtains the complete list of performance information that was constructed during the previous collection setup. This list is encoded into binary format, and passed to the data manager nodes in the monitoring hierarchy, along with the monitoring hierarchy structure itself. The managing systems decode the monitoring hierarchy and the list of statistics from the central coordinator node's message, determines which systems report to it, and forward the list of statistics to these systems.

Each system in the monitoring hierarchy decodes the list of performance statistics from its data manager node (or the central coordinator Node, in the case of data manager nodes), and constructs a table in shared memory for these statistics. This table contains entries for each statistic along with indicator flags for the following information:

- whether this system is capable of providing the information at this time,
- whether this system is providing this information to its managing node, and
- whether this statistic is to be recorded in the performance information archive

Each node determines if the information exists at the present time of the system, and sets the first field accordingly. The provision field is set to TRUE, and the archive field is set to FALSE.

Each system then reports its success or failure in starting performance collection to its data manager node. The data manager nodes collect all responses from their reporters, combine them into a single message, and relay it to the central coordinator node. The central coordinator Node then prepares an overall response, and sends the results back to this caller of the **PtpeColStart** subroutine.

Should at least one of the systems in the Performance Toolbox Parallel Extensions monitoring hierarchy successfully complete the startup duties, this subroutine will report a successful completion of the startup. Should none of the systems successfully complete the startup duty, this subroutine will report a failure. Any systems that failed the startup are provided in the lists anchored at data manager nodes and nodes. The data manager nodes list will indicate the data manager nodes that could not complete the startup tasks, which may have prevented the systems that reported to these nodes from also completing the task. nodes will contain the list of any other systems that could not carry out the startup tasks. Entries in each list will have their respective result codes set to indicate the reason for failure.

Upon completion of the subroutine, the nodes that successfully responded to the command will have started the PTPE daemons for collection and summarization of performance data, however, no performance information is collected and summarized at this time. Collection and summarization do not begin until the application issues a **PtpeColEnableStats** or a **PtpeColEnableAllStats** call to make a set of performance statistics available for collection.

PtpeColStart requires that a PTPE API session was previously established with the **PtpeOpenSession** subroutine.

This subroutine performs the same function as the **ptpectrl -c** command. This subroutine cannot be issued from a PTPE read-only session.

Return Codes

Upon successful completion, this subroutine returns one of two values:

PTPE_SUCCESS	All systems in the monitoring hierarchy successfully completed collection start. Any systems that did not successfully carry out the setup duties appear in the *mgrs* and *nodes*. lists.

PTPE_SUCCESS_BADR	All systems in the monitoring hierarchy successfully completed collection start. An error occurred while reading the complete response from the central coordinator node.

If an error has occurred, the return code is set to one of the following values to indicate the cause of the error:

PTPE_LIMITED	At least one system in the monitoring hierarchy successfully completed collection start. Any systems that did not successfully carry out the setup duties appear in the **_mgrs_** and **_nodes_** lists. PTPE will consider that start *successful* in this case.
PTPE_LIMITED_BADR	At least one system in the monitoring hierarchy successfully completed collection start. An error occurred while reading the complete response from the central coordinator node. PTPE will consider the start a *failure* in this case, but will not automatically shut down the PTPE daemons on the nodes that successfully started collection.
PTPE_API_FAILED	The central coordinator node has indicated that the start function failed (none of the systems in the monitoring hierarchy could successfully complete the function). Any data managers systems that could not carry out the start function are listed in the *mgrs* list; any other systems that failed the function are listed in the *nodes* list.
PTPE_API_FAILED_BADR	The central coordinator node has indicated that the start function failed (none of the systems in the monitoring hierarchy could successfully complete the function). The subroutine could not obtain the list of the systems that failed the start function. *mgrs* and *nodes* are not modified.
PTPE_COL_ACTIVE	Performance information collection and summarization is currently active in the monitoring hierarchy. The start function cannot be performed while collection and summarization are active. *mgrs* and *nodes* are not modified.
PTPE_COL_ERROR	Performance information collection and summarization is current in an indeterminate state. The start function cannot be performed while this state persists. The application should attempt to clear this condition by issuing a **PtpeColStop** call. *mgrs* and *nodes* are not modified.
PTPE_NO_CONNECT	Could not establish a network connection to the central coordinator node. *mgrs* and *nodes* are not modified.
PTPE_BAD_SEND	Could not successfully transmit the start command to the central coordinator node. *mgrs* and *nodes* are not modified.

PTPE_BAD_RECEIVE	Error in receiving the reply to the start command from the central coordinator node. It is impossible to determine if the start function succeeded or failed; the subroutine treats it as a failure. *mgrs* and *nodes* are not modified.
PTPE_FILE_ERROR	An error occurred in creating an AIX file to contain information retrieved from the System Data Repository. A shortage of space or inodes in the **/tmp** filesystem is the most likely source of this error.
PTPE_RONLY_SESS	The application attempted to issue this subroutine from a PTPE read-only session. A regular session must be established to issue this instruction. Ensure that the application is executing in the proper SP system partition.
PTPE_TIMEOUT	A reply to the start command was not received from the central coordinator node in the time allowed. *mgrs* and *nodes* are not modified.
PTPE_CMGR_NOEXEC	Could contact the central coordinator node, but could not execute the collection daemon **spdmcold** on the *mgrs* and *nodes* are not modified.
PTPE_CMGR_CONTACT	The collection daemon **spdmcold** was successfully started on the central coordinator node, but the communication link with the daemon was lost. *mgrs* and *nodes* are not modified.
PTPE_INV_HIERARCHY	The monitoring hierarchy contains a node that is not a member of the currently active system partition. The hierarchy must be repaired by removing any nodes that are not members of the active system partition before this subroutine can be used.
PTPE_INV_PTR	data manager nodes, nodes, or *sblock* has a NULL value.
PTPE_NO_SESSION	A PTPE API session was not previously established by this application (see “PtpeOpenSession” on page 277).
PTPE_LOCKED	Another PTPE API application has exclusive use of the monitoring hierarchy.
PTPE_NO_LOCK_OBJ	The PTPE data classes do not exist in the System Data Repository (see “ptpeconf” on page 86).
PTPE_NO_HIERARCHY	A monitoring hierarchy does not exist. This subroutine cannot function without a monitoring hierarchy (see “ptpehier” on page 99).
PTPE_NO_MEMORY	Could not allocate enough memory to initiate the start function. *mgrs* and *nodes* are not modified.
PTPE_MEMORY	An internal error occurred.
PTPE_SDR	An unexpected error occurred while obtaining information from the PTPE data classes in the System Data Repository. These data classes might not exist,

of the System Data Repository might be experiencing difficulties.

Examples

```
#include <spdm.h>

session_ptr_t   sblock;
host_list_t     mgrs, nodes;
int             rc;

rc = PtpeOpenSession(&sblock);
if (rc != PTPE_SUCCESS) {
     printf("Cannot open session\n");
     return(0);
}
mgrs = (host_list_t) NULL;
nodes = (host_list_t) NULL;
rc = PtpeColStart(sblock, &mgrs, &nodes);
if (rc != PTPE_SUCCESS) {
    /* scan mgrs and show which systems */
    /* failed                           */
}
```

Related Information

- “PtpeOpenSession” on page 277
- “PtpeFirstHost” on page 237
- “PtpeNextHost” on page 273
- “PtpeFindHost” on page 233
- “PtpeFreeHostList” on page 241
- “PtpeGetHost” on page 245
- “PtpeGetHostResult” on page 247
- “PtpeColSetup” on page 211
- “PtpeColStop” on page 220

PtpeColStop

Purpose

Instructs all systems in the Performance Toolbox Parallel Extensions monitoring hierarchy to cease collection, summarization, and recording of performance information.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeColStop(sblock, mgrs, nodes)
session_ptr_t   sblock;
host_list_t     *mgrs;
host_list_t     *nodes;
```

Parameters

sblock
: The address of a session control block. This information block is created and initialized by the **PtpeOpenSession** subroutine.

mgrs
: A pointer to a host_list_t data type, which should be set to a NULL value before invoking this subroutine. Upon completion of the subroutine, an anchor point for a host list may be placed at this location, containing the list of any data manager nodes that failed to carry out the shutdown command.

nodes
: A pointer to a host_list_t data type, which should be set to a NULL value before invoking this subroutine. Upon completion of the subroutine, an anchor point for a host list may be placed at this location, containing the list of any reporter nodes that failed to carry out the shutdown command.

Description

PtpeColStop instructs all systems in the Performance Toolbox Parallel Extensions monitoring hierarchy to stop any performance information collection, summarization, and recording that may be active. This subroutine requires that a monitoring hierarchy be in place (see “ptpehier” on page 99), and that performance information collection and summarization was previously begun (see “ptpectrl” on page 88) and the **PtpeColStart** subroutine).

A shutdown command is encoded, along with a binary representation of the monitoring hierarchy, and the complete message is transmitted to the monitoring hierarchy's central coordinator node. The central coordinator node decodes the instructions and the monitoring hierarchy from the message, determines which systems in the hierarchy are the managing systems, and relays the shutdown command to these systems. The managing systems, in turn, relay the shutdown command to all nodes in their monitoring group. All nodes report their success or failure in shutting down performance collection and archiving to their managing systems, which reports the list of successes or failures back to the central coordinator Node. The central coordinator node determines overall success or failure of the request, constructs a complete list of all successes and failures, and sends this result and list back to this subroutine.

If all nodes that were actively collecting and summarizing performance information report a successful shut down of the PTPE daemons, this subroutine will report a successful completion of the shutdown. Should any of the nodes be unable to shut down collection and summarization, this subroutine will report a failure. Any systems that failed the shutdown are provided in the lists anchored at data manager nodes and nodes. The data manager nodes list will indicate the data manager nodes that could not completely shut down collection and summarization, which may have prevented the systems that reported to these nodes from also completing the task. nodes will contain the list of any other systems that could not shut down collection and summarization. Entries in each list will have their respective result codes set to indicate the reason for failure.

In cases where the shutdown was considered successful, any systems that succeeded in the shutdown attempt will cease to forward all available performance information to their data manager nodes for collection and summarization. These systems also cease to record their performance information to the archive, if archiving was active before **PtpeColStop** was called.

If the **ptpectrl -c** command or the **PtpeColStart** subroutine was interrupted before completion, collection and summarization may be in an indeterminate state. While collection is in this indeterminate state, API subroutines -- such as **PtpeColSetup** and **PtpeColStart** -- will fail with an error code of PTPE_COL_ERROR. This indeterminate state exists until it is explicitly cleared. An application can issue the **PtpeColStop** subroutine to clear this indeterminate state.

PtpeColStop requires that a PTPE API session was previously established with the **PtpeOpenSession** subroutine.

This subroutine performs the same function as the **ptpectrl -s** command. This subroutine cannot be issued from a PTPE read-only session.

Return Codes

Upon successful completion, this subroutine returns one of two values:

PTPE_SUCCESS	All systems in the monitoring hierarchy that were collecting and summarizing performance information have reported a successful shutdown.
PTPE_SUCCESS_BADR	All systems in the monitoring hierarchy that were collecting and summarizing performance information have reported a successful shutdown. An error occurred while reading the full response from the central coordinator node.

If an error has occurred, the return code is set to one of the following values to indicate the cause of the error:

PTPE_LIMITED	Some systems in the monitoring hierarchy successfully shut down collection. Any systems that did not successfully carry out the shutdown duties appear in

	the **_mgrs_** and **_nodes_** lists. PTPE will consider the shutdown a *failure* in this case.
PTPE_LIMITED_BADR	Some of the systems in the monitoring hierarchy successfully shut down collection. An error occurred while reading the list of any failing data manager and reporter nodes from the central coordinator node. PTPE will consider the shutdown a *failure* in this case.
PTPE_API_FAILED	The central coordinator node has indicated that the collection shutdown has failed (at least half of the systems in the monitoring hierarchy could not successfully shut down collection). Any data manager nodes that could not shut down collection are listed in the *mgrs* host list; any other systems that failed to shut down collection are listed in the *nodes* list.
PTPE_API_FAILED_BADR	The central coordinator node has indicated that the shutdown function has failed (at least half of the systems in the monitoring hierarchy could not successfully shut down collection). The subroutine could not obtain the list of the systems that failed to shut down collection. *mgrs* and *nodes* are not modified.
PTPE_COL_OFF	Performance information collection and summarization is not currently active in the monitoring hierarchy. Collection cannot be shut down while collection is inactive. *mgrs* and *nodes* are not modified.
PTPE_NO_CONNECT	Could not establish a network connection to the central coordinator node. *mgrs* and *nodes* are not modified.
PTPE_BAD_SEND	Could not successfully transmit the shutdown command to the central coordinator node. *mgrs* and *nodes* are not modified.
PTPE_BAD_RECEIVE	Error in receiving the reply to the shutdown command from the central coordinator node. It is impossible to determine if the shutdown request succeeded or failed; the subroutine treats it as a failure. *mgrs* and *nodes* are not modified.
PTPE_RONLY_SESS	The application attempted to issue this subroutine from a PTPE read-only session. A regular session must be established to issue this instruction. Ensure that the application is executing in the proper SP system partition.
PTPE_TIMEOUT	A reply to the shutdown command was not received from the central coordinator node in the time allowed. *mgrs* and *nodes* are not modified.
PTPE_CMGR_NOEXEC	Could contact the central coordinator node, but could not execute the collection daemon **spdmcold** on the central coordinator node. The data manager nodes and nodes are not modified.

PTPE_CMGR_CONTACT	The collection daemon **spdmcold** was successfully started on the central coordinator node, but the communication link with the daemon was lost. *mgrs* and *nodes* are not modified.
PTPE_INV_HIERARCHY	The monitoring hierarchy contains a node that is not a member of the currently active system partition. The hierarchy must be repaired by removing any nodes that are not members of the active system partition before this subroutine can be used.
PTPE_INV_PTR	mgrs, nodes, or *sblock* has a NULL value.
PTPE_NO_SESSION	A PTPE API session was not previously established by this application (see “PtpeOpenSession” on page 277).
PTPE_LOCKED	Another PTPE API application has exclusive use of the monitoring hierarchy.
PTPE_NO_LOCK_OBJ	The PTPE data classes do not exist in the System Data Repository (see “ptpeconf” on page 86).
PTPE_NO_HIERARCHY	A monitoring hierarchy does not exist. This subroutine cannot function without a monitoring hierarchy (see “ptpehier” on page 99
PTPE_NO_MEMORY	Could not allocate enough memory to initiate the setup function. *mgrs* and *nodes* are not modified.
PTPE_MEMORY	An internal error occurred.
PTPE_SDR	An unexpected error occurred while obtaining information from the PTPE data classes in the System Data Repository. These data classes might not exist, or the System Data Repository might be experiencing difficulties.

Examples

```
#include <spdm.h>

session_ptr_t    sblock;
host_list_t      mgrs, nodes;
int              rc;

rc = PtpeOpenSession(&sblock);
if (rc != PTPE_SUCCESS) {
    printf("Cannot open session\n");
    return(0);
}
mgrs = (host_list_t) NULL;
nodes = (host_list_t) NULL;
rc = PtpeColSttop(sblock, &mgrs, &nodes);
if (rc != PTPE_SUCCESS) {
    /* scan mgrs and show which systems */
    /* failed                           */
}
```

Related Information

- “PtpeOpenSession” on page 277
- “PtpeFirstHost” on page 237
- “PtpeNextHost” on page 273
- “PtpeFindHost” on page 233
- “PtpeFreeHostList” on page 241
- “PtpeGetHost” on page 245
- “PtpeGetHostResult” on page 247
- “PtpeColSetup” on page 211
- “PtpeColStart” on page 215

PtpeDelHostFromList

Purpose

Removes a system's entry from an existing host list.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeDelHostFromList(hostname, hanchor)
char          *hostname;
host_list_t   hanchor;
```

Parameters

hostname
: Points to a NULL-terminated character string of up to PTPE_NMLN characters in length, which contains the network name of the system to be removed from the host list.

hanchor
: The anchor point of an existing host list that contains an entry for *hostname*.

Description

PtpeDelHostFromList removes the entry for the system specified by *hostname* from the host list anchored by *hanchor*. The memory used to store the system's information, and any statistics that may have been associated with that system, is freed. The “current” entry pointers in the host list anchor are reset to the beginning of the host list, or to indicate an empty host list if the last entry is removed from the list by this subroutine.

This subroutine should be used when a system will no longer be part of API commands that are sent to other systems that are members of the host list.

hostname should be specified in the same manner as the system is known to the Performance Toolbox for AIX monitoring hierarchy, and in exactly the same format as it was added to the host list by the **PtpeAddHostToList** subroutine. If the system is defined in the monitoring hierarchy by its fully qualified network name, such as spnode05.ibm.com, the same format should be used when deleting the system from the host list. If a system is known by more than one name to the network, the name that should be used by this subroutine is the name by which the system is known to the monitoring hierarchy. If the name used by this subroutine is not the same name, in the same format, as used by the monitoring hierarchy, **PtpeDelHostFromList** will not be able to locate the system in the host list.

An error is returned if *hanchor* references an empty host list, or if *hostname* cannot be found in the *hanchor* host list.

Return Codes

Upon successful completion, this subroutine returns the value of PTPE_SUCCESS. If an error has occurred, the return code is set to one of the following values to indicate the cause of the error:

PTPE_INV_HOSTLIST	*hanchor* has a NULL value.
PTPE_INV_HOSTNAME	*hostname* has a NULL value, or points to a NULL character string.
PTPE_HOST_NOT_FOUND	The system specified by *hostname* could not be found in the *hanchor* host list.
PTPE_EMPTY	*hanchor* is an empty host list.
PTPE_BAD_LOC_PTR	The memory pointers in *hanchor* have been corrupted.

Examples

```
#include <spdm.h>

host_list_t   hanchor;
char          delhost[PTPE_NMLN];
int           rc;

strcpy(delhost, "spnode05.ibm.com");
rc = PtpeDelHostFromList(delhost, hanchor);
switch (rc) {
     case PTPE_SUCCESS:
          break;
     case PTPE_HOST_NOT_FOUND:
          /* handle missing host error */
     case PTPE_EMPTY:
          /* handle empty list error */
     default:
          /* handle general error */
}
```

Related Information

- "PtpeInitHostList" on page 265
- "PtpeAddHostToList" on page 114
- "PtpeFindHost" on page 233

PtpeDelStatFromList

Purpose

Removes a statistics entry from an existing statistic list.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeDelStatFromList(statname, sanchor)
char        *statname;
stat_list_t  sanchor;
```

Parameters

statname
: Points to a NULL-terminated character string of up to PTPE_NMLN characters in length, which contains the full context path of the statistic to be removed from the statistics list.

sanchor
: The anchor point of an existing statistics list that contains an entry for *statname*.

Description

PtpeDelStatFromList removes the entry for the statistic specified by *statname* from the statistics list anchored by *sanchor*. The memory used to store the statistics information is freed. The "current" entry pointers in the statistics list anchor are reset to the beginning of the statistics list, or to indicate an empty statistics list if the last entry is removed from the list by this subroutine.

This subroutine should be used when a statistic will no longer be part of API commands that involve other statistics that are members of the statistics list.

statname specifies the full context path of the statistic being removed from the list. This name is provided in **Spmi** format, relative to the top context, and must match exactly to the format used when the statistics was added to the statistics list with the **PtpeAddStatToList** subroutine. The statistic path name does not include the **/host/**<*hostname*> prefix used by the Performance Toolbox **Rsi** programming library. No verification is performed on the format of the statistic name, nor does the API confirm that such a statistic exists, at the time the statistic is accepted for processing by this subroutine.

If the statistics list was previously assigned to a system by the **PtpeAssignStatsToHost** subroutine, the deletion performed by this subroutine occurs in the "unassigned" statistics list *sanchor* only; the copy of the statistics list that was previously assigned to a system remains unchanged by this subroutine.

An error is returned if *sanchor* references an empty host list, or if *hostname* cannot be found in the *sanchor* host list.

Return Codes

Upon successful completion, this subroutine returns the value of PTPE_SUCCESS. If an error has occurred, the return code is set to one of the following values to indicate the cause of the error:

PTPE_INV_STATLIST	*sanchor* has a NULL value.
PTPE_INV_STATNAME	*statname* has a NULL value, or points to a NULL character string.
PTPE_STAT_NOT_FOUND	The statistic specified by *statname* could not be found in the *sanchor* host list.
PTPE_EMPTY	*sanchor* is an empty host list.
PTPE_BAD_LOC_PTR	The memory pointers in *sanchor* have been corrupted.

Examples

```
#include <spdm.h>

stat_list_t    sanchor;
char           delstat[PTPE_STNL];
int            rc;

strcpy(delstat, "Mem/Virt/%free");
rc = PtpeDelStatFromList(delstat, sanchor);
switch (rc) {
     case PTPE_SUCCESS:
          break;
     case PTPE_STAT_NOT_FOUND:
          /* handle missing host error */
     case PTPE_EMPTY:
          /* handle empty list error */
     default:
          /* handle general error */
}
```

Related Information

- “PtpeInitStatList” on page 267
- “PtpeAddStatToList” on page 116
- “PtpeFindStat” on page 235
- “PtpeAssignStatsToHost” on page 170
- “PtpeGetHostStatList” on page 250

PtpeEmptyHostList

Purpose

Clears the contents of a host list.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeEmptyHostList(hanchor)
host_list_t     *hanchor;
```

Parameters

hanchor
The anchor point of an existing non-empty host.

Description

PtpeEmptyHostList clears out all entries in the host list anchored at *hanchor*. The memory used by the host list for these entries is freed, including the memory used to contain any statistics lists that may have been assigned to system entries in the host list.

An error results if this subroutine is performed upon a non-existent host list.

Return Codes

If an invalid host list was referenced by the *hanchor* parameter, or the *hanchor* parameter is NULL, this subroutine returns a value of PTPE_INV_HOSTLIST. In all other cases, this subroutine returns a value of PTPE_SUCCESS.

Examples

```
#include <spdm.h>

host_list_t   hanchor;
int           rc;

rc = PtpeInitHostList(&hanchor);
rc = PtpeAddHostToList("spnode05.ibm.com",
                       hanchor);
rc = PtpeEmptyHostList(&hanchor);
if (rc != PTPE_SUCCESS) {
/* handle error condition */
}
```

Related Information

- "PtpeInitHostList" on page 265
- "PtpeFreeHostList" on page 241
- "PtpeAddHostToList" on page 114
- "PtpeDelHostFromList" on page 225

PtpeEmptyStatList

Purpose

Clears the contents of a statistics list.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeEmptyStatList(sanchor)
stat_list_t    *sanchor;
```

Parameters

sanchor
The anchor point of an existing non-empty statistics list.

Description

PtpeEmptyStatList clears out all entries in t statistics list anchored at *sanchor*. The memory used by the statistics list for these entries is freed.

If the statistics list was previously assigned to a system by the **PtpeAssignStatsToHost** subroutine, or if this statistic was retrieved from a system's host list entry by the **PtpeGetHostStatList** subroutine, the test performed by this subroutine occurs in the “unassigned” statistics list *sanchor* only. The copy of the statistics list assigned to a system remains unchanged by this subroutine.

Statistics lists assigned to systems cannot be cleared using this subroutine; instead, these lists must be removed from the system's host list entry by using the **PtpeRemoveStatsFromHost** subroutine.

An error results if this subroutine is performed upon a non-existent statistics list.

Return Codes

If an invalid statistics list was referenced by the *sanchor* parameter, or the *sanchor* parameter is NULL, this subroutine returns a value of PTPE_INV_STATLIST. In all other cases, this subroutine returns a value of PTPE_SUCCESS.

Examples

```
#include <spdm.h>

stat_list_t    sanchor;
int            rc;

rc = PtpeInitStatList(&sanchor);
rc = PtpeAddStatToList("Mem/Virt/%free",
                        sanchor);
rc = PtpeEmptyStatList(&sanchor);
if (rc != PTPE_SUCCESS) {
/* handle error condition */
}
```

Related Information

- “PtpeInitStatList” on page 267
- “PtpeAddStatToList” on page 116
- “PtpeDelStatFromList” on page 227
- “PtpeAssignStatsToHost” on page 170
- “PtpeFreeStatList” on page 243
- “PtpeGetHostStatList” on page 250
- “PtpeFindStat” on page 235
- “PtpeFirstStat” on page 239
- “PtpeGetHostStatList” on page 250
- “PtpeRemoveStatsFromHost” on page 298

PtpeFindHost

Purpose

Locates a specific system in an existing host list, and positions the host list anchor pointers for further operations on that system's host list entry.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeFindHost(hostname, hanchor)
char           *hostname;
host_list_t    hanchor;
```

Parameters

hostname
Points to a NULL-terminated character string of up to PTPE_NMLN characters in length, which contains the network name of the system to be located in the host list.

hanchor
The anchor point of an existing host list that contains an entry for *hostname*.

Description

PtpeFindHost scans the host list anchored at *hanchor* from beginning to end for an entry that contains information for the system specified by *hostname*. If an entry is found in the host list, the pointers in the host list anchor are updated to point to the system's entry as the “current” entry for the list.

After this subroutine has located an entry for a system, any subsequent API subroutines that perform operations on the “current” entry in a host list will be performed upon this system's entry in the host list. This behavior will continue until another subroutine that modifies the setting for the “current” host list entry is executed.

hostname should be specified in the same manner as the system is known to the Performance Toolbox for AIX monitoring hierarchy, and in exactly the same format as it was added to the host list by the **PtpeAddHostToList** subroutine. If the system is defined in the monitoring hierarchy by its fully qualified network name, such as spnode05.ibm.com, the same format should be used when locating the system to the host list. If a system is known by more than one name to the network, the name that should be used by this subroutine is the name by which the system is known to the monitoring hierarchy. If the name used by this subroutine is not the same name, in the same format, as used by the monitoring hierarchy, **PtpeFindHost** will not be able to locate the system in the host list.

If the host list is empty, or the system does not have an entry in the host list, an error is returned, and the "current" pointer in the anchor is reset to the beginning of the host list.

Return Codes

Upon successful completion, this subroutine returns the value of PTPE_SUCCESS. If an error has occurred, the return code is set to one of the following values to indicate the cause of the error:

PTPE_INV_HOSTLIST	*hanchor* has a NULL value.
PTPE_INV_HOSTNAME	*hostname* has a NULL value, or points to a NULL character string.
PTPE_HOST_NOT_FOUND	The system specified by *hostname* could not be found in the *hanchor* host list.
PTPE_EMPTY	*hanchor* is an empty host list.
PTPE_BAD_LOC_PTR	The memory pointers in *hanchor* have been corrupted.

Examples

```
#include <spdm.h>

host_list_t    hanchor;
char           findme[PTPE_NMLN];
int            rc;

strcpy(findme, "spnode05,ibm.com");
rc = PtpeFindHost(findme, hanchor);
switch (rc) {
     case PTPE_SUCCESS:
          break;
     case PTPE_HOST_NOT_FOUND:
          /* handle missing host error */
     case PTPE_EMPTY:
          /* handle empty list error */
     default:
          /* handle general error */
}
```

Related Information

- "PtpeInitHostList" on page 265
- "PtpeAddHostToList" on page 114
- "PtpeDelHostFromList" on page 225
- "PtpeFirstHost" on page 237
- "PtpeNextHost" on page 273

PtpeFindStat

Purpose

Locates a specific statistic in an existing statistics list, and positions the statistics list anchor pointers for further operations on that statistic's list entry.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeFindStat(statname, sanchor)
char          *statname;
stat_list_t   sanchor;
```

Parameters

hostname
Points to a NULL-terminated character string of up to PTPE_STLN characters in length, which contains the full context path of the statistic to be located in the statistics list.

sanchor
The anchor point of an existing statistics list that contains an entry for *statname*.

Description

PtpeFindStat scans the statistics list anchored at *sanchor* from beginning to end for an entry that contains information for the statistic specified by *statname*. If an entry is found in the statistics list, the pointers in the statistics list anchor are updated to point to the statistic's entry as the “current” entry for the list.

After this subroutine has located an entry for a statistic, any subsequent API subroutines that perform operations on the “current” entry in a statistics list will be performed upon this statistic's entry in the statistics list. This behavior will continue until another subroutine that modifies the setting for the “current” statistics list entry is executed.

statname specifies the full context path of the statistic being located into the list. This name is provided in **Spmi** format, relative to the top context, and must match exactly to the format used when the statistics was added to the statistics list with the **PtpeAddStatToList** subroutine. The statistic path name does not include the **/host/**<*hostname*> prefix used by the Performance Toolbox **Rsi** programming library. No verification is performed on the format of the statistic name, nor does the API confirm that such a statistic exists, at the time the statistic is accepted for processing by this subroutine.

If the statistics list was previously assigned to a system by the **PtpeAssignStatsToHost** subroutine, the search perform subroutine occurs in the “unassigned” statistics list *sanchor* only; the copy of the statistics list that was previously assigned to a system remains unchanged by this subroutine.

If the statistics list is empty, or the statistic does not have an entry in the statistics list, an error is returned, and the “current” pointer in the anchor is reset to the beginning of the statistics list.

Return Codes

Upon successful completion, this subroutine returns the value of PTPE_SUCCESS. If an error has occurred, the return code is set to one of the following values to indicate the cause of the error:

PTPE_INV_STATLIST	*sanchor* has a NULL value.
PTPE_INV_STATNAME	*statname* has a NULL value, or points to a NULL character string.
PTPE_STAT_NOT_FOUND	The system specified by *statname* could not be found in the *sanchor* host list.
PTPE_EMPTY	*sanchor* is an empty host list.
PTPE_BAD_LOC_PTR	The memory pointers in *sanchor* have been corrupted.

Examples

```
#include <spdm.h>

stat_list_t    sanchor;
char           findme[PTPE_STNL];
int            rc;

strcpy(findme, "Mem/Virt/%free");
rc = PtpeFindStat(findme, sanchor);
switch (rc) {
     case PTPE_SUCCESS:
          break;
     case PTPE_STAT_NOT_FOUND:
          /* handle missing host error */
     case PTPE_EMPTY:
          /* handle empty list error */
     default:
          /* handle general error */
}
```

Related Information

- “PtpeInitStatList” on page 267
- “PtpeAddStatToList” on page 116
- “PtpeDelStatFromList” on page 227
- “PtpeAssignStatsToHost” on page 170
- “PtpeGetHostStatList” on page 250

PtpeFirstHost

Purpose

Positions the host list anchor pointers for operations upon the first entry in the host list.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeFirstHost(hanchor)
host_list_t    hanchor;
```

Parameters

hanchor
The anchor point of an existing, non-empty host list.

Description

PtpeFirstHost updates the pointers in the host list anchor *hanchor* to point to the first entry in the host list as the "current" host list entry. Any subsequent API subroutines that perform operation upon the "current" host list entry will be performed upon the first entry in the host list. This behavior continues until the "current" entry pointers are reset using by another API subroutine.

An error occurs if this subroutine is executed upon an empty host list.

Return Codes

Upon successful completion, this subroutine returns the value of PTPE_SUCCESS. If an error has occurred, the return code is set to one of the following values to indicate the cause of the error:

PTPE_INV_HOSTLIST	*hanchor* has a NULL value.
PTPE_EMPTY	*hanchor* is an empty host list.

Examples

```
#include <spdm.h>

host_list_t    hanchor;

rc = PtpeFirstHost(hanchor);
if (rc != PTPE_SUCCESS) {
/* handle error */
}
```

Related Information

- “PtpeInitHostList” on page 265
- “PtpeAddHostToList” on page 114
- “PtpeDelHostFromList” on page 225
- “PtpeFindHost” on page 233
- “PtpeNextHost” on page 273
- “PtpeIsLastHost” on page 269

PtpeFirstStat

Purpose

Positions the statistics list anchor pointers for operations upon the first entry in the statistics list.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeFirstStat(sanchor)
stat_list_t sanchor;
```

Parameters

sanchor
The anchor point of an existing, non-empty statistics list.

Description

PtpeFirstStat updates the pointers in the statistics list anchor *sanchor* to point to the first entry in the host list as the “current” statistics list entry. Any subsequent API subroutines that perform operation upon the “current” statistics list entry will be performed upon the first entry in the statistics list. This behavior continues until the “current” entry pointers are reset using by another API subroutine.

If the statistics list was previously assigned to a system by the **PtpeAssignStatsToHost** subroutine, the positioning performed by this subroutine occurs in the “unassigned” statistics list *sanchor* only; the copy of the statistics list that was previously assigned to a system remains unchanged by this subroutine.

An error occurs if this subroutine is executed upon an empty statistics list.

Return Codes

Upon successful completion, this subroutine returns the value of PTPE_SUCCESS. If an error has occurred, the return code is set to one of the following values to indicate the cause of the error:

PTPE_INV_STATLIST	*sanchor* has a NULL value.
PTPE_EMPTY	*sanchor* is an empty statistics list.

Examples

```
#include <spdm.h>

stat_list_t     sanchor;

rc = PtpeFirstStat(sanchor);
if (rc != PTPE_SUCCESS) {
/* handle error */
}
```

Related Information

- “PtpeInitStatList” on page 267
- “PtpeAddStatToList” on page 116
- “PtpeDelStatFromList” on page 227
- “PtpeAssignStatsToHost” on page 170
- “PtpeGetHostStatList” on page 250
- “PtpeIsLastStat” on page 271
- “PtpeGetHostStatList” on page 250
- “PtpeFindStat” on page 235

PtpeFreeHostList

Purpose

Frees the memory used to store a host list and its anchor point.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeFreeHostList(hanchor)
host_list_t   *hanchor;
```

Parameters

hanchor
The address of an existing host list anchor point.

Description

PtpeFreeHostList frees all memory used to store any contents of the host list pointed to by *hanchor*, including an statistics lists that may have been assigned to system entries in the host list. The host list anchor point is also freed, making the anchor point unusable until a new anchor point is allocated with the **PtpeInitHostList** subroutine.

An error occurs if this subroutine is performed on a non-existent host list.

Return Codes

If an invalid host list was referenced by the *hanchor* parameter, or the *hanchor* parameter is NULL, this subroutine returns a value of PTPE_INV_HOSTLIST. In all other cases, this subroutine returns a value of PTPE_SUCCESS.

Examples

```
#include <spdm.h>

host_list_t   hanchor;
int           rc;

rc = PtpeInitHostList(&hanchor);
rc = PtpeAddHostToList("spnode05.ibm.com",
                       hanchor);
rc = PtpeFreeHostList(&hanchor);
if (rc != PTPE_SUCCESS) {
/* handle error condition */
}
```

Related Information

- “PtpeInitHostList” on page 265
- “PtpeAddHostToList” on page 114
- “PtpeDelHostFromList” on page 225
- “PtpeEmptyHostList” on page 229

PtpeFreeStatList

Purpose

Frees the memory used to store a statistics list and its anchor point.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeFreeHostList(sanchor)
stat_list_t    *sanchor;
```

Parameters

sanchor
The address of an existing statistics list anchor point.

Description

PtpeFreeHostList frees all memory used to store any of the host list pointed to by *sanchor*, including an statistics lists that may have been assigned to system entries in the host list. The host list anchor point is also freed, making the anchor point unusable until a new anchor point is allocated with the **PtpeInitStatList** subroutine.

Statistics lists assigned to systems cannot be freed using this subroutine; instead, these lists must be removed from the system's host list entry by using the **PtpeRemoveStatsFromHost** subroutine.

An error occurs if this subroutine is performed on a non-existent host list.

Return Codes

If an invalid statistics list was referenced by the *sanchor* parameter, or the *sanchor* parameter is NULL, this subroutine returns a value of PTPE_INV_STATLIST. In all other cases, this subroutine returns a value of PTPE_SUCCESS.

Examples

```
#include <spdm.h>

stat_list_t    sanchor;
int            rc;

rc = PtpeInitStatList(&sanchor);
rc = PtpeAddStatToList("Mem/Virt/%free",
                       sanchor);
rc = PtpeFreeStatList(&sanchor);
if (rc != PTPE_SUCCESS) {
/* handle error condition */
}
```

Related Information

- “PtpeInitStatList” on page 267
- “PtpeAddStatToList” on page 116
- “PtpeDelStatFromList” on page 227
- “PtpeAssignStatsToHost” on page 170
- “PtpeEmptyStatList” on page 231
- “PtpeGetHostStatList” on page 250
- “PtpeRemoveStatsFromHost” on page 298

PtpeGetHost

Purpose

Returns the network name of the system whose information is contained in the “current” entry in a host list.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeGetHost(hostname, hanchor)
char        *hostname;
host_list_t  hanchor;
```

Parameters

hostname
Points to a NULL-terminated character string of up to PTPE_NMLN characters in length, which will contain the network name of the system of the system from the “current” entry in the host list.

hanchor
The anchor point of an existing non-empty host.

Description

PtpeGetHost returns the network name that was stored in the “current” entry of the host list *hanchor*. The calling subroutine must provide sufficient memory to store the network name, and provide a pointer to this memory area in the *hostname* parameter. An error results if this subroutine is performed on an empty host list.

Return Codes

Upon successful completion, this subroutine returns the value of PTPE_SUCCESS, and the memory area referenced by *hostname* is modified to contain the network name of the system in the “current” entry of the host list. If an error has occurred, the return code is set to one of the following values to indicate the cause of the error:

PTPE_INV_HOSTLIST	*hanchor* has a NULL value.
PTPE_EMPTY	*hanchor* is an empty host list.
PTPE_BAD_LOC_PTR	The memory pointers in *hanchor* have been corrupted.
PTPE_MEMORY	The subroutine could not write the system name to the memory location provided in *hostname*.

Examples

```
#include <spdm.h>

host_list_t    hanchor;
char           this_host[PTPE_NMLN];
int            rc;

rc = PtpeNextHost(hostlist);
if (rc == PTPE_SUCCESS) {
     rc = PtpeGetHost(this_host, hanchor);
     printf("Host is %s\n", this_host);
}
```

Related Information

- “PtpeInitHostList” on page 265
- “PtpeAddHostToList” on page 114
- “PtpeDelHostFromList” on page 225
- “PtpeFirstHost” on page 237
- “PtpeNextHost” on page 273
- “PtpeFindHost” on page 233
- “PtpeGetHostResult” on page 247
- “PtpeGetHostStatList” on page 250

PtpeGetHostResult

Purpose

Determines the result of a previous Performance Toolbox Parallel Extensions API subroutine upon the currently referenced entry in a host list.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeGetHostResult(hanchor, result)
host_list_t     hanchor;
int             *result;
```

Parameters

hanchor
The anchor point of an existing non-empty host.

result
The address of an integer, where the result code for the “current” entry in the host list will be placed.

Description

PtpeGetHostResult should be used upon a "reply" host list from a previous API subroutine that issued commands to control performance information or retrieve performance information from the Performance Toolbox Parallel Extensions monitoring hierarchy. **PtpeGetHostResult** obtains the result code of the operation, which is placed by these API subroutines in a system's host list entry, and copies the result code to the memory location referenced by *result*.

The result code returned indicates whether or not the system in the “current” host list entry for *hanchor* successfully carried out the previous API request. If the system failed to carry out the request, the result code will indicate the nature of the failure on that system.

An error results if this subroutine is performed on an empty host list. If this subroutine is used on any host list other than a host list returned as a reply from a previous API subroutine, the result code will be PTPE_SUCCESS for all entries in the host list.

Upon completion of **PtpeGetHostResult** the *result* parameter takes one of the following values:

PTPE_SUCCESS	The operation was successful.
PTPE_DAEMON_ERROR	The PTPE daemons on this node received an unrecognized or corrupted instruction, or received an enhanced instruction from a later release of PTPE, which it could not carry out.

PTPE_FILE_ERROR	The node could not create a file it needed to obtain information from the System Data Repository.
PTPE_AGAIN	The node was not in a state from which it could accept API requests. This condition can occur when the node has not fully completed initializing the PTPE daemons. Retry the request on this node at a later time.
PTPE_API_FAILED	The operation failed on this node.
PTPE_ARCH_ACTIVE	Recording was already active, and another request to begin recording performance information to the archive was received
PTPE_ARCH_OFF	Recording was inactive, and another request to stop recording performance information to the archive was received
PTPE_STAT_NOT_FOUND	At least one statistic in the host's statistic list could not be found on the node
PTPE_INV_STATLIST	None of the statistics in the host's statistic list could not be found on the node
PTPE_NO_CONTACT	Could not establish a network connection with this host to perform the operation
PTPE_NO_EXEC	A host connection was established, but the PTPE API handler could not be started on the system to fulfil the request.

Return Codes

Upon successful completion, this subroutine returns the value of PTPE_SUCCESS, signaling that the operation was successful and the memory area referenced by *result* is modified to contain the result code from the “current” entry in the host list. If an error has occurred, return code is set to one of the following values to indicate the cause of the error:

PTPE_INV_HOSTLIST	*hanchor* has a NULL value.
PTPE_INV_PTR	*result* is a NULL value.
PTPE_EMPTY	*hanchor* is an empty host list.
PTPE_BAD_LOC_PTR	The memory pointers in *hanchor* have been corrupted.

In all error cases, the memory area referenced by result is set to the value of PTPE_INV_HOSTLIST.

Examples

```
#include <spdm.h>

session_ptr_t   sblock;
host_list_t     targets, reply;
char            this_host[PTPE_NMLN];
int             rc, result;

rc = PtpeArchStartHosts(sblock, targets,
                        &reply);
rc = PtpeFirstHost(reply);
for (;;) {
     rc = PtpeGetHost(this_host, reply);
     rc = PtpeGetHostResult(reply,&result);
     if (rc == PTPE_SUCCESS) {
         printf("%s: code %d\n",this_host,
                          result);
     }
     rc = PtpeIsLastHost(reply);
     if (rc == PTPE_TRUE) {
         break;
     }
rc = PtpeNextHost(reply);
}
```

Related Information

- “PtpeInitHostList” on page 265
- “PtpeGetHost” on page 245
- “PtpeFirstHost” on page 237
- “PtpeNextHost” on page 273
- “PtpeFindHost” on page 233
- “PtpeColSetup” on page 211
- “PtpeColStart” on page 215
- “PtpeColStop” on page 220
- “PtpeArchStartHosts” on page 156
- “PtpeArchStopHosts” on page 165
- “PtpeColGetStats” on page 198
- “PtpeArchGetStats” on page 138
- “PtpeColQueryStats” on page 206
- “PtpeArchQueryStats” on page 147

PtpeGetHostStatList

Purpose

Makes a copy of a statistics list that has been assigned to the currently referenced system in a host list.

Library

libptpe.a

Syntax

```
#include <spdm.h>
int PtpeGetHostStatList(sanchor, hanchor)
stat_list_t    sanchor;
host_list_t    hanchor;
```

Parameters

sanchor
: Points to the statistic list anchor point. This parameter must reference an initialized statistics list before the subroutine is called.

hanchor
: The anchor point of a non-empty host list.

Description

PtpeGetHostStatList obtains a copy of the statistics list that has been assigned to the “current” entry in the host list *hanchor*. The anchor of the copy list is returned to the caller in the *sanchor* parameter.

The application should ensure that the *hanchor* host list references the appropriate entry by using the **PtpeFirstHost, PtpeNextHostList,** and **PtpeFindStat** subroutines. If care is not taken, the application may retrieve the wrong statistics list from the *hanchor* list.

PTPE API subroutines such as **PtpeColGetStats** and **PtpeArchGetStats** provide a host list to the calling application. Potentially, each entry in this reply host list contains a list of statistics and their values assigned to them. However, the API only provides subroutines to scan through, and obtain information from, a statistics list that has not been assigned to a host list entry.

In order to scan statistics lists returned from API subroutines, and retrieve statistical values from these lists, the application should get a copy of the statistics list using the **PtpeGetHostStatList** subroutine.

When the application has completed its use of the statistic list copy, the copy should be freed using the **PtpeFreeStatList** subroutine.

An error occurs when this subroutine is performed on an empty or uninitialized host list.

Return Codes

Upon successful completion, this subroutine returns the value of PTPE_SUCCESS, and the *sanchor* parameter is modified to point to a copy of the “current” host list entry's statistic list. If an error has occurred, the return code is set to one of the following values to indicate the cause of the error:

PTPE_INV_HOSTLIST	*hanchor* has a NULL value.
PTPE_INV_PTR	*sanchor* has a NULL value.
PTPE_INV_STATLIST	The currently referenced entry in *hanchor* does not have a statistics list assigned to it.
PTPE_EMPTY	*hanchor* is an empty host list.
PTPE_BAD_LOC_PTR	The memory pointers in *hanchor* have been corrupted.

Examples

```
#include <spdm.h>

host_list_t     hanchor;
stat_list_t     sanchor;

rc = PtpeFindHost("spnode05.ibm.com",
                   hanchor);
sanchor = (stat_list_t) NULL;
rc = PtpeInitStatList(&sanchor);
rc = PtpeGetHostStatList(sanchor, hanchor);
if (rc != PTPE_SUCCESS) {
            /* handle error condition */
}
```

Related Information

- “PtpeInitHostList” on page 265
- “PtpeFreeHostList” on page 241
- “PtpeFirstHost” on page 237
- “PtpeNextHost” on page 273
- “PtpeFindHost” on page 233
- “PtpeFirstStat” on page 239
- “PtpeNextStat” on page 275
- “PtpeFindStat” on page 235
- “PtpeGetStatName” on page 252
- “PtpeGetStatType” on page 258
- “PtpeGetStatValueLong” on page 263
- “PtpeGetStatValueFloat” on page 261

PtpeGetStatName

Purpose

Returns the full context path of the statistic whose information is contained in the "current" entry in a statistics list.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeGetStatName(statname, sanchor)
char         *statname;
stat_list_t   sanchor;
```

Parameters

statname
Points to a character string of up to PTPE_STNL characters in length, which will contain the full context path of the system of the statistic from the "current" entry in the statistics list.

sanchor
The anchor point of an existing non-empty statistics list.

Description

PtpeGetStatName returns the full context path that stored in the "current" entry of the statistics list *sanchor*. The calling subroutine must provide sufficient memory to store the statistic context, and provide a pointer to this memory area in the *statname* parameter.

An error results if this subroutine is performed on an empty statistics list.

Return Codes

Upon successful completion, this subroutine returns the value of PTPE_SUCCESS, and the memory area referenced by *statname* is modified to contain the full context of the statistic in the "current" entry of the statistics list. If an error has occurred, the return code is set to one of the following values to indicate the cause of the error:

PTPE_INV_STATLIST	*sanchor* has a NULL value.
PTPE_EMPTY	*sanchor* is an empty statistics list.
PTPE_BAD_LOC_PTR	The memory pointers in *sanchor* have been corrupted.
PTPE_MEMORY	The subroutine could not write the statistic name to the memory location provided in *statname*.

Examples

```
#include <spdm.h>

stat_list_t    sanchor;
char           this_stat[PTPE_STNL];
int            rc;

rc = PtpeNextStat(sanchor);
if (rc == PTPE_SUCCESS) {
     rc = PtpeGetStatName(this_stat,
                          sanchor);
     printf("Statistics is %s\n",
                           this_stat);
}
```

Related Information

- “PtpeInitStatList” on page 267
- “PtpeAddStatToList” on page 116
- “PtpeDelStatFromList” on page 227
- “PtpeAssignStatsToHost” on page 170
- “PtpeNextStat” on page 275
- “PtpeGetHostStatList” on page 250
- “PtpeGetStatResult” on page 254
- “PtpeFindStat” on page 235
- “PtpeFirstStat” on page 239

PtpeGetStatResult

Purpose

Determines the result of a previous Performance Toolbox Parallel Extensions API subroutine upon the currently referenced entry in a statistics list.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeGetStatResult(sanchor, result)
stat_list_t   sanchor;
int           *result;
```

Parameters

sanchor
The anchor point of an existing non-empty statistics list.

result
The address of an integer, where the result of the “current” entry in the statistics list will be placed.

Description

PtpeGetStatResult should be used upon a statistics retrieved from a *reply* host list from a previous API subroutine that issued commands to control performance information or retrieve performance information from the Performance Toolbox Parallel Extensions monitoring hierarchy.The **PtpeGetHostStatList** subroutine can be used to obtain a the statistics list from a system's entry in a *reply* host list. **PtpeGetStatResult** obtains the result code of the operation which is placed by these API subroutines in a statistics's statistics list entry, and copies the result code to the memory location referenced by *result*.

The result code returned indicates whether or not the statistic in the “current” host list entry for *sanchor* was successfully located the previous API request. If the system failed to locate the statistic, the result code will reflect this.

An error results if this subroutine is performed on an empty statistics list. Results are unpredictable if this subroutine is used upon a statistics list that was not retrieved from a "reply" host list.

Upon completion of **PtpeGetStatResult** the *result* parameter takes one of the following values:

PTPE_SUCCESS	Statistic operation was successful.
PTPE_TIME_APPROX	Statistic operation was successful, but PTPE could not find an entry for the statistic with the exact timestamp specified, so the entry with the closest matching timestamp was used.

PTPE_STAT_NOT_FOUND	The statistic could not be located on the node or in that node's archive

Return Codes

Upon successful completion, this subroutine returns the value of PTPE_SUCCESS, signaling that the statistic operation was successful. The memory area referenced by *result* is modified to contain the result code from the "current" entry in the statistics list. Otherwise, the return code is set to one of the following values:

PTPE_INV_STATLIST	*sanchor* has a NULL value.
PTPE_INV_PTR	*result* is a NULL value.
PTPE_EMPTY	*sanchor* is an empty statistics list.
PTPE_BAD_LOC_PTR	The memory pointers in *sanchor* have been corrupted.

In all error cases except PTPE_INV_PTR, the memory area referenced by *result* is set to the value of PTPE_INV_STATLIST.

Examples

```
#include <spdm.h>

session_ptr_t    sblock;
host_list_t      targets, reply;
stat_list_t      sanchor;
char             this_host[PTPE_NMLN];
char             this_stat[PTPE_STNL];
int              rc, result;
```

Related Information

- "PtpeInitStatList" on page 267
- "PtpeAddStatToList" on page 116
- "PtpeDelStatFromList" on page 227
- "PtpeAssignStatsToHost" on page 170
- "PtpeNextStat" on page 275
- "PtpeGetHostStatList" on page 250
- "PtpeFindStat" on page 235
- "PtpeFirstStat" on page 239
- "PtpeColGetStats" on page 198
- "PtpeArchGetStats" on page 138
- "PtpeColQueryStats" on page 206
- "PtpeArchQueryState" on page 144

PtpeGetStatTime

Purpose

Retrieves the timestamp from the currently referenced statistic entry in a statistics list.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeGetStatTime(tstamp, sanchor)
stat_list_t    sanchor;
struct tm      *tstmp;
```

Parameters

sanchor
The address of an existing statistics list anchor point.

tstmp
The address of a struct tm structure, contain the value of the *timestamp* if this subroutine successfully completes. The format of the tm structure is defined in the <time.h> header file.

Description

PtpeGetStatTime retrieves the timestamp recorded in “current” entry of the statistic list anchored at *sanchor*. This timestamp is recorded in an integer format in the entry, and is converted to struct tm format before being provided to the caller.

When an entry is created in a statistics list by the **PtpeAddStatToList** subroutine, the entry's timestamp field is initialized to a value of -1. If the timestamp has not been set by the **PtpeSetStatTime** subroutine, or updated by other subroutines that obtain statistic information from the Performance Toolbox Parallel Extensions monitoring hierarchy (such as **PtpeArchGetStats**), this subroutine will return a struct value that indicates a date of 31 December, 1969. Otherwise, the data and time returned is the value set in the currently referenced entry in *sanchor*.

This subroutine returns an error if it is performed on an invalid or empty statistics list.

Return Codes

Upon successful completion, this subroutine returns the value of PTPE_SUCCESS, and the timestamp from the “current” entry in the statistics list is placed in the memory location pointed to by *tstmp*. If an error has occurred, the memory area indicated by *tstmp* is not modified, and the return code is set to one of the following values to indicate the cause of the error:

PTPE_INV_PTR	*tstmp* contains a NULL value.
PTPE_INV_STATLIST	*sanchor* has a NULL value.

PTPE_EMPTY	*sanchor* is an empty host list.
PTPE_BAD_LOC_PTR	The memory pointers in *sanchor* have been corrupted.
PTPE_MEMORY	An error occurred when copying the timestamp value to the memory location indicated by the *tstmp* parameter.

Examples

```
#include <spdm.h>

stat_list_t    sanchor;
struct tm      tstmp;
char           statname[PTPE_STNL];

rc = PtpeFirstStat(sanchor);
for(;;) {
      rc = PtpeGetStatName(statname, sanchor);
      rc = PtpeGetStatTime(&tstmp, sanchor);
      if (rc != PTPE_SUCCESS) {
          /* handle error */
      }
      else{
            printf("Time on %s is %n",
                  statname, ctime(tstmp));
      }
      rc = PtpeIsLastStat(sanchor);
      if (rc == PTPE_TRUE) {
          break;
      }
      rc = PtpeNextStat(sanchor);
}
```

Related Information

- “PtpeInitStatList” on page 267
- “PtpeFirstStat” on page 239
- “PtpeNextStat” on page 275
- “PtpeFindStat” on page 235
- “PtpeGetStatName” on page 252
- “PtpeGetHostStatList” on page 250
- “PtpeRemoveStatsFromHost” on page 298
- “PtpeGetStatType” on page 258
- “PtpeGetStatValueLong” on page 263
- “PtpeGetStatValueFloat” on page 261
- “PtpeGetStatResult” on page 254

PtpeGetStatType

Purpose

Indicates the data type used to store the currently reference statistic in a statistics list.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeGetStatType(stype, sanchor)
ptpe_data_id   *stype;
stat_list_t    sanchor;
```

Parameters

sanchor
: The address of an existing statistics list anchor point.

stype
: The address of a ptpe_data_id, which will contain a data type identifier upon successful completion of this subroutine.

Description

PtpeGetStatType returns an identifier that indicate data type used by a statistic. The statistic used in this subroutine is the “current” statistic entry in the statistics list anchored at *sanchor*. Statistics are one of the following types:

PTPE_LONG	Indicates data of integer or long integer format.
PTPE_FLOAT	Indicates data of float or double format.
PTPE_NONE	Indicates that the data type has not been set yet for the statistic.

PtpeGetStatType allows an application to test the t data used to store the performance statistic in the statistics list, so that the proper API subroutine can be used to obtain the actual value of the statistic from the list (see “PtpeGetStatValueLong” on page 263 and “PtpeGetStatValueFloat” on page 261). Using a subroutine that retrieves the wrong data type from the list may yield incorrect data values to the application.

When an entry is created for a statistic by the **PtpeAddStatToList** subroutine, the data type is set by default to PTPE_NONE. If the statistic type has not been set to a specific type by the **PtpeSetStatType** subroutine, or by any other API subroutine that retrieves statistic information from the Performance Toolbox Parallel Extension monitoring hierarchy (such as **PtpeArchGetStats**), this subroutine will return this de initialization value.

Return Codes

Upon successful completion, this subroutine returns the value of PTPE_SUCCESS, and places the data type of the statistic at the location referenced by the *stype* parameter. If an error has occurred, the memory referenced by *stype* is not modified, and the return code is set to one of the following values to indicate the cause of the error:

PTPE_INV_STATLIST	*sanchor* has a NULL value.
PTPE_INV_PTR	*stype* has a NULL value.
PTPE_EMPTY	*sanchor* is an empty host list.
PTPE_BAD_LOC_PTR	The memory pointers in *sanchor* have been corrupted.
PTPE_MEMORY	An error occurred when the subroutine attempted to place the data type information in the memory location referenced by *stype*.

Examples

```
#include <spdm.h>

stat_list_t   sanchor;
ptpe_data_id  stype;
char          sname[PTPE_STNL];
long          ldata;
float         fdata;
int           rc;

rc = PtpeFirstStat(sanchor);
for (;;) {
      rc = PtpeGetStatName(sname, sanchor);
      rc = PtpeGetStatType(&stype, sanchor);
      if (rc != PTPE_SUCCESS) {
          /* handle error */
      }
      else {
           if (rc == PTPE_LONG) {
           rc = PtpeGetStatValueLong(
                &ldata, sanchor);
           }
           if (rc == PTPE_FLOAT) {
           rc = PtpeGetStatValueFloat(
                &fdata, sanchor);
           }
      }
      rc = PtpeIsLastStat(sanchor);
      if (rc == PTPE_TRUE) {
          break;
      }
      rc = PtpeNextStat(sanchor);
}
```

Related Information

- “PtpeInitStatList” on page 267
- “PtpeFirstStat” on page 239
- “PtpeNextStat” on page 275
- “PtpeFindStat” on page 235
- “PtpeAddStatToList” on page 116
- “PtpeDelStatFromList” on page 227
- “PtpeRemoveStatsFromHost” on page 298
- “PtpeGetStatType” on page 258
- “PtpeGetStatName” on page 252
- “PtpeGetStatValueLong” on page 263
- “PtpeGetStatValueFloat” on page 261
- “PtpeGetStatResult” on page 254
- “PtpeStatIsLong” on page 305
- “PtpeStatIsFloat” on page 303
- “PtpeSetStatTime” on page 300

PtpeGetStatValueFloat

Purpose

Retrieves the value stored in the currently referenced entry in a statistics list.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeGetStatValueFloat(sval, sanchor)
stat_list_t   sanchor;
float         *sval;
```

Parameters

sanchor
The anchor point for a non-empty statistics list.

sval
The address of a float. Upon successful completion of this subroutine, the value stored for the currently referenced entry in *sanchor* will be stored at this location.

Description

PtpeGetStatValueFloat is used by an application to obtain the value stored for the "current" entry in the statistics list anchored by *sanchor*. When this subroutine is performed upon a valid, non-empty statistics list, the value stored in the "current" entry is copied to the memory location referenced by the *sval* parameter.

This subroutine returns a value in floating point data format, and should be used only when the "current" entry is of the data type PTPE_FLOAT. The API subroutines **PtpeGetStatType** and **PtpeStatIsFloat** should be used to verify that the statistic is of the proper type before issuing this subroutine. This subroutine will not report an error if it is issued upon a statistic of another type, but will cast the value found in the "current" entry to a float data type.

When a statistic entry is created by the **PtpeAddStatToList** subroutine, the value for the statistic is initialized to zero. If the statistic value has not been set by the **PtpeSetStatValue (MISSING THIS ROUTINE)** subroutine, or the statistic value was not obtained by an API subroutine (like the **PtpeArchGetStats** subroutine), a value of zero will be obtained. It will be impossible to determine whether this value contains the initialization value, or whether the value of the statistic is truly zero, so caution must be used when issuing this subroutine and analyzing its results.

An error results if this subroutine is performed on an empty or an uninitialized statistics list.

Return Codes

Upon successful completion, this subroutine returns the value of PTPE_SUCCESS, and the value of the “current” statistic list entry is placed in the memory location referenced by *sval*. If an error has occurred, the return code is set to one of the following values to indicate the cause of the error:

PTPE_INV_STATLIST	*sanchor* has a NULL value.
PTPE_INV_PTR	*sval* has a NULL value.
PTPE_EMPTY	*sanchor* is an empty host list.
PTPE_BAD_LOC_PTR	The memory pointers in *sanchor* have been corrupted.
PTPE_MEMORY	An error occurred while writing the value of the statistic to the memory location referenced by *sval*.

Examples

```
#include <spdm.h>

stat_list_t   sanchor;
float         fdata;
int           rc;

rc = PtpeStatIsFloat(sanchor);
if (rc == PTPE_TRUE) {
     rc = PtpeGetStatValueFloat(&fdata,
          sanchor);
     if (rc != PTPE_SUCCESS) {
         /* handle error */
     }
     else {
           printf("Value is %10.2f\n", fdata);
     }
}
```

Related Information

- “PtpeInitStatList” on page 267
- “PtpeFirstStat” on page 239
- “PtpeNextStat” on page 275
- “PtpeFindStat” on page 235
- “PtpeAddStatToList” on page 116
- “PtpeDelStatFromList” on page 227
- “PtpeStatIsLong” on page 305
- “PtpeStatIsFloat” on page 303
- “PtpeGetStatType” on page 258
- “PtpeGetStatName” on page 252
- “PtpeGetHostStatList” on page 250
- “PtpeGetStatValueLong” on page 263

PtpeGetStatValueLong

Purpose

Retrieves the value stored in the currently referenced entry in a statistics list.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeGetStatValueLong(sval, sanchor)
stat_list_t    sanchor;
long           *sval;
```

Parameters

sanchor
The anchor point for a non-empty statistics list.

sval
The address of a long integer. Upon successful completion of this subroutine, the value stored for the currently referenced entry in *sanchor* will be stored at this location.

Description

PtpeGetStatValueLong is used by an application to obtain the value stored for the “current” entry in the statistics list anchored by *sanchor*. When this subroutine is performed upon a valid, non-empty statistics list, the value stored in the “current” entry is copied to the memory location referenced by the *sval* parameter.

This subroutine returns a value in long integer format, and should be used only when the “current” entry is of the data type PTPE_LONG. The API subroutines **PtpeGetStatType** and **PtpeStatIsLong** should be used to verify that the statistic is of the proper type before issuing this subroutine. This subroutine will not report an error if it is issued upon a statistic of another type, but will cast the value found in the “current” entry to a long data type.

When a statistic entry is created by the **PtpeAddStatToList** subroutine, the value for the statistic is initialized to zero. If the statistic value has not been set by the **PtpeSetStatValue (MISSING THIS ROUTINE)** subroutine, or the statistic value was not obtained by an API subroutine (like the **PtpeArchGetStats** subroutine), a value of zero will be obtained. It will be impossible to determine whether this value contains the initialization value, or whether the value of the statistic is truly zero, so caution must be used when issuing this subroutine and analyzing its results.

An error results if this subroutine is performed on an empty or an uninitialized statistics list.

Return Codes

Upon successful completion, this subroutine returns the value of PTPE_SUCCESS, and the value of the “current” statistic list entry is placed in the memory location referenced by *sval*. If an error has occurred, the return code is set to one of the following values to indicate the cause of the error:

PTPE_INV_STATLIST	*sanchor* has a NULL value.
PTPE_INV_PTR	*sval* has a NULL value.
PTPE_EMPTY	*sanchor* is an empty host list.
PTPE_BAD_LOC_PTR	The memory pointers in *sanchor* have been corrupted.
PTPE_MEMORY	An error occurred while writing the value of the statistic to the memory location referenced by *sval*.

Examples

```
#include <spdm.h>

stat_list_t    sanchor;
long           ldata;
int            rc;

rc = PtpeStatIsLong(sanchor);
if (rc == PTPE_TRUE) {
     rc = PtpeGetStatValueLong(&ldata,
          sanchor);
     if (rc != PTPE_SUCCESS) {
         /* handle error */
     }
     else {
          printf("Value is %ld\n", ldata);
     }
}
```

Related Information

- “PtpeInitStatList” on page 267
- “PtpeFirstStat” on page 239
- “PtpeNextStat” on page 275
- “PtpeFindStat” on page 235
- “PtpeAddStatToList” on page 116
- “PtpeDelStatFromList” on page 227
- “PtpeStatIsLong” on page 305
- “PtpeStatIsFloat” on page 303
- “PtpeGetStatType” on page 258
- “PtpeGetStatName” on page 252
- “PtpeGetHostStatList” on page 250
- “PtpeGetStatValueLong” on page 263
- “PtpeGetStatValueFloat” on page 261

PtpeInitHostList

Purpose

Allocates an anchor point for a host list, and sets the anchor point initial values to indicate an empty list.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeInitHostList(hanchor)
host_list_t    *hanchor;
```

Parameters

sanchor
: Points to the host list anchor point at the completion of the subroutine. This parameter must point to a NULL anchor point before this subroutine is called.

Description

PtpeInitHostList allocates a host list anchor point, and sets the anchor to indicate that the list is empty. This subroutine must be called before a host list can be used by other PTPE library subroutines that manipulate host lists.

Return Codes

Upon successful completion, this subroutine returns the value of PTPE_SUCCESS. If an error has occurred, the return code is set to one of the following values to indicate the cause of the error:

PTPE_NO_MEMORY	An anchor point could not be allocated
PTPE_INV_PTR	The *hanchor* parameter is NULL, or the memory at the location indicated by *sanchor* is not NULL

Examples

```
#include <spdm.h>
host_list_t    myhosts;
int            rc;

myhosts = (host_list_t) NULL;
rc = PtpeInitHostList(&myhosts);
if (rc != PTPE_SUCCESS) {
/* handle error */
}
```

Related Information

- “PtpeEmptyHostList” on page 229
- “PtpeFreeHostList” on page 241
- “PtpeAddHostToList” on page 114
- “PtpeQueryAvailHosts” on page 284

PtpeInitStatList

Purpose

Allocates an anchor point for a statistics list, and sets the anchor point initial values to indicate an empty list.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeInitStatList(sanchor)
stat_list_t *sanchor;
```

Parameters

sanchor
Points to the statistic list anchor point at the completion of the subroutine. This parameter must point to a NULL anchor point before this subroutine is called.

Description

PtpeInitStatList allocates a statistics list anchor point, and sets the anchor to indicate that the list is empty. This subroutine must be called before a statistics list can be used by other PTPE library subroutines that manipulate statistics lists.

A statistics list allocated by this subroutine is said to be “unassigned.” In other words, the list has not been attached or "assigned" to a system at this time.

Return Codes

Upon successful completion, this subroutine returns the value of PTPE_SUCCESS. If an error has occurred, the return code is set to one of the following values to indicate the cause of the error:

PTPE_NO_MEMORY	An anchor point could not be allocated
PTPE_INV_PTR	The *sanchor* parameter is NULL, or the memory at the location indicated by *hanchor* is not NULL

Examples

```
#include <spdm.h>
stat_list_t    mystats;
int            rc;

mystats = (host_list_t) NULL;
rc = PtpeInitStatList(&mystats);
if (rc != PTPE_SUCCESS) {
/* handle error */
}
```

Related Information

- “PtpeEmptyStatList” on page 231
- “PtpeFreeStatList” on page 243
- “PtpeAddStatToList” on page 116
- “PtpeDelStatFromList” on page 227
- “PtpeAssignStatsToHost” on page 170

PtpeIsLastHost

Purpose

Determines if the host list is currently positioned at the last entry of the list.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeIsLastHost(hanchor)
host_list_t    hanchor;
```

Parameters

hanchor
The anchor point of an existing, non-empty host list.

Description

PtpeIsLastHost determines if the "current" element in the host list *hanchor* is the last entry in the host list. In cases where this subroutine is performed on a valid, non-empty host list, the subroutine provides a true or false answer.

An error results if this subroutine is performed on an empty host list.

Return Codes

Upon successful completion, this subroutine returns one of the following values:

PTPE_TRUE	The "current" entry is the last entry in the host list.
PTPE_FALSE	The "current" entry is not the last entry in the host list.

If an error has occurred, the return code is set to one of the following values to indicate the cause of the error:

PTPE_INV_HOSTLIST	*hanchor* has a NULL value.
PTPE_EMPTY	*hanchor* is an empty host list.
PTPE_BAD_LOC_PTR	The memory pointers in *hanchor* have been corrupted.

Examples

```
#include <spdm.h>

host_list_t    hanchor;
int            rc;

cp 14
for (;;) {
     rc = PtpeIsLastHost(hanchor);
     if (rc!=PTPE_TRUE || rc!=PTPE_FALSE){
          /* handle error */
     }
     else {
          if (rc == PTPE_TRUE) {
              break;
          }
         /* do normal processing */
         rc = PtpeNextHost(hanchor);
     }
}
```

Related Information

- "PtpeInitHostList" on page 265
- "PtpeAddHostToList" on page 114
- "PtpeDelHostFromList" on page 225
- "PtpeFirstHost" on page 237
- "PtpeFindHost" on page 233
- "PtpeNextHost" on page 273

PtpeIsLastStat

Purpose

Determines if the statistics list is currently positioned at the last entry of the list.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeIsLastStat(sanchor)
stat_list_t   sanchor;
```

Parameters

sanchor
The anchor point of an existing, non-empty statistics list.

Description

PtpeIsLastStat determines if the “current” element in the statistics list *sanchor* is the last entry in the statistics list. In cases where this subroutine is performed on a valid, non-empty statistics list, the subroutine provides a true or false answer.

An error results if this subroutine is performed on an empty statistics list.

If the statistics list was previously assigned to a system by the **PtpeAssignStatsToHost** subroutine, the test performed b subroutine occurs in the “unassigned” statistics list *sanchor* only; the copy of the statistics list that was previously assigned to a system remains unchanged by this subroutine.

Return Codes

Upon successful completion, this subroutine returns one of the following values:

PTPE_TRUE	The “current” entry is the last entry in the statistics list.
PTPE_FALSE	The “current” entry is not the last entry in the statistics list.

If an error has occurred, the return code is set to one o the following values to indicate the cause of the error:

PTPE_INV_STATLIST	*sanchor* has a NULL value.
PTPE_EMPTY	*sanchor* is an empty statistics list.
PTPE_BAD_LOC_PTR	The memory pointers in *sanchor* have been corrupted.

Examples

```
#include <spdm.h>

stat_list_t    sanchor;
int            rc;

for (;;) {
     rc = PtpeIsLastStat(sanchor);
     if (rc!=PTPE_TRUE || rc!=PTPE_FALSE){
         /* handle error */
   }
   else {
        if (rc == PTPE_TRUE) {
             break;
        }
        /* do normal processing */
        rc = PtpeNextStat(sanchor);
        }
}
```

Related Information

- “PtpeInitStatList” on page 267
- “PtpeAddStatToList” on page 116
- “PtpeDelStatFromList” on page 227
- “PtpeAssignStatsToHost” on page 170
- “PtpeNextStat” on page 275
- “PtpeGetHostStatList” on page 250
- “PtpeFindStat” on page 235
- “PtpeFirstStat” on page 239

PtpeNextHost

Purpose

Positions the host list anchor pointers for operations upon the entry following the current entry in the host list.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeNextHost(hanchor)
host_list_t    hanchor;
```

Parameters

hanchor
The anchor point of an existing, non-empty host list.

Description

PtpeNextHost updates the pointers in the host list anchor *hanchor* to point to the entry in the host list immediately following the entry marked as “current.” This makes the next entry in the list the new “current” entry. Any subsequent API subroutines that perform operation upon the “current” host list entry will be performed upon the this entry in the host list. This behavior continues until the “current” entry pointers are reset using by another API subroutine.

If the “current” pointers in the host list anchor point to the last entry in the host list prior to the call to this subroutine, the pointers are not modified. An error results if this subroutine is used on an empty host list.

Return Codes

Upon successful completion, this subroutine returns the value of PTPE_SUCCESS. If an error has occurred, the return code is set to one of the following values to indicate the cause of the error:

PTPE_INV_HOSTLIST	*hanchor* has a NULL value.
PTPE_LIST_END	The internal pointers are already positioned at the last entry in the host list.
PTPE_EMPTY	*hanchor* is an empty host list.
PTPE_BAD_LOC_PTR	The memory pointers in *hanchor* have been corrupted.

Examples

```
#include <spdm.h>

host_list_t    hanchor;
int            rc;

for(;;) {
     rc = PtpeNextHost(hanchor);
     if (rc == PTPE_LIST_END) {
          break;
     }
     if (rc != PTPE_SUCCESS) {
         /* handle error */
     }
     /* do normal operation on list entry */
}
```

Related Information

- “PtpeInitHostList” on page 265
- “PtpeAddHostToList” on page 114
- “PtpeDelHostFromList” on page 225
- “PtpeFirstHost” on page 237
- “PtpeFindHost” on page 233
- “PtpeIsLastHost” on page 269

PtpeNextStat

Purpose

Positions the statistics list anchor pointers for operations upon the entry following the current entry in the statistics list.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeNextStat(sanchor)
stat_list_t    sanchor;
```

Parameters

sanchor
The anchor point of an existing, non-empty statistics list.

Description

PtpeNextStat updates the pointers in the statistics list anchor *sanchor* to point to the entry in the statistics list immediately following the entry marked as “current.” This makes the next entry in the list the new “current” entry. Any subsequent API subroutines that perform operation upon the “current” statistics list entry will be performed upon the this entry in the statistics list. This behavior continues until the “current” entry pointers are reset using by another API subroutine.

If the “current” pointers in the statistics list anchor point to the last entry in the statistics list prior to the call to this subroutine, the pointers are not modified. An error results if this subroutine is used on an empty statistics list.

If the statistics list was previously assigned to a system by the **PtpeAssignStatsToHost** subroutine, the positioning performed by this subroutine occurs in the “unassigned” statistics list *sanchor* only; the copy of the statistics list that was previously assigned to a system remains unchanged by this subroutine.

Return Codes

Upon successful completion, this subroutine returns the value of PTPE_SUCCESS. If an error has occurred, the return code is set to one of the following values to indicate the cause of the error:

PTPE_INV_STATLIST	*sanchor* has a NULL value.
PTPE_LIST_END	The internal pointers are already positioned at the last entry in the statistics list.
PTPE_EMPTY	*sanchor* is an empty statistics list.
PTPE_BAD_LOC_PTR	The memory pointers in *sanchor* have been corrupted.

Examples

```
#include <spdm.h>

stat_list_t   sanchor;
int           rc;

for(;;) {
      rc = PtpeNextStat(sanchor);
      if (rc == PTPE_LIST_END) {
          break;
      }
      if (rc != PTPE_SUCCESS) {
      /* handle error */
      }
      /* do normal operation on list entry */
}
```

Related Information

- “PtpeInitStatList” on page 267
- “PtpeAddStatToList” on page 116
- “PtpeDelStatFromList” on page 227
- “PtpeAssignStatsToHost” on page 170
- “PtpeGetHostStatList” on page 250
- “PtpeIsLastStat” on page 271
- “PtpeFindStat” on page 235
- “PtpeFirstStat” on page 239

PtpeOpenSession

Purpose

Establishes a session with the Performance Toolbox Parallel Extensions.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeOpenSession(sblock)
session_ptr_t   *sblock;
```

Parameters

sblock
: A pointer to a session_ptr_t data type, which should be set to a NULL value before invoking this subroutine. Upon successful completion of the subroutine, a session control information block will be placed at this location.

Description

PtpeOpenSession establishes a session between the application and the Performance Toolbox Parallel Extensions. The session permits the application to control performance information collection, summarization, and archiving in the PTPE monitoring hierarchy. The session also permits an application to obtain current or archived performance information from the systems in the monitoring hierarchy.

When a session is established, PTPE grants sole control of the monitoring hierarchy to the application for the duration of the session. This means that other commands, such as **ptpehier** or **ptpectrl**, cannot modify the hierarchy structure, or change the current status of performance information collection and archiving. Other PTPE API applications are also restricted from controlling performance information collection and archiving while the session is held. If multiple PTPE API applications must execute in parallel, the applications should coordinate their use of sessions, ensuring that one application releases the session for the other application(s) to acquire. Applications must recognize that no other application will be able to acquire a session while a session is held by another application, and should be coded to handle this eventuality.

Only users that are members of the **perfmon** user group can establish a session with PTPE. The user of the application must be a member of the **perfmon** user group, and have set that group to be their effective user group for this subroutine to establish a session.

A session cannot be established if another application holds a session, or if the application attempts to acquire a session while either the **ptpectrl** or **ptpehier** commands are executing. A session also cannot be established if the **ptpeconf** command was never executed to create the necessary PTPE data classes in the System Data Repository.

Return Codes

Upon successful completion, a return code of PTPE_SUCCESS is returned to the caller. If an error occurred, one of the following return codes is provided:

PTPE_LOCKED	Another application currently holds a PTPE API session. This application must wait for the other application to release the session.
PTPE_OVERWRITE	The application already has an active session.
PTPE_AUTH	User is not a member of the **perfmon** user group, or has not made the **perfmon** group the effective group for this application.
PTPE_INV_PTR	*sblock* is not a NULL value.
PTPE_NO_MEMORY	Could not allocate enough memory to initiate the session.
PTPE_MEMORY	An internal error occurred.
PTPE_NO_LOCK_OBJ	The PTPE data classes do not exist in the System Data Repository. The classes may not have been created, or the application may be running in an incorrect system partition.
PTPE_SDR	An unexpected error occurred while obtaining information from the PTPE data classes in the System Data Repository. These data classes might not exist, or the System Data Repository might be experiencing difficulties.

Examples

```
#include <spdm.h>

session_ptr_t   sblock;
int             rc;

sblock = (session_ptr_t) NULL;
rc = PtpeOpenSession(&sblock);
switch(rc) {
case PTPE_SUCCESS:
     break;
case PTPE_OVERWRITE:
     /* already had a session */
     break;
case PTPE_LOCKED:
     /* wait for session to become free */
     /* then retry the session          */
default:
     /* handle error condition *.
}
```

Related Information

- “PtpeCloseSession” on page 176
- “ptpectrl” on page 88
- “ptpehier” on page 99
- “ptpeconf” on page 86
- “ptpegroup” on page 97
- The AIX **newgrp** command

PtpeQueryAllHostStatus

Purpose

Reports the status of performance information collection and archiving on all systems in the Performance Toolbox Parallel Extensions monitoring hierarchy.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeQueryAllHostStatus(sblock, reply)
session_ptr_t   sblock;
host_list_t     *reply;
```

Parameters

sblock
: The address of a session control block. This information block is created and initialized by the **PtpeOpenSession** subroutine.

reply
: A pointer to a host_list_t data type, which should be set to a NULL value before invoking this subroutine. Upon successful completion of the subroutine, an anchor point for a host list will be placed at this location. This list will contain entries for each system in the monitoring hierarchy. Each entry will contain a result code to indicate whether or not the system was successful in carrying out the request.

Description

PtpeQueryAllHostStatus returns the current status of performance information collection and archiving on all systems in the Performance Toolbox Parallel Extensions monitoring hierarchy. This call is similar to the **PtpeQueryHostStatus** subroutine, except that this subroutine automatically queries all systems in the monitoring hierarchy.

The application must first establish a PTPE API session by using the **PtpeOpenSession** subroutine.

When **PtpeQueryAllHostStatus** is executed, the subroutine relays the query to the monitoring hierarchy's central coordinator node. The central coordinator node forwards the query to all data manager nodes in the hierarchy, which then relay the query to those systems in their reporting groups.

Each system in the monitoring hierarchy replies with a code that indicates its current collection and archiving status. This code is constructed by ORing together the following values:

PTPE_SAMPLE	The system is currently supplying performance information (set by all systems whenever performance information collection is active)

PTPE_COLLECT	The system is currently supplying summary performance information (set by managing systems when performance information collection is active)
PTPE_ARCHIVE	The system is currently recording performance information to the archive

If collection or archiving is not active on the system, the system replies with the following code:

PTPE_INACTIVE	The system is not currently supplying performance information or archiving information.

The system then indicates its success or failure in the effort to its data manager node.

Data manager nodes construct partial host lists, with an entry for each system in its reporting group that was targeted for the command. The system's status or failure reply is placed in the partial host list. This list is then relayed to the central coordinator node, who constructs a complete host list. The complete host list is transmitted back to the **PtpeQueryHostStatus** subroutine, which provides it to the application in the *reply* parameter.

PtpeQueryHostStatus will provide an indication of the overall success or failure of the query in the return code:

PTPE_SUCCESS	All systems in the *targets* list successfully responded to the query.
PTPE_LIMITED	Some systems in the *targets* list were unable to respond successfully to the query.
PTPE_API_FAILED	All systems in the *targets* list were unable to respond successfully to the query.

To determine the status of a system, and to determine which systems were not successful, the *reply* list should be scanned, and the results for each system in the list checked using the **PtpeGetHostResult** subroutine. A negative value for a result indicates an error. A value of zero or greater reflects the current status of the system. In non-error cases, the result should be ANDed with PTPE_SAMPLE, PTPE_COLLECT, or PTPE_ARCHIVE to determine the system's current status. This subroutine cannot be issued from a PTPE read-only session.

Return Codes

Upon successful completion, a return code of PTPE_SUCCESS is returned to the caller. If an error occurred, one of the following return codes is provided:

PTPE_SUCCESS_BADR	The central coordinator node has indicated that all systems in the *targets* list responded successfully to the query, but an error occurred in reading the *reply* list response from the central coordinator Node.
PTPE_LIMITED	Some of the systems in the *targets* list were unable to reply to the query. The list *reply* points to contains all systems involved in the command; the application can determine which systems failed by performing the

	PtpeGetHostResult subroutine on all systems in the *reply* list.
PTPE_LIMITED_BADR	Some systems in the *targets* list were unable to respond to the query, and an error occurred in reading the *reply* list response from the central coordinator node.
PTPE_API_FAILED	The central coordinator node has indicated that all systems in the *targets* list failed to respond to the query. *reply* contains all systems involved in the query, along with the reasons for the failure on these systems. The application can determine the cause of the error by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_API_FAILED_BADR	The central coordinator node has indicated that all systems failed to respond to the query, and an error occurred in reading the *reply* list from the central coordinator node.
PTPE_NO_CONNECT	Could not establish a network connection to the central coordinator node. *reply* list not modified.
PTPE_BAD_SEND	Could not successfully transmit the command to the central coordinator node. *reply* is not modified.
PTPE_BAD_RECEIVE	Error in receiving the reply to the command from the central coordinator node. It is impossible to determine if the query was successful. *reply* is not modified.
PTPE_RONLY_SESS	The application attempted to issue this subroutine from a PTPE read-only session. A regular session must be established to issue this instruction. Ensure that the application is executing in the proper SP system partition.
PTPE_TIMEOUT	A reply to the command was not received from the central coordinator node in the time allowed. reply is not modified.
PTPE_INV_HIERARCHY	The monitoring hierarchy contains a node that is not a member of the currently active system partition. The hierarchy must be repaired by removing any nodes that are not members of the active system partition before this subroutine can be used.
PTPE_INV_PTR	*sblock* has a NULL value, or *reply* is not NULL.
PTPE_NO_SESSION	A PTPE API session was not previously established by this application (see “PtpeOpenSession” on page 277)
PTPE_NO_HIERARCHY	A monitoring hierarchy does not exist. This subroutine cannot function without a monitoring hierarchy (see “ptpehier” on page 99)
PTPE_NO_MEMORY	Could not allocate enough memory to initiate the query. *reply* is not modified.
PTPE_MEMORY	An internal error occurred.

PTPE_BAD_LOC_PTR	The internal pointers in the *targets* anchor have been corrupted.
PTPE_SDR	An unexpected error occurred while obtaining information from the PTPE data classes in the System Data Repository. These data classes might not exist, or the System Data Repository might be experiencing difficulties.

Examples

```
#include <spdm.h>

session_ptr_t   sblock;
host_list_t     reply;
int             rc;

/* set up session and "targets" earlier */
reply = (host_list_t) NULL;
rc = PtpeQueryAllHostStatus(sblock, &reply);
if (rc != PTPE_SUCCESS) {
     printf("Not all systems succeded.\n");
}
/* loop through "reply" and check results */
/* on each system with PtpeGetHostResult  */
```

Related Information

- "PtpeOpenSession" on page 277
- "PtpeColQueryState" on page 203
- "PtpeArchQueryState" on page 144
- "PtpeQueryHostStatus" on page 293
- "PtpeColStart" on page 215
- "PtpeColStop" on page 220
- "PtpeArchStartHosts" on page 156
- "PtpeArchStartAllHosts" on page 152
- "PtpeArchStopHosts" on page 165
- "PtpeArchStopAllHosts" on page 161
- "PtpeInitHostList" on page 265
- "PtpeAddHostToList" on page 114
- "PtpeFirstHost" on page 237
- "PtpeNextHost" on page 273
- "PtpeFindHost" on page 233
- "PtpeFreeHostList" on page 241
- "PtpeGetHost" on page 245
- "PtpeGetHostResult" on page 247
- "ptpectrl" on page 88

PtpeQueryAvailHosts

Purpose

Retrieves the list of systems in the Performance Toolbox Parallel Extensions monitoring hierarchy.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeQueryAvailHosts(sblock, hanchor)
session_ptr_t        sblock;
host_list_t          hanchor;
```

Parameters

sblock
: Points to a PTPE API session control information block. This block is created and initialized using the **PtpeOpenSession** subroutine.

hanchor
: An anchor point for an empty host list. The host list is created by the **PtpeInitHostList** subroutine.

Description

PtpeQueryAvailHosts retrieves the monitoring hierarchy currently in use by the Performance Toolbox Parallel Extensions. A host list is constructed, containing one entry for each host list in the monitoring hierarchy. The host list is then attached to the anchor point provided by the caller. This host list can be used in later PTPE API subroutines to send instructions to the systems in the monitoring hierarchy.

The host list anchor must reference an empty host list, or this subroutine returns with an error. This subroutine cannot be issued from a PTPE read-only session.

Return Codes

Upon successful completion, this subroutine returns the value of PTPE_SUCCESS, and *hanchor* will anchor a host list containing entries for all systems in the monitoring hierarchy. If an error has occurred, the return code is set to one of the following values to indicate the cause of the error:

PTPE_NO_SESSION	A session was not previously established using the **PtpeOpenSession** subroutine.
PTPE_INV_HIERARCHY	The monitoring hierarchy contains a node that is not a member of the currently active system partition. The hierarchy must be repaired by removing any nodes that are not members of the active system partition before this subroutine can be used.
PTPE_INV_HOSTLIST	*hanchor* does not reference an empty host list.

PTPE_INV_PTR	*sblock* or *hanchor* is a NULL value.
PTPE_SDR	An unexpected error occurred while the subroutine had access to the System Data Repository.
PTPE_MEMORY	An internal memory usage error occurred.
PTPE_RONLY_SESS	The application attempted to issue this subroutine from a PTPE read-only session. A regular session must be established to issue this instruction. Ensure that the application is executing in the proper SP system partition.

Examples

```
#include <spdm.h>

session_ptr_t   sblock;
host_list_t     hanchor;
int             rc;

rc = PtpeQueryAvailHosts(sblock, hanchor);
if (rc != PTPE_SUCCESS) {
/* handle error */
}
```

Related Information

- "PtpeOpenSession" on page 277
- "PtpeInitHostList" on page 265
- "PtpeEmptyHostList" on page 229

PtpeQueryAvailStats

Purpose

Retrieves the list of the statistics available for collection, summarization, or archival on one or more systems in the monitoring hierarchy.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeQueryAvailStats(sblock, targets, reply)
session_ptr_t   sblock;
host_list_t     targets;
host_list_t     *reply;
```

Parameters

sblock
The address of a session control block. This information block is created and initialized by the **PtpeOpenSession** subroutine.

targets
The anchor of a host list that contains at least one entry. This list should not contain any systems that have statistics assigned to them.

reply
A pointer to a host_list_t data type, which should be set to a NULL value before invoking this subroutine. Upon successful completion of the subroutine, an anchor point for a host list will be placed at this location. This list will contain entries for each system in the *targets* list. A statistics list will be assigned to all systems that respond successfully to the query, and will contain the list of statistics available on those systems for collection, summarization, and archival.

Description

PtpeQueryAvailStats asks the Performance Toolbox Parallel Extensions to provide the list of statistics available for collection, summarization, and archival by one or more systems in the monitoring hierarchy.

The application must first establish a PTPE API session by using the **PtpeOpenSession** subroutine. The application must also construct a host list for the query, containing an entry for each system that the application needs to query. This list is then provided in the *targets* parameter.

When **PtpeQueryAvailStats** is executed, the subroutine relays the query command, along with the host list of systems to query, to the monitoring hierarchy's central coordinator node. The central coordinator node determines which nodes are data managers for the systems in the *targets* list, and forwards the query to only those data manager nodes that manage systems in the query list. The data manager nodes then relay the query to those systems in their reporting groups that have been listed in the targets list.

Each system in the *targets* list constructs a statistics list, containing one entry for each statistics that system can provide to Performance Toolbox Parallel Extensions for collection, summarization, or archival. On systems that also double as data manager nodes, this list will also include any summary statistics that the node prepares. This list is then relayed back to the data manager node.

Data manager nodes construct partial host lists, with an entry for each system in its reporting group that was targeted for the query. The statistics lists received from each node are assigned to their respective host list entries in the data manager node's partial host list. This list is then relayed to the central coordinator node, who constructs a complete host list. The complete host list is transmitted back to the **PtpeQueryAvailStats** subroutine, which provides it to the application in the *reply* parameter.

PtpeQueryAvailStats will provide an indication of the overall success or failure of the query in the return code:

PTPE_SUCCESS	All systems in the *targets* list successfully responded to the query.
PTPE_LIMITED	Some systems in the *targets* list were unable to respond successfully to the query.
PTPE_API_FAILED	All systems in the *targets* list were unable to respond successfully to the query.

To determine which systems were not successful, the reply list should be scanned, and the results for each system in the list checked using the **PtpeGetHostResult** subroutine. The result code will provide an indication of the error for the system. To determine what statistics are available on those systems that successfully responded to the query, extract the statistics list for each entry by using the **PtpeGetHostStatList** subroutine, and scan the list using the statistics list scanning subroutines.

Return Codes

Upon successful completion, a return code of PTPE_SUCCESS is returned to the caller. If an error occurred, one of the following return codes is provided:

PTPE_SUCCESS_BADR	The central coordinator node has indicated that all systems in the *targets* list responded successfully to the query, but an error occurred in reading the *reply* list response from the central coordinator Node. The application should treat this as an error, since the contents of the *reply* list cannot be guaranteed to be accurate.
PTPE_LIMITED	Some of the systems in the *targets* list were unable to reply to the query. The list *reply* points to contains all systems involved in the query; the application can determine which systems failed by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_LIMITED_BADR	Some systems in the *targets* list were unable to respond to the query, and an error occurred in reading the *reply* list response from the central coordinator node. The application should treat this as an error,

	since the contents of the *reply* list cannot be guaranteed to be accurate.
PTPE_API_FAILED	The central coordinator node has indicated that all systems in the *targets* list failed to respond to the query. *reply* contains all systems involved in the query, along with the reasons for the failure on these systems. The application can determine the cause of the error by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_API_FAILED_BADR	The central coordinator node has indicated that all systems failed to respond to the query, and an error occurred in reading the *reply* list from the central coordinator node. The application should treat this as an error, since the contents of the *reply* list cannot be guaranteed.
PTPE_STATE	Performance information collection and summarization is not currently active in the monitoring hierarchy.
PTPE_NO_CONNECT	Could not establish a network connection to the central coordinator node. *reply* list not modified.
PTPE_BAD_SEND	Could not successfully transmit the query to the central coordinator node. *reply* is not modified.
PTPE_BAD_RECEIVE	Error in receiving the reply to the query from the central coordinator node. It is impossible to determine if the query was successful. *reply* is not modified.
PTPE_TIMEOUT	A reply to the query was not received from the central coordinator node in the time allowed. reply is not modified.
PTPE_INV_HIERARCHY	The monitoring hierarchy contains a node that is not a member of the currently active system partition. The hierarchy must be repaired by removing any nodes that are not members of the active system partition before this subroutine can be used.
PTPE_INV_HOSTLIST	*targets* either contains a NULL value, or does not anchor a valid host list.
PTPE_INV_PTR	*sblock* has a NULL value, or *reply* is not NULL.
PTPE_EMPTY	*targets* is an empty host list.
PTPE_HOST_NOT_FOUND	One or more systems listed in *targets* could not be found in the monitoring hierarchy. *reply* contains the list of systems that could not be found in the hierarchy.
PTPE_NO_SESSION	A PTPE API session was not previously established by this application (see “PtpeOpenSession” on page 277).
PTPE_NO_HIERARCHY	A monitoring hierarchy does not exist. This subroutine cannot function without a monitoring hierarchy (see “ptpehier” on page 99).

PTPE_NO_MEMORY	Could not allocate enough memory to initiate the query. *reply* is not modified.
PTPE_MEMORY	An internal error occurred.
PTPE_BAD_LOC_PTR	The internal pointers in the *targets* anchor have been corrupted.
PTPE_SDR	An unexpected error occurred while obtaining information from the PTPE data classes in the System Data Repository. These data classes might not exist, or the System Data Repository might be experiencing difficulties.

Examples

```
#include <spdm.h>

session_ptr_t   sblock;
host_list_t     targets, reply;
int             rc;

/* set up session and "targets" earlier */
reply = (host_list_t) NULL;
rc = PtpeQueryAvailStats(sblock, targets,
                         &reply);
if (rc != PTPE_SUCCESS) {
     printf("Not all systems succeded.\n");
}
/* loop through "reply" and check results */
/* on each system with PtpeGetHostResult  */
```

Related Information

- “PtpeOpenSession” on page 277
- “PtpeInitHostList” on page 265
- “PtpeAddHostToList” on page 114
- “PtpeFirstHost” on page 237
- “PtpeNextHost” on page 273
- “PtpeFindHost” on page 233
- “PtpeFreeHostList” on page 241
- “PtpeGetHost” on page 245
- “PtpeGetHostResult” on page 247
- “PtpeInitStatList” on page 267
- “PtpeAddStatToList” on page 116
- “PtpeFirstStat” on page 239
- “PtpeNextStat” on page 275
- “PtpeFindStat” on page 235
- “PtpeFreeStatList” on page 243
- “PtpeAssignStatsToHost” on page 170

- “PtpeGetHostStatList” on page 250
- “PtpeGetStatName” on page 252
- “PtpeGetStatResult” on page 254

PtpeQueryHostRates

Purpose

Reports the current time intervals used by the monitoring hierarchy for sampling and recording of performance information.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeQueryHostRates(sblock, srate, arate)
session_ptr_t   sblock;
int             *srate;
int             *arate;
```

Parameters

sblock
The address of a session control block. This information block is created and initialized by the **PtpeOpenSession** subroutine.

srate
A pointer to an integer. Upon successful completion of this routine, the current time between samples of performance information used by the monitoring hierarchy will be stored at this location. The time is expressed in units of seconds.

arate
A pointer to an integer. Upon successful completion of this routine, the current time between recording of performance information to the archive used by the monitoring hierarchy will be stored at this location. The time is expressed in units of seconds.

Description

PtpeQueryHostRates obtains the current time intervals between performance information samples and recordings in the monitoring hierarchy, after collection and archiving have been started. Both intervals are expressed in units of seconds. The recording interval should always be an even multiple of the data sampling rate.

All systems in the performance monitoring hierarchy use the same sampling and recording rates.

Return Codes

Upon successful completion, this subroutine returns the value of PTPE_SUCCESS. If an error has occurred, the return code is set to one of the following values to indicate the cause of the error:

PTPE_INV_PTR	Any values for *sblock*, *srate*, or *arate* are NULL.

PTPE_NO_SESSION	A PTPE API session was not established by this application (see “PtpeOpenSession” on page 277)
PTPE_NO_LOCK_OBJ	The PTPE data classes do not exist in the System Data Repository (see “ptpeconf” on page 86).
PTPE_NO_HIERARCHY	A monitoring hierarchy does not exist. This subroutine cannot function without a monitoring hierarchy (see “ptpehier” on page 99)
PTPE_NO_MEMORY	Could not allocate enough memory to initiate the query. *reply* is not modified.
PTPE_MEMORY	An internal error occurred.
PTPE_SDR	An unexpected error occurred while obtaining information from the PTPE data classes in the System Data Repository. These data classes might not exist, or the System Data Repository might be experiencing difficulties.
PTPE_STATE	Performance information collection and summarization is not currently active in the monitoring hierarchy.

Examples

```
#include <spdm.h>

session_ptr_t   sblock;
int             srate;
int             arate;
int             rc;

/* set up PTPE session and start collection earlier in the code */
if (rc != PTPE_SUCCESS) {
     printf("Could not obtain sampling and archiving rates\n");
}
else{
     printf("Current sampling rate is %d seconds\n",srate);
     printf("Current recording rate is %d seconds\n",arate);
}
```

Related Information

- “PtpeOpenSession” on page 277
- “PtpeArchStartHosts” on page 156
- “PtpeArchStartAllHosts” on page 152
- “PtpeChangeHostRates” on page 172
- “PtpeColStart” on page 215

PtpeQueryHostStatus

Purpose

Reports the status of performance information collection and archiving on specific systems in the Performance Toolbox Parallel Extensions monitoring hierarchy.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeQueryHostStatus(sblock, targets, reply)
session_ptr_t   sblock;
host_list_t     targets;
host_list_t     *reply;
```

Parameters

sblock
: The address of a session control block. This information block is created and initialized by the **PtpeOpenSession** subroutine.

targets
: The anchor of a host list that contains at least one entry. All systems in this list should not have statistics lists assigned to them.

reply
: A pointer to a host_list_t data type, which should be set to a NULL value before invoking this subroutine. Upon successful completion of the subroutine, an anchor point for a host list will be placed at this location. This list will contain entries for each system in the *targets* list. Each entry will contain a result code to indicate whether or not the system was successful in carrying out the request.

Description

PtpeQueryHostStatus returns the current status of performance information collection and archiving on the systems provided in the *targets* parameter.

The application must first establish a PTPE API session by using the **PtpeOpenSession** subroutine. The application must also construct a host list for the command, containing an entry for each system that the application needs to query. This list is then provided in the *targets* parameter.

When **PtpeQueryHostStatus** is executed, the subroutine relays the query, along with the host list of systems involved, to the monitoring hierarchy's central coordinator node. The central coordinator node determines which nodes are data managers for the systems in the *targets* list, and forwards the query to only those data manager nodes that manage systems in the host list. The data manager nodes then relay the query to those systems in their reporting groups that have been listed in the targets list.

Each system in the *targets* list replies with a code that indicates its current collection and archiving status. This code is constructed by ORing together the following values:

PTPE_SAMPLE	The system is currently supplying performance information (set by all systems whenever performance information collection is active)
PTPE_COLLECT	The system is currently supplying summary performance information (set by managing systems when performance information collection is active)
PTPE_ARCHIVE	The system is currently recording performance information to the archive If collection or archiving is not active on the system, the system replies with the following code:
PTPE_INACTIVE	The system is not currently supplying performance information or archiving information.

The system then indicates its success or failure in the effort to its data manager node.

Data manager nodes construct partial host lists, with an entry for each system in its reporting group that was targeted for the command. The system's status or failure reply is placed in the partial host list. This list is then relayed to the central coordinator node, who constructs a complete host list. The complete host list is transmitted back to the **PtpeQueryHostStatus** subroutine, which provides it to the application in the *reply* parameter.

PtpeQueryHostStatus will provide an indication of the overall success or failure of the query in the return code:

PTPE_SUCCESS	All systems in the *targets* list successfully responded to the query.
PTPE_LIMITED	Some systems in the *targets* list were unable to respond successfully to the query.
PTPE_API_FAILED	All systems in the *targets* list were unable to respond successfully to the query.

To determine the status of a system, and to determine which systems were not successful, the *reply* list should be scanned, and the results for each system in the list checked using the **PtpeGetHostResult** subroutine. A negative value for a result indicates an error. A value of zero or greater reflects the current status of the system. In non-error cases, the result should be ANDed with PTPE_SAMPLE, PTPE_COLLECT, or PTPE_ARCHIVE to determine the system's current status. This subroutine cannot be issued from a PTPE read-only session.

Return Codes

Upon successful completion, a return code of PTPE_SUCCESS is returned to the caller. If an error occurred, one of the following return codes is provided:

PTPE_SUCCESS_BADR	The central coordinator node has indicated that all systems in the *targets* list responded successfully to the query, but an error occurred in reading the *reply* list response from the central coordinator Node.

PTPE_LIMITED	Some of the systems in the *targets* list were unable to reply to the query. The list *reply* points to contains all systems involved in the command; the application can determine which systems failed by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_LIMITED_BADR	Some systems in the *targets* list were unable to respond to the query, and an error occurred in reading the *reply* list response from the central coordinator node.
PTPE_API_FAILED	The central coordinator node has indicated that all systems in the *targets* list failed to respond to the query. *reply* contains all systems involved in the query, along with the reasons for the failure on these systems. The application can determine the cause of the error by performing the **PtpeGetHostResult** subroutine on all systems in the *reply* list.
PTPE_API_FAILED_BADR	The central coordinator node has indicated that all systems failed to respond to the query, and an error occurred in reading the *reply* list from the central coordinator node.
PTPE_NO_CONNECT	Could not establish a network connection to the central coordinator node. *reply* list not modified.
PTPE_BAD_SEND	Could not successfully transmit the command to the central coordinator node. *reply* is not modified.
PTPE_BAD_RECEIVE	Error in receiving the reply to the command from the central coordinator node. It is impossible to determine if the query was successful. *reply* is not modified.
PTPE_RONLY_SESS	The application attempted to issue this subroutine from a PTPE read-only session. A regular session must be established to issue this instruction. Ensure that the application is executing in the proper SP system partition.
PTPE_TIMEOUT	A reply to the command was not received from the central coordinator node in the time allowed. reply is not modified.
PTPE_INV_HIERARCHY	The monitoring hierarchy contains a node that is not a member of the currently active system partition. The hierarchy must be repaired by removing any nodes that are not members of the active system partition before this subroutine can be used.
PTPE_INV_HOSTLIST	*targets* either contains a NULL value, or does not anchor a valid host list.
PTPE_INV_PTR	*sblock* has a NULL value, or *reply* is not NULL.
PTPE_EMPTY	*targets* is an empty host list.
PTPE_HOST_NOT_FOUND	One or more systems listed in *targets* could not be found in the monitoring hierarchy. *reply* contains the list of systems that could not be found in the hierarchy.

PTPE_NO_SESSION	A PTPE API session was not previously established by this application (see “PtpeOpenSession” on page 277).
PTPE_NO_HIERARCHY	A monitoring hierarchy does not exist. This subroutine cannot function without a monitoring hierarchy (see “ptpehier” on page 99)
PTPE_NO_MEMORY	Could not allocate enough memory to initiate the query. *reply* is not modified.
PTPE_MEMORY	An internal error occurred.
PTPE_BAD_LOC_PTR	The internal pointers in the *targets* anchor have been corrupted.
PTPE_SDR	An unexpected error occurred while obtaining information from the PTPE data classes in the System Data Repository. These data classes might not exist, or the System Data Repository might be experiencing difficulties.

Examples

```
#include <spdm.h>

session_ptr_t   sblock;
host_list_t     targets, reply;
int             rc;

/* set up session and "targets" earlier */
reply = (host_list_t) NULL;
rc = PtpeQueryHostStatus(sblock, targets,
                         &reply);
if (rc != PTPE_SUCCESS) {
     printf("Not all systems succeeded.\n");
}
/* loop through "reply" and check results */
/* on each system with PtpeGetHostResult  */
```

Related Information

- “PtpeOpenSession” on page 277
- “PtpeArchQueryStats” on page 147
- “PtpeArchStartHosts” on page 156
- “PtpeQueryAllHostStatus” on page 280
- “PtpeColStart” on page 215
- “PtpeColStop” on page 220
- “PtpeArchStartAllHosts” on page 152
- “PtpeArchStopHosts” on page 165
- “PtpeArchStopAllHosts” on page 161
- “PtpeInitHostList” on page 265
- “PtpeAddHostToList” on page 114

- “PtpeFirstHost” on page 237
- “PtpeNextHost” on page 273
- “PtpeFindHost” on page 233
- “PtpeFreeHostList” on page 241
- “PtpeGetHost” on page 245
- “PtpeGetHostResult” on page 247
- “ptpectrl” on page 88

PtpeRemoveStatsFromHost

Purpose

Removes any assigned statistics from the currently referenced system in a host list.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeRemoveStatsFromList(hanchor)
host_list_t   hanchor;
```

Parameters

hanchor
The anchor of a non-empty host list.

Description

PtpeRemoveStatsFromHost removes any statistics that may have been assigned to the “current” entry in the *hanchor* host list. A statistics list may have been assigned explicitly by the application to a system by use of the **PtpeAssignStatsToHost** subroutine, or may have been provided by an API subroutine that retrieves performance information from the Performance Toolbox Parallel Extension monitoring hierarchy (such as **PtpeArchGetStats** and **PtpeColGetStats).** Once a statistics list has been assigned to a system, that list is used by all subsequent API subroutines until the list is removed with this subroutine. In order to modify the statistics associated with a system's entry in a host list, the statistics list must be removed by this subroutine, and a modified list assigned using the **PtpeAssignStatsToHost** subroutine. An error results if this subroutine is performed upon an empty or uninitialized host list.

Return Codes

Upon successful completion, this subroutine returns the value of PTPE_SUCCESS. If an error has occurred, the return code is set to one of the following values to indicate the cause of the error:

PTPE_INV_HOSTLIST	*hanchor* is a NULL value.
PTPE_INV_STATLIST	The “current” entry in the *hanchor* host list does not have a statistics list assigned to it.
PTPE_EMPTY	*hanchor* is an empty host list.
PTPE_BAD_LOC_PTR	The memory pointers in *sanchor* have been corrupted.

Examples

```
#include <spdm.h>

host_list_t   hanchor;
int           rc;

rc = PtpeFindHost("spnode05.ibm.com",
                  hanchor);
rc = PtpeRemoveStatsFromHost(hanchor);
switch(rc) {
case PTPE_SUCCESS:
     break;
case PTPE_INV_STATLIST:
     printf("No statistics to remove\n");
     break;
default:
     /* handle error condition */
}
```

Related Information

- “PtpeInitHostList” on page 265
- “PtpeInitStatList” on page 267
- “PtpeAssignStatsToHost” on page 170
- “PtpeFirstHost” on page 237
- “PtpeNextHost” on page 273
- “PtpeFindHost” on page 233
- “PtpeAddHostToList” on page 114
- “PtpeDelHostFromList” on page 225

PtpeSetStatTime

Purpose

Sets the timestamp in the currently referenced entry of a statistics list.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeSetStatTime(whence, tstmp, sanchor)
int            whence;
struct tm      tstmp;
stat_list_t    sanchor;
```

Parameters

whence
Determines how the timestamp is set in the currently referenced entry of *sanchor*. Can contain one of three values:

PTPE_EARLIEST	The *tstmp* parameter is ignored, and the statistic timestamp is set to retrieve the earliest recorded value for this statistic when API operations are performed using this statistics list.
PTPE_LATEST	The *tstmp* parameter is ignored, and the statistic timestamp is set to retrieve the last recorded value for this statistic when API operations are performed using this statistics list.
PTPE_MATCH	The statistic timestamp is set to the value indicated by the *tstmp* parameter.

tstmp
The address of a struct tm structure, which contains the timestamp to set in the currently referenced entry of *sanchor*. The format of the tm structure is defined in the <**time.h**> header file.

sanchor
The address of an existing statistics list anchor point.

Description

PtpeSetStatTime sets the timestamp in the “current” entry of the statistics list anchored at *sanchor*. How the timestamp is set depends upon the setting of the *whence* parameter (see the description in the Parameters section).

The setting of the timestamp dictates which of the values recorded in performance information archives will be obtained by API subroutines such as **PtpeArchGetStats.** If the *whence* parameter is set to PTPE_EARLIEST, the timestamp provided by the caller is ignored, and the statistic entry is set to retrieve the earliest recorded observance of this statistic in a performance information archive. Setting *whence* to PTPE_LATEST also ignores the timestamp provided by the caller, and sets the statistic entry to retrieve the last recorded observance for the statistic in a performance information archive. When the *whence* parameter is set to PTPE_MATCH, the statistic's timestamp is set to the value provided by the caller, and will instruct subsequent API subroutines to attempt to locate the entry recorded for this time in a performance information archive.

PTPE API subroutines will attempt to match the timestamps provided by the caller exactly when the *whence* parameter is set to PTPE_MATCH; in most cases, this will not be possible. When an exact match for the time cannot be located, the API will find the closest match that does not exceed the specified time. When examining the statistics list returned by API subroutines, the application should examine the statistic's result code with the **PtpeGetStatResult** subroutine. A result code of PTPE_TIME_APPROX will indicate that an exact match for the timestamp could not be located, and an approximate time was returned in its place.

Return Codes

Upon successful completion, this subroutine returns one of the following values:

PTPE_SUCCESS	The timestamp provided in *tstmp* was successfully converted to a statistical timestamp value and stored in the “current” statistic list entry.
PTPE_TIME_APPROX	The timestamp provided in *tstmp* could not be accurately converted to a statistical timestamp before being stored in the “current” statistics list entry, so the subroutine needed to substitute an approximated timestamp in its place.

If an error has occurred, the return code is set to one of the following values to indicate the cause of the error:

PTPE_INV_PTR	*tstmp* has a NULL value, or *whence* does not contain a valid value.
PTPE_INV_STATLIST	*sanchor* has a NULL value.
PTPE_EMPTY	*sanchor* is an empty host list.
PTPE_BAD_LOC_PTR	The memory pointers in *sanchor* have been corrupted.
PTPE_MEMORY	An internal memory usage error occurred.

Examples

```
#include <spdm.h>

stat_list_t    sanchor;
struct tm      tstmp;
int            whence;
int            rc;

/* Get statistic value for 2:30pm on October 1, 1995 */
tstmp.tm_sec = 0;
tstmp.tm_min = 30;
tstmp.tm_hour = 14;
tstmp.tm_mday = 1;
tstmp.tm_mon = 9;
tstmp.tm_year = 95;

rc = PtpeFindStat("Mem/Virt/%free", sanchor);
rc = PtpeSetStatTime(PTPE_MATCH, tstmp,
                     sanchor);
switch(rc) {
case PTPE_SUCCESS:
     break;
case PTPE_TIME_APPROX:
     printf("Had to use an approximate ");
     printf("timestamp\n");
     break;
default:
     /* handle error condition */
}
```

Related Information

- “PtpeInitStatList” on page 267
- “PtpeFirstStat” on page 239
- “PtpeNextStat” on page 275
- “PtpeFindStat” on page 235
- “PtpeAddStatToList” on page 116
- “PtpeGetStatType” on page 258
- “PtpeStatIsLong” on page 305
- “PtpeStatIsFloat” on page 303
- “PtpeArchGetStats” on page 138

PtpeStatIsFloat

Purpose

Tests if the currently referenced statistic in a statistics list is of a float or double data type.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeStatIsFloat(sanchor)
stat_list_t     sanchor;
```

Parameters

sanchor
The address of an existing statistics list anchor point.

Description

PtpeStatIsLong examines the data type used to store the statistic at the “current” location in the statistics list anchored at *sanchor*. When this subroutine is performed on a valid, non-empty statistics list, the subroutine will indicate whether or not the data type used by the “current” entry is PTPE_FLOAT.

When an entry is created for a statistic by the **PtpeAddStatToList** subroutine, the data type is set by default to PTPE_NONE. If the statistic type has not been set to a specific type by the **PtpeSetStatType** subroutine, or by any other API subroutine that retrieves statistic information from the Performance Toolbox Parallel Extension monitoring hierarchy (such as **PtpeArchGetStats**), this subroutine will return the value of PTPE_FALSE.

This subroutine returns an error when performed on an uninitialized or empty statistics list.

Return Codes

Upon successful completion, this subroutine returns one of the following values:

PTPE_TRUE	The data type used to store the currently referenced statistic is PTPE_FLOAT.
PTPE_FALSE	The data type is either PTPE_LONG, or has not been set by another API subroutine,

If an error has occurred, the return code is set to one of the following values to indicate the cause of the error:

PTPE_INV_STATLIST	*sanchor* has a NULL value.
PTPE_EMPTY	*sanchor* is an empty host list.
PTPE_BAD_LOC_PTR	The memory pointers in *sanchor* have been corrupted.

PTPE_MEMORY	An internal memory usage error occurred.

Examples

```
#include <spdm.h>

stat_list_t   sanchor;
long          ldata;
float         fdata;
int           rc;

rc = PtpeStatIsFloat(sanchor);
if (rc != PTPE_TRUE || rc != PTPE_FALSE) {
     /* handle error */
}
else {
     if (rc == PTPE_TRUE) {
         rc = PtpeGetStatValueFloat(&fdata,
                           sanchor);
         }
}
```

Related Information

- “PtpeInitStatList” on page 267
- “PtpeFirstStat” on page 239
- “PtpeNextStat” on page 275
- “PtpeFindStat” on page 235
- “PtpeAddStatToList” on page 116
- “PtpeDelStatFromList” on page 227
- “PtpeStatIsLong” on page 305
- “PtpeGetStatType” on page 258
- “PtpeGetStatName” on page 252
- “PtpeGetHostStatList” on page 250
- “PtpeGetStatValueLong” on page 263
- “PtpeGetStatValueFloat” on page 261

PtpeStatIsLong

Purpose

Tests if the currently referenced statistic in a statistics list is of an integer or long integer data type.

Library

libptpe.a

Syntax

```
#include <spdm.h>

int PtpeStatIsLong(sanchor)
stat_list_t     sanchor;
```

Parameters

sanchor
The address of an existing statistics list anchor point.

Description

PtpeStatIsLong examines the data type used to store statistic at the “current” location in the statistics list anchored at *sanchor*. When this subroutine is performed on a valid, non-empty statistics list, the subroutine will indicate whether or not the data type used by the “current” entry is PTPE_LONG.

When an entry is created for a statistic by the **PtpeAddStatToList** subroutine, the data type is set by default to PTPE_NONE. If the statistic type has not been set to a specific type by the **PtpeSetStatType** subroutine, or by any other API subroutine that retrieves statistic information from the Performance Toolbox Parallel Extension monitoring hierarchy (such as **PtpeArchGetStats**), this subroutine will return the value PTPE_FALSE.

This subroutine returns an error when performed on an uninitialized or empty statistics list.

Return Codes

Upon successful completion, this subroutine returns one of the following values:

PTPE_TRUE	The data type used to store the currently referenced statistic is PTPE_LONG.
PTPE_FALSE	The data type is either PTPE_FLOAT, or has not been set by another API subroutine, If an error has occurred, the return code is set to one of the following values to indicate the cause of the error:
PTPE_INV_STATLIST	*sanchor* has a NULL value.
PTPE_EMPTY	*sanchor* is an empty host list.
PTPE_BAD_LOC_PTR	The memory pointers in *sanchor* have been corrupted.
PTPE_MEMORY	An internal memory usage error occurred.

Examples

```
#include <spdm.h>

stat_list_t    sanchor;
long           ldata;
float          fdata;
int            rc;

rc = PtpeStatIsLong(sanchor);
if (rc != PTPE_TRUE || rc != PTPE_FALSE) {
    /* handle error */
}
else {
    if (rc == PTPE_TRUE) {
rc = PtpeGetStatValueLong(&ldata,
                          sanchor);
     }
}
```

Related Information

- "PtpeInitStatList" on page 267
- "PtpeFirstStat" on page 239
- "PtpeNextStat" on page 275
- "PtpeFindStat" on page 235
- "PtpeAddStatToList" on page 116
- "PtpeDelStatFromList" on page 227
- "PtpeRemoveStatsFromHost" on page 298
- "PtpeGetStatType" on page 258
- "PtpeStatIsFloat" on page 303

Chapter 9. Diagnosing PTPE Problems and Messages

This chapter contains suggestions for diagnosing problems encountered while using various PTPE functions. It also includes a list of all error messages, their meanings, and steps you can follow to correct the conditions that generate these messages.

Diagnosing PTPE Problems

This section contains information to help you diagnose problems you may encounter while installing or running PTPE. It helps you to identify the cause of the problem and includes a procedure to follow if you should require assistance from IBM Support.

Identifying PTPE Problems

PTPE problems are indicated by an error message returned when you enter a command. These messages are entered in the logs. You can look messages up in "PTPE Messages" on page 330, where explanations and recovery suggestions are provided.

Getting Help From IBM

If you require help from IBM in resolving a PTPE problem, you can get assistance by calling IBM Support at the numbers listed below.

Before you call, be sure you have the following information:

1. Your access code (customer number).
2. The PTPE product number: **5765-D51**.

   This is important information that will speed the correct routing of your call.
3. The name and version of the operating system you are using.
4. A telephone number where you can be reached.

The person with whom you speak will ask for the above information and then give you a time period during which an IBM SP representative will call you back.

In the United States:

The number for IBM software support is **1-800-237-5511**.
The number for IBM hardware support is **1-800-IBM-SERV**.

Outside the United States, contact your local IBM Service Center.

Sending Problem Data to IBM

You may be asked to produce a system dump and send it to the IBM support office. Refer to *IBM Parallel System Support Programs for AIX: Diagnosis and Messages Guide* for instructions on how to produce this information.

To send the data to IBM, label the tape or diskette with the problem number and mail it to:

IBM RS/6000 Scalable POWERparallel Systems Dept. 39KA, M/S P961, Bldg. 415 522 South Road Poughkeepsie, N.Y. 12601-5400 ATTN: APAR Processing

Opening a Problem Management Record (PMR)

A PMR is an online software record used to keep track of software problems reported by customers.

Follow your local support/service procedures for opening a PMR.

Note: To aid in quick problem determination and resolution, it will be very useful to have the SDR data specific to the problem included in the PMR.

SDR data can be obtained through the **splstdata** command. Use the appropriate command flag to view data relevant to the problem. For example:

splstdata -e	Lists environment choices
splstdata -n	Lists node information
splstdata -s	Lists switch information

For more information on **splstdata**, refer to *IBM Parallel System Support Programs for AIX: Command and Technical Reference*.

Error Logging

Error logging is a convenient way to record error information in persistent storage for later reference and debugging. Refer to the *IBM Parallel System Support Programs for AIX: Diagnosis and Messages Guide* for a general discussion of error logging for the SP

Identifying PTPE Error Log Entries

PTPE error log entries are generated for the software class with the resource name, **perfmon**. To obtain the error log entries PTPE generated for a node, issue the **errpt** command as follows:

```
errpt -a -N perfmon > ptpe.err
```

The PTPE daemons generate error log entries under two conditions:

Change in collection or archiving state
Whenever the PTPE daemons begin or stop collection or archiving of performance collection

Daemon failure
Whenever the PTPE daemons encounter unexpected conditions resulting in daemon failure, or the inability of a daemon to handle a request.

Log Entries for Change in Collection or Archiving State

PTPE generates these log entries in the following format:

```
------------------------------------------------------
LABEL: PERFMON_COLLECT_ST
IDENTIFIER: 650AFF61

Date/Time:       Fri Jul 19 16:51:57
Sequence Number: 3167
Machine Id:      000131141000
Node Id:         avenger
Class:           S
Type:            UNKN
Resource Name:   perfmon

Description
PERFORMANCE TOOLBOX PARALLEL EXT. EVENT

Probable Causes
NONE

Failure Causes
NONE

  Recommended Actions
  NONE

Detail Data
DETECTING MODULE
LPP=PTPE,Fn=spdmspld.c,SID=1.34,L#=254,
EVENT DATA:
Performance monitor data sampling started
--------------------------------------------------------
```

Error log records that indicate a change in the current performance information collection status are given the label, PERFMON_COLLECT_ST, while those indicating a change in the current performance information archiving status have a PERFMON_ARCHIVE_ST label. The description code used in either case is PERFORMANCE TOOLBOX PARALLEL EXT. EVENT. Event data may vary, depending on the event being recorded:

Performance monitor data sampling started
This entry is made by the PTPE **spdmspld** daemon when it executes and begins collecting performance information from the node. An entry of this type should appear on all nodes within the monitoring hierarchy after performance information collection has begun.

Performance monitor data sampling terminated
This entry is made by the PTPE **spdmspld** daemon when it halts itself, either in response to an error or in response to a stop request. An entry of this type should appear on all nodes within the monitoring hierarchy after performance information collection has been shut down.

Performance monitor aggregate data collection started
This entry is made by the PTPE **spdmcold** daemon when it executes and begins performing manager duties in a monitoring hierarchy. An entry of this type should appear on all data manager nodes within the monitoring hierarchy, as well as the g node, after performance

information collection has begun. This entry should not appear in the error log of a node that is not configured as a data manager or the g in the monitoring hierarchy, unless the node was previously configured as such a node, and the error log was not cleared out after the hierarchy was changed.

Performance monitor aggregate data collection terminated
This entry is made by the PTPE **spdmcold** daemon when it halts itself, either in response to an error or in response to a stop request. An entry of this type should appear on all data manager nodes within the monitoring hierarchy, as well as the g, after performance information collection has been shut down. This entry should not appear in the error log of a node that is not configured as a data manager or g in the monitoring hierarchy, unless the node was previously configured as such a node, and the error log was not cleared out after the hierarchy was changed.

Performance monitor archiving of data started or resumed
This entry is made by the PTPE **spdmspld** daemon when it begins recording performance information to the archive. This daemon also generates an entry of this type whenever it resumes recording performance data, should the daemon have temporarily suspended recording due to filesystem space limitations. An entry of this type should appear on any nodes in a monitoring hierarchy after performance information archiving has begun.

Performance monitor archiving of data terminated
This entry is made by the PTPE **spdmspld** daemon when it ceases recording performance information to the archive. This daemon also generates an entry of this type whenever it suspends recording of performance data due to filesystem space limitations. An entry of this type should appear on any nodes within the monitoring hierarchy after performance information archiving has been shut down for the node.

Log Entries for Daemon Failure

PTPE generates these log entries in the following format:

```
-----------------------------------------------------------
LABEL:  PERFMON_ERROR_ER
IDENTIFIER: 673DC205

Date/Time:       Fri Jul 19 16:27:41
Sequence Number: 3160
Machine Id:      000131141000
Node Id:         avenger
Class:           S
Type:            PERM
Resource Name:   perfmon

Description
PERFORMANCE TOOLBOX PARALLEL EXT. ERROR

Probable Causes
SOFTWARE PROGRAM

Failure Causes
SOFTWARE SUBTASK


Recommended Actions
PERFORM PROBLEM DETERMINATION PROCEDURES
IDENTIFY OFFENDING SOFTWARE COMPONENT
IF PROBLEM PERSISTS THEN DO THE FOLLOWING
CONTACT YOUR LOCAL IBM SERVICE REPRESENTATIVE

Detail Data
DETECTING MODULE
LPP=PTPE,Fn=debug.c,SID=1.9,L#=202,
EVENT REASON CODE:
spdmspld : 2516-746 Cannot connect to socket file
/var/ha/soc/em.RMIBM.PSSP.harmld.part1:
   No such file or directory
-----------------------------------------------------------
```

Records of this type are given a label of `PERFMON_ERROR_ER` and a description code of `PERFORMANCE TOOLBOX PARALLEL EXT. ERROR`. The detecting module file name will always be listed as `debug.c`, because of the structure of the PTPE source code, but the daemon that is making the error log entry will always record its name in the Event Reason Code. The text of the Event Reason Code varies, depending upon the error encountered. The contents of the Event Reason Code come from the PTPE message catalog. For a more complete description of the error being recorded and recommendations for corrective action to take, consult the Performance "PTPE Messages" on page 330.

Diagnosing PTPE Command Problems

Error conditions encountered during PTPE command execution can be divided into two groups: those common to most PTPE commands, and those that are unique to certain commands. The following discussions of both types of error conditions include suggestions for solving these problems, or bypassing them.

Common PTPE Command Errors

These error messages can be returned by most PTPE command:

User is not authorized to use this command

User is not a member of the perfmon user group

Most PTPE commands can only be issued by members of the **perfmon** user group. This error occurs if the user not a member of this group, but more frequently, the user is not running with the **perfmon** group as the primary group.

Verify that the user is a member of the **perfmon** user group, and that the user has set that group as the primary group. To change the primary group, use the **newgrp** command:

```
newgrp perfmon
```

User is not authorized to use this command

Unexpected System Data Repository Error

The user may be a member of the **perfmon** user group, and have the primary group correctly set, however the user's account may not have sufficient privileges to issue SDR commands. Refer to the *IBM Parallel System Support Programs for AIX: Administration Guide* for information on SDR access privileges, and certify that the user account has these privileges, or create a new user account with these privileges, and assign it to the **perfmon** user group.

Unexpected System Data Repository Error

Cannot contact the System Data Repository

This error can occur when the System Data Repository, or the control workstation, is temporarily off-line. This error can also occur if the **sdrd** daemon on the local node has terminated or lost contact with its server system. For assistance in resolving System Data Repository problems, refer to *IBM Parallel System Support Programs for AIX: Administration Guide*

This command cannot be issued from a read-only PTPE session

Commands that set controls within the monitoring hierarchy of a specific system partition, or alter the hierarchy structure, must be issued from nodes within that system partition. Nodes outside the system partition cannot issue these commands. When they attempt to do so, the above error results.

The user issued a command to control or alter the hierarchy from another SP system partition. For example, such an error would be caused by attempting to start collection on partition “syspar2” from a node (not the control workstation) on partition “syspar1” while the SP_NAME environment variable is set to “syspar2.”

Ensure that the system partition variables are set correctly.

Performance information collection is not in the proper state to issue this command

Performance information archiving is not in the proper state to issue this command

Certain commands cannot be issued while collection or archiving are active. These include commands that attempt to restart collection, restart archiving, or alter the structure of the monitoring hierarchy. Conversely, certain actions cannot be issued when collection or archiving are disabled. For example, collection cannot be disabled when collection is inactive, archiving cannot be disabled if either archiving or collection is inactive, and the daemon intervals cannot be altered if collection is inactive.

Check the current status of collection by issuing

```
ptpectrl -q
```

Verify that collection or archiving is in the proper state for the command you wish to issue.

Cannot perform the requested action because another application holds a PTPE session

Most PTPE commands attempt to obtain a PTPE session as part of their operation. However, only one PTPE application or command may possess a PTPE session at any one time, regardless of where they execute within the same system partition. For this reason, commands and applications may occasionally fail to obtain the PTPE session because another PTPE command or application currently holds it. In such cases, the command should be executed at a later time. If you find that the session cannot be obtained for a prolonged period of time, search for other PTPE applications that may be executing, and which may also be holding the PTPE session for longer periods than "Controlling Sessions" on page 48 recommends.

ptpectrl Command Errors

Many errors encountered by the **ptpectrl** command result in messages that are explained in "PTPE Messages" on page 330. Consult this section for recommended actions for the more common **ptpectrl** error conditions.

The following scenarios cover more complex error conditions that may be encountered by the **ptpectrl** command, or the PTPE daemons it attempted to contact in the monitoring hierarchy.

Symptoms of Daemon Failure During Collection Startup: A node may report that performance information collection was successfully started during the initial startup phase of the **ptpectrl -a** or **ptpectrl -c** command, but may report that the collection daemons have terminated during the statistic enablement phase of the command. The most likely causes of this failure are:

1. Inability to create or update the **/var/adm/ptpe/perftab** file. During collection startup, if this table does not exist on the node, the PTPE daemons attempt to create it after the initial phase of collection startup has completed. If the file cannot be created or updated, the daemons terminate, creating an entry in that node's error log to explain the error. If the file does not exist on the node, the filesystem containing the **/var/adm/ptpe** directory does not have enough space, and should be extended. If the file exists, the permissions on the file may have been changed to prohibit further modification, and should be reset to `-rw-r--r--`.
2. Spmi error - After the completion of the first phase of collection startup, the PTPE daemons attempt to register their new statistics with Performance Toolbox for AIX, using the Spmi interface. If an error occurred during this effort, the daemons many not be able to continue, and will terminate. Again, the daemons will create an entry in that node's error log to explain the error. If an Spmi error has occurred, certain Spmi shared memory may need to be

refreshed or removed to correct the problem. To refresh the **xmservd** daemon and remove unused shared memory, send a SIGINT signal to the **xmservd** process on that node with the `kill -1` command. Consult the *IBM Performance Toolbox for AIX: Guide and Reference* for further recommendations on handling the error.

In either case, the error log on the failing node should be examined to determine the cause of the error.

Symptoms of Reporter Node Failure After Collection or Archiving Startup Process Completes: PTPE is not informed when nodes fail, or are purposely taken offline. When a node fails, the global status information for PTPE, which is stored in the System Data Repository, is not modified. The **spdmcold** daemon running on the failing node's data manager node will notice a lack of input from the failed node, but it will not be able to distinguish this event from slow network traffic or a network failure. The data manager node will compensate for the missing input by excluding the node from the calculation of the statistical averages, but no other action is taken.

Because the global status information is not modified, the **ptpectrl** command may encounter difficulties when shutting off performance information collection or archiving. If a node has failed and has not been restarted, the manager daemon will report that it cannot issue the shutdown request to the node, recoding this in its own error log. The **ptpectrl** command will report that it failed in executing the request on the failed node. Subsequent **ptpectrl** commands may report that performance information archiving is still active, or that it cannot determine the status of performance information archiving.

This problem can usually be cleared by temporarily shutting down performance information collection with the **ptpectrl -s** command, then restarting collection again after the reporting node has been brought back online. Shutting down collection will terminate all daemons that record performance information to the archives, and should reset the global PTPE status information stored in the System Data Repository. If the problem is not cleared by temporarily shutting down performance information collection, further steps need to be taken. These steps are listed below. However, use these steps only as a last resort.

1. Shut down performance information collection, using the **ptpectrl -s** command.
2. Ensure that all PTPE daemons have terminated within the monitoring hierarchy. This is easily accomplished with the following **dsh** command:

   ```
   dsh -a ps -ef : grep spdm
   ```

   If any nodes show that the **spdmspld** sampler daemon or the **spdmcold** manager daemon are executing, terminate them using the **kill** command, either directly on those nodes or remotely with the **dsh** command. Do not issue the **kill -9** command on these daemons, for that will prevent the daemons from cleaning up their shared memory resources, and make it difficult or impossible to restart the PTPE daemons without clearing out unused shared memory on the nodes.
3. Reset the global status information stored for PTPE in the System Data Repository. The following command must be executed only within the same system partition as the one where PTPE was previously running:

   ```
   /usr/lpp/ssp/bin/SDRChangeAttrValues SPDM active=0 archive=0
   ```

These steps should reset PTPE, allowing you to restart collection and archiving within that system partition. However, keep in mind that these steps are only to be taken as a last resort, when all other methods to reset PTPE have failed.

If a reporter node should fail and then be restarted before you need to take any corrective action, do not attempt to restart the PTPE daemons manually. The **spdmspld** data sampling daemon cannot be restarted from the AIX command line or from a shell script. This daemon expects input in a specific format from a socket in order to complete its setup. While the daemon may execute for a short time if started from the command line, the daemon will eventually timeout while waiting for input on an unestablished socket connection, and terminate once again.

Symptoms of Data Manager Node Failure After Collection or Archiving Startup Process Completes: As with reporter nodes, PTPE is not informed when a data manager node in the monitoring hierarchy fails, or is purposely taken offline. The global status information, maintained for PTPE by the System Data Repository, is not updated to reflect the failure.

When a data manager node fails, the nodes in its hierarchy reporting group begin to encounter errors when reporting their performance information to the data manager node. The **spdmspld** daemons expect to occasionally encounter a problem transmitting this data, due to increased network traffic or other temporary network difficulty, and create an entry in their own error logs when such events occur. However, if the reporter nodes are unable to transmit their performance information in five consecutive attempts, the nodes will shut down the **spdmspld** daemon, recoding this event in the error log on that node. The exact time between failure of the data manager node and the subsequent shutdown of the data sampling daemons will depend on the current setting of the data sampling interval for the monitoring hierarchy. The g will not detect the failure of the data manager node, but will recognize that the data manager has not reported new data, and will exclude the data manager from calculations of the summary data that it prepares. Applications will encounter `PTPE_NO_CONNECT` errors when attempting to obtain performance information or status from the data manager node, or any nodes reporting to the data manager in the monitoring hierarchy.

Because the global status information is not modified, the **ptpectrl** command may encounter difficulties when shutting off performance information collection or archiving. If a data manager node has failed and has not been restarted, the g will report that it cannot issue the shutdown request to the node, recoding this in its own error log. The **ptpectrl** command will report that it failed in executing the request on the failed node, as well as reporting failures for all nodes that are members of the data manager's group. Subsequent **ptpectrl** commands may report that performance information archiving is still active, or that it cannot determine the status of performance information archiving.

You can resolve this problem by temporarily shutting down performance information collection using the **ptpectrl -s** command, then restarting collection when the data manager node has been brought back online. Shutting down collection will terminate all daemons that record performance information to the archives, and should reset the global PTPE status information stored in the System Data Repository.

If the problem is not cleared up by temporarily shutting down performance information collection, use the steps listed in “Symptoms of Data Manager Node

Failure After Collection or Archiving Startup Process Completes" to reset the PTPE status stored in the System Data Repository. Use these steps only as a last resort.

If a data manager node should fail and then be restarted before you need to take any corrective action, do not attempt to restart the PTPE daemons manually. Like the **spdmspld** daemon, the **spdmcold** manager daemon cannot be restarted from the AIX command line or from a shell script. This daemon expects input in a specific format from a socket in order to complete its setup. While the daemon may execute for a short time if started from the command line, the daemon will eventually timeout while waiting for input on an unestablished socket connection, and terminate once again.

Symptoms of Failure of Central Coordinator Node After Collection or Archiving Startup Process Completes: Failure of the g node is similar to a failure in a data manager node, except the effects are felt throughout the monitoring hierarchy. As with all other node failures, the PTPE global status information is not updated. If not detected in time, the **spdmcold** daemons on all data manager nodes in the hierarchy shut down, similar to the manner in which the **spdmspld** daemons on reporter nodes shut down after repeated failures in contacting the manager. After five unsuccessful attempts to transmit the summary performance information to the central coordinator, the **spdmcold** daemons on all data manager nodes will shut down. The **spdmspld** daemons on all reporter nodes will shut down in a similar fashion after five failed attempts to forward their information to their data managers. Therefore, all PTPE daemons on all nodes in the monitoring hierarchy should shut down after ten data sampling intervals have expired. However, PTPE will still believe that performance information collection or archiving is still active within the monitoring hierarchy. All application requests and commands made to the hierarchy should fail with a `PTPE_NO_CONNECT` error to the g, and to all other nodes in the monitoring hierarchy. The application will not be able to contact any node within the hierarchy.

You can resolve this problem by temporarily shutting down performance information collection using the **ptpectrl -s** command, then restarting collection when the g node has been brought back online. Shutting down collection will terminate all daemons that record performance information to the archives, and should reset the global PTPE status information stored in the System Data Repository.

If the problem is not resolved by temporarily shutting down performance information collection, use the steps listed in "Symptoms of Reporter Node Failure After Collection or Archiving Startup Process Completes" on page 314 to reset PTPE status stored in the System Data Repository. Use these steps only as a last resort.

If the g node should fail and then be restarted before you need to take any corrective action, do not attempt to restart the PTPE daemons manually. The **spdmspld** and **spdmcold** daemons cannot be restarted from the AIX command line or from a shell script. These daemons expect input in a specific format from a socket in order to complete its setup. While the daemon may execute for a short time if started from the command line, the daemon will eventually timeout while waiting for input on an unestablished socket connection, and terminate once again.

Symptoms of Interference from PTPE API Applications: Each instance of the **ptpectrl** command obtains a PTPE session for the duration of the command. This session is not released until all options have been performed, no matter how many options are provided. Within that period of time, other PTPE applications and commands will not be able to obtain a PTPE session. This effectively prevents them from altering the hierarchy structure or the status of the hierarchy while the command runs.

After the command runs, however, the PTPE session can be acquired by any application or PTPE command. These commands and applications may alter the controls set by the **ptpectrl** command, or even shut down collection and alter the monitoring hierarchy. As a result, you may discover the monitoring hierarchy in a different state, or even in a different structure, than you previously left it.

The PTPE monitoring hierarchy is a shared resource, and you should plan for this eventuality.

Performance Information Archives Not Modified After Archiving Started On Node: When **ptpectrl -**r is issued, the command instructs all nodes within the monitoring hierarchy to begin recording performance information to the archive. In certain instances, you may discover that no performance information is being recorded to the archive file **/var/adm/ptpe/perflog** on some or all of the nodes in the monitoring hierarchy. The two most common reasons for this are:

1. Lack of space

   When the filesystem containing the **/var/adm/ptpe/perflog** file reaches 95% of its capacity, the PTPE daemons disable recording of performance information to the archive file. This is done to prevent PTPE from consuming all the filesystem space. If the filesystem grows, or space is made so that the filesystem is less than 95% used, the daemons will resume recoding performance information without user intervention.
2. Statistics have bee restricted by another application

   Another PTPE application, or another instance of the **ptpectrl** command, may have explicitly disabled some or all performance statistics from being recorded to the archive file. Since the PTPE monitoring hierarchy is a resource shared by all users of the **perfmon** user group, you should plan for this eventuality.

ptpehier Command Errors

Many errors encountered by the **ptpehier** command result in messages that are explained in “PTPE Messages” on page 330. Consult this section for recommended actions for the more common **ptpehier** error conditions.

The following scenarios cover more complex error conditions that may be encountered by the **ptpehier** command, or by the PTPE daemons it attempts to contact in the monitoring hierarchy.

Attempting to Include Nodes Outside Current System Partition: **ptpehier** operates on the system partition currently active in the command's environment. When automatically creating a hierarchy, the command will only include in the monitoring hierarchy nodes belonging to that system partition.

When creating a reporting hierarchy manually with the **ptpehier -i** command, the user can only specify nodes that belong to the current system partition. If the user attempts to include a node outside the current system partition, an error will result when attempting to record the hierarchy format in the System Data Repository.

Users of the **ptpehier** command should ensure that the command is being issued from within the proper system partition, and should also be familiar with the nodes available within that partition.

Using Node Names Different from The Node's Reliable Hostname: When creating a reporting hierarchy manually with the **ptpehier -i** command, the user may only specify node names that match those used as *reliable hostnames* in the System Data Repository. Partial domain name specifications are permitted. For example, `node01.ibm` would be accepable for the `node01.ibm.com` system, but alternate names are not permitted. If `node01.ibm.com` is also known as `prl105.ibm.com`, the name used as the reliable hostname must be specified. If the user does not follow this convention, an error will occur when the command attempts to record the hierarchy to the System Data Repository.

Users of **ptpehier** should have knowledge of each node's reliable hostname, and use that name in creating the monitoring hierarchy.

Including Nodes with Back-Level Operating Systems in The Monitoring Hierarchy: When using the **ptpehier -e** or **ptpehier -f** commands, the hierarchy will only contain nodes which have Parallel System Support Programs Version 2.2 installed on the node. Back-level nodes will be excluded from the hierarchy.

When using the **ptpehier -i** command, the hierarchy will be checked for back-level nodes. If any nodes in the hierarchy do not have the appropriate level of PSSP installed, the hierarchy will not be considered valid, and will not be stored in the System Data Repository.

Attempting to Alter Monitoring Hierarchy while Collection or Archiving Active in System Partition: A monitoring hierarchy's structure cannot be altered while performance information collection or archiving is active in that hierarchy. Node assignments cannot be changed, and nodes cannot be added to or removed from the hierarchy. This is an intended function of the **ptpehier** command: to prevent confusion within the PTPE daemons should the reporting paths suddenly become altered during their operation.

To alter the hierarchy's structure, performance information collection and archiving must be shut down.

ptpedump and spdm_dump Command Errors

Many errors encountered by these commands result in messages that are explained in "PTPE Messages" on page 330. Consult this section for recommended actions for the more common error conditions.

The following scenarios cover more complex error conditions that may be encountered by the these commands.

Incomplete Text File Output: The **ptpedump** command makes use of the **spdm_dump** utility on each node where it has been targeted to run, redirecting the output of that utility to a text file. The **spdm_dump** utility reads records from the archive file **/var/adm/ptpe/perflog** sequentially.

Because the **spdm_dump** utility writes its results to standard output, it does not need to check for existing space. The **ptpedump** command has no understanding of the inner format of the archive file, and cannot anticipate how much disk space is needed for the text dump.

If the filesystem containing the **/var/adm/ptpe** directory has a limited amount of space, it is possible for the resulting text file to exceed the storage capacity of the filesystem. A truncated text file may result. To correct this, add more space to the filesystem containing the **/var/adm/ptpe** directory, or execute the **spdm_dump** utility locally on the node, and redirect the output of the utility to a filesystem that can accept the text file.

Missing Text File Output: If the **ptpedump** command cannot contact a node to generate the requested dump, the command indicates this with an error message. If you cannot locate the text dump file on the node after the command has completed, the **ptpedump** command most likely failed in contacting the node. Three possible sources of error are:

1. Node failure

   If a node failed, the **ptpedump** command would not be successful in creating a text dump file on that node.

2. Network failure

   Failures in the network also prevent **ptpectrl** from contacting a node and initiating a dump of the archive file.

3. dsh error

   ptpedump uses **dsh** to invoke the **spdm_dump** utility on the targeted nodes. If the **dsh** command failed, **ptpedump** would be unsuccessful in contacting the target nodes to generate the dump. For an explanation of the possible causes of a **dsh** error, see the *IBM Parallel System Support Programs for AIX: Administration Guide*

Diagnosing PTPE API Problems

This section addresses error conditions that might occur when a program uses the PTPE application programming interface.

User Permission Problems

In order to use the PTPE API, you must be a member of the **perfmon** user group, and specify **perfmon** as your primary group.

You must also have sufficient privileges to use the System Data Repository. Refer to the *IBM Parallel System Support Programs for AIX: Administration Guide* for details on SDR privileges.

Read-Only Sessions Problems

If an application is executing on a node where the environment has been set to use a system partition other than the node's own system partition, a PTPE application will only be able to acquire a read-only PTPE session. This is indicated by a `PTPE_RONLY_SESS` return code from the **PtpeOpenSession** call. Since the application does not hold control over the monitoring hierarchy of that system partition in such a session, it can only issue the following query routines:

1. **PtpeArchQueryState**
2. **PtpeColQueryState**

3. **PtpeQueryHostRates**

All other control and query routines, other than the **PtpeCloseSession** routine, will fail with a return code of `PTPE_RONLY_SESS`, indicating that the application does not have a proper session to issue the request. Host list and statistics list interfaces within the PTPE API are not disabled.

Read-only sessions should only be used to query basic status information about the monitoring hierarchy in another system partition. If an application needs to exercise control over the monitoring hierarchy in another system partition, the application should be executed on a node within that system partition.

Invalid Pointer Problems

The PTPE API avoids writing to memory that has not been allocated by ensuring that pointers it receives are either initialized or cleared out before use. If the subroutine you are using expects this condition and finds that local variables have not been set to zero values, the API subroutine may return a **PTPE_INV_PTR** code. Chapter 8, "The PTPE API Subroutines" on page 113, states whether subroutines expect parameters to contain NULL or non-NULL values.

Determining the Cause of API Failures

This section discusses how an application can determine the success or failure of an API request, how to isolate the cause of the error, and when the user should consult the error logs on nodes in the monitoring hierarchy to identify the cause of an error.

All interfaces that contact the PTPE monitoring hierarchy to set hierarchy controls or obtain performance information return two information elements to the calling application to indicate success or failure of the request:

1. A return code

   This code, which is described in the documentation for the routine and listed in the <spdm.h> header file, indicates the overall success or failure of the request. Applications should use this to determine whether or not the API request could be carried out by the monitoring hierarchy.

2. A host list

   This list contains entries for each node that was targeted for the API request. If performance information was requested from these nodes, statistics lists will be attached to the nodes' entries if the nodes successfully carried out the request.

   Even if a PTPE API routine indicated complete success of a request (PTPE_SUCCESS), the application should scan the reply list and verify the results from each node. The reply list can be scanned with the PtpeFirstHost, PtpeNextHost, PtpeFindHost, and PtpeIsLastHost routines. The results from each node within the list can be obtained using the PtpeGetHostResult routine. (See subroutine man pages or *Performance Toolbox Parallel Extensions for AIX Guide and Reference* to to learn when a host list is generated.)

API Return Codes: The following return codes indicate that the monitoring hierarchy was successfully contacted, and reflect the degree of success with which the nodes could carry out the request:

PTPE_SUCCESS	The API request was successfully completed on all nodes targeted for the request. The reply host list contains the actual results from each node.
PTPE_LIMITED	The API request was successfully completed on some of the nodes targeted for the request. The reply host list contains the actual results from each node, an can be scanned to determine which nodes failed.
PTPE_API_FAILED	The API request failed on all nodes targeted for the request. The reply list contains the results from these failing nodes.

On some occasions, the PTPE library will receive an indication of the overall success or failure of an API request, but it may encounter difficulties in retrieving the reply host list containing the results from each node. Upon receipt of one of these return codes, the application may attempt to retry the request. If the error persists, check the error log on the central coordinator node for the most recent **perfmon** resource entries for a possible explanation of a data transmission error, and the error log on the node executing the application for any entries that might indicate a network error. These replies are:

PTPE_SUCCESS_BADR	The API request was successfully completed on all nodes targeted for the request. However, the reply list could not be obtained. For API requests that set hierarchy controls, the application can assume that the request was successful. For API requests that query node status or performance information, the application should treat this as an error.
PTPE_LIMITED_BADR	The API request was successfully completed on some of the nodes targeted for the request. However, the reply list could not be obtained. Upon receipt of this error, the application cannot know which systems succeeded in the request and which ones failed, and should therefore treat this condition as an error, regardless of the request made. The error logs on each node within the monitoring hierarchy can be examined, looking for **perfmon** resource entries that might indicate which nodes failed the request, but this is usually not possible from an application.
PTPE_API_FAILED_BADR	The API request failed on all nodes targeted for the request. This return code can be treated the same as a PTPE_API_FAILED return code.

When the following return code is received, the application cannot know whether the monitoring hierarchy received the API request correctly, or to what extent the hierarchy could complete the request. The application may attempt the API request again, but if the condition persists, the error log should be examined on the central coordinator, looking for **perfmon** resource entries that might indicate the cause of the error. The error log on the node executing the application can also be checked for entries that would reflect network errors:

PTPE_BAD_RECEIVE	The library did not receive a valid response from the central coordinator node. It cannot be determined if the central coordinator received the API request in full,

or whether any of the request was carried out. Applications should treat this as an error.

If the library cannot transmit the API request to the central coordinator node, one of the following error codes may be returned. These errors may be caused by temporary conditions, such as socket unavailability on the central coordinator or network traffic problems. The application can choose to reissue the request, but if the conditions persist, the error log on the node executing the application should be checked for indications of network problems. The error log on the central coordinator node can also be checked, but may yield little information. The application should also be checked to make sure the session control information block, created by the **PtpeOpenSession** routine, is not being inadvertently modified:

PTPE_NO_CONNECT	The library could not establish a connection with the g node. The g may be offline, or it may be experiencing network difficulties. PTPE may also have been uninstalled on the g. The session control block in the user's application may also be be damaged, or the application may be using an invalid session control block. Ensure that the application is valid, then check the error logs on both the g node and the local node to detect possible networking errors.
PTPE_BAD_SEND	The application was able to establish a network connection to the g node, but the library could not transmit the full request. The network connection may have been dropped. Check the error log on the g node for **perfmon** resource entries that might indicate reasons for the connection being terminated. Also check the error log on the local node for possible network errors.

When the application receives any other return code from an API control request or query, the application should assume that the request was not transmitted to the monitoring hierarchy. In these cases, the application should be checked for errors.

API Host Lists: Some of the possible results that can be obtained from the **PtpeGetHostResult** routine are listed below. When one of these results is noted, the user should consult error logs on one or more systems to determine the source of the error.

PTPE_DAEMON_ERROR	The PTPE daemons on this node received an unknown, unsupported, or corrupted instruction. This code can indicate that the versions of PTPE on the node issuing the request and the node receiving the request do not match; specifically, the node receiving the request may be running a back-leveled version. If this is not the case, the request was corrupted in transmission. In either case, the error log on the node returning this error should be checked for **perfmon** resource entries that should explain the cause of the error.

PTPE_NO_CONTACT	The manager of a node that was targeted for the request could not establish a network connection to the node. The target node may be experiencing network difficulty, or PTPE may have been uninstalled on that node. The error log on the data manager node should be checked for **perfmon** resource entries that might indicate the cause of the error. The error log on the target node can also be checked for evidence of network difficulty.
PTPE_NO_EXEC	The manager of a node that was targeted for the request established a network connection to the node, but the node could not start the appropriate PTPE daemon to handle the request. The node may be experiencing high workload that might prevent the daemon from executing, or the daemons may have been removed from their expected location in the **/usr/lpp/ptpe/bin** directory. The error log on the failing node should be checked for **perfmon** resource entries that might indicate the cause of the error.
PTPE_API_FAILED	The application made a request to the targeted node that the node could not fulfil. The application should be checked to ensure it is making an appropriate request to the node.

Diagnosing PTPE Daemon Problems

Daemon failures might be indicated whenever an API call returns a message about an unusual node failure, or whenever a command lists the node as a failing node. Check the g or the data manager node whenever that node is listed as the failing node, or when the error message cites a communication failure from a reporting node in the monitoring hierarchy. Check the reporting node whenever it is citing as failing.

spdmd

This is the PTPE master daemon, which operates as a subserver of the **inetd** master daemon on all nodes in the monitoring hierarchy.

spdmd executes on a node when:

- An API request is made of that node,
- The node is a data manager or the central coordinator, and an API request is made of a node reporting to that system,
- The node is a data manager or the g, and a reporter node is submitting its performance information to that node for aggregation.

spdmd will execute under the following conditions:

- The node is offline,
- PTPE has been uninstalled on the node,
- The **spdmd** entry has been removed from, or commented out of, the **/etc/inetd.conf** file, and the **inetd** daemon was refreshed,

- Another application is using the socket reserved for **spdmd** in the **/etc/services** file,
- The node is not a target for an API request, and is not the data manager of a node targeted for an API request.

In the first four cases, PTPE will not be able to generate an error on the node where the error occurred. The data manager of the node may detect that the node did not accept the connection or respond favorably to the request, and may generate a **perfmon** resource entry in its own error log.

The **spdmd** daemon may also indicate an error if it needed to execute another PTPE daemon to handle the API request, but failed in the attempt. In this case, the daemon will create a **perfmon** resource entry in the node's error log, as well as pass back a PTPE_NO_EXEC response to the API request.

Symptoms of spdmd daemon failure include:

- The **PtpeGetHostResult** routine returns a PTPE_NO_CONNECT error for the system,
- The **ptpectrl** command indicates that the node failed during the first stage of collection startup, or reports an **spdmd** error on the node.

spdmcold

spdmcold, the PTPE collector daemon runs on the central coordinator node and all data manager nodes in the monitoring hierarchy. It prepares the averaged performance statistics for the monitoring hierarchy. This daemon also executes when the monitoring hierarchy is being initialized by the **PtpeColSetup** subroutine, or by the **ptpectrl -i** command.

spdmcold will not execute under the following conditions:

- PTPE has been uninstalled on the node,
- The daemon's binary file has been removed from its default location in the **/usr/lpp/ptpe/bin** directory,
- The node is not a data manager or the g in the monitoring hierarchy
- The nodes is a data manager or the g, but performance information collection is not active
- The node is a data manager or the g, but the monitoring hierarchy is not currently being initialized
- The node is a data manager or the g, but the **spdmd** daemon failed to execute on the node when collection was started or initialization was started

Only the final condition is in error. Use the methods described in “spdmd” on page 323 to determine the cause of the **spdmd** daemon error.

spdmcold may start, but fail, under the following conditions:

- The daemon did not receive all the input it expected from the socket connection during startup, or the socket connection was unexpectedly terminated
- The daemon believed that an **spdmcold** daemon was already running on the node, because of information remaining in the **/etc/perf/spdm.pid** file, and could not terminate the other instance of the daemon

- The daemon could not establish itself as a Spmi dynamic data supplier daemon during startup, possibly because certain Spmi shared memory remained active on the node from a previous execution of the daemon,
- The daemon could not forward a collection startup or a hierarchy initialization request to any of the nodes that report to it in the monitoring hierarchy
- The daemon could not find enough space in the filesystem containing the **/var/adm/ptpe** directory to record the statistic translation file **perftab**
- The **spdmcold** daemon on the g node terminated abnormally, and the daemon on this node could not report its summary performance information to the g for five consecutive attempts,
- The shared memory used by the daemon is removed by an external process, such as the **ipcrm** command, which would cause a segmentation fault
- The daemon cannot report the initialization results or confirmation of collection startup to its superior node, or to the library if the daemon is executing on the g
- An external process sends a termination or abort signal to the daemon, which the daemon would interpret as a request for termination.

In each of the following cases, the **spdmcold** daemon should create a **perfmon** resource entry in the node's error log to explain the reason for the daemon's termination. The following information should also be checked on a node where the **spdmcold** daemon has failed abnormally:

- The **/etc/perf/spdm.pid** file. This file should not exist if neither the **spdmcold** and **spdmspld** daemons are active on the node. The file contains the process identifiers of these daemons, as well as the shared memory segment identifier for the statistics table used by the **spdmcold** daemon.

  If these daemons are not currently running on the node but the file exists, examine this file. Check for the presence of the shared memory identifier listed in the file. It will be contained in an entry starting with the letter M. If the shared memory segment is still active on the node, remove it with the **ipcrm -m** command. After verifying that the shared memory identifier no longer exists on the node, or determining that none is listed in this file, remove the file.
- The **/etc/perf/ptpe.shseg** file. This file is used as a token file for the **spdmcold** daemon's Spmi shared memory. If the **spdmcold** daemon is not active on this node and this file exists, remove the file.
- The **xmservd** daemon. Occasionally, this daemon retains information from a prior execution of the **spdmcold** daemon. If the **xmservd** daemon is running on the node when the **spdmcold** daemon is inactive, instruct the **xmservd** daemon to refresh itself by determining its process identifier, then issue a **kill -1** command to the daemon.

The **ptpectrl -s** command and the **PtpeColStop** API routine are the only means that should be used to terminate the **spdmcold** daemon on a data manager or g. However, if you must terminate the daemon manually, do not issue the **kill -9** command on the daemon. This will not permit the daemon to properly discard its shared memory, remove its temporary work files, or properly update the **/etc/perf** directory before termination. This can lead later instances of the daemon to assume that the earlier instance is still active, or can cause failures in the Spmi initialization of the daemon at a later time.

Symptoms of spdmcold failure include:

- **PtpeGetHostResult** indicates a PTPE_NO_EXEC error for this node in response to a **PtpeColSetup** or **PtpeColStart** request
- Widespread failures in the **spdmspld** daemons on the nodes reporting to this node, if this node is a data manager
- Widespread failures in the **spdmcold** daemons on the data manager nodes if this node is the g
- The results of **PtpeQueryHostStatus** on the node will not indicate PTPE_COLLECT, even though collection is active in the hierarchy and this node is either a data manager or the g
- **PtpeColGetStats** will be unable to retrieve the summary performance statistics from a data manager node or the g, even though performance information collection is active
- Summary performance statistics do not seem to be appended to the performance information archive when archiving is active, if these statistics have been enabled.

spdmspld

The PTPE sampler daemon runs on all nodes within the monitoring hierarchy. It obtains all the performance information from the node, forwards this information to the node's data manager node, and records the information to that node's performance information archive.

spdmspld also executes when the monitoring hierarchy is being initialized by the **PtpeColSetup** subroutine or by the **Ptpectrl -i** command.

spdmspld will not execute under the following conditions:

- PTPE has been uninstalled on the node
- The daemon's binary file has been removed from its default location in the **/usr/lpp/ptpe/bin** directory
- Performance information collection is not active
- The monitoring hierarchy is not currently being initialized
- The **spdmd** daemon failed to execute on the node when collection was started or initialization was started.

Only the final condition is in error. Use the methods described in “spdmd” on page 323 to determine the cause of the **spdmd** daemon error.

spdmspld may start, but fail, under the following conditions:

- The daemon did not receive all the input it expected from the socket connection during startup, or the socket connection was unexpectedly terminated
- The daemon believed that an **spdmspld** daemon was already running on the node, because of information remaining in the **/etc/perf/spdm.pid** file, and could not terminate the other instance of the daemon
- The daemon could not establish itself as a Spmi data consumer daemon during startup
- The daemon could not find enough space in the filesystem containing the **/var/adm/ptpe** directory to record the statistic translation file **perftab**

- The **spdmcold** daemon on the node's data manager terminated abnormally, and the daemon on this node could not report its performance information to the g for five consecutive attempts
- The shared memory used by the daemon is removed by an external process, such as the **ipcrm** command, which would cause a segmentation fault
- The daemon cannot report the initialization results or confirmation of collection startup to its superior node, or to the library if the daemon is executing on the g
- An external process sends a termination or abort signal to the daemon, which the daemon would interpret as a request for termination.

In each of the following cases, the **spdmspld** daemon should create a **perfmon** resource entry in the node's error log to explain the reason for the daemon's termination. The following information should also be checked on a node where the **spdmspld** daemon has failed abnormally:

- The **/etc/perf/spdm.pid** file. This file should not exist if neither the **spdmcold** nor the **spdmspld** daemons are active on the node. The file contains the process identifiers of these daemons, as well as the shared memory segment identifier for the statistics table used by the **spdmcold** daemon. If this node is only a reporter node within the monitoring hierarchy, and the **spdmspld** daemon is not active on this node, remove this file. If this node is also a data manager or the g, consult the previous information on **spdmcold** errors.
- The **xmservd** daemon. Occasionally, this daemon retains information from a prior execution of the **spdmcold** daemon, which can also cause errors in subsequent data consumer requests. If the **xmservd** daemon is running on the node when the **spdmspld** daemon is inactive, instruct the **xmservd** daemon to refresh itself by determining its process identifier, then issue a **kill -1** command to the daemon.

The **ptpectrl -s** command and the **PtpeColStop** API routine are the only means that should be used to terminate the **spdmspld** daemon on any node in the monitoring hierarchy. However, if you must terminate the daemon manually, do not issue the **kill -9** command on the daemon. This will not permit the daemon to properly discard its shared memory, remove its temporary work files, or properly update the **/etc/perf** directory before termination. This can lead later instances of the daemon to assume that the earlier instance is still active, or can cause failures in the Spmi initialization of the daemon at a later time.

Symptoms of spdmspld failure include:

- **PtpeGetHostResult** indicates a PTPE_NO_EXEC error for this reporting node in response to a **PtpeColSetup** or **PtpeColStart** request
- The node's performance information does not appear to be reflected in the summary performance statistics prepared by the node's data manager
- The results of a **PtpeQueryHostStatus** on the node will not indicate PTPE_SAMPLE, even though collection is active within the hierarchy
- **PtpeColGetStats** will be unable to retrieve the basic performance statistics from a data manager node or the g, even though performance information collection is active

- Basic performance statistics do not seem to be appended to the performance information archive when archiving is active, if these statistics have been enabled.

spdmapid

The PTPE programming library request handler runs on any node that responds to a programming library request. The daemon obtains performance data recorded in the archives of the node, enables or restricts information from being collected or archived, and (on data manager nodes) relays requests on to other nodes for further processing.

spdmapid executes on a node under the following conditions:

- The node is a target of an API request
- The node is a data manager node, and at least one of the nodes is a target of an API request
- The node is the g, and at least one node in the monitoring hierarchy is targeted for an API request.

spdmapid will not execute on a node under the following conditions:

- PTPE has been uninstalled on the node,
- The daemon's binary file has been removed from its default location in the **/usr/lpp/ptpe/bin** directory,
- The node is not the target of an API request, and is not the data manager of a node targeted for an API request,
- The **spdmd** daemon failed to execute on the node when collection was started or initialization was started

Only the final condition is in error. Use the methods described in “spdmd” on page 323 to determine the cause of the **spdmd** daemon error.

spdmapid may start, but fail, under the following conditions:

- The daemon did not receive all the input it expected from the socket connection to its data manager or to the application
- The daemon was executing on the g, and could not establish connections to any of the data managers that needed to be contacted to carry out the request
- The daemon was executing on a data manager node, and could not establish connections to any of the reporter nodes that were targeted for the request
- The API requested to set controls for collection and archiving, but the **spdmspld** daemon could not attach to the shared memory areas used by the **spdmcold** and **spdmspld** daemons on the node to store control information, indicating that these other daemons may have terminated
- The API requested to set controls for collection and archiving, and the **spdmcold** or **spdmspld** binary files have been removed from their default location in the **/usr/lpp/ptpe/bin** directory
- The API requested performance information stored in the archive, but the daemon could not open the statistics table file **/var/adm/ptpe/perftab**
- The API requested performance information stored in the archive, but the daemon could not open the archive file **/var/adm/ptpe/perflog**

- The daemon cannot report its results to its superior node, or to the library if the daemon is executing on the g

In each of the following cases, the **spdmapid** daemon should create a **perfmon** resource entry in the node's error log to explain the reason for the daemon's failure.

Symptoms of spdmapid failure include:

- A failure reported for the node during the statistics enablement phase of collection start, or a failure reported while starting or stopping archiving from the **ptpectrl** command
- The **PtpeGetHostResult** command reports a PTPE_NO_EXEC or PTPE_DAEMON_ERROR error for the node after an API request has been made to the node
- An application request returns a code of PTPE_BAD_SEND, PTPE_BAD_RECEIVE, PTPE_SUCCESS_BADR, PTPE_LIMITED_BADR, or PTPE_API_FAILED_BADR, which may indicate a failure in the **spdmapid** daemon on the g node

spdmtrmd

The PTPE termination daemon runs on a node whenever performance information collection is being shut down. It is responsible for forcing the other PTPE daemons to exit.

spdmtrmd will not execute under the following conditions:

- PTPE has been uninstalled on the node,
- The daemon's binary file has been removed from its default location in the **/usr/lpp/ptpe/bin** directory,
- The **spdmd** daemon failed to execute on the node when collection was started or initialization was started.

Only the final condition is in error. Use the methods described in "spdmd" on page 323 to determine the cause of the **spdmd** daemon error.

spdmtrmd may start, but fail, under the following conditions:

- The daemon did not receive all the input it expected from the socket connection during startup, or the socket connection was unexpectedly terminated
- The **/etc/perf/spdm.pid** file cannot be located on the node, making it impossible for the daemon to locate the **spdmspld** and **spdmcold** daemons
- The **/etc/perf/spdm.pid** file exists, but the daemon cannot send a termination signal to the processes listed in that file, implying that the **spdmcold** and **spdmspld** daemons are no longer active on the node
- The daemon cannot report its status to its superior node, or to the library if the daemon is executing on the g

In each of the following cases, the **spdmtrmd** daemon should create a **perfmon** resource entry in the node's error log to explain the reason for the daemon's error. The following information should also be checked on a node where the **spdmtrmd** daemon has failed abnormally:

- The **/etc/perf/spdm.pid** file. This file should exist if the **spdmcold** or **spdmspld** daemons are active on the node. The file contains the process identifiers of these daemons, as well as the shared memory segment identifier for the statistics table used by the **spdmcold** daemon.

If the file does not exist, check if the **spdmcold** or **spdmspld** daemons are active, using the **ps** command. If the daemons are active, you will need to terminate these daemons manually, using the **kill** command. Do not terminate these daemons with the **kill -9** command option, because this will not permit the daemons to properly discard their shared memory, or remove their temporary work files before termination. This can lead later failures of the **spdmcold** or **spdmspld** daemons when collection is restarted within the monitoring hierarchy.

Symptoms of spdmtrmd failure include:

- The **spdmcold** or the **spdmspld** daemons remain operational on a node, even after the **ptpectrl -s** command or the **PtpeColStop**subroutine has been issued
- The **PtpeGetHostResult** routine returns a PTPE_NO_EXEC error for a node after the **PtpeColStop** routine has been executed

PTPE Messages

Not all PTPE messages are included in this listing.

- **Informational** messages provide status or usage information and do not require an explanation. These messages are not included here.
- **Error** messages indicate a possible problem or error has occurred. They require further explanation to help you understand the cause and recover from the situation. These messages are included, beginning on page 331.

The following message indicates that the system cannot find the message catalog:

```
Msg nnnn not found
```

This could be caused by an error or it might indicate that your language is not supported.

If this happens, perform these steps to access the catalog:

1. Check the LANG environmental variable. If it is coded **En_US, en_US** or **C.**,

   a. Check for a directory with your language variable name in **/usr/lib/nls/msg/** and ensure that it contains the message catalogs.

   If the LANG environmental variable is coded other than **En_US, en_US** or **C.**, then your language is not supported.

   a. In this case you can access the messages by creating a link from the directory with your language variable name in **/usr/lib/nls/msg/** to the **/usr/lib/nls/msg/en_US** directory. If you prefer, you can also link each message catalog separately.

Error Messages

2516-014 ptpectrl: Problem contacting the System Data Repository.

Explanation: Could not get a session with the System Data Repository (SDR), or cannot locate necessary information in the SDR.

Action: Check if SDR is off-line or damaged; verify that the classes SPDM and SPDM_NODES exist in the SDR

2516-015 ptpectrl: Cannot find the reporting hierarchy.

Explanation: Could not find SPDM_NODES objects in the System Data Repository, or error getting it from the SDR.

Action: Check if the SDR is off-line; verify that the SPDM_NODES class exists in the SDR; verify that SPDM_NODES class objects exist in the SDR; build a reporting hierarchy and save it

2516-016 ptpectrl: Cannot contact the Central Coordinator Node.

Explanation: Cannot establish a socket connection to the g node.

Action: Try again later; check that g node is up and not fenced off; have system administrator check for other network errors on the g; have system administrator tune the network

2516-017 ptpectrl: Not enough memory available.

Explanation: Memory allocation failed

Action: Try again later; have system administrator check for applications that are consuming large amounts of memory

2516-018 ptpectrl: Internal memory error.

Explanation: The ptpectrl command used its own memory incorrectly

Action: Note the conditions that caused the error and contact IBM Service

2516-019 ptpectrl: Cannot send data to the Central Coordinator Node.

Explanation: An error occurred during a write to the g node. Socket connection may have dropped unexpectedly

Action: Check that central coordinator node is up and not fenced off; have system administrator check for other network errors on the central coordinator node.

2516-020 ptpectrl: Central Coordinator Node did not confirm message.

Explanation: central coordinator node did not reply to the data sent to it in time. Socket may have dropped unexpectedly, or the **spdmcold** daemon on the central coordinator node may have been killed.

Action: Check that Central Coordinator Node is up and not fenced off; have system administrator check for other network errors on the central coordinator node; check error logs on the central coordinator node.

2516-021 ptpectrl: Central Coordinator Node confirmation was corrupted.

Explanation: The confirmation message from the Central Coordinator Node was not in the expected format - socket may have dropped unexpectedly, or some other application may have seized the socket

Action: Check that Central Coordinator Node is up and not fenced off; have system administrator check for other network errors on the Central Coordinator Node

2516-022 ptpectrl: *node name* **could not be contacted, or a failure occurred while running the spdmd daemon on that node.**

Explanation: The Performance Monitor daemon registered with **inetd** did not start, or failed after starting.

Action: Verify that the node is online; verify that the Performance Monitor is installed on the indicated node; verify that **spdmd** has an entry in both the **/etc/inetd.conf** and **/etc/services** files on the node; try again later; have system administrator check for other network errors on the node.

2516-023 ptpectrl: A failure occurred while running the spdmcold daemon on node *name***.**

Explanation: The collector daemon (**spdmcold**) did not start, or failed after starting.

Action: Verify that the Performance Monitor is installed on the g node; check that g is up and not fenced off; try again later; have system administrator check for other network errors on the g.

2516-024 ptpectrl: Central Coordinator Node replied with an invalid response.

Explanation: the g node sent back an unknown code to this command - another application may be sending information on the socket, or the socket may have dropped unexpectedly

Action: Check that g node is up and not fenced off;

check error logs on the g; have system administrator check for other network errors on the g.

2516-025 ptpectrl: Manager node *name* failed.

Explanation: The data manager node reported that it could not continue, or the g node lost contact with that manager node

Action: Revise the reporting hierarchy to use another system as the manager node for the group; check error logs on the failing manager node; verify that the manager node is up and not fenced off; have system administrator check for other network errors on the manager node.

2516-026 ptpectrl: System *node name* failed.

Explanation: The system reported that it could not continue, or that system's manager node lost contact with that system

Action: Check error logs on both the failing system and that system's manager node; verify that the system is up and not fenced off; have system administrator check for other network errors on the data manager node.

2516-027 ptpectrl: Performance information collection is running. The action you have requested cannot be carried out when performance information is running.

Explanation: Performance information collection is running - user cannot attempt a setup or another start while the collection is running.

Action: Run the setup or start after collection has been stopped.

2516-028 ptpectrl: Performance information collection is not running. The action you have requested cannot be carried out unless performance information collection is active.

Explanation: Performance information collection is not running - user cannot stop collection, since it isn't on

Action: Do not attempt to stop collection when it is not running.

2516-029 ptpectrl: Cannot obtain list of failing systems.

Explanation: An error occurred in reading the failing node list from the g node. This message will be paired with another message to indicate if the related startup, setup, or stop command succeeded or failed. However, some systems did not succeed in the attempt, but the command cannot isolate these failures for the user.

Action: Check error logs on all data manager nodes to see if a cause for the error may be logged there; verify that the g node is up and not fenced off.

2516-030 ptpectrl: Cannot determine current status of performance information collection.

Explanation: The Performance Monitor cannot determine if performance information collection is currently running on all systems.

Action: Issue

```
ptpectrl -s
```

to shut down all performance information collection; verify that a SPDM object class exists in the System Data Repository (SDR); verify that an object of the SPDM class exists in the SDR; verify that the SDR is online.

2516-031 ptpectrl: Performance information archiving is active. The action you have requested cannot be carried out while performance information is being archived.

Explanation: Performance information collection is not running - user cannot start archiving this information if it is not being collected first

Action: Start collecting performance information by issuing

```
ptpectrl -c
```

.

2516-032 ptpectrl: Performance information archiving is not active. The action you have requested cannot be carried out unless performance information archiving is active.

Explanation: An attempt was made to stop archiving performance information when this information was not being archived

Action: Do not attempt to stop archiving when it is not on.

2516-033 ptpectrl: All systems failed to start performance information archiving.

Explanation: **ptpectrl -r** failed on all systems in the reporting hierarchy

Action: Check error logs on all systems to determine the cause of the error; verify that performance information is being collected on all nodes.

2516-034 ptpectrl: All systems failed to stop performance information archiving.

Explanation: **ptpectrl -t** failed on all systems in the reporting hierarchy

Action: Check error logs on all systems to determine the cause of the error; verify that performance information is being collected on all nodes; verify that performance information is being archived on all nodes.

2516-035 ptpectrl: Setup failed.

Explanation: **ptpectrl -a** or **ptpectrl -i** failed

Action: Consult other error messages to help determine the cause of failure.

2516-036 ptpectrl: Could not start collecting performance information.

Explanation: **ptpectrl -a** or **ptpectrl -c** failed

Action: Consult other error messages to help determine the cause of failure.

2516-037 ptpectrl: Could not stop collecting performance information.

Explanation: **ptpectrl -s** failed

Action: Consult other error messages to help determine the cause of failure.

2516-038 ptpectrl: Could not start archiving performance information.

Explanation: **ptpectrl -r** failed

Action: Consult other error messages to help determine the cause of failure.

2516-039 ptpectrl: Could not stop archiving performance information.

Explanation: **ptpectrl -t** failed

Action: Consult other error messages to help determine the cause of failure.

2516-040 ptpectrl: Cannot locate message catalog.

Explanation: Message catalog **/usr/lib/nls/msg/**<**lang**>**/ptpectrl.cat** cannot be found, or cannot be opened

Action: Verify that the message catalog exists for your specific locale; verify that the message catalog has read permission.

2516-041 ptpectrl: Cannot use -s along with -a, -i, -c, -f, -l, -m, -n, -r or -v options.

Explanation: User used mutually exclusive parameters

Action: Use command with proper options

2516-042 ptpectrl: Cannot use -t along with -a, -i, -c, -f, -l, -m, -n, -r or -v options.

Explanation: User used mutually exclusive parameters

Action: Use command with proper options

2516-043 ptpectrl: Missing operands.

Explanation: User failed to provide any parameters to the command

Action: Use command with proper options

2516-044 ptpectrl: The Performance Toolbox Parallel Extensions Information Object, class SPDM, does not exist.

Explanation: The global information object for the Performance Toolbox Parallel Extensions does not exist in the System Data Repository (SDR).

Action: Create a reporting hierarchy; recreate the reporting hierarchy; check if SDR is off-line or damaged; verify that the class SPDM exists in the SDR

2516-045 ptpectrl: Another Performance Toolbox Parallel Extensions application is currently running. Please wait for that application to complete, then run this command again.

Explanation: Another application has exclusive use of the reporting hierarchy

Action: Wait for the application to complete, then try again later.

2516-046 ptpectrl: Unexpected error - *function name* **returned code** *function return code***.**

Explanation: An unanticipated error occurred, and the command could not proceed

Action: Note the function name and the return code, make note of the system conditions when the error occurred, and contact IBM Service.

2516-047 ptpectrl: Cannot use *option* **along with any other option.**

Explanation: User specified mutually exclusive parameters

Action: Use command with proper options

2516-048 ptpectrl: A failure occurred while running the spdmapid daemon on node *name*.

Explanation: The API handing daemon, **spdmapid**, could not be started on the indicated node, or failed after starting - the daemon might be missing, or the node may not be able to start any more processes.

Action: Check if the indicated node is unable to start any new processes; verify that the named node is up and not fenced off.

2516-049 ptpectrl: User is not authorized to use this command.

Explanation: User is not a member of the **perfmon** user group, or is not currently running as a member of the **perfmon** user group

Action: User must issue

```
newgrp perfmon
```

Add user to the **perfmon** user group; verify that the user is a member of the **perfmon** user group

2516-050 ptpectrl: The performance information archives could not be erased on the above systems.

Explanation: Attempt to erase the performance information archive on one or more nodes failed. Network traffic may have prevented the request.

Action: Rerun the request; make sure that the failing systems are up and not fenced off; have system administrator check for network errors on the failing systems

2516-051 ptpectrl: Performance information collection is currently in Error state. Please take corrective action.

Explanation: The state of performance information collection is unclear - some systems may be collecting information while others are not.

Action: Issue

```
ptpectrl -s
```

to shut down all performance information collection; verify that a SPDM object class exists in the System Data Repository (SDR); verify that an object of the SPDM class exists in the SDR; verify that the SDR is online; note the conditions that caused the error and contact IBM service.

2516-052 ptpectrl: Performance information archiving is currently in Error state. Please take corrective action.

Explanation: The state of performance information archiving is unclear - some systems may be archiving information while others are not.

Action: Issue

```
ptpectrl -t
```

to shut down all performance information collection; verify that a SPDM object class exists in the System Data Repository (SDR); verify that an object of the SPDM class exists in the SDR; verify that the SDR is online; note the conditions that caused the error and contact IBM service.

2516-053 ptpectrl: Cannot determine current status of performance information archiving.

Explanation: The Performance Monitor cannot determine if performance information collection is currently running on all systems.

Action: Issue

```
ptpectrl -s
```

to shut down all performance information collection; verify that a SPDM object class exists in the System Data Repository (SDR); verify that an object of the SPDM class exists in the SDR; verify that the SDR is online.

2516-054 ptpectrl: *command line* option was provided more than once.

Explanation: The caller used the same option on the command line more than once.

Action: Issue the command with the proper combination of parameters; Issue

```
ptpectrl -h
```

to view the proper command combinations.

2516-055 ptpectrl: *option* is not a valid interval specification for the -f option.

Explanation: The interval specification for the **-f** option does not conform to the expected format of <number>,<number>.

Action: Issue the command with the proper interval specification.

2516-056 ptpectrl: Could not alter the performance information reporting and recording frequencies.

Explanation: An error occurred within the reporting hierarchy that prevented the command from updating the data reporting and recording intervals. Some of the systems may have modified their frequencies, but some may not have updated their intervals.

Action: Attempt the command again; verify that all nodes in the reporting hierarchy are operational and not fenced off; verify that the data sampling and collection daemons are active on all systems in the reporting hierarchy.

2516-057 ptpectrl: Could not obtain the performance information reporting and recording frequencies.

Explanation: Could not retrieve the current performance information reporting and recording frequencies from the System Data Repository.

Action: Verify that a SPDM object class exists in the System Data Repository (SDR); verify that an object of the SPDM class exists in the SDR; verify that the SDR is online.

2516-058 ptpectrl: Cannot locate a statistics list file in the System Data Repository. Initialize the reporting hierarchy with ptpectrl -i.

Explanation: The collection start command cannot locate the list of statistics in the System Data Repository. The most likely cause of the error is that **ptpectrl -i** was not executed before **ptpectrl -c**. Another cause would be if the file **SPDM_STATS** was removed from the System Data Repository with the SDRDeleteFile command.

Action: Issue the

```
ptpectrl -i
```

or the

```
ptpectrl -a
```

command to initialize the hierarchy; verify that the **SPDM_STATS** file is not being removed from the System Data Repository.

2516-059 ptpectrl: The System Data Repository cannot create a copy of the statistics list file on this node. The /tmp filesystem may be full.

Explanation: The System Data Repository (SDR) was unable to create an AIX file to contain the contents of the SDR's **SPDM_STATS** file. The command attempts to create this file in **/tmp**.

Action: Verify that there is a reasonable amount of space available in the **/tmp** filesystem.

2516-060 ptpectrl: Node *name* was not ready to accept the request.

Explanation: The command could not be carried out on the named node because that system was not ready to accept requests at that point in time. The delay is most likely the result of a Spmi delay.

Action: Reissue the command at a later time.

2516-061 ptpectrl: Node *name* reported that performance information archiving is active on that node.

Explanation: The command could not complete successfully because the named node is archiving performance information.

Action: Shut down archiving before attempting the command; verify that you are issuing the proper command.

2516-063 ptpectrl: The performance data collection daemons have terminated

Explanation: While attempting a command on the node, the request handling daemon found that the collector daemon (**spdmcold**), the data sampling daemon (**spdmspld**), or both, were no longer running. The request could not be completed on this node.

Action: Examine earlier output from the command for any indication of error on this node; examine the error logs on the node for information that might explain why the daemons shut down; perform the appropriate corrective action for any errors found on the node.

2516-064 *Daemon name*: **The following node is not a member of the currently active system partition:** *hostname*.

Explanation: A node has been included in the monitoring hierarchy, but that node is not a member of the system partition that is currently active.

Action: Verify that the command is executing in the proper system partition; verify that the node is a member of the system partition; verify that the name used to identify the node within the monitoring hierarchy is also the reliable_hostname for the node in the System Data Repository; repair the monitoring hierarchy; remove the node from the monitoring hierarchy

2516-075 ptpectrl: Cannot use -m along with -a, -c, -e, -h, -i, -q, -r, -s, -t, or -v options.

Explanation: User used mutually exclusive parameters

Action: Use command with proper options

2516-076 ptpectrl: Some of the statistics listed in the file *configuration filename* could not be found on *node name*.

Explanation: The node is not capable of providing some of the statistics listed in the configuration file. Therefore, some of the statistics were not enabled or disabled as the configuration file requested. The statistics that could be found were enabled or disabled as requested.

Action: none

2516-077 ptpectrl: Cannot locate or open configuration file.

Explanation: The **ptpectrl -v** command cannot locate a **ptpe.cf** file, either in the user's HOME directory, or in the default **/etc/perf** directory

Action: Verify that the **ptpe.cf** file exists, either in the user's HOME directory or the default **/etc/perf** directory; ensure that the **ptpe.cf** file allows read permission to the root user; ensure that the access control lists (ACLs) for the directory containing the **ptpe.cf** file permit read permission to the root user.

2516-078 ptpectrl: Cannot access or open file *filename* for reading.

Explanation: The file cannot be accessed, or cannot be opened for reading by the **ptpectrl** command.

Action: Ensure that the file name is correct; ensure that the file exists has read permissions set; check the access control lists (ACLs) for the directory and make sure the root user has read permission on the directory containing the file.

2516-079 ptpectrl: Cannot use *option* with the *option* option.

Explanation: User used mutually exclusive parameters

Action: Use command with proper options

2516-080 ptpectrl: Node *name* reported a general error. Check the error logs on that node for more information.

Explanation: The indicated node reported a general failure. The error log entries on that node may contain additional information that would indicate the actual error.

Action: Consult the error logs on the indicated node for more information; make sure the node is up and not fenced off; check the node for networking errors; check if performance information collection is active for the node; check that the Performance Toolbox Parallel Extensions daemons are installed and available on that node.

2516-081 ptpectrl: Archiving is not active on node *name*

Explanation: The **ptpectrl** command attempted to enable or disable statistics for archiving on the indicated node, but performance information archiving does not appear to be active on this node.

Action: Verify that performance information archiving is active; check the error log on the indicated node to see if an error occurred while starting archiving; check if **/var** on the indicated node is nearly full.

2516-082 ptpectrl: None of the statistics listed in the *configuration file* exist on node *name*.

Explanation: The **ptpectrl** command attempted to enable or disable statistics for collection and/or archiving on the indicated node, but none of the statistics listed in the **ptpe.cf** file could be found on that node.

Action: Verify that the **ptpe.cf** file entries are correct; if the reporting hierarchy has changed recently, execute **ptpectrl -i** to initialize the hierarchy.

2516-083 ptpectrl: The following systems reported a failure in disabling statistics for archiving:

Explanation: Some systems in the reporting hierarchy were unable to enable performance statistics for archiving. A list of these systems may follow this message, if the **ptpectrl** command was able to receive this information from the g node.

Action: Examine the failures for each node for clues on how to repair the problem; if the hierarchy has been modified recently, issue

```
ptpectrl -i
```

to initialize the hierarchy; check for networking errors.

2516-084 ptpectrl: The following systems reported a failure in enabling statistics for archiving:

Explanation: Some systems in the reporting hierarchy were unable to enable performance statistics for archiving. A list of these systems may follow this message, if the **ptpectrl** command was able to receive this information from the g node.

Action: Examine the failures for each node for clues on how to repair the problem; if the hierarchy has been modified recently, issue

```
ptpectrl -i
```

to initialize the hierarchy; check for networking errors.

2516-085 ptpectrl: The following systems reported a failure in disabling statistics for collection.

Explanation: Some systems in the reporting hierarchy were unable to enable performance statistics for collection. A list of these systems may follow this message, if the **ptpectrl** command was able to receive this information from the g node.

Action: Examine the failures for each node for clues on how to repair the problem; if the hierarchy has been modified recently, issue

```
ptpectrl -i
```

to initialize the hierarchy; check for networking errors.

2516-086 ptpectrl: The following systems reported a failure in enabling statistics for collection.

Explanation: Some systems in the reporting hierarchy were unable to enable performance statistics for collection. A list of these systems may follow this message, if the **ptpectrl** command was able to receive this information from the g node.

Action: Examine the failures for each node for clues on how to repair the problem; if the hierarchy has been modified recently, issue

```
ptpectrl -i
```

to initialize the hierarchy; check for networking errors.

2516-087 ptpectrl: All systems reported a failure in disabling statistics for archiving.

Explanation: All systems in the reporting hierarchy were unable to enable performance statistics for archiving. A list of these systems may follow this message, if the **ptpectrl** command was able to receive this information from the g node.

Action: Examine the failures for each node for clues on how to repair the problem; if the hierarchy has been modified recently, issue

```
ptpectrl -i
```

to initialize the hierarchy; check for networking errors.

2516-088 ptpectrl: All systems reported a failure in enabling statistics for archiving.

Explanation: All systems in the reporting hierarchy were unable to enable performance statistics for archiving. A list of these systems may follow this message, if the **ptpectrl** command was able to receive this information from the g node.

Action: Examine the failures for each node for clues on how to repair the problem; if the hierarchy has been modified recently, issue

```
ptpectrl -i
```

to initialize the hierarchy; check for networking errors.

2516-089 ptpectrl: All systems reported a failure in disabling statistics for collection.

Explanation: All systems in the reporting hierarchy were unable to enable performance statistics for collection. A list of these systems may follow this message, if the **ptpectrl** command was able to receive this information from the g node.

Action: Examine the failures for each node for clues on how to repair the problem; if the hierarchy has been modified recently, issue

```
ptpectrl -i
```

to initialize the hierarchy; check for networking errors.

2516-090 ptpectrl: All systems reported a failure in enabling statistics for collection.

Explanation: All systems in the reporting hierarchy were unable to enable performance statistics for collection. A list of these systems may follow this message, if the **ptpectrl** command was able to receive this information from the g node.

Action: Examine the failures for each node for clues on how to repair the problem; if the hierarchy has been modified recently, issue

```
ptpectrl -i
```

to initialize the hierarchy; check for networking errors.

2516-091 ptpectrl: Could not enable all statistics for archiving.

Explanation: An error occurred while attempting to set up the command to enable all statistics for archiving, or an error occurred in the reporting hierarchy while enabling all statistics. Messages that precede this message may indicate the source of the error.

Action: Consult the recommended actions for any of the preceding messages.

2516-092 ptpectrl: Could not enable all statistics for collection.

Explanation: An error occurred while attempting to set up the command to enable all statistics for collection, or an error occurred in the reporting hierarchy while enabling all statistics. Messages that precede this message may indicate the source of the error.

Action: Consult the recommended actions for any of the preceding messages.

2516-093 ptpectrl: Could not selectively restrict statistics from the *configuration file* **file for archiving.**

Explanation: An error occurred while attempting to set up the command to disable specific statistics for archiving, or an error occurred in the reporting hierarchy while disabling these statistics. Messages that precede this message may indicate the source of the error.

Action: Consult the recommended actions for any of the preceding messages.

2516-094 ptpectrl: Could not selectively enable statistics from the *configuration file* **for archiving.**

Explanation: An error occurred while attempting to set up the command to enable specific statistics for archiving, or an error occurred in the reporting hierarchy while enabling these statistics. Messages that precede this message may indicate the source of the error.

Action: Consult the recommended actions for any of the preceding messages.

2516-095 ptpectrl: Could not selectively restrict statistics from the *configuration file* **file for collection.**

Explanation: An error occurred while attempting to set up the command to disable specific statistics from collection, or an error occurred in the reporting hierarchy while disabling these statistics. Messages that precede this message may indicate the source of the error.

Action: Consult the recommended actions for any of the preceding messages.

2516-096 ptpectrl: Could not selectively enable statistics from the *configuration file* **for collection.**

Explanation: An error occurred while attempting to set up the command to enable specific statistics for collection, or an error occurred in the reporting hierarchy while enabling these statistics. Messages that precede this message may indicate the source of the error.

Action: Consult the recommended actions for any of the preceding messages.

2516-097 ptpectrl: None of the statistics listed in the configuration file *configuration file* **were found in the statistics list used by the reporting hierarchy.**

Explanation: No matches could be found for any of the entries provided in the **ptpe.cf** file.

Action: Verify the entries in the **ptpe.cf** file; ensure that any wildcards used in statistic names are used correctly.

2516-098 ptpectrl: *ptpe.cf file entry* **is not a valid entry - ignoring.**

Explanation: The entry within the **ptpe.cf** file is not in a valid format.

Action: Correct the format of the entry in the **ptpe.cf** file.

2516-099 ptpectrl: Statistic *statistic name* **not known - ignoring.**

Explanation: A statistic name, listed in the user's **ptpe.cf** file, cannot be located in the summary statistics list file retrieved from the System Data Repository.

Action: Make sure the statistic name is correct; replace the name, if incorrect, with the correct statistic name.

2516-100 ptpectrl: Cannot open the following file for reading: *filename*

Explanation: The command created a temporary file and attempted to open that file as a read-only file, but the open attempt failed.

Action: Rerun the command. If the condition persists, verify that no other commands, such as **skulker**, are running and possibly removing this file before it can be used.

2516-120 ptpehier: Cannot specify *option* **with any other arguments.**

Explanation: User specified a command line option that does not permit other options, along with other options

Action: Rerun command with proper parameters; check command usage

2516-121 ptpehier: Cannot use “-e” , “-f,” or “-i” options together.

Explanation: User specified mutually exclusive parameters on the command line

Action: Rerun without mutually exclusive parameters; check command usage

2516-122 ptpehier: Another Performance Toolbox Parallel Extensions application is currently running. Please wait for that application to complete, then run this command again.

Explanation: Another application has exclusive use of the reporting hierarchy

Action: Wait for the application to complete, then try again later.

2516-123 ptpehier: The Performance Toolbox Parallel Extensions Information Object, class SPDM, does not exist.

Explanation: The global information object for the Performance Toolbox Parallel Extensions does not exist in the System Data Repository (SDR).

Action: Create a reporting hierarchy; recreate the reporting hierarchy; check if SDR is off-line or damaged; verify that the class SPDM exists in the SDR

2516-124 ptpehier: Unexpected System Data Repository error.

Explanation: Could not get a session with the System Data Repository (SDR), cannot locate necessary information in the SDR.

Action: Check if SDR is off-line or damaged; verify that the filesystem used by the SDR has some space available; verify that the classes SPDM and SPDM_NODES exist in the SDR

2516-125 ptpehier: Cannot find the reporting hierarchy.

Explanation: Could not find SPDM_NODES objects in the System Data Repository, or error getting it from the SDR.

Action: Check if the SDR is off-line; verify that the SPDM_NODES class exists in the SDR; verify that SPDM_NODES class objects exist in the SDR; build a reporting hierarchy and save it

2516-126 ptpehier: Not enough memory available.

Explanation: Memory allocation failed

Action: Try again later; have system administrator check for applications that are consuming large amounts of memory

2516-127 ptpehier: Unexpected error - *function name* **returned code** *function return code***.**

Explanation: An unanticipated error occurred, and the command could not proceed

Action: Note the function name and the return code, make note of the system conditions when the error occurred, and contact IBM Service.

2516-128 ptpehier: Performance information collection is running.

Explanation: Performance information collection is running - user cannot attempt to alter the hierarchy while collection is active.

Action: Stop collection and then modify the hierarchy

2516-129 ptpehier: Cannot determine current status of performance information collection.

Explanation: The Performance Monitor cannot determine if performance information collection is currently running on all systems.

Action: Issue

```
ptpectrl -s
```

to shut down all performance information collection; verify that a SPDM object class exists in the System Data Repository (SDR); verify that an object of the SPDM class exists in the SDR; verify that the SDR is online.

2516-130 ptpehier: The host specified as the preferred Central Coordinator: *node name* **is either not listed in the System Data Repository Nodes object class, or has not installed the proper level of Parallel System Support Programs.**

Explanation: The hostname specified as the preferred g cannot be found in the System Data Repository (SDR), or the node does not have PSSP version 2.2 or later installed.

Action: Verify the hostname. Ensure that the hostname is the same one used by the System Data Repository to refer to that node. Verify that the node has version 2.2 of the Parallel System Support Programs installed. Ensure that the information listed for the node in the System Data Repository's Node class is correct. Verify the reporting hierarchy input.

2516-131 ptpehier: Too many arguments provided, or options are missing.

Explanation: The caller provided too many arguments to the command - the command accepts only one argument after all options have been specified, which is the name of the system to use as the central coordinator.

Action: Check command usage

2516-132 ptpehier: Invalid option *option* provided.

Explanation: User specified an unknown option to this command

Action: Specify proper options; run **ptpehier -h** to determine valid options

2516-133 ptpehier: Missing required operands.

Explanation: User failed to supply at least one of the required operands

Action: Specify proper options; run **ptpehier -h** to determine valid options

2516-134 ptpehier: Unable to print reporting hierarchy.

Explanation: The **ptpehier** command encountered an error while printing the reporting hierarchy

Action: Make sure a reporting hierarchy exists by examining the SPDM_NODES object class in the System Data Repository; check that the system has not run out of virtual memory

2516-135 ptpehier: Error in hierarchy input.

Explanation: The reporting hierarchy input provided to the command contains a syntax error

Action: Check for badly placed "{" and "}" characters in the input; ensure that all "{" and "}" characters are on a separate line and not preceded by spaces, tabs, or other characters; make sure that at least one host name is listed between each set of "{" and "}" lines; check other output from the command for clues to the error

2516-136 ptpehier: A Central Coordinator name is required for this command, but one was not provided on the command line.

Explanation: The user failed to specify a central coordinator node when one was required.

Action: Verify the command parameters; provide the central coordinator node name on the command line and rerun the command.

2516-137 ptpehier: *node name* has been designated both a data manager node and the Central Coordinator.

Explanation: The user specified a host as the central coordinator of the reporting hierarchy on the command line, but also specified it as a group manager in the input.

Action: Change the name of the central coordinator on the command line and rerun the command; change the order of the hosts specified for a node group in the input and rerun the command.

2516-138 ptpehier: *option* option was provided more than once.

Explanation: The caller used the same option on the command line more than once.

Action: Issue the command with the proper combination of parameters; Issue

```
ptpehier -h
```

to view the proper command combinations.

2516-139 ptpehier: User is not authorized to use this command.

Explanation: User is not a member of the **perfmon** user group, or is not currently running as a member of the **perfmon** user group

Action: User must execute

```
newgrp perfmon
```

add user to the **perfmon** user group; verify that the user is a member of the **perfmon** user group

2516-140 ptpehier: The host specified as the preferred Central Coordinator: *node name* was not also listed in the reporting hierarchy input.

Explanation: The host name provided does not appear in the reporting hierarchy input, provided by the user through standard input.

Action: Verify the reporting hierarchy input.

2516-141 ptpehier: The following host is either not listed in the System Data Repository, or has not installed the proper level of the Parallel System Support Programs.

Explanation: The specified hostname cannot be found in the System Data Repository (SDR), or the node does not have PSSP version 2.2 or later installed.

Action: Verify the hostname. Ensure that the hostname is the same one used by the System Data Repository to refer to that node. Verify that the node has version 2.2 of the Parallel System Support Programs installed. Ensure that the information listed for the node in the System Data Repository's Node class is correct. Verify the reporting hierarchy input.

2516-142 ptpehier: Syntax error in the input. The last node group was not terminated with the "}" marker.

Explanation: The user's input contained an error. The final node group in the hierarchy was not terminated with the end of group marker character, "}."

Action: Verify the input format; rerun the command with the proper input.

2516-143 ptpehier: Syntax error in line *number* **of the input. "{" or end of input marker expected. Contents of the failing line:** *contents***.**

Explanation: The user's input contained an error. The "{" character was expected, but was not found, or did not appear alone on a line.

Action: Verify the input format; rerun the command with the proper input.

2516-144 ptpehier: Syntax error in line *number* **of the input. "{" not expected within a group definition. Contents of the failing line:** *contents***.**

Explanation: The user's input contained an error. The "{" character cannot appear within a group definition, before the "}" character to close the group definition.

Action: Verify the input format; rerun the command with the proper input.

2516-145 ptpehier: Syntax error in line *number* **of the input. No members were provided for a group.**

Explanation: The user's input contained an error. An empty group was specified in the input. At least one host name must appear between the "{" and "}" lines in the input.

Action: Verify the input format; rerun the command with the proper input.

2516-147 ptpehier: The following host name was provided more than once: *nodename*

Explanation: A host was listed more than once within the hierarchy input.

Action: Correct the input to provide the host name once, and retry the command.

2516-148 ptpehier: Node number *number* **does not have an entry in the System Data Repository. This node will not be included in the monitoring hierarchy.**

Explanation: A node number was listed in the Adapter class of the System Data Repository (SDR), but an entry for this node number did not exist in the node class in the SDR.

Action: To include this node in the monitoring hierarchy, update the SDR by adding an entry in the Node class for the number listed, then reconstruct the hierarchy.

2516-210 ptpeconf: Invalid option *option* **provided.**

Explanation: User specified an unknown option to this command

Action: Specify proper options; run **ptpeconf -h** to determine valid options

2516-211 ptpeconf: Cannot specify *option* **and** *option* **options at the same time.**

Explanation: User specified mutually exclusive parameters on the command line

Action: Rerun without mutually exclusive parameters; check command usage

2516-212 ptpeconf: Cannot establish session with the System Data Repository server.

Explanation: **SDROpenSession** returned with an error code of 80, indicating that it could not connect to the System Data Repository server

Action: Check if the SDR is off-line or damaged; contact the system administrator and report the SDR problem.

2516-213 ptpeconf: System Data Repository authorization failure.

Explanation: **SDROpenSession** returned with an error code of 82, indicating that the application or the user is not authorized to use the SDR

Action: Ensure that the user has sufficient privileges to use the SDR; ensure that the permissions on the **ptpeconf** binary are correct

2516-214 ptpeconf: Problem contacting the System Data Repository.

Explanation: Could not get a session with the System Data Repository (SDR).

Action: Check if SDR is off-line or damaged; contact the system administrator and report difficulty in using the SDR.

2516-215 ptpeconf: Unexpected error - *function name* **returned code** *return code***.**

Explanation: An unanticipated error occurred, and the command could not proceed.

Action: Note the function name and the return code, make note of the system conditions when the error occurred, and contact IBM Service.

2516-216 ptpeconf: *option* **option was provided more than once.**

Explanation: The caller used the same option on the command line more than once.

Action: Issue the command with the proper combination of parameters; Issue **ptpeconf -h** to view the proper command combinations.

2516-310 ptpegroup: This command can only be executed by the root user.

Explanation: Command was executed by a user without root permissions

Action: Rerun the command as root or a user with equivalent permissions.

2516-311 ptpegroup: The user group perfmon already exists.

Explanation: The user group that the command would create already exists

Action: Make sure that this command is being run on the proper system; check that the **/etc/group** and **/etc/security/group** files are not corrupted; ensure that the **/etc/group** and **/etc/security/groups** files agree on a list of valid user groups.

2516-312 ptpegroup: Incorrect number of parameters.

Explanation: The user invoked the command with more than one parameter.

Action: use the command with the proper parameters.

2516-313 ptpegroup: Invalid parameter specified: *parameter***.**

Explanation: The user provided a parameter other than **-h**.

Action: invoke command with the proper parameters.

2516-314 ptpegroup: Cannot create group perfmon. Exit code from mkgroup is *exit code***.**

Explanation: **mkgroup** failed to create the user group.

Action: Check that the **/etc/group** and **/etc/security/group** files are not corrupted; ensure that **/etc/group** and **/etc/security/groups** files agree on a list of valid user groups.

2516-315 ptpegroup: Cannot modify perfmon administrator. Exit code from chgrpmem command is *exit code***.**

Explanation: **chgrpmem** failed to modify the administrator of the user group.

Action: Check that the **/etc/group** and **/etc/security/group** files are not corrupted; ensure that the **/etc/group** and **/etc/security/groups** files agree on a list of valid user groups.

2516-316 ptpegroup: Cannot add users to perfmon group. Exit code from chgrpmem command is *exit code***.**

Explanation: **chgrpmem** failed to add users to the user group.

Action: Check that the **/etc/group** and **/etc/security/group** files are not corrupted; ensure that the **/etc/group** and **/etc/security/groups** files agree on a list of valid user groups.

2516-360 ptpedump: This command can only be executed by members of the perfmon user group.

Explanation: command was executed by a user without **perfmon** group permissions

Action: Verify that the user is a member of the **perfmon** user group; execute **newgrp** to change to the **perfmon** user group; verify that the **perfmon** user group exists; ensure that the **/etc/security/group** and **/etc/group** files agree that the **perfmon** user group exists.

2516-361 ptpedump: Invalid parameter specified: *parameter***.**

Explanation: The user provided a parameter other than **-h** or **-n**

Action: Invoke command with the proper parameters

2516-362 ptpedump: Unknown host ***node name*****.**

Explanation: The host name specified by the user is unknown to this system

Action: Verify that the host name is correct; reissue the command using the proper host name; verify that the host is operational and connected to the network

2516-363 spdm_dump: Cannot allocate memory for archive translation tables.

Explanation: The program could not allocate enough memory to store the translation tables used by the performance archive file.

Action: Contact the system administrator to report the shortage of memory.

2516-364 spdm_dump: Cannot read translation table information from the performance information archive file ***filename*****.**

Explanation: The program could not read the translation table from the archive file.

Action: The archive file may be corrupted or truncated; check the error log for filesystem errors with this filesystem.

2516-365 spdm_dump: Cannot locate or open the performance information archive file ***filename*****.**

Explanation: The program could not open the performance information archive file for reading.

Action: Ensure that the file exists; ensure that the file grants read permissions; ensure that the filesystem containing the file is mounted.

2516-366 ptpedump: Performance information archive files do not exist on the systems listed below. A text archive file will not be created on these systems.

Explanation: The file **/var/adm/ptpe/perflog** does not exist on one or more systems where **ptpedump** was to execute.

Action: Make sure that the system was expected to have a performance information archive file on it; determine if the archive file was previously removed by the **ptpectrl -e** command.

2516-367 ptpedump: Parameter ***parameter*** **specified more than once.**

Explanation: Parameter was specified more than once on the command line.

Action: Invoke command with the proper parameters.

2516-368 ptpedump:Parameters “-c” and “-s” cannot be used together.

Explanation: User specified mutually exclusive parameters to the command.

Action: Use the **-h** parameter to view the acceptable parameter to the **ptpedump** command; issue the **ptpedump** command with the proper parameters.

2516-369 ptpedump: Host names were not provided.

Explanation: User specified the **ptpedump -n** command, but did not provide any host names on the command line.

Action: Reissue the command with the appropriate parameters and host names.

2516-401 ptpertm: Cannot start process ***resource monitor name*****.**

Explanation: The named resource monitor could not be started.

Action: Check that the data supplier does exist out in **/usr/lpp/ssp/bin/haemRM** file system. Check with administrator for possible network problems. Look into error log for any possible network, socket errors.

2516-402 ptpertm: Cannot send command ***command name*** **to resource monitor.**

Explanation: Unable to send the specified command to the resource monitor.

Action: Check with administrator for possible network problems. Make sure no other process's are trying to use the same port.

2516-403 ptpertm: Unexpected error in the select() system call - error code ***return code*****.**

Explanation: Select error.

Action: The socket the select is being done on, could have closed. Check with administrator for possible network problems. Look into error log for any possible network, socket errors. If problem persists, contact IBM Service.

2516-404 ptpertm: Resource monitor daemon did not reply within ***timeout period*** **seconds after a command was issued to it. Daemon may be hung or dead.**

Explanation: The resource monitor daemon did not reply with a confirmation message after a command was sent to it.

Action: Verify that the resource monitor daemon is running on this node. Check the error logs on this node for any error message from the resource monitor daemon. Check for network problems or high system loads that might have delayed the resource monitor from running.

2516-405 ptpertm: An error occurred receiving the response from the resource monitor daemon: *error message***.**

Explanation: An error occurred reading the response from the resource monitor daemon through the socket connection to it. The socket may have been closed, the resource monitor daemon may have terminated prematurely, or high system loads may have prevented the data from being read.

Action: Verify that the resource monitor daemon is running on this node. Check the error logs on this node for any error message from the resource monitor daemon. Check for network problems or high system loads that might have delayed the resource monitor from running.

2516-406 ptpertm: Did not receive a full reply from the resource monitor daemon (*number* **of** *number* **expected bytes were received).**

Explanation: An incomplete response was received from the resource monitor daemon. The resource monitor daemon may have terminated prematurely, or the socket connection to the resource monitor daemon may have been dropped.

Action: Verify that the resource monitor daemon is running on this node. Check the error logs on this node for any error message from the resource monitor daemon. Check for network problems or high system loads that might have delayed the resource monitor from running.

2516-407 ptpertm: Cannot create a socket to communicate with the resource monitor daemon: *system error message***.**

Explanation: The **socket()** system call failed. The user may not have any open file descriptors available, but this is unlikely. Check the text of the error message for more information.

Action: Perform any corrective action associated with the error described in the message text. If the problem persists, contact IBM Service.

2516-408 ptpertm: Cannot connect to socket file *name of resource monitor's socket file: system error message***.**

Explanation: The **connect()** system call failed. A socket connection to the resource monitor daemon could not be established. The resource monitor daemon may not be active, or the socket may have been dropped.

Action: Check the node's error logs for possible error messages from the resource monitor daemon, or for indications of network problems.

2516-409 ptpertm: Error sending command *command code***/of size** *size of control message* **to resource monitor daemon (***number* **bytes sent). Error message:** *system error message*

Explanation: Error sending command message through the socket to the resource monitor daemon. The resource monitor daemon may have terminated prematurely, or the socket connection may have been dropped.

Action: Check the error log on this node for any possible error messages from the resource monitor daemon. Verify that the resource monitor daemon is running on this node. Check the error log for other network related problems. If problem persists, contact IBM service.

2516-410 ptpertm: Command terminated by unexpected signal *symbolic name* **(***numeric value of signal***).**

Explanation: The command received a signal that it did not expect. The command may have encountered an error that caused the generation of the signal, or the process may have been terminated by the **kill** command by another user.

Action: Check error output of the error logs on the node for any indications of errors encountered by the command before it terminated and perform any associated corrective action. Check if another user terminated the command. If the problem persists, contact IBM Service.

2516-411 ptpertm: Could not instruct the resource monitor daemon *daemon name* **to stop sending performance information.**

Explanation: The command could not instruct the resource monitor daemon to stop sending performance information to Spmi. The resource monitor daemon may have terminated prematurely, or an error may have occurred while sending data to or receiving confirmation from the daemon.

Action: Check the error output of the error logs on this node for other error messages from the command or the resource monitor daemon to determine the actual cause of the error. Verify that the resource monitor daemon was active at the time the command attempted to send this instruction. Check for socket or network errors. If the problem persists, contact IBM Service.

2516-412 ptpertm: Cannot allocate memory to store the node's partition name.

Explanation: The command failed to allocate memory to store the node's partition name. Another process may be consuming large amounts of virtual memory, but this is an unlikely cause.

Action: Use the **vmstat** command or Performance Toolbox for AIX to check if the amount of free virtual memory is low, and if the system is doing a large amount of paging. If the virtual memory figures do not show a problem, this error is most likely caused by an error in the command's source code - contact IBM Service.

2516-413 ptpertm: Cannot obtain the node's partition name.

Explanation: The command cannot obtain the node's partition name from the System Data Repository (SDR). If the command is being executed from the command line, the user may not have sufficient privilege to use SDR commands. The SDR server may also be offline, or network problems may be prohibiting the command from contacting the SDR server.

Action: If running the command from the command line, the user should verify that the user's account has sufficient privilege to use System Data Repository commands. Verify that the SDR is online, and that this node is not fenced off from the rest of the partition. Check in the error log on this node for messages that may indicate network problems with this node, and also check for any error log entries from the System Data Repository.

2516-414 ptpertm: Cannot invoke the resource monitor daemon *name of resource monitor daemon***.**

Explanation: The resource monitor daemon could not be started by this command, or the command could not determine whether or not the daemon was already running.

Action: Check other error output or error log entries from this command for indications of the actual cause of the error, and take the appropriate corrective action.

2516-415 ptpertm: Cannot instruct the resource monitor daemon to begin sending all its performance information.

Explanation: A failure occurred while trying to instruct the resource monitor daemon to make its performance information available.

Action: Check other error output or error log entries from this command for indications of the actual cause of the error, and take the appropriate corrective action.

2516-416 ptpertm: Error occurred while becoming a daemon process - continuing. Failing routine: *name* **Error:** *message***.**

Explanation: The **dae_init()** internal routine failed. This routine was called to convert the command to a daemon. The cause of the failure should be explained in the error message.

Action: None - the command will continue.

2516-417 ptpertm: Cannot set options on the socket connection to the resource monitor daemon - aborting.

Explanation: The SO_KEEPALIVE option could not be set on the socket connection between the **ptpertm** command and the resource monitor daemon. Without this option, the system could drop the socket connection, which would possibly terminate the resource monitor daemon.

Action: Retry the command. If the problem persists, contact IBM Service.

2516-418 ptpertm: A failure occurred while connecting to the System Data Repository server.

Explanation: The System Data Repository (SDR) reported a server error when the **ptpertm** command attempted to open a session with the SDR.

Action: Verify that the SDR is online, and that the node has not been fenced off from the rest of the partition. Retry the command.

2516-419 ptpertm: The user or the process running this command does not have authority to use the System Data Repository.

Explanation: The System Data Repository rejected the request to establish a session, because the user or the process starting the **ptpertm** command does not have sufficient privileges to use the System Data Repository.

Action: If the command is being executed from the command line, ensure that the user has sufficient privileges.

2516-420 ptpertm: An unexpected error occurred while establishing a session with the System Data Repository. Error code: *error code***.**

Explanation: An unexpected error occurred when contacting the System Data Repository (SDR).

Action: Verify that the SDR is online, and that the node has not been fenced off from the rest of the partition.

2516-421 ptpertm: Unable to initialize System Performance Measurement Interface (Spmi). Error: *Spmi error message (Spmi error code)*

Explanation: An error occurred when **ptpertm** attempted to initialize itself as a Performance Toolbox dynamic data supplier using the System Performance Measurement Interface library (Spmi).

Action: Consult the IBM Performance Toolbox for AIX manual for an explanation of the error. If the error persists, or if the Performance Toolbox documentation indicates that the cause is a programming error, contact IBM service.

2516-422 ptpertm: Unable to terminate System Performance Measurement Interface (Spmi). Error: *Spmi error message (Spmi error code)*

Explanation: An error occurred when **ptpertm** attempted to shut down the System Performance Measurement Interface library (Spmi).

Action: Consult the IBM Performance Toolbox for AIX manual for an explanation of the error. Repair any system conditions that may have caused the error. If the error persists, or if the Performance Toolbox documentation indicates that the cause is a programming error, contact IBM service.

2516-460 spdmdctrl: Cannot read the *inetd configuration file***. Update this file manually, and refresh the inetd daemon to enable** *inetd_subserver_name* **subserver.**

Explanation: The **inetd** daemon's configuration file could not be found, or could not be read.

Action: Add the following entry to the end of the **inetd** configuration file manually:

```
spdmd stream tcp nowait root /usr/lpp/ptpe/bin/spdmd spdmd
```

After making this update, refresh the **inetd** daemon twice with the following command to enable the PTPE subserver on this node:

```
refresh -s inetd
```

2516-461 spdmdctrl: Do not have permission to modify the *inetd configuration file***. Update this file manually, and refresh the inetd daemon to enable the inetd_subserver_name subserver.**

Explanation: Permissions on the **/etc/inetd.conf** file do not allow it to be modified by **spdmdctrl**.

Action: Add the following entry to the end of the **inetd** configuration file manually:

```
spdmd stream tcp nowait root /usr/lpp/ptpe/bin/spdmd spdmd
```

After making this update, refresh the **inetd** daemon twice with the following command to enable the PTPE subserver on this node:

```
refresh -s inetd
```

2516-462 spdmdctrl: Failed to add an entry for the *inetd_subserver_name* **subserver to the** *inetd configuration file***. Update this file manually, and refresh the inetd daemon to enable the subserver.**

Explanation: Could not add an entry to the **/etc/inetd.conf** file for the named subserver.

Action: Add the following entry to the end of the **inetd** configuration file manually:

```
spdmd stream tcp nowait root /usr/lpp/ptpe/bin/spdmd spdmd
```

After making this update, refresh the **inetd** daemon twice with the following command to enable the PTPE subserver on this node:

```
refresh -s inetd
```

2516-463 spdmdctrl: Cannot read the file *filename***. Update this file manually, and refresh the inetd daemon to remove the** *inetd_subserver_name* **subserver.**

Explanation: The **inetd** daemon's configuration file, **/etc/inetd.conf**, could not be found, or could not be read.

Action: Remove any entries for the named subserver from the **/etc/services** file manually, and refresh the **inetd** daemon.

2516-464 spdmdctrl: Do not have permission to modify the *inetd configuration file***. Update this file manually, and refresh the inetd daemon to to remove the** *inetd_subserver_name* **subserver.**

Explanation: Permissions on the **/etc/inetd.conf** file do not allow it to be modified by **spdmdctrl**.

Action: Remove any entries for the named subserver from the **/etc/inetd.conf** file manually, and refresh the **inetd** daemon.

2516-465 spdmdctrl: Cannot make a backup copy of the *inetd configuration file***. Update this file manually, and refresh the inetd daemon to to remove the** *inetd_subserver_name* **subserver.**

Explanation: Permissions on the **/etc/inetd.conf** file do not allow it to be modified by **spdmdctrl**.

Action: Remove any entries for the named subserver from the **/etc/inetd.conf** file manually, and refresh the **inetd** daemon.

2516-466 spdmdctrl: Failed to remove the entry for the *inetd_subserver_name* subserver from the *inetd configuration file*. Update this file manually, and refresh the inetd daemon to to remove the subserver.

Explanation: Could not delete the entry from the **/etc/inetd.conf** file for the named subserver.

Action: Remove any entries for the named subserver from the **/etc/inetd.conf** file manually, and refresh the **inetd** daemon.

2516-467 spdmdctrl: Cannot create an object in the System Data Repository of the class, *classname*, with attributes, *attributes*.

Explanation: A failure occurred in creating an object of the named System Data Repository class. This object is required to reserve a port for use by the PTPE programs.

Action: Verify that the System Data Repository is online; verify that the **sdrd**daemon is active on this node; retry the command.

2516-468 spdmdctrl: The refresh of the inetd daemon failed. Refresh this daemon manually.

Explanation: The command,

```
/bin/refresh -s inetd
```

failed.

Action: Issue the command manually.

2516-469 spdmdctrl: Internal error in the *function name* routine. Contact IBM Service.

Explanation: A coding error was made in this command.

Action: Contact IBM Service to resolve the problem.

2516-470 spdmdctrl: Cannot remove the entry for the *inetd_subserver_name* subserver from the *server configuration file*.

Explanation: The entry for the named **inetd** subserver could not be removed from the services configuration file.

Action: Verify that the named file exists; verify that the file has its permissions set to permit this command to modify its contents; verify that the directory containing this file has its permissions set to permit this command to create a file in this directory

2516-471 spdmdctrl: Cannot determine the number for this node.

Explanation: The

```
/usr/lpp/ssp/bin/node_number
```

command could not be found, or could not provide a valid node number.

Action: Ensure that the **install_cw** command has been executed on the control workstation.

2516-472 spdmdctrl: No options were provided to this command.

Explanation: The caller must provide an option to this command.

Action: Issue

```
spdmdctrl -h
```

to get a list of valid command options; issue this command with the appropriate options.

2516-473 spdmdctrl: Too many options were provided to this command.

Explanation: Only one option can be provided to this command.

Action: Issue

```
spdmdctrl -h
```

to get a list of valid command options; issue this command with the appropriate options.

2516-474 spdmdctrl: The inetd subserver, *inetd_subserver_name*, is active on this node. Stop this subserver and retry this command.

Explanation: An attempt was made to add or delete the PTPE subsystem to this node while that subsystem's **inetd** subserver was active on the node.

Action: Stop the subserver by issuing

```
stopsrc -t
```

Then retry the command.

2516-475 spdmdctrl: A port number for the *inetd_subserver_name* inetd subserver has not been reserved by the control workstation.

Explanation: A port number must be reserved by the control workstation for the PTPE subsystem. This port has not been reserved at this time.

Action: Issue

```
spdmdctrl -a
```

on the control workstation, and then retry the command on this node.

2516-476 spdmdctrl: Cannot register the following inetd subserver on this node *inetd_subserver_name port_number network_protocol*.

Explanation: The subserver could not be registered in the **/etc/services** and **/etc/inetd.conf** files on this node. If the word, *NOPORT*, is displayed, then the command could not find an available port number for PTPE to use.

Action: Contact IBM Service.

2516-477 spdmdctrl: Could not register the inetd subserver, *host name*, **with the System Resource Controller on this node.**

Explanation: The attempt to register the PTPE **inetd** subserver with the System Resource Controller failed.

Action: Perform this registry manually by issuing

```
mkserver -t
```

2516-478 spdmdctrl: Invalid option specified to this command.

Explanation: An invalid option was specified.

Action: Issue

```
spdmdctrl -h
```

to get a list of valid command options; issue this command with the appropriate options.

2516-610 *daemon name*: **Unknown command** *command name* **received on socket** *socket number*.

Explanation: Unknown command received on socket from **process_request** routine.

Action: Look at the **/etc/services** file, make sure no other process are trying to use the same port number as the Performance Monitor. (Look for **spdmd** in **/etc/services** file.)

2516-611 *daemon name*: **Read failure.**

Explanation: Error reading the message passed through the socket.

Action: Have the system administrator check for network errors on the specified Node. Problem is either the socket has closed or there is nothing to read on the specified socket.

2516-612 *daemon name*: **archiving already active when issuing** *command* **on node** *name*.

Explanation: Archiving already active when issuing archive start command(**ptpectrl -r**).

Action: Stop archiving before issuing another archive start command.

2516-613 *daemon name*: *command name* **unable to create archive active file** *filename*.

Explanation: Specified command was unable to create archive active file.

Action: Check directory permissions for the **/var/adm/ptpe** file structure.

2516-614 *Daemon name*: *command name* **unable to find archive active file on node** *name*.

Explanation: Specified command was unable to find archive active file.

Action: Try issuing the start archiving command, see if archive active file gets generated.

2516-615 *Daemon name*: *command name* **unable to delete archive active file** *filename*.

Explanation: Specified command was unable to delete archive active file.

Action: Check specified archive active file permissions.

2516-616 *Daemon name*: **Invalid command** *name* **in routine(***name***).**

Explanation: Invalid command issued.

Action: This should never occur, check that no other process trying to use same port has performance monitor.

2516-617 *Daemon name*: **Cannot locate** *node name* **in hierarchy, error:** *number* **in routine(***name***).**

Explanation: Cannot find specified node in the hierarchy.

Action: Check hierarchy to see if it reflects desired hierarchy. If not, delete hierarchy and then create desired hierarchy.

2516-618 *Daemon name*: **Couldn't find node** *name*, **errno =** *number*.

Explanation: Couldn't find node when checking for it's type.

Action: Check with administrator for possible network problems.

2516-619 *Daemon name*: **Cannot get identifier to collector daemon's shared memory. spdmcold may have died or had its shared memory removed.**

Explanation: Specified daemon cannot get identifier to the collector's shared memory segment.

Action: Verify that the **spdmcold** daemon is running on this node; shut down collection and restart.

2516-620 *Daemon name*: **Cannot attach to collector daemon's shared memory (key** *key***). spdmcold may have died or had its shared memory removed.**

Explanation: Specified daemon cannot attach to collector's shared memory of key specified.

Action: Verify that the **spdmcold** daemon is running on the node; verify that a shared memory segment exists with the key listed in this error log entry by using the **ipcs** command; if the shared memory exists, note the conditions that caused the error and contact IBM service - if the shared memory does not exist, shut down collection and restart.

2516-621 *Daemon name*: **Cannot get identifier to sampler daemon's shared memory. spdmspld may have died or had its shared memory removed.**

Explanation: Specified daemon cannot get identifier to the sampler's shared memory segment.

Action: Verify that the **spdmspld** daemon is running on this node; shut down collection and restart.

2516-622 *Daemon name*: **Cannot attach to sampler daemon's shared memory (key** *key***). spdmspld daemon may have died or had its shared memory removed.**

Explanation: Specified daemon cannot attach to sampler's shared memory of key specified.

Action: Verify that the **spdmspld** daemon is running on the node; verify that a shared memory segment exists with the key listed in this error log entry by using the **ipcs** command; if the shared memory exists, note the conditions that caused the error and contact IBM service - if the shared memory does not exist, shut down collection and restart.

2516-623 *Daemon name*: **error** *number* **in routine (***name***).**

Explanation: Error occurred when searching for each statistic in node's archive list with statistics in either the collector's or sampler's shared memory.

Action: Make sure you have a shared memory segment, if so try shutting down collection and restarting collection. If still error, note the conditions that caused the error and contact IBM Service.

2516-624 *Daemon name*: **Couldn't find node** *name***, errno =** *number***.**

Explanation: Could not find node in hierarchy when trying to find what type of node it is.

Action: Check that desired node is found in the hierarchy.

2516-625 *Daemon name*: **cannot get node statistics list, errno =** *number***.**

Explanation: Cannot get the node's statistics list, error returned should give more on problem found.

Action: Determine what to do, depends on error returned. Note the conditions that caused the error and contact IBM Service.

2516-626 *Daemon name*: **error opening archive file** *filename* **in routine (***name***).**

Explanation: Error occurred when trying to open the archive file to read or write to the code table.

Action: Verify that the archive file has read and write permissions.

2516-627 *Daemon name*: **Cannot allocate memory for archive code table error:** *number***.**

Explanation: Error when allocating archive code table.

Action: Check free space for the **/var/adm/ptpe** file structure.

2516-628 *Daemon name*: **Cannot read archive code table.**

Explanation: Error occurred when trying to read archive code table.

Action: Check directory permissions for the **/var/adm/ptpe** file structure.

2516-629 *Daemon name*: **error locating start of archive file.**

Explanation: Error occurred locating start of archive file.

Action: Make sure that the system was expected to have a performance information archive file on it. If not stop archiving and then restart archiving, see if perflog file is generated in the **/var/adm/ptpe** file structure.

2516-630 *Daemon name*: **error locating end of archive file.**

Explanation: Error occurred locating end of archive file.

Action: Make sure that the system was expected to have a performance information archive file on it, if not, try stopping archiving then restart archiving and see if **perflog** file is generated in the **/var/adm/ptpe** file system.

2516-631 *Daemon name*: **error locating last record in archive file.**

Explanation: Error occurred locating last record in archive file.

Action: Make sure that the system was expected to have a performance information archive file on it.

2516-632 *Daemon name*: **Cannot determine current position in the archive file.**

Explanation: Cannot determine current position in the archive file.

Action: Make sure that the system was expected to have a performance information archive file on it.

2516-633 *Daemon name*: **error reading archive file.**

Explanation: Error reading archive file.

Action: Verify that the archive file has read permission.

2516-634 *Daemon name*: **Cannot locate reporter node** *name* **in hierarchy.**

Explanation: Cannot locate reporter node specified in the hierarchy.

Action: Check that the hierarchy exists, if so, issue the **ptpectrl -i** to initialize the hierarchy and then restart collection.

2516-635 *Daemon name*: **NULL hierarchy pointer passed to routine (***name***).**

Explanation: NULL hierarchy pointer passed to **send_hierarchy_to_caller** routine.

Action: Try shutting down collection, check hierarchy, if hierarchy is setup right issue the setup command (**ptpectrl -i**) and then start collection again.

2516-636 *Daemon name*: **Error making a message of size(***size***) out of reply hierarchy for node** *name***.**

Explanation: Error making a hierarchy message for the specified node.

Action: Either the socket has closed or there is no message to read on the socket. Check with the system administrator for possible network problems.

2516-637 *Daemon name*: **Error sending message to our manager from node** *name***.**

Explanation: Error sending message to our manager on a specified node.

Action: Have the system administrator check for network errors on the specified node.

2516-638 *Daemon name*: **Unknown node type** *where* **in routine(***name***).**

Explanation: The daemons cannot correctly identify the node's responsibilities in the monitoring hierarchy. The message sent to this node may have been corrupted.

Action: Check with administrator for possible network problems.

2516-639 *Daemon name*: **Unable to send back hierarchy from node** *name***.**

Explanation: Error occurred when trying to send hierarchy data message to the manager node.

Action: Either the socket has closed or there is no message to read on the socket. Check with the system administrator for possible network problems.

2516-640 *Daemon name*: **Error deleting process identifier for** *name* **daemon from file:** *error code***.**

Explanation: Error occurred when trying to delete the pid for the specified daemon targeted for kill.

Action: Check that a daemon is still running on the nodes. If so, it is likely that you will have to physically kill the process.

2516-641 *Daemon name*: **Invalid Argument Count:** *number***.**

Explanation: Invalid argument count, the only valid argument count is 3 (program name, socket descriptor, and SPDM command).

Action: Ensure that no other process trying to use the same port as PTPE. Might result when daemon is executed from the command line with no arguments.

2516-642 *Daemon name*: **All** *number* **failed in routine(***name***).**

Explanation: All the specified nodes have failures, depending on input from the individual reporters, so all failed is returned to **spdmctrl**.

Action: Look into the error log to see what the possible causes are. The action taken will depend on the node failures.

2516-643 *Daemon name*: *number* **out of** *number* **total failures exceeds maximum of** *percent***% in routine(***name***).**

Explanation: The maximum number of failures has been exceeded so thresh exceeded is sent back to **spdmctrl**.

Action: Look into the error log to see what the possible causes are. The action taken will depend on the node failures.

2516-644 *Daemon name*: **Error: Collector daemon was started on reporter node.**

Explanation: A collector daemon should never be running on a reporter node.

Action: Make sure that a collector daemon has not been started from the command line. If collection is shut down and a collector daemon is still running the process will have to be killed manually.

2516-645 *Daemon name*: **Cannot determine the name of this node's manager.**

Explanation: Could not find out the name of the data manager node. If the daemon name is **spdmcold**, this means that a data manager node cannot get the name of the central coordinator node from examining the socket. If the daemon name is **spdmspld**, this means that a node cannot determine the name of its data manager node.

Action: Either the socket has closed or there is no message to read on the socket. Check with the administrator for possible network problems with the node's data manager node.

2516-646 *Daemon name*: **Cannot save process identifier in routine(***name***):** *error number***.**

Explanation: Error occurred when attempting to access, create, open, lock or unlock the SPDM pid file.

Action: ensure that the file exists; ensure that the file grants read/write permissions; ensure that the filesystem containing the file is mounted.

2516-647 *Daemon name*: **Invalid message size of** *size* **for context and statistics data.**

Explanation: Invalid message size when reading in the size of the message for context and statistics data.

Action: Check that node is up and not fenced off; have system administrator check for other network errors on the node. Make sure no other process is trying to use the same port as the **spdmd** daemon.

2516-648 *Daemon name*: **Unable to allocate memory of size** *size* **in routine(***name***), error:** *error number***.**

Explanation: Memory allocation failed

Action: Try again later; have system administrator check for applications that are consuming large amounts of memory.

2516-649 *Daemon name*: **Unable to read statistics message.**

Explanation: Read failed.

Action: Have system administrator check for other network errors on the node. Either the socket has closed or there is no message to read on the socket.

2516-650 *Daemon name*: **Invalid statistics line size** *size***.**

Explanation: Statistics line size is greater that the maximum line size.

Action: Note the conditions that caused the error and contact IBM Service

2516-651 *Daemon name*: **Read all** *number* **statistics, but haven't received DATA END yet. Quitting.**

Explanation: Read all the statistics but did not receive the data end response.

Action: Either the socket has closed or there is no message to read on the socket. Check with the administrator for possible network problems

2516-652 *Daemon name*: **Received more statistics than the maximum** *number* **that can be handled. Quitting.**

Explanation: Received more statistics than the maximum statistics size.

Action: Either the socket has closed or there is no message to read on the socket. Check with the administrator for possible network problems

2516-653 *Daemon name*: **Cannot get statistics from data manager.**

Explanation: Unable to get the statistics from the data manager.

Action: Either the socket has closed or there is no message to read on the socket. Check with the administrator for possible network problems with the Regular Manager.

2516-654 *Daemon name*: **Giving up after** *number* **unsuccessful attempts to send statistics data to my manager.**

Explanation: The number of times it was unable to send data to the data manager node exceeds _SPDM_MAX_RETRIES, the manager no longer cares about the data, so break out of the loop and terminate.

Action: Check with administrator for possible network problems.

2516-655 *Daemon name*: **Error in routine(***error***).**

Explanation: Error when trying to perform specified routine on shared memory. Possible lock_cmd's are lock, unlock or update.

Action: Note the conditions that caused the error and contact IBM Service.

2516-656 *Daemon name*: **Cannot setup statistic archive codes**

Explanation: Not able to setup statistics archive codes.

Action: Check directory permissions and free space in the **/var/adm/ptpe** file structure.

2516-657 *Daemon name*: **String length** *length* **is invalid.**

Explanation: When initializing aggregate list the read_reply on the caller socket receives an error.

Action: Have system administrator check for other network errors on the node, make sure no other process are using the same port as the **spdmd** daemon.

2516-658 *Daemon name*: **Cannot add statistic data for** *statistic name* **to archive set.**

Explanation: Error adding statistic data to archive set

Action: Look into the error log and see possible causes to the errors adding to the archive set.

2516-659 *Daemon name*: **Did not write data to archive file.**

Explanation: Did not write data to archive file.

Action: Check directory permissions and free space in the **/var/adm/ptpe** file structure.

2516-660 *Daemon name*: **Socket** *number* **no longer connected.
Peer** *remote node name* **may have closed connection prematurely or died unexpectedly.**

Explanation: A socket connection, previously established by the daemon to another system, has been dropped. The most likely cause is a failure in the other system, or a failure in the daemon running on that other system.

Action: Check the error logs on the remote system for an explanation of the failure; ensure that the remote system is up and not fenced off.

2516-661 *Daemon name*: **Connected to node** *name* **on socket** *number* **expected to be connected to node** *data manager node name***.**

Explanation: Not connected to expected host

Action: Check the hierarchy to make sure the hierarchy hasn't been altered since collection has started, if so, stop collection, setup hierarchy to desired hierarchy, then issue a setup followed by a start collection command.

2516-662 *Daemon name*: **Cannot connect to Central Coordinator** *name***.**

Explanation: Cannot connect to the specified node type.

Action: Check with administrator for possible network problems. Look into error log for any possible network, socket errors.

2516-663 *Daemon name*: **Central Coordinator DID NOT get our data.**

Explanation: Central coordinator did not get the data sent to it.

Action: Check with administrator for possible network problems. Look into error log for any possible network, socket errors.

2516-664 *Daemon name*: **Spmi error in** *library routine* **-** *error message* **from Spmi interface**

Explanation: Specified error occurred when issuing an Spmi call.

Action: Note error, make sure that IBM Performance Toolbox for AIX is installed.

2516-665 *Daemon name*: **Cannot stop** *resource monitor name* **data supplier.**

Explanation: The sampler daemon cannot stop the dynamic data supplier daemon.

Action: Look at error log for any possible errors, the dynamic data supplier daemon will have to be killed manually.

2516-666 *Daemon name*: **Unable to initialize Spmi interface**

Explanation: Sampler daemon unable to initialize the Spmi interface used in IBM Performance Toolbox for AIX.

Action: Make sure that the Performance Aide for AIX is installed on the node.

2516-667 *Daemon name*: **Cannot invoke** *name* **resource monitor.**

Explanation: Sampler daemon cannot invoke the specified resource monitor.

Action: Check error log for possible causes, make sure that the resource monitor daemon is found the **/usr/lpp/ssp/bin/haemRM** file system.

2516-668 *Daemon name*: **received** *reply* **instead of** *reply* **from collector for list received.**

Explanation: The collector daemon did not receive the list of statistics from the sampler daemon.

Action: Check with administrator for possible network problems. Look into error log for any possible network, socket errors. Possible that the socket has closed, or no message to read on the socket.

2516-669 *Daemon name*: **Cannot send reply** *reply* **on socket** *descriptor***.**

Explanation: Cannot send the specified data on the specified socket

Action: Check with administrator for possible network problems. Look into error log for any possible network, socket errors.

2516-670 *Daemon name*: **SpmiPathAddSetStat failed for** *number* **statistics in routine(***name***).**

Explanation: **SpmiPathAddSetStat** routine failed with specified errors.

Action: Possible for some statistics to not be found on every given node, but collection is still continued for statistics that are found. If count is high, make sure resource monitor was started if so, then individual lpp modules might not be loaded.

2516-671 *Daemon name*: **Cannot find statistic** *name* **in list.**

Explanation: Cannot find the statistic requested in the list specified.

Action: This should never happen, if so, note the conditions that caused the error and contact IBM Service.

2516-672 *Daemon name*: **Error** *return* **in setup_sampling();.**

Explanation: Incorrect response sent back from setup command.

Action: Check error log for possible failures.

2516-673 *Daemon name*: **Error killing COLLECTOR process** *process id*: *error code***.**

Explanation: Terminator daemon unable to kill collector daemon process.

Action: The process will have to be killed manually.

2516-674 *Daemon name*: **Cannot start process** *resource monitor name***.**

Explanation: The resource monitor passed in cannot be started.

Action: Check that the data supplier does exist in **/usr/lpp/ssp/bin/haemRM** file system. Check with administrator for possible network problems. Look in error log for any possible network, socket errors.

2516-675 *Daemon name*: **Cannot start** *name* **resource monitor.**

Explanation: Unable to connect to the resource monitor specified.

Action: Check that the resource monitor does exist in **/usr/lpp/ssp/bin/haemRM** file system. Check with administrator for possible network problems. Look into error log for any possible network, socket errors.

2516-678 *Daemon name*: **data manager node** *name* **received errors checking for reporter nodes.**

Explanation: Error finding who the specified node are under the data manager node.

Action: Check with administrator for possible network problems.

2516-679 *Daemon name*: **Cannot get hierarchy from reporter** *name* **on socket** *number*.

Explanation: Error getting hierarchy from specified node type on the specified socket

Action: Check that reporter node is up and not fenced off; have system administrator check for other network errors on the node

2516-680 *Daemon name*: **Cannot send command** *name* **to resource monitor.**

Explanation: Unable to send the specified command to the resource monitor.

Action: Check with administrator for possible network problems. Make sure no other processes are trying to use the same port.

2516-681 *Daemon name*: **Unexpected error in the select() system call - error code** *number*.

Explanation: Select error.

Action: The socket the select is being done on, could have closed. Check with administrator for possible network problems. Look into error log for any possible network, socket errors.

2516-682 *Daemon name*: **select timed out after** *number* **seconds waiting on socket** *number*. **socket number that is waiting for select**

Explanation: Select timed out after a predetermined time interval.

Action: The socket the select is being done on, could have closed. Check with administrator for possible network problems. Look into error log for any possible network, socket errors.

2516-683 *Daemon name*: **Error reading resource monitor response from socket** *number*: *error number*.

Explanation: Error reading resource monitor from the specified socket

Action: Ensure that the **harmld** resource monitor is active; check error log for *harmld* entries and take appropriate corrective action.

2516-684 *Daemon name*: **Expected to read** *number* **bytes, but read** *number* **bytes from socket** *number*.

Explanation: When reading from the socket, more was expected. expected daemons are starting up, also check error log for any possible errors that could have caused the problem.

Action: Look for any other *perfmon* error log entries generated at the same time to see if there is any corrective action that can be taken. (This might be a symptom of a larger problem.)

2516-685 *Daemon name*: **system() exited with** *number* **return code.**

Explanation: When starting to run process in the background the system() exited.

Action: Make sure the data supplier daemon is in **/usr/lpp/ptpe/bin** file structure. Check with system administrator for possible network problems .

2516-686 *Daemon name*: **Error getting a UNIX socket** *error code*.

Explanation: Error getting a UNIX domain socket.

Action: Check with administrator for possible network problems.

2516-687 *Daemon name*: **unable to unlock archive file** *filename*: *error number*.

Explanation: Error occurred when unlocking the archive file.

Action: Note the conditions that caused the error and contact IBM Service.

2516-688 *Daemon name*: **Error sending command** *command code***of size** *number* **to resource monitor daemon (***number* **bytes sent). Error message:** *message*.

Explanation: Error sending command message through the socket to the resource monitor daemon. The resource monitor daemon may have terminated prematurely, or the socket connection may have been dropped.

Action: Check the error log on this node for any possible error messages from the resource monitor daemon. Verify that the resource monitor daemon is running on this node. Check the error log for other network related problems. If problem persists, contact IBM service.

2516-689 *Daemon name*: **Unknown data type** *type* **in routine(***name***).**

Explanation: Unknown data type in specified routine

Action: Note the conditions that caused the error and contact IBM Service.

2516-690 *Daemon name*: **no space for statistic** *statistic name* **in the archive set.**

Explanation: archive_set is about to overflow

Action: Note the conditions that caused the error and contact IBM Service.

2516-691 *Daemon name*: **unable to lock archive file** *filename*: *error number*.

Explanation: Error occurred when locking the archive file.

Action: Note the conditions that caused the error and contact IBM Service.

2516-692 *Daemon name*: **Wrote** *number bytes written* **out of** *total* **bytes into archive file:** *error number*.

Explanation: Write error.

Action: Check free space for the **/var/adm/ptpe** file structure. Socket might have closed in the middle of writing to archive file. Check with system administrator for possible network problems.

2516-693 *Daemon name*: **Cannot stat() file** *filename*: *error number*.

Explanation: Cannot successfully stat() the archive file for some reason

Action: Check error log for possible causes. Check with the system administrator for possible network problems.

2516-694 *Daemon name*: **Cannot** *system call* **Buffer size***number* **(read** *number bytes***) from archive file** *filename*: *error number*.

Explanation: Read error reading from archive file.

Action: Check with the system administrator for possible network problems.

2516-695 *Daemon name*: **Cannot get hierarchy from manager** *node name* **on socket** *number*.

Explanation: Error getting hierarchy from specified node type on the specified socket.

Action: Check that data manager node is up and not fenced off; have system administrator check for other network errors on the data manager node.

2516-696 *Daemon name*: **Node** *name* **designated as both g and data manager. This is an invalid configuration.**

Explanation: Node cannot be both a g and data manager.

Action: Change hierarchy so that this is not the case.

2516-697 *Daemon name*: **Central Coordinator** *name* **received errors checking for data managers.**

Explanation: Error determining the specified node types.

Action: Check with administrator for possible network problems.

2516-698 *Daemon name*: **Central Coordinator** *name* **is reporting to itself: ERROR.**

Explanation: Specified data manager node cannot report back to itself.

Action: Should never happen, if so, note the conditions that caused the error and contact IBM Service.

2516-699 *Daemon name*: **Attempted to reference node list pointer of NULL in iteration** *number*.

Explanation: Attempting to reference node list pointer of NULL when walking through hierarchy looking for data managers.

Action: Check that the hierarchy is what is expected, it might be possible that the hierarchy has been changed. If hierarchy is correct, then stop collection, then issue a setup command, then start collection again.

2516-700 *Daemon name*: **Central Coordinator** *name* **received errors checking for reporting data managers.**

Explanation: Invalid argument has been returned back from the **spdm_get_reporters** routine.

Action: Check that the hierarchy is what is expected, it might be possible that the hierarchy has been changed. If hierarchy is correct, then stop collection, then issue a setup command, then restart collection.

2516-701 *Daemon name*: **Not connecting to manager** *name* **because of connection errors on socket** *number*.

Explanation: Connect to node failures.

Action: Check with administrator for possible network problems.

2516-702 *Daemon name*: **At least one Manager Node failed to connect.**

Explanation: At least one of the data managers failed to connect.

Action: Check with administrator for possible network problems. Make sure no other processes are trying to use the same port number.

2516-703 *Daemon name*: **Error sending hierarchy message to reporting node** *name* **on socket** *number*.

Explanation: Error sending hierarchy message to reporting node specified on specified socket number.

Action: Check that node is up and not fenced off; have system administrator check for other network errors on the node. Make sure no other process is trying to use the same port as the **spdmd** daemon.

2516-704 *Daemon name*: *occurrence number* **Error** *number* **making a message out of hierarchy for manager** *name*.

Explanation: Error making a message out of the hierarchy for the manager.

Action: Either the socket has closed or there is no message to read on the socket. Check with the system administrator for possible network problems.

2516-705 *Daemon name*: **socket** *number* **is connected to** *node name* **and not** *node name* **as expected.**

Explanation: Socket is not connected to the correct node.

Action: Either the socket has closed or there is no message to read on the socket. Check with the system administrator for possible network problems.

2516-706 *Daemon name*: **Failed to read reply from** *node name* **in routine(***name***).**

Explanation: Failed to read reply from specified node type.

Action: Check that node is up and not fenced off; have system administrator check for other network errors on the node. Make sure no other process is trying to use the same port as the **spdmd** daemon.

2516-707 *Daemon name*: **Received reply** *reply* **from** *node name* **in routine(***name***). routine where error occurred**

Explanation: Error receiving specified reply on the specified node type.

Action: Either the socket has closed or there is no message to read on the socket. Check with the system administrator for possible network problems.

2516-708 *Daemon name*: **Unknown reply** *reply* **from** *node name* **in routine(***name***).**

Explanation: Unknown reply in specified routine.

Action: Either the socket has closed or there is no message to read on the socket. Check with the system administrator for possible network problems.

2516-709 *Daemon name*: **unable to add reporter** *name* **hierarchy to manager** *name* **hierarchy.**

Explanation: Unable to add reporter's hierarchy to the manager's hierarchy.

Action: Either the socket has closed or there is no message to read on the socket. Check with the system administrator for possible network problems.

2516-710 *Daemon name*: **Cannot allocate memory reconstructing hierarchy on node** *name*.

Explanation: Memory allocation failed.

Action: Try again later; have system administrator check for applications that are consuming large amounts of memory.

2516-711 *Daemon name*: **Cannot open System Data Repository: return code from SDROpenSession =** *error number*.

Explanation: Could not open the System Data Repository (SDR).

Action: Check if SDR is off-line or damaged. Note the conditions that caused the error and contact IBM Service.

2516-712 *Daemon name*: **Error returning from routine(***name***) System Data Repository. errno =** *number*.

Explanation: Could not perform specified function on the System Data Repository (SDR).

Action: Check if SDR is off-line or damaged.

2516-713 *Daemon name*: **Length of failing reporter name is invalid** *length*.

Explanation: Error reading reply on the socket.

Action: Check with administrator for possible network problems.

2516-714 *Daemon name*: **Unable to read a failing reporter name.**

Explanation: Error in read_string_reply reading a failing reporter name.

Action: Check with administrator for possible network problems.

2516-715 *Daemon name*: **Unable to read context name string length.**

Explanation: Error reading reply on socket.

Action: Check with administrator for possible network problems.

2516-716 *Daemon name*: **Length of context name is invalid** *length*.

Explanation: Error reading reply on socket.

Action: Check with administrator for possible network problems.

2516-717 *Daemon name*: **Unable to read description string length for** *name*.

Explanation: Error reading reply on socket.

Action: Check with administrator for possible network problems.

2516-718 *Daemon name*: **Length of** *name* **description is invalid** *length*.

Explanation: Error reading reply on socket.

Action: Check with administrator for possible network problems.

2516-719 *Daemon name*: **Unable to read description for** *name*.

Explanation: Error reading reply on socket.

Action: Check with administrator for possible network problems.

2516-720 *Daemon name*: **Cannot read ASN-1 number for** *name*.

Explanation: Error reading reply on socket.

Action: Check with administrator for possible network problem.

2516-721 *Daemon name*: **Cannot read number of subcontexts for context** *name*.

Explanation: Error reading reply on socket.

Action: Check with administrator for possible network problem.

2516-722 *Daemon name*: **Invalid number of subcontexts** *number* **for context** *name*.

Explanation: Error reading reply on socket.

Action: Check with administrator for possible network problem.

2516-723 *Daemon name*: **Cannot get minimum statistic** *name* **from socket.**

Explanation: Error reading reply on socket.

Action: Check with administrator for possible network problems.

2516-724 *Daemon name*: **Cannot read statistic** *name* **value type.**

Explanation: Error reading reply on socket.

Action: Check with administrator for possible network problem.

2516-725 *Daemon name*: **Unable to open statistic file** *filename* **with mode** *mode*: *error number*.

Explanation: The **/tmp/manager.stat.file** statistics file cannot be found, or cannot be opened

Action: Verify that the statistic file exists for your specific locale; verify that the statistic file has read permission.

2516-726 *Daemon name*: **execl failed: Executing command** *command* **with parameters** *socket_string* **and** *comm_string* **error:** *error number*.

Explanation: Execution of specified program failed with specified error.

Action: make sure the executable is in the **/usr/lpp/ptpe/bin** file system.

2516-727 *Daemon name*: **semget error:** *error number*

Explanation: Error getting a set of semaphores.

Action: Note the conditions that caused the error and contact IBM Service.

2516-728 *Daemon name*: **semop() error on lock:** *error number***.**

Explanation: Semaphore operations error on lock.

Action: Note the conditions that caused the error and contact IBM Service.

2516-729 *Daemon name*: **Cannot remove semaphore** *id*: *error number***.**

Explanation: Semaphore control operations error on specified semid.

Action: Note the conditions that caused the error and contact IBM Service.

2516-730 *Daemon name*: **Invalid input parameter** *name* **in routine(***name***).**

Explanation: Invalid input parameter specified in specified routine.

Action: Note the conditions that caused the error and contact IBM Service.

2516-731 *Daemon name*: **Cannot combine REGULAR_NODE and AGGREGATE_TABLE parameters in routine(***name***).**

Explanation: Error combining REGULAR_NODE and AGGREGATE_TABLE parameters on a REGULAR_NODE.

Action: Note the conditions that caused the error and contact IBM Service.

2516-732 *Daemon name*: **Cannot create a shared memory key. errno =** *error number***.**

Explanation: Error creating a shared memory key.

Action: Note the conditions that caused the error and contact IBM Service.

2516-733 *Daemon name*: **Cannot create a shared memory of size** *memory requirement*: *error number***.**

Explanation: Error creating shared memory.

Action: Note the conditions that caused the error and contact IBM Service.

2516-734 *Daemon name*: **Cannot attach to shared memory with id** *shared memory id*: *error number***.**

Explanation: Cannot do specified command with shared memory id.

Action: Note the conditions that caused the error and contact IBM Service.

2516-735 *Daemon name*: **Cannot detach shared memory at** *pointer* *error number***.**

Explanation: Cannot do specified command with shared memory at specified location.

Action: Note the conditions that caused the error and contact IBM Service.

2516-736 *Daemon name*: **Cannot get shared memory id from socket.**

Explanation: Cannot get shared memory id from socket.

Action: Check with administrator for possible network problems. Make sure no other process's are trying to use the same port.

2516-737 *Daemon name*: **Cannot get message size from socket.**

Explanation: Cannot get message size from socket.

Action: Check with administrator for possible network problems. Make sure no other process's are trying to use the same port.

2516-738 *Daemon name*: **Cannot get the name of the local host:** *error number***.**

Explanation: Unable to get hostname.

Action: Check with administrator for possible network problems. Possible SDR error not being able to get the hostname.

2516-739 *Daemon name*: **Failed getpeername call for socket** *socket number*: *error number***.**

Explanation: Failure to return the name of the node the process is connected to on the socket.

Action: Either the socket has closed or there is no message to read on the socket. Check with the system administrator for possible network problems.

2516-740 *Daemon name*: **routine(***name***) failed for** *node name*: *error number***.**

Explanation: Specified routine failed with specified error.

Action: Either the socket has closed or there is no message to read on the socket. Check with the system administrator for possible network problems.

2516-741 *Daemon name*: **Error in routine(***name***)** *command* **on socket** *number*: **Broken pipe.**

Explanation: Error sending specified in string and command on a socket and the pipe is broken.

Action: Either the socket has closed or there is no message to read on the socket. Check with the system administrator for possible network problems.

2516-742 *Daemon name*: **Error in routine(***name***)** *command* **on socket** *number*: *error number* **(***number* **bytes sent).**

Explanation: Error sending a specified string and command on a socket.

Action: Either the socket has closed or there is no message to read on the socket. Check with the system administrator for possible network problems.

2516-743 *Daemon name*: **Error sending** *length* **bytes of data to node** *name*. **The socket was closed unexpectedly by the remote node.**

Explanation: Error sending specified in string on a socket and the pipe is broken.

Action: The socket has closed unexpectedly. Check with the system administrator for possible network problems.

2516-744 *Daemon name*: **Error sending message size** *command* **to node socket** *number*: *error number*. *number* **bytes sent.**

Explanation: Error sending a specified string and command on a socket.

Action: Either the socket has closed or there is no message to read on the socket. Check with the system administrator for possible network problems.

2516-746 *Daemon name*: **Cannot connect to socket file** *filename*: *error message*.

Explanation: The connect() system call failed. A socket connection to the resource monitor daemon could not be established. The resource monitor daemon may not be active, or the socket may have been dropped.

Action: Check the node's error logs for possible error messages from the resource monitor daemon, or for indications of network problems.

2516-747 *Daemon name*: **Error in routine(***name***) from socket** *number error number*.

Explanation: Error sending specified in string from socket.

Action: Either the socket has closed or there is no message to read on the socket. Check with the system administrator for possible network problems.

2516-748 *Daemon name*: **error in set_not_found_fields().**

Explanation: Error in set_not_found_fields.

Action: Note the conditions that caused the error and contact IBM Service.

2516-749 *Daemon name*: **cannot make hierarchy for reporting node** *name*

Explanation: Error making the hierarchy for the specified reporting node.

Action: Check that the hierarchy exists, if so, issue

```
ptpectrl -i
```

to initialize the hierarchy.

2516-750 *Daemon name*: **Cannot find free ASN-1 number for statistic** *name*.

Explanation: Unable to find a free ASN-1 number for the specified statistic.

Action: Note the conditions that caused the error and contact IBM Service

2516-751 *Daemon name*: **Cannot find context** *name* **in statistics file.**

Explanation: Unable to find the specified context name in the statistics file.

Action: Note the conditions that caused the error and contact IBM Service

2516-752 *Daemon name*: **Cannot find parent for context** *name*.

Explanation: Cannot find parent for specified context.

Action: Note the conditions that caused the error and contact IBM Service

2516-753 *Daemon name*: **Daemon received unexpected signal** *number* **- aborting.**

Explanation: The daemon received a signal that caused it to terminate, but the signal was not one of the expected termination signals. Signal may have been generated by a memory access problem, an outside source, or a coding error.

Action: Check if someone issued a **kill** call on the daemon named; If no one killed the daemon, and the signal value is not 33, contact IBM service.

2516-754 *Daemon name*: **Error making a message of size(*size*) out of archive data for node *name*.**

Explanation: Error making a hierarchy message for the specified node.

Action: Either the socket has closed or there is no message to read on the socket. Check with the system administrator for possible network problems.

2516-755 *Daemon name*: **Cannot connect to Manager *name*.**

Explanation: Cannot connect to the specified node type.

Action: Check with administrator for possible network problems.

2516-756 *Daemon name*: **Error killing SAMPLER process *process id*: *error number*.**

Explanation: Terminator daemon unable to kill sampler daemon process.

Action: The process will have to be killed manually.

2516-757 *Daemon name*: **Not connecting to node *name* because of connection errors on socket *number*.**

Explanation: Connect to node failures

Action: Check with administrator for possible network problems.

2516-758 *Daemon name*: **At least one node failed to connect.**

Explanation: At least one of the reporter nodes failed to connect.

Action: Check with administrator for possible network problems. Make sure no other process's are trying to use the same port number.

2516-759 *Daemon name*: **At least node failed to start archiving.**

Explanation: At least one of the reporter nodes failed to start archiving.

Action: Check with administrator for possible network problems. Make sure no other process's are trying to use the same port number.

2516-760 *Daemon name*: **At least one Manager Node failed to start archiving.**

Explanation: At least one of the data manager nodes failed to start archiving.

Action: Check with administrator for possible network problems. Make sure no other process's are trying to use the same port number.

2516-761 *Daemon name*: **At least one node failed to reply with *replay*.**

Explanation: At least one of the nodes failed to reply with _SPDM_INPUT_CONFIRMED.

Action: Check with administrator for possible network problems. Make sure no other process's are trying to use the same port number.

2516-762 *Daemon name*: **At least one data manager node failed to reply with *reply*.**

Explanation: At least one of the data manager nodes failed to reply with _SPDM_INPUT_CONFIRMED.

Action: Check with administrator for possible network problems. Make sure no other process's are trying to use the same port number.

2516-763 *Daemon name*: **Error sending hierarchy message to data manager *name* on socket *number*.**

Explanation: Error sending hierarchy message to specified data manager on specified socket number

Action: Check that node is up and not fenced off; have system administrator check for other network errors on the node. Make sure no other process is trying to use the same port as the **spdmd** daemon.

2516-764 *Daemon name*: **unable to add new manager *name* hierarchy to manager *name* hierarchy.**

Explanation: Unable to add reporter's hierarchy to the manager's hierarchy.

Action: Check error log for network errors on two nodes listed, as well as for *perfmon* entries on the reporting node indicating reporting node failure, and take appropriate corrective action. If none generated, contact IBM Service.

2516-765 *Daemon name*: **Unable to read context name.**

Explanation: Error reading reply on socket.

Action: Check with administrator for possible network problems.

2516-766 *Daemon name*: **Unable to read long context name string length.**

Explanation: Error reading reply on socket.

Action: Check with administrator for possible network problems.

2516-767 *Daemon name*: **Unable to read long context name.**

Explanation: Error reading reply on socket.

Action: Check with administrator for possible network problems.

2516-768 *Daemon name*: **Unable to read statistic name string length.**

Explanation: Error reading reply on socket.

Action: Check with administrator for possible network problems.

2516-769 *Daemon name*: **Unable to read statistic name.**

Explanation: Error reading reply on socket.

Action: Check with administrator for possible network problems.

2516-770 *Daemon name*: **Cannot read number of statistics for context** *name***.**

Explanation: Error reading reply on socket.

Action: Check with administrator for possible network problem.

2516-771 *Daemon name*: **Invalid number of statistics** *number* **for context** *name***.**

Explanation: Error reading reply on socket.

Action: Check with administrator for possible network problem.

2516-772 *Daemon name*: **Cannot get maximum statistic** *name* **from socket.**

Explanation: Error reading reply on socket.

Action: Check with administrator for possible network problem.

2516-773 *Daemon name*: **Cannot get data type of statistic** *name* **from socket.**

Explanation: Error reading reply on socket.

Action: Check with administrator for possible network problem.

2516-774 *Daemon name*: **Unable to open context file** *filename* **with mode** *mode***:** *error number***.**

Explanation: **/tmp/manager.cont.file** statistics file cannot be found, or cannot be opened.

Action: Verify that the context file exists for your specific locale; verify that the context file has read and write permission.

2516-775 *Daemon name*: **semop() error on unlock:** *error number***.**

Explanation: Semaphore operations error on unlock.

Action: Note the conditions that caused the error and contact IBM Service.

2516-776 *Daemon name*: **Cannot remove shared memory with id** *id***:** *error number***.**

Explanation: Cannot do specified command with shared memory id.

Action: Note the conditions that caused the error and contact IBM Service.

2516-777 *Daemon name*: **Cannot save shared memory id(***id***) to file allocated by the** *routine* **call.**

Explanation: Error saving shared memory id to file.

Action: Note the conditions that caused the error and contact IBM Service.

2516-778 *Daemon name*: **The sampler daemon spdmapid was already running. PID number:** *id***. The attempt to shut down this daemon failed with an error:** *error number***.**

Explanation: When trying to kill an already running sampler daemon before start_sampling() is called, an error occurred.

Action: The process will have to be killed manually.

2516-779 *Daemon name*: **The collector daemon spdmcold was already running. PID number:** *id***. The attempt to shut down this daemon failed with an error:** *error number***.**

Explanation: an error occurred when trying to kill an already running collector daemon before saving the process id to the file.

Action: The process will have to be killed manually

2516-780 *Daemon name*: **The inetd subserver daemon spdmd failed on reporting node** *name***.**

Explanation: The **spdmd** daemon failed to start on the named node, or failed before starting the proper handler daemon on that node.

Action: Check the error log on the named node for an explanation of the failure.

2516-781 *Daemon name*: **Cannot get statfs() information about the filesystem containing the directory** *name***.**

Explanation: The daemon attempted to get information on the filesystem containing the named directory with the statfs() function. The attempt failed.

Action: Verify that the directory exists on the node; verify that the **/var** filesystem is mounted.

2516-782 *Daemon name*: **Archive code table requires** *size* **blocks. The filesystem containing the directory /var/adm/ptpe has** *number* **blocks available.**

Explanation: The filesystem containing the named directory does not have enough space to contain the archive codes table. This file is required to start performance information collection.

Action: Increase the amount of space in the filesystem, or mount a new file-system on the named directory.

2516-783 *Daemon name*: **Cannot read system partition information from manager's socket message.**

Explanation: The daemon could not extract the name of the SP partition from the message sent to this node by its data manager node.

Action: Check for network related errors on this system and the system's data manager node.

2516-784 *Daemon name*: **The routine,** *Spmi Library routine name***, could not lock the Spmi shared memory for** *number* **seconds. Assuming a failure in Spmi and terminating.**

Explanation: The Spmi routine failed continuously for the stated period of time because another process held a shared memory lock within the Spmi. This may indicate an error in Spmi itself. The daemon cannot continue.

Action: The Spmi shared memory on the node may be damaged, or another process may be using the Spmi incorrectly. Another error log entry should follow this one, indicating the process that has locked the Spmi. Try to terminate this process, or refresh it with a **SIGINT** if the process is the **xmservd** daemon. Shut down collection and restart.

Appendixes

Appendix A. PTPE Files

The directories, files and processes used by PTPE are described here.

What PTPE Creates at Installation

The following files, commands, and directories are created when PTPE is installed.

Files, Directories, and Libraries

/usr/lpp/ptpe
: Location of the PTPE installable image, commands, and daemons.

/var/adm/ptpe
: Location of the PTPE information archive for the node.

/var/adm/ptpe/perftab
: The PTPE archived statistics translation table. This file contains the mapping between the **Spmi**names for performance statistics and the identification codes used for these statistics in the performance information archive on the node.

/usr/include/spdm.h
: The PTPE C language header file, providing definitions of the data types and prototypes of the PTPE API library subroutines.

/usr/lib/libptpe.a
: The PTPE C language programming interface.

/usr.lpp/ptpe/samples
: Sample API and configuration files.

/usr/lpp/ssp/info2
: PTPE hypertext information database files for InfoExplorer retrieval

/usr/lpp/ssp/docs/sp_perf_parallel_ext.ps
: This manual in printable PostScript format.

Manual pages for PTPE commands and subroutines are included with Parallel System Support Programs man pages in **/usr/lpp/ssp/man**.

Commands and Utilities

/usr/sbin/ptpectrl
: The control command for PTPE. This command controls the current status of performance information sampling, collection, and recording in the entire system.

/usr/lpp/ptpe/bin/ptpeconf
: The configuration command for PTPE. This command creates the necessary data classes within the System Data Repository.

/usr/lpp/ptpe/bin/ptpegroup
: A user group creation command, which creates the **perfmon** user group. Only users of this group, with the group set as their primary group, can execute any PTPE commands or use its programming library.

/usr/lpp/ptpe/bin/ptpehier
The hierarchy construction command of PTPE. This command permits the user to create a monitoring hierarchy, using a set of standard methods or using input provided by the caller.

/usr/lpp/ptpe/bin/ptpedump
A distributed performance information archive dump utility. This utility dumps a text version of the archives maintained by one or more nodes to files on these systems.

/usr/lpp/ptpe/bin/spdm_dump
A local performance information archive dump utility. This utility dumps a text version of the archive maintained by the local node to standard output.

Daemons

/usr/lpp/ptpe/bin/spdmd
The PTPE master daemon. This daemon receives all PTPE requests for the node, validates the request, and invokes the appropriate daemon process to handle the request.

/usr/lpp/ptpe/bin/spdmcold
The PTPE collector daemon. This daemon is executed on the central coordinator node and all data manager nodes in the monitoring hierarchy, and prepares the averaged performance statistics for the monitoring hierarchy.

/usr/lpp/ptpe/bin/spdmspld
The PTPE sampler daemon. This daemon is executed on all nodes within the monitoring hierarchy. This daemon obtains all the performance information from the node, forwards this information to the node's data manager node, and records the information to that node's performance information archive.

/usr/lpp/ptpe/bin/spdmapid
The PTPE programming library request handler. This daemon executes whenever the node is involved in the response to a programming library request. It obtains performance data recorded in the archives of a node, enables or restricts information from being collected or archived, and (on data manager nodes) relays requests on to other nodes for further processing.

/usr/lpp/ptpe/bin/spdmtrmd
The PTPE termination daemon. This daemon executes on a node whenever performance information collection is being shut down, and is responsible for forcing the other PTPE daemons to exit.

Message Catalogs

/usr/lib/nls/msg/En_US/ptpe.cat

/usr/lib/nls/msg/en_US/ptpe.cat

/usr/lib/nls/msg/C/ptpe.cat
The message catalog for Performance Toolbox Parallel Extensions for AIX. Other message catalogs may be provided in other directories for other locales.

What PTPE Creates During Use

These files are not created until PTPE is started.

/var/adm/ptpe/perflog
The PTPE information archive. This file is created on each node in the monitoring hierarchy after collection has begun. The file contains a statistics code translation table, plus any performance information that was recorded by this node.

/etc/perf/spdm.pid
Ths file containing the process identifiers of any PTPE daemons currently executing on the node. This file should not remain after performance information collection has been shut down.

/etc/perf/ptpe.shseg
A file used by the **Spmi** library when creating the shared memory to store the SP-specific performance data. This file should not remain after performance information collection has been shut down, or the **ptpertm** command has completed.

/tmp/spdm.trace
The PTPE daemon tracing file. This file will only be present if the daemons encounter unexpected error conditions. The messages in this file should also be mirrored in the node's error log.

/tmp/manager.cont.file

/tmp/manager.cont.file.uniq

/tmp/manager.cont.file.sorted

/tmp/manager.stat.file

/tmp/manager.stat.file.uniq

/tmp/manager.stat.file.sorted

/tmp/manager.stat.file.transposed

/tmp/current.stats
Work files used by the PTPE daemons when initializing the monitoring hierarchy. The files should only exist on the node during initialization phase (see “ptpectrl” on page 88), and be removed after that phase has completed.

Appendix B. PTPE Sample Application Program

This sample PTPE application can be found in **/usr/lpp/ptpe/samples/ptpe_sample.c**.

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

/*
 * @(#)04   1.3 src/perfmon/pdm/samples/ptpe_sample.c, \
 * perfmon, perfmon_rloc 8/28/96 21:11:48
 *
 * Module Name: ptpe_example.c
 *
 * Component:   perfmon
 *
 * Description: This module demonstrates how the Performance Toolbox Parallel
 *              Extensions programming library could be utilized to start
 *              performance information aggregation and retention within a
 *              monitoring hierarchy, and how it might also be used to retrieve
 *              performance information stored within the hierarchy's perfor-
 *              mance information archives.
 *
 *              It attempts to demonstrate the relationship between the
 *              host_list_t and the stat_list_t data types, and how assignments
 *              are made between these types in order to specify the statistics
 *              and the nodes involved in API requests.  It also attempts to
 *              demonstrate how an application should "navigate" within these
 *              lists to obtain information and make assignments.
 *
 * Routines:    sig_setup                  exit_hdlr
 *              print_host                 print_stat
 *              print_result               select_hosts
 *              setup_stats                set_stats_time
 *              assign_stats               get_different_times
 *              retry_test                 main
 *
 * Macros:      SUCCESS_TEST               LIMITED_TEST
 *
* Disclaimer:  THE SOURCE CODE EXAMPLES PROVIDED BY IBM ARE ONLY INTENDED TO
 *              ASSIST IN THE DEVELOPMENT OF A WORKING SOFTWARE PROGRAM.  THE
 *              SOURCE CODE EXAMPLES DO NOT FUNCTION AS WRITTEN:  ADDITIONAL
 *              CODE IS REQUIRED.  IN ADDITION, THE SOURCE CODE EXAMPLES MAY
 *              NOT COMPILE AND/OR BIND SUCCESSFULLY AS WRITTEN.
 *
 *              INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THE
 *              SOURCE CODE EXAMPLES, BOTH INDIVIDUALLY AND AS ONE OR MORE
 *              GROUPS, "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED
 *              OR IMPLIED, INCLUDING, BUT NOT LIMITED TO THE IMPLIED
 *              WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 *              PURPOSE.  THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE
 *              OF THE SOURCE CODE EXAMPLES, BOTH INDIVIDUALLY AND AS ONE OR
 *              MORE GROUPS, IS WITH YOU.  SHOULD ANY PART OF THE SOURCE CODE
 *              EXAMPLES PROVE DEFECTIVE, YOU (AND NOT IBM OR AN AUTHORIZED IBM
 *              DEALER) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING,
 *              REPAIR OR CORRECTION.
 *
 *              IBM does not warrant that the contents of the source code
 *              examples, whether individually or as one or more groups, will
 *              meet your requirements or that the source code examples are
 *              error-free.
 *
 *              IBM may make improvements and/or changes in the source code
 *              examples at any time.
 *
 *              Changes may be made periodically to the information in the
 *              source code examples; these changes may be reported, for the
 *              sample device drivers included herein, in new editions of the
 *              examples.
 *
 *              References in the source code examples 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 licensed program in the source code
 *              examples is not intended to state or imply that only IBM's
 *              licensed program may be used.  Any functionally equivalent
 *              program may be used.
 */

#include        <stdio.h>
#include        <spdm.h>            /* PTPE data types, functions, etc.     */
#include        <time.h>
#include        <sys/time.h>
/****************************************************************************
 *
 * Global Data Areas
 *
 ***************************************************************************/
/*
 * Session control information block - will be created and initialized by the
 * PtpeOpenSession routine, discarded by the PtpeCloseSession routine, and
 * required by any routine that controls collection, controls archiving, or
 * requests information from the monitoring hierarchy.
 */
session_ptr_t   sblock;

/*
 * The names of statistics that will be retrieved from the performance infor-
 * mation archives on some of the nodes within the monitoring hierarchy.
 * Notice that the statistic names use the Performance Toolbox "Spmi" naming
 * convention (the statistic names are relative to the TOP context, not
 * absolute path names from /hosts as "Rsi" uses).
 */
#define         NUM_EX_STATS        4
char            test_stats[NUM_EX_STATS][PTPE_STNL] = {
                    "Proc/runque",
                    "PagSp/%totalfree",
                    "Mem/Real/%free",
                    "CPU/gluser"
                };

/*
 * Function prototypes
 */
extern void print_host(host_list_t);
extern void print_stat(char *, stat_list_t);
extern void print_result(int);
extern void select_hosts(host_list_t, host_list_t *);
extern void setup_stats(stat_list_t *);
extern void set_stats_time(stat_list_t, int, struct tm *);
extern void assign_stats(host_list_t, stat_list_t);
extern void get_different_times(host_list_t, stat_list_t, struct tm *);

/*****************************************************************************
 *
 * Macro Name:      SUCCESS_TEST
 *
 * Description:     Tests the return code from an API routine to determine if
 *                  the routine was completed successfully.  If an API routine
 *                  is not successful, this macro forces the application to
 *                  drop the session and abort.
 *
 *                  Notice that if the session is dropped after collection or
 *                  archiving has been activated, collection or archiving
 *                  REMAINS active after the session is released.  If this
 *                  macro forces an abort after collection or archiving has
 *                  been activated, run "ptpectrl -s" to stop collection and
 *                  archiving.
 *
 * Usage:           SUCCESS_TEST(calling_func, api_routine, return_code)
 *
 *                  where:  calling_func    Is the name of the routine making
 *                                          the API request.
 *                          api_routine     Is the name of the API routine that
 *                                          "calling_func" was using.
 *                          return_code     Is the return code from
 *                                          "api_routine".
 *
 * Return Codes:    None
 *
 *****************************************************************************/

#define SUCCESS_TEST(calling_func, api_routine, return_code)          \
{                                                                     \
        if (rc != PTPE_SUCCESS && rc != PTPE_LIMITED) {               \
                printf("Unexpected error in %s(): %s() returned ",    \
                       calling_func, api_routine);                    \
                print_result(return_code);                            \
                printf("RESULTS NOT AS EXPECTED - TERMINATING.\n");   \
                exit_hdlr(-1);                                        \
        }                                                               \
}
/*****************************************************************************
 *
 * Macro Name:      LIMITED_TEST
 *
 * Description:     Tests if the return code from a preceeding PTPE API library
 *                  routine indicated that all hosts targeted for the request
 *                  did not repsond favorably.  If at least one node did not
 *                  respond favorably, the reply list is displayed for the
 *                  user.  The user should examine the list to detect which
 *                  failed, and for what reason.
 *
 * Usage:           LIMITED_TEST(api_routine, return_code, reply1, reply2)
 *
 *                  where:  api_routine     Is the name of the API routine that
 *                                          was issued.
 *                          return_code     Is the return code from
 *                                          "api_routine".
 *                          reply1          Is the reply host list generated by
 *                                          most PTPE API calls, or the
 *                                          manager failure lists from
 *                                          PtpeColStart / PtpeColStop.  If
 *                                          NULL, no reply lists are printed.
 *                          reply2          Is NULL for most calls, or the
 *                                          non-manager failure lists from
 *                                          PtpeColStart / PtpeColStop
 *
 * Return Codes:    None
 *****************************************************************************/

#define LIMITED_TEST(api_routine, return_code, reply1, reply2)             \
{                                                                           \
    if (return_code == PTPE_LIMITED) {                                      \
        printf("Not all hosts in the target list were successful in\n");    \
        printf("\t performing the %s request.\n", api_routine);             \
        if (reply1 == (host_list_t) NULL) {                                 \
            printf("\t Check previous host list output for error notices\n");\
        }                                                                   \
        else {                                                              \
            if (reply2 == (host_list_t) NULL) {                             \
                printf("\t Hosts that failed the request were:\n");         \
                print_host(reply1);                                         \
            }                                                               \
            else {                                                          \
                printf("\t Managers that failed the request were:\n");      \
                print_host(reply1);                                         \
                printf("\t Non-managers that failed:\n");                   \
                print_host(reply2);                                         \
            }                                                               \
        }                                                                   \
        printf("\t Continuing with application.\n");                        \
    }                                                                       \
}
/*****************************************************************************
 *
 * Function Name:   sig_setup
 *
 * Description:     Sets up the handlers for specific signals.  This is a
 *                  simplistic handler, used for the purposes of demonstration.
 *                  An actual application would consider using a more
 *                  sophisticated setup.
 *
 * Usage:           (void) sig_setup()
 *
 * Return Codes:    None
 *
 ******************************************************************************/

void
sig_setup()

{
    extern void             exit_hdlr();

    signal(SIGHUP, exit_hdlr);
    signal(SIGINT, exit_hdlr);
    signal(SIGQUIT, exit_hdlr);
    signal(SIGABRT, exit_hdlr);
    signal(SIGBUS, exit_hdlr);
    signal(SIGSEGV, exit_hdlr);
    signal(SIGTERM, exit_hdlr);
    signal(SIGXCPU, exit_hdlr);
    signal(SIGDANGER, exit_hdlr);
    signal(SIGPRE, exit_hdlr);
    return;

}
/******************************************************************************
 *
 * Function Name:   exit_hdlr
 *
 * Description:     Termination and signal handler for this application.  The
 *                  primary responsibility of this handler is to close the
 *                  PTPE session when ending the application or terminating
 *                  terminating the application because of a signal.  This is
 *                  necessary, to ensure the global PTPE status information is
 *                  consistent and the ensure that other PTPE applications will
 *                  be able to properly acquire a session.
 *
 * Usage:           (void) exit_hdlr(int sigval)
 *
 *                  where:  sigval  If this is a negative value, it specifies
 *                                  the exit value for the application.  If
 *                                  this is a positive value, it specifies the
 *                                  numeric value of the signal that was
 *                                  received by the application (this is
 *                                  supplied by the operating system).
 *
 * Return Codes:    None
 *
 ******************************************************************************/

void
exit_hdlr(int sigval)
```

```
{

    host_list_t             mgrs, others, reply;
    int                     rc;

    /*
     * Inform the user that the application is terminating, and give
     * appropriate warnings.
     */
     if (sigval < 0) {
         printf ("Application terminating\n");
     }
     else {
         printf("Application terminating because of receipt of signal %d\n",
                 sigval);
     }
     if (sblock != (session_ptr_t) NULL) {
        /*
         * Try to shut down performance information collection and archiving.
         * If neither is active, the routine will return an error, but the
         * error will be ignored.
         */
        mgrs = others = reply = (host_list_t) NULL;
       (void) PtpeArchStopAllHosts(sblock, &reply);
       (void) PtpeFreeHostList(&reply);
       (void) PtpeColStop(sblock, &mgrs, &others);
       (void) PtpeFreeHostList(&mgrs);
       (void) PtpeFreeHostList(&others);
       /*
        * Release the PTPE API session.
        */
       (void) PtpeCloseSession(&sblock);
     }
    /*
     * Terminate the application.
     */
    printf("Verify that performance information collection and archiving \n");
    printf("have been shut down with the 'ptpectrl -q' command.\n");
   exit(sigval);
}
/*****************************************************************************
 *
 * Function Name:   print_host
 *
 * Description:     Prints the contents of a host list, along with any
 *                  reated statistics and reults.
 *
 * Usage:           (void) print_host(host_list_t hosts);
 *
 *                  where:  hosts   Contains the list of hosts to print
 *
 * Return Codes:    None
 *
 *****************************************************************************/

void
print_host(host_list_t hosts)
```

```
{

    int             rc;
    int             breakout;
    int             result;
    char            hostname[PTPE_NMLN];
    stat_list_t     stats;
    extern void     print_result(int);

    /*
     * In printing the contents of the host list, we force the internal
     * pointers of the host list to reference the first entry.  We then extract
     * the name of the first entry from the host list and print it.  We then
     * move to the next entry and repeat the process until we encounter the
     * end of the list.
     *
     * If any of the host entries also contain statistics (which they should,
     * after PtpeArchGetStats is called), the associated statistics list is
     * retrieved and printed out as well.
     */
    breakout = 0;
    rc = PtpeFirstHost(hosts);
    switch (rc) {
        case PTPE_SUCCESS:  break;
        case PTPE_EMPTY:    printf("No entries in host list.\n");
                            break;
        default:            printf("Unexpected failure in print_host(): ");
                            printf("PtpeFirstHost() returned ");
                            print_result(rc);
                            return;
    }
    stats = (stat_list_t) NULL;
    rc = PtpeInitStatList(&stats);
    SUCCESS_TEST("print_host", "PtpeInitStatList", rc);
    printf("Contents of host list:\n");
    for (;;) {
        bzero(hostname, PTPE_NMLN);
        rc = PtpeGetHost(hostname, hosts);
        SUCCESS_TEST("print_host", "PtpeGetHost", rc);
        rc = PtpeGetHostResult(hosts, &result);
        SUCCESS_TEST("print_host", "PtpeGetHostResult", rc);
        printf("\t Host Name: %s -- results: ", hostname);
        print_result(result);
        rc = PtpeGetHostStatList(stats, hosts);
        switch (rc) {
            case PTPE_SUCCESS:  (void) print_stat("\t\t", stats);
                                break;
            case PTPE_EMPTY:    printf("\t\t No statistics listed.\n");
                                break;
            default:            SUCCESS_TEST("print_host",
                                             "PtpeGetHostStatList", rc);
        }
        rc = PtpeEmptyStatList(&stats);
        SUCCESS_TEST("print_host", "PtpeEmptyStatList", rc);
        rc = PtpeIsLastHost(hosts);
        switch (rc) {
            case PTPE_TRUE:     breakout=1;
                                break;
            case PTPE_FALSE:    break;
            default:            SUCCESS_TEST("print_host", "PtpeIsLastHost",
                                              rc);
        }
        if (breakout != 0) {
            break;
        }
        rc = PtpeNextHost(hosts);
        SUCCESS_TEST("print_host", "PtpeNextHost", rc);
  }
  return;

}
/*****************************************************************************
 *
 * Function Name:   print_stat
 *
 * Description:     Prints the contents of a statistic list, along with any
 *                  results
 *
 * Usage:           (void) print_stat(char *indent, stat_list_t stats);
 *
 *                  where:  indent  Just a character string to keep the output
 *                                  format consistent... usually a tab
 *                          stats   Contains the list of statistics to print
 *
 * Return Codes:    None
 *
 *****************************************************************************/

void
print_stat(char *indent, stat_list_t stats)

{

    int             rc;
    int             breakout;
    char            statname[PTPE_STNL];
    char            *timestr;
    unsigned int    result;
    int             sresult;
    float           fdata;
    long            ldata;
    struct tm       tms;
    extern void     print_result(int);

   /*
    * The printing of the statistics list is handled in the same manner as the
    * printing of the host list: reset to the first entry, retrieve the values
    * from the first entry, print the values, move to the next entry, and
    * repeat until the end of the statistics list is found.
    *
    * In order to properly interpret the value of the statistic, the data type
    * of the statistic must be determined.
    */
   rc = PtpeFirstStat(stats);
   switch (rc) {
       case PTPE_SUCCESS:  break;
       case PTPE_EMPTY:    printf("No entries in statistics list.\n");
                           break;
       default:            printf("Unexpected failure in print_stat(): ");
                           printf("PtpeFirstStat() returned ");
                           print_result(rc);
                           return;
    }
    printf("%sStatistics listed:\n", indent);
    breakout = 0;
    for (;;) {
        bzero(statname, PTPE_STNL);
        rc = PtpeGetStatName(statname, stats);
        SUCCESS_TEST("print_stat", "PtpeGetStatName", rc);
        rc = PtpeGetStatResult(stats, &sresult);
        SUCCESS_TEST("print_stat", "PtpeGetStatResult", rc);
        printf("%s\t Statistic: %s -- results: ", indent, statname);
        print_result(sresult);
        bzero((char *) &tms,; sizeof(struct tm));
        rc = PtpeGetStatTime(&tms,; stats);
        SUCCESS_TEST("print_stat", "PtpeGetStatTime", rc);
        timestr = (char *) NULL;
        timestr = asctime(&tms);
        rc = strlen(timestr);
        *(timestr + (rc -1)) = (char) NULL;
        switch (rc) {
            case PTPE_TRUE: rc = PtpeGetStatValueLong(&ldata,; stats);
                            SUCCESS_TEST("print_stat", "PtpeGetStatValueLong",
                                         rc);
                            printf("%s\t\t PTPE_LONG, value %d, time %s\n",
                                   indent, ldata, timestr);
                            break;
            case PTPE_FALSE: break;
            default:        SUCCESS_TEST("print_stat", "PtpeStatIsLong", rc);
        }
        rc = PtpeStatIsFloat(stats);
        switch (rc) {
            case PTPE_TRUE: rc = PtpeGetStatValueFloat(&fdata, stats);
                            SUCCESS_TEST("print_stat", "PtpeGetStatValueFloat",
                                         rc);
                            printf("%s\t\t PTPE_FLOAT, value %f, time %s\n",
                                   indent, fdata, timestr);
                            break;
            case PTPE_FALSE: break;
            default:        SUCCESS_TEST("print_stat", "PtpeStatIsFloat", rc);
        }
        rc = PtpeIsLastStat(stats);
        switch (rc) {
           case PTPE_TRUE:     breakout=1;
                                break;
            case PTPE_FALSE:    break;
            default:            SUCCESS_TEST("print_stat", "PtpeIsLastStat",
                                rc);
        }
        if (breakout != 0) {
            break;
        }
        rc = PtpeNextStat(stats);
        SUCCESS_TEST("print_stat", "PtpeNextStat", rc);
    }

}

/**************************************************************************
 *
 * Function Name:   print_result
 *
 * Description:     Translates result code back into its symbolic name.
 *
 * Usage:           (void) print_result(int code);
 *
 *                  where:  code    Retrieved via PtpeGetResult()
 *
 * Return Codes:    None.
 *
 **************************************************************************/

void
print_result(int code)

{

     int         status;

     /*
      * Return codes from PtpeQuery*HostStatus are > 0, unless an error
      * occured.
      */
     if (code > 0) {
         if (code == PTPE_INACTIVE) {
             printf("inactive\n");
         }
        if (code & PTPE_SAMPLE) {
            printf("sampling ");
            status = 1;
        }
        if (code & PTPE_COLLECT) {
             printf("collecting ");
             status = 1;
        }
        if (code & PTPE_ARCHIVE) {
            printf("archiving ");
            status = 1;
        }
       if (status == 1) {
           printf("\n");
       }
       return;
    }
   /*
    * All other routines should provide one of the following codes (so will
    * PtpeQuery*HostStatus in error cases).
    */
   switch(code) {
       case PTPE_SUCCESS:          printf("PTPE_SUCCESS\n");
                                   break;
       case PTPE_SUCCESS_BADR:     printf("PTPE_SUCCESS_BADR\n");
```

```
                            break;
case PTPE_INV_HOSTNAME:     printf("PTPE_INV_HOSTNAME\n");
                            break;
case PTPE_INV_HOSTLIST:     printf("PTPE_INV_HOSTLIST\n");
                            break;
case PTPE_INV_STATNAME:     printf("PTPE_INV_STATNAME\n");
                            break;
case PTPE_INV_STATLIST:     printf("PTPE_INV_STATLIST\n");
                            break;
case PTPE_NO_MEMORY:        printf("PTPE_NO_MEMORY\n");
                            break;
case PTPE_HOST_NOT_FOUND:   printf("PTPE_HOST_NOT_FOUND\n");
                            break;
case PTPE_STAT_NOT_FOUND:   printf("PTPE_STAT_NOT_FOUND\n");
                            break;
case PTPE_EMPTY:            printf("PTPE_EMPTY\n");
                            break;
case PTPE_BAD_LOC_PTR:      printf("PTPE_BAD_LOC_PTR\n");
                            break;
case PTPE_NO_SESSION:       printf("PTPE_NO_SESSION\n");
                            break;
case PTPE_SDR:              printf("PTPE_SDR\n");
                            break;
case PTPE_NO_HIERARCHY:     printf("PTPE_NO_HIERARCHY\n");
                            break;
case PTPE_NO_STATFILE:      printf("PTPE_NO_STATFILE\n");
                            break;
case PTPE_NO_LOCK_OBJ:      printf("PTPE_NO_LOCK_OBJ\n");
                            break;
case PTPE_NO_CONNECT:       printf("PTPE_NO_CONNECT\n");
                            break;
case PTPE_MEMORY:           printf("PTPE_MEMORY\n");
                            break;
case PTPE_BAD_SEND:         printf("PTPE_BAD_SEND\n");
                            break;
case PTPE_BAD_RECEIVE:      printf("PTPE_BAD_RECEIVE\n");
                            break;
case PTPE_TIMEOUT:          printf("PTPE_TIMEOUT\n");
                            break;
case PTPE_INV_PTR:          printf("PTPE_INV_PTR\n");
                            break;
case PTPE_LIST_END:         printf("PTPE_LIST_END\n");
                            break;
case PTPE_API_FAILED:       printf("PTPE_API_FAILED\n");
                            break;
case PTPE_API_FAILED_BADR:  printf("PTPE_API_FAILED_BADR\n");
                            break;
case PTPE_LIMITED:          printf("PTPE_LIMITED\n");
                            break;
case PTPE_LIMITED_BADR:     printf("PTPE_LIMITED_BADR\n");
                            break;
case PTPE_ARCH_ACTIVE:      printf("PTPE_ARCH_ACTIVE\n");
                            break;
case PTPE_ARCH_OFF:         printf("PTPE_ARCH_OFF\n");
                            break;
case PTPE_ARCH_ERROR:       printf("PTPE_ARCH_ERROR\n");
                            break;
case PTPE_COL_ACTIVE:       printf("PTPE_COL_ACTIVE\n");
```

```
                                 break;
       case PTPE_COL_OFF:         printf("PTPE_COL_OFF\n");
                                 break;
       case PTPE_COL_ERROR:       printf("PTPE_COL_ERROR\n");
                                 break;
       case PTPE_NO_CONTACT:      printf("PTPE_NO_CONTACT\n");
                                 break;
       case PTPE_NO_EXEC:         printf("PTPE_NO_EXEC\n");
                                 break;
       case PTPE_CANT_CLOSE:      printf("PTPE_CANT_CLOSE\n");
                                 break;
       case PTPE_DAEMON_ERROR:    printf("PTPE_DAEMON_ERROR\n");
                                 break;
       case PTPE_STATE:           printf("PTPE_STATE\n");
                                 break;
       case PTPE_TIME_APPROX:     printf("PTPE_TIME_APPROX\n");
                                 break;
       case PTPE_LOCKED:          printf("PTPE_LOCKED\n");
                                 break;
       case PTPE_AUTH:            printf("PTPE_AUTH\n");
                                 break;
       case PTPE_FILE_ERROR:      printf("PTPE_FILE_ERROR\n");
                                 break;
       case PTPE_AGAIN:           printf("PTPE_AGAIN\n");
                                 break;
       case PTPE_RONLY_SESS:      printf("PTPE_RONLY_SESS\n");
                                 break;
       case PTPE_NOT_IMP:         printf("PTPE_NOT_IMP\n");
                                 break;
   }
  return;

}
/*****************************************************************************
 *
 * Function Name:   select_hosts
 *
 * Description:     Selects up to 3 hosts from a host list, and adds those
 *                  host to an initialized host list.
 *
 * Usage:           (void) select_hosts(host_list_t full, host_list_t *part);
 *
 *                  where:  full    List of hosts to select from
 *                          part    Pointer to an uninitialized host list
 *
 * Notes:           The first three hosts in the host list are selected.  If
 *                  the host list does not contain three hosts, all hosts are
 *                  selected.
 *
 * Return Codes:    None
 *
 *****************************************************************************/
void
select_hosts(host_list_t full, host_list_t *part)

{

   int                     rc;
   int                   i;
   char                  hostname[PTPE_NMLN];

   bzero(hostname, PTPE_NMLN);
   rc = PtpeInitHostList(part);
   SUCCESS_TEST("select_hosts", "PtpeInitHostList", rc);
   rc = PtpeFirstHost(full);
   SUCCESS_TEST("select_hosts", "PtpeFirstHost", rc);
   for (i = 0 ; i < 3 ; i++) {
       rc = PtpeIsLastHost(full);
       if (rc == PTPE_TRUE) {
           break;
       }
       if (rc == PTPE_FALSE) {
           rc = PtpeGetHost(hostname, full);
           SUCCESS_TEST("select_hosts", "PtpeGetHost", rc);
           rc = PtpeAddHostToList(hostname, *part);
           SUCCESS_TEST("select_hosts", "PtpeAddHostToList", rc);
           rc = PtpeNextHost(full);
           SUCCESS_TEST("select_hosts", "PtpeNextHost", rc);
       }
       else {
           SUCCESS_TEST("select_hosts", "PtpeIsLastHost", rc);
       }
    }
    return;

}
/*****************************************************************************
 *
 * Function Name:   setup_stats
 *
 * Description:     Prepares a list of statistics.  This list will contain the
 *                  statistics that the application wishes to retrieve from
 *                  a node's performance information archive.
 *
 * Usage:           (void) setup_stats(stat_list_t *slist);
 *
 *                  where:  slist       Is a pointer to a statistics list
 *                                      anchor which has not been initialized.
 *
 * Notes:
 *
 * Return Codes:    None
 *
 *****************************************************************************/

void
setup_stats(stat_list_t *slist)

{

    int                 rc;
    int                 i;

    rc = PtpeInitStatList(slist);
    SUCCESS_TEST("setup_stats", "PtpeInitStatList(slist)", rc);
    for (i = 0 ; i < NUM_EX_STATS ; i++) {
        rc = PtpeAddStatToList(test_stats[i], *slist);
        SUCCESS_TEST("setup_stats", "PtpeInitStatList(slist)", rc);
    }
    return;

}
/*****************************************************************************
 *
 * Function Name:   set_stats_time
 *
 * Description:     Set the timestamp on statistics within a statistics list
 *                  according to the caller's specification.
 *
 * Usage:           set_stats_time(stat_list_t slist, int whence,
 *                                 struct tm *tstamp)
 *
 *                  where:  slist       Contains statistics to be timestamped
 *                          whence      Is one of PTPE_MATCH, PTPE_EARLIEST, or
 *                                      PTPE_LATEST
 *                          tstamp      When "whence" is PTPE_MATCH, this
 *                                      references the time to set the
 *                                      statistic's timestamp to
 *
 * Notes:
 *
 * Return Codes:    None
 *
 *****************************************************************************/

void
set_stats_time(stat_list_t slist, int whence, struct tm *tstamp)

{

    int                     rc;

    /*
     * This routine sets the timestamps of all statistics to the same value.
     * This is not a requirement - you may set the timestamps to differing
     * values and differing "whence" indications.  This example uses the same
     * timestamp and "whence" values for simplicity sake.
     */
    printf("\t Setting timestamps for all statistics in list.\n");
    rc = PtpeFirstStat(slist);
    SUCCESS_TEST("set_stats_time", "PtpeFirstStat", rc);
    for (;;) {
         rc = PtpeSetStatTime(whence, tstamp, slist);
         prinf("\t\t PtpeSetStatTime() returned: ");
         print_result(rc);
         SUCCESS_TEST("set_stats_time", "PtpeSetStatTime", rc);
         rc = PtpeIsLastStat(slist);
         if (rc != PTPE_TRUE && rc != PTPE_FALSE) {
             SUCCESS_TEST("set_stats_time", "PtpeIsLastStat", rc);
         }
         if (rc == PTPE_TRUE) {
             break;
         }
         rc = PtpeNextStat(slist);
```

```
            SUCCESS_TEST("set_stats_time", "PtpeNextStat", rc);
        }

}
/******************************************************************************
 *
 * Function Name:   assign_stats
 *
 * Description:     This routine assigns a statistics list to each entry in a
 *                  host list.  Statistics lists must be assigned to entries in
 *                  a host list whenever the application wants to perform
 *                  operations on specific statistics on these nodes.
 *
 * Usage:           (void) assign_stats(host_list_t targets, stat_list_t slist);
 *
 *                  where:  targets     Is the list of nodes involved in later
 *                                      API calls.
 *                          slist       Is the list of statistics the applica-
 *                                      tion will retrieve from these nodes.
 *
 * Notes:
 *
 * Return Codes:    None
 *
 ******************************************************************************/

void
assign_stats(host_list_t targets, stat_list_t slist)

{

    int             rc;

    /*
     * This example assigns the same statistics list to all nodes listed in a
     * host list.  This is not a requirement - each node within a host list
     * may have any statistics list assigned to it, and none of the lists need
     * to match.  The same list is used here for coding simplicity.
     */
    rc = PtpeFirstHost(targets);
    SUCCESS_TEST("assign_stats", "PtpeFirstHost()", rc);
    for (;;) {
        /*
         * Before assigning the statistics list, make sure that the entry in
         * the host list does not already contain a statistics list.
         */
        (void) PtpeRemoveStatsFromHost(targets);
        rc = PtpeAssignStatsToHost(slist, targets);
        SUCCESS_TEST("assign_stats", "PtpeAssignStatsToHost()", rc);
        rc = PtpeIsLastHost(targets);
        if (rc != PTPE_TRUE && rc != PTPE_FALSE) {
            SUCCESS_TEST("assign_stats", "PtpeIsLastHost()", rc);
        }
        if (rc == PTPE_TRUE) {
            break;
        }
        rc = PtpeNextHost(targets);
        SUCCESS_TEST("assign_stats", "PtpeNextHost()", rc);
    }
    return;

}
/*****************************************************************************
 *
 * Function Name:   get_different_times
 *
 * Description:     Uses PtpeArchGetStats() to retrieve the earliest and
 *                  latest entries for specific statistics in a host's
 *                  performance information archive, as well as those entries
 *                  that match the specified timestamps.
 *
 * Usage:           get_different_times(host_list_t targets, stat_list_t slist,
 *                                      struct tm *tms)
 *
 *                  where:  targets     Are the hosts to contact to get the
 *                                      statistics
 *                          slist       Contains the statistics to retrieve
 *                          tms         Specifies the timestamp to match
 *
 * Notes:
 *
 * Return Codes:    None.
 *
 *****************************************************************************/

void
get_different_times(host_list_t targets, stat_list_t slist, struct tm *tms)

{

    int                     rc;
    host_list_t             reply;
    extern session_ptr_t    sblock;

    reply = (host_list_t) NULL;

    /*
     * First attempt - set statistic timestamp to PTPE_MATCH, and retrieve the
     * values in the nodes' performance information archive that is closest to
     * the specified timestamp value.
     */
    printf("Attempting to get statistics that MATCH the timestamps provided ");
    printf(" - timestamp\n\t requested is %s.\n", asctime(tms));
    set_stats_time(slist, PTPE_MATCH, tms);
    assign_stats(targets, slist);
    rc = PtpeArchGetStats(sblock, targets, &reply);
    printf("\t PtpeArchGetStats() returned: ");
    print_result(rc);
    printf("Host list returned by PtpeArchGetStats():\n");
    print_host(reply);
    SUCCESS_TEST("get_different_times", "PtpeArchGetStats(1)", rc);
    LIMITED_TEST("PtpeArchGetStats(1)", rc, ((host_list_t) NULL),
                ((host_list_t) NULL));
    rc = PtpeFreeHostList(&reply);
    SUCCESS_TEST("get_different_times", "PtpeFreeHostList(1)", rc);
    /*
     * Second attempt - set statistics timestamp to PTPE_EARLIEST, and retrieve
     * the earliest entries for these statistics in the nodes' performance
     * information archives.
     */
    printf("Attempting to get EARLIEST entry for statistics \n");
    set_stats_time(slist, PTPE_EARLIEST, tms);
    assign_stats(targets, slist);
    rc = PtpeArchGetStats(sblock, targets, &reply);
    printf("\t PtpeArchGetStats() returned: ");
    print_result(rc);
    printf("Host list returned by PtpeArchGetStats():\n");
    print_host(reply);
    SUCCESS_TEST("get_different_times", "PtpeArchGetStats(2)", rc);
    LIMITED_TEST("PtpeArchGetStats(2)", rc, ((host_list_t) NULL),
                 ((host_list_t) NULL));
    rc = PtpeFreeHostList(&reply);
    SUCCESS_TEST("get_different_times", "PtpeFreeHostList(2)", rc);

    /*
     * Third attempt - set statistics timestamp to PTPE_LATEST, and find the
     * last entry for each statistic in the nodes' performance information
     * archives.
     */
     printf("Attempting to get LATEST entry for statistics \n");
     set_stats_time(slist, PTPE_LATEST, tms);
     assign_stats(targets, slist);
     rc = PtpeArchGetStats(sblock, targets, &reply);
     printf("\t PtpeArchGetStats() returned: ");
     print_result(rc);
     printf("Host list returned by PtpeArchGetStats():\n");
     print_host(reply);
     SUCCESS_TEST("get_different_times", "PtpeArchGetStats(3)", rc);
     LIMITED_TEST("PtpeArchGetStats(3)", rc, ((host_list_t) NULL),
                  ((host_list_t) NULL));
     rc = PtpeFreeHostList(&reply);
     SUCCESS_TEST("get_different_times", "PtpeFreeHostList(3)", rc);

     return;
}
/******************************************************************************
 *
 * Function Name:   retry_test
 *
 * Description:     Checks if any hosts replying to a PTPE API routine have
 *                  indicated that the routine should be retried at a later
 *                  time.
 *
 * Usage:           int = retry_test(host_list_t reply)
 *
 *                  where:  reply       Is the reply host list generated from
 *                                      a PTPE API library routine.
 *
 * Notes:           Notice that this routine does not validate its parameters.
 *                  The caller must validate the parameters before calling this
 *                  routine.
 *
 * Return Codes:    (int)  0            All nodes have responded with a reply
 *                                      other than "retry the routine"
 *                  (int)  1           At least one node within the reply list
 *                                      responed with a "retry the routine"
 *
 ******************************************************************************/

int
retry_test(host_list_t reply)

{

    int                     rc;
    int                     result;

    rc = PtpeFirstHost(reply);
    SUCCESS_TEST("retry_test", "PtpeFirstHost", rc);
    for (;;) {
        result = 0;
        rc = PtpeGetHostResult(reply, &result);
        SUCCESS_TEST("retry_test", "PtpeGetHostResult", rc);
        if (result == PTPE_AGAIN) {
            rc = PtpeFirstHost(reply);
            SUCCESS_TEST("retry_test", "PtpeFirstHost", rc);
            return(1);
        }
        rc = PtpeIsLastHost(reply);
        if (rc != PTPE_TRUE && rc != PTPE_FALSE) {
            SUCCESS_TEST("retry_test", "PtpeIsLastHost", rc);
        }
        if (rc == PTPE_TRUE) {
            break;
        }
        rc = PtpeNextHost(reply);
        SUCCESS_TEST("retry_test", "PtpeNextHost", rc);
    }
    rc = PtpeFirstHost(reply);
    SUCCESS_TEST("retry_test", "PtpeFirstHost", rc);
    return(0);

}
/******************************************************************************
 *
 *     -- MAIN ROUTINE -- MAIN ROUTINE -- MAIN ROUTINE -- MAIN ROUTINE --
 *
 ******************************************************************************/

main(int arcg, char **argv)

{

    int                     rc;
    int                     i, j;
    int                     done;
    int                     srate, arate;
    host_list_t             full, targets, reply;
    host_list_t             mgrs, others;
    stat_list_t             slist;
    struct timeval          tp;
    struct tm              *tms;
    extern session_ptr_t    sblock;
    extern void             sig_setup();

    /*
       * Set up an API session.  A session is required to permit the application
       * to issue commands to the monitoring hierarchy.
       */
    sig_setup();
    printf("Establishing an API session now.\n");
    sblock = (session_ptr_t) NULL;
    rc = PtpeOpenSession(&sblock);
    SUCCESS_TEST("main", "PtpeOpenSession", rc);

    /*
       * Set up a host list to be used in obtaining performance information from
       * the performance information archives.  Will use the PtpeQueryAvailHosts
       * API routine to get a list of the hosts that are available within the
       * monitoring hierarchy, and will then pick three nodes from this list.
       * Later on, this program will obtain performance information from the
       * archives on these three systems.
       */
    full = targets = reply = (host_list_t) NULL;
    rc = PtpeInitHostList(&full);;
    SUCCESS_TEST("main", "PtpeInitHostList(full)", rc);
    printf("Obtaining list of available hosts.\n");
    rc = PtpeQueryAvailHosts(sblock, &full);
    SUCCESS_TEST("main", "PtpeQueryAvailHosts", rc);
    printf("Setting up target host list for later use.\n");
    select_hosts(full, &targets);
    printf("The host list we will be using in this program is:\n");
    print_host(targets);
    /*
       * Set up a list of statistics to retrieve from the archives of these
       * nodes that were just selected.
       */
    printf("Setting up the statistics list.  This list will contain the\n");
    printf("\t statistics that will be obtained from the performance\n");
    printf("\t archives on the nodes that were selected in the previous\n");
    printf("\t step.\n");
    slist = (stat_list_t) NULL;
    setup_stats(&slist);
    printf("The statistics that will be retrieved from the archives are:\n");
    print_stat("\t", slist);

    /*
       * Nodes and statistics have been selected.  Start collection and archiving
       * in the monitoring hierarchy, and enable all statistics within the
       * monitoring hierarchy for collectioon and archiving.
       */
    printf("Starting collection in the monitoring hierarchy now.\n");
    printf("\t (this may take a few moments...)\n");
    mgrs = others = (host_list_t) NULL;
    rc = PtpeColStart(sblock, &mgrs,; &others);
    if (mgrs != (host_list_t) NULL) {
        printf("The following manager nodes did not start collection:\n");
        print_host(mgrs);
    }
```

```
if (others != (host_list_t) NULL) {
    printf("The following nodes did not start collection:\n");
    print_host(others);
}
SUCCESS_TEST("main", "PtpeColStart", rc);
LIMITED_TEST("PtpeColStart", rc, mgrs, others);
(void) PtpeFreeHostList(&mgrs);
(void) PtpeFreeHostList(&others);
/*
 * Note that the reply list from the following routine is checked for any
 * "not ready" responses from nodes, and that the application repeatidly
 * issues the routine until all nodes respond with an answer other than
 * "not ready".
 */
 printf("Enabling all available statistics for collection.\n");
 reply = (host_list_t) NULL;
 for (;;) {
     (void) PtpeFreeHostList(&reply);
      rc = PtpeColEnableAllStats(sblock, full, &reply);
      switch (rc) {
          case PTPE_LIMITED:
          case PTPE_API_FAILED:       i = retry_test(reply);
                                      if (i == 0) {
                                          done = 1;
                                      }
                                      break;
          default:                    done = 1;
                                      break;
      }
      if (done == 1) {
          break;
      }
      printf("All hosts are not ready to handle PTPE requests at this \n");
      printf("\t time.  Will pause for 15 seconds and try request again.\n");
      sleep(15);
 }
/*
 * Now start archiving in the hierarchy, and enable all available
 * statistics for archiving.
 */
 (void) PtpeFreeHostList(&reply);
 printf("Starting performance information archiving in the hierarchy.\n");
 rc = PtpeArchStartAllHosts(sblock, &reply);
 SUCCESS_TEST("main", "PtpeArchStartAllHosts", rc);
 LIMITED_TEST("PtpeArchStartAllHosts", rc, reply, ((host_list_t) NULL));
 (void) PtpeFreeHostList(&reply);
 printf("Enabling all available statistics for archiving.\n");
 rc = PtpeArchEnableAllStats(sblock, full, &reply);
 SUCCESS_TEST("main", "PtpeArchEnableAllStats", rc);
 LIMITED_TEST("PtpeArchEnableAllStats", rc, reply, ((host_list_t) NULL));
(void) PtpeFreeHostList(&reply);

/*
 * Set up a timestamp to retrieve the statistics recorded by each node in
 * the target list a minute or so into the future.
 */
 bzero((char *) &tp, sizeof(struct timeval));
 tms = (struct tm *) NULL;
 rc = gettimeofday(&tp, (struct timezone *) NULL);
 if (rc != 0) {
     printf("\t FAILURE IN gettimeofday - TERMINATING PROGRAM.\n");
     (void) PtpeFreeHostList(&targets);
     (void) PtpeFreeHostList(&full);
     (void) PtpeFreeStatList(&slist);
     exit_hdlr(-1);
 }
 tp.tv_sec += 85;
 tms = localtime((time_t *) &(tp;tv_sec));

/*
 * Now the program should wait for some new data to be recordedin the
 * performance information archives.  The archiving rate has not been
 * changed from the default, but to avoid making assumptions about the
 * default archiving rate, the application will retrieve the current
 * archiving rate, and pause itself for three archiving intervals to
 * Let data accumulate.
 */
 rc = PtpeQueryHostRates(sblock, &srate, &arate);
 SUCCESS_TEST("main", "PtpeQueryHostRates", rc);
 printf("Pausing %d seconds to allow performance information", (arate * 3));
 printf("\n\t to accumulate in the archive.\n");
 sleep(arate * 3);

/*
 * Now attempt to get the statistics that were specified from the archives
 * on the nodes that were selected.
 */
 get_different_times(targets, slist, tms);

/*
 * Shut down collection and archiving, and exit.
 */
printf("Shutting down archiving... ");
fflush(stdout);
rc = PtpeArchStopAllHosts(sblock, &reply);
SUCCESS_TEST("main", "PtpeArchStopAllHosts", rc);
LIMITED_TEST("PtpeArchStopAllHosts", rc, reply, ((host_list_t) NULL));
printf("and collection.\n");
rc = PtpeColStop(sblock, &mgrs,; &others);
SUCCESS_TEST("main", "PtpeColStop", rc);
LIMITED_TEST("PtpeColStop", rc, reply, ((host_list_t) NULL));
(void) PtpeFreeHostList(&others);
(void) PtpeFreeHostList(&mgrs);
(void) PtpeFreeHostList(&targets);
(void) PtpeFreeHostList(&reply);
(void) PtpeFreeHostList(&full);
(void) PtpeFreeStatList(&slist);
rc = PtpeCloseSession(&sblock);
if (rc != PTPE_SUCCESS) {
    printf("COULD NOT CLOSE SESSION - error code is ");
    print_result(rc);
}
else {
    printf("Program successfully completed.\n");
}
return(0);
```

```
}
```

Bibliography

This bibliography helps you find product documentation related to the RS/6000 SP hardware and software products.

You can find most of the IBM product information for RS/6000 SP products on the World Wide Web. Formats for both viewing and downloading are available.

PSSP documentation is shipped with the PSSP product in a variety of formats and can be installed on your system. The man pages for public code that PSSP includes are also available online.

You can order hard copies of the product documentation from IBM. This bibliography lists the titles that are available and their order numbers.

Finally, this bibliography contains a list of non-IBM publications that discuss parallel computing and other topics related to the RS/6000 SP.

Finding Documentation on the World Wide Web

Most of the RS/6000 SP hardware and software books are available from the IBM RS/6000 web site at **http://www.rs6000.ibm.com**. You can view a book or download a Portable Document Format (PDF) version of it. At the time this manual was published, the full path to the "RS/6000 SP Product Documentation Library" page was **http://www.rs6000.ibm.com/resource/aix_resource/sp_books**. However, the structure of the RS/6000 web site can change over time.

Accessing PSSP Documentation Online

On the same medium as the PSSP product code, IBM ships PSSP man pages, HTML files, and PDF files. In order to use these publications, you must first install the **ssp.docs** file set.

To view the PSSP HTML publications, you need access to an HTML document browser such as Netscape. The HTML files and an index that links to them are installed in the **/usr/lpp/ssp/html** directory. Once installed, you can also view the HTML files from the RS/6000 SP Resource Center.

If you have installed the SP Resource Center on your SP system, you can access it by entering the **/usr/lpp/ssp/bin/resource_center** command. If you have the SP Resource Center on CD-ROM, see the **readme.txt** file for information about how to run it.

To view the PSSP PDF publications, you need access to the Adobe Acrobat Reader 3.0.1. The Acrobat Reader is shipped with the AIX Version 4.3 Bonus Pack and is also freely available for downloading from the Adobe web site at URL **http://www.adobe.com**.

Manual Pages for Public Code

The following manual pages for public code are available in this product:

SUP	/usr/lpp/ssp/man/man1/sup.1
NTP	/usr/lpp/ssp/man/man8/xntpd.8
	/usr/lpp/ssp/man/man8/xntpdc.8
Perl (Version 4.036)	/usr/lpp/ssp/perl/man/perl.man
	/usr/lpp/ssp/perl/man/h2ph.man

	/usr/lpp/ssp/perl/man/s2p.man
	/usr/lpp/ssp/perl/man/a2p.man
Perl (Version 5.003)	Man pages are in the /usr/lpp/ssp/perl5/man/man1 directory

Manual pages and other documentation for **Tcl**, **TclX**, **Tk**, and **expect** can be found in the compressed **tar** files located in the **/usr/lpp/ssp/public** directory.

RS/6000 SP Planning Publications

This section lists the IBM product documentation for planning for the IBM RS/6000 SP hardware and software.

IBM RS/6000 SP:

- *Planning, Volume 1, Hardware and Physical Environment*, GA22-7280
- *Planning, Volume 2, Control Workstation and Software Environment*, GA22-7281

RS/6000 SP Hardware Publications

This section lists the IBM product documentation for the IBM RS/6000 SP hardware.

IBM RS/6000 SP:

- *Planning, Volume 1, Hardware and Physical Environment*, GA22-7280
- *Planning, Volume 2, Control Workstation and Software Environment*, GA22-7281
- *Maintenance Information, Volume 1, Installation and Relocation*, GA22-7375
- *Maintenance Information, Volume 2, Maintenance Analysis Procedures*, GA22-7376
- *Maintenance Information, Volume 3, Locations and Service Procedures*, GA22-7377
- *Maintenance Information, Volume 4, Parts Catalog*, GA22-7378

RS/6000 SP Switch Router Publications

The RS/6000 SP Switch Router is based on the Ascend GRF switched IP router product from Ascend Communications, Inc.. You can order the SP Switch Router as the IBM 9077.

The following publications are shipped with the SP Switch Router. You can also order these publications from IBM using the order numbers shown.

- *Ascend GRF Getting Started*, GA22-7368
- *Ascend GRF Configuration Guide*, GA22-7366
- *Ascend GRF Reference Guide*, GA22-7367
- *IBM SP Switch Router Adapter Guide*, GA22-7310.

RS/6000 SP Software Publications

This section lists the IBM product documentation for software products related to the IBM RS/6000 SP. These products include:

- IBM Parallel System Support Programs for AIX (PSSP)
- IBM LoadLeveler for AIX (LoadLeveler)
- IBM Parallel Environment for AIX (Parallel Environment)
- IBM General Parallel File System for AIX (GPFS)

- IBM Engineering and Scientific Subroutine Library (ESSL) for AIX
- IBM Parallel ESSL for AIX
- IBM High Availability Cluster Multi-Processing for AIX (HACMP)
- IBM Client Input Output/Sockets (CLIO/S)
- IBM Network Tape Access and Control System for AIX (NetTAPE)

PSSP Publications

IBM RS/6000 SP:

- *Planning, Volume 2, Control Workstation and Software Environment*, GA22-7281

PSSP:

- *Installation and Migration Guide*, GA22-7347
- *Administration Guide*, SA22-7348
- *Managing Shared Disks*, SA22-7349
- *Performance Monitoring Guide and Reference*, SA22-7353
- *Diagnosis Guide*, GA22-7350
- *Command and Technical Reference*, SA22-7351
- *Messages Reference*, GA22-7352

RS/6000 Cluster Technology (RSCT):

- *Event Management Programming Guide and Reference*, SA22-7354
- *Group Services Programming Guide and Reference*, SA22-7355

As an alternative to ordering the individual books, you can use SBOF-8587 to order the PSSP software library.

LoadLeveler Publications

LoadLeveler:

- *Using and Administering*, SA22-7311
- *Diagnosis and Messages Guide*, GA22-7277

GPFS Publications

GPFS:

- *Installation and Administration Guide*, SA22-7278

Parallel Environment Publications

Parallel Environment:

- *Installation Guide*, GC28-1981
- *Hitchhiker's Guide*, GC23-3895
- *Operation and Use, Volume 1*, SC28-1979
- *Operation and Use, Volume 2*, SC28-1980
- *MPI Programming and Subroutine Reference*, GC23-3894
- *MPL Programming and Subroutine Reference*, GC23-3893
- *Messages*, GC28-1982

As an alternative to ordering the individual books, you can use SBOF-8588 to order the PE library.

Parallel ESSL and ESSL Publications

- *ESSL Products: General Information*, GC23-0529
- *Parallel ESSL: Guide and Reference*, SA22-7273
- *ESSL: Guide and Reference*, SA22-7272

HACMP Publications

HACMP:

- *Concepts and Facilities*, SC23-1938
- *Planning Guide*, SC23-1939
- *Installation Guide*, SC23-1940
- *Administration Guide*, SC23-1941
- *Troubleshooting Guide*, SC23-1942
- *Programming Locking Applications*, SC23-1943
- *Programming Client Applications*, SC23-1944
- *Master Index and Glossary*, SC23-1945
- *HANFS for AIX Installation and Administration Guide*, SC23-1946
- *Enhanced Scalability Installation and Administration Guide*, SC23-1972

CLIO/S Publications

CLIO/S:

- *General Information*, GC23-3879
- *User's Guide and Reference*, GC28-1676

NetTAPE Publications

NetTAPE:

- *General Information*, GC23-3990
- *User's Guide and Reference*, available from your IBM representative

AIX and Related Product Publications

For the latest information on AIX and related products, including RS/6000 hardware products, see *AIX and Related Products Documentation Overview*, SC23-2456. You can order a hard copy of the book from IBM. You can also view it online from the "AIX Online Publications and Books" page of the RS/6000 web site, at URL **http://www.rs6000.ibm.com/resource/aix_resource/Pubs**.

Red Books

IBM's International Technical Support Organization (ITSO) has published a number of redbooks related to the RS/6000 SP. For a current list, see the ITSO website, at URL **http://www.redbooks.ibm.com**.

Non-IBM Publications

Here are some non-IBM publications that you may find helpful.

- Almasi, G., Gottlieb, A., *Highly Parallel Computing*, Benjamin-Cummings Publishing Company, Inc., 1989.
- Foster, I., *Designing and Building Parallel Programs*, Addison-Wesley, 1995.
- Gropp, W., Lusk, E., Skjellum, A., *Using MPI*, The MIT Press, 1994.
- Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard, Version 1.1*, University of Tennessee, Knoxville, Tennessee, June 6, 1995.
- Message Passing Interface Forum, *MPI-2: Extensions to the Message-Passing Interface, Version 2.0*, University of Tennessee, Knoxville, Tennessee, July 18, 1997.
- Ousterhout, John K., *Tcl and the Tk Toolkit*, Addison-Wesley, Reading, MA, 1994, ISBN 0-201-63337-X.
- Pfister, Gregory, F., *In Search of Clusters*, Prentice Hall, 1998.

Glossary of Terms and Abbreviations

This glossary includes terms and definitions from:

- The *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994.
- The *American National Standard Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies can be purchased from the American National Standards Institute, 1430 Broadway, New York, New York 10018. Definitions are identified by the symbol (A) after the definition.
- The *ANSI/EIA Standard - 440A: Fiber Optic Terminology* copyright 1989 by the Electronics Industries Association (EIA). Copies can be purchased from the Electronic Industries Association, 2001 Pennsylvania Avenue N.W., Washington, D.C. 20006. Definitions are identified by the symbol (E) after the definition.
- The *Information Technology Vocabulary* developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions of published parts of this vocabulary are identified by the symbol (I) after the definition; definitions taken from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol (T) after the definition, indicating that final agreement has not yet been reached among the participating National Bodies of SC1.

The following cross-references are used in this glossary:

Contrast with. This refers to a term that has an opposed or substantively different meaning.
See. This refers the reader to multiple-word terms in which this term appears.
See also. This refers the reader to terms that have a related, but not synonymous, meaning.
Synonym for. This indicates that the term has the same meaning as a preferred term, which is defined in the glossary.

This section contains some of the terms that are commonly used in the SP publications.

IBM is grateful to the American National Standards Institute (ANSI) for permission to reprint its definitions from the American National Standard *Vocabulary for Information Processing* (Copyright 1970 by American National Standards Institute, Incorporated), which was prepared by Subcommittee X3K5 on Terminology and Glossary of the American National Standards Committee X3. ANSI definitions are preceded by an asterisk (*).

Other definitions in this glossary are taken from *IBM Vocabulary for Data Processing, Telecommunications, and Office Systems* (SC20-1699) and *IBM DATABASE 2 Application Programming Guide for TSO Users* (SC26-4081).

A

adapter. An adapter is a mechanism for attaching parts. For example, an adapter could be a part that electrically or physically connects a device to a computer or to another device. In the SP system, network connectivity is supplied by various adapters, some optional, that can provide connection to I/O devices, networks of workstations, and mainframe networks. Ethernet, FDDI, token-ring, HiPPI, SCSI, FCS, and ATM are examples of adapters that can be used as part of an SP system.

address. A character or group of characters that identifies a register, a device, a particular part of storage, or some other data source or destination.

AFS. A distributed file system that provides authentication services as part of its file system creation.

AIX. Abbreviation for Advanced Interactive Executive, IBM's licensed version of the UNIX operating system. AIX is particularly suited to support technical computing applications, including high function graphics and floating point computations.

Amd. Berkeley Software Distribution automount daemon.

API. Application Programming Interface. A set of programming functions and routines that provide access between the Application layer of the OSI seven-layer model and applications that want to use the network. It is a software interface.

application. The use to which a data processing system is put; for example, a payroll application, an airline reservation application.

application data. The data that is produced using an application program.

ARP. Address Resolution Protocol.

ATM. Asynchronous Transfer Mode. (See *TURBOWAYS 100 ATM Adapter*.)

Authentication. The process of validating the identity of a user or server.

Authorization. The process of obtaining permission to perform specific actions.

B

batch processing. * (1) The processing of data or the accomplishment of jobs accumulated in advance in such a manner that each accumulation thus formed is processed or accomplished in the same run. * (2) The processing of data accumulating over a period of time. * (3) Loosely, the execution of computer programs serially. (4) Computer programs executed in the background.

BMCA. Block Multiplexer Channel Adapter. The block multiplexer channel connection allows the RS/6000 to communicate directly with a host System/370 or System/390; the host operating system views the system unit as a control unit.

BOS. The AIX Base Operating System.

C

call home function. The ability of a system to call the IBM support center and open a PMR to have a repair scheduled.

CDE. Common Desktop Environment. A graphical user interface for UNIX.

charge feature. An optional feature for either software or hardware for which there is a charge.

CLI. Command Line Interface.

client. * (1) A function that requests services from a server and makes them available to the user. * (2) A term used in an environment to identify a machine that uses the resources of the network.

Client Input/Output Sockets (CLIO/S). A software package that enables high-speed data and tape access between SP systems, AIX systems, and ES/9000 mainframes.

CLIO/S. Client Input/Output Sockets.

CMI. Centralized Management Interface provides a series of SMIT menus and dialogues used for defining and querying the SP system configuration.

connectionless. A communication process that takes place without first establishing a connection.

connectionless network. A network in which the sending logical node must have the address of the receiving logical node before information interchange can begin. The packet is routed through nodes in the network based on the destination address in the packet. The originating source does not receive an acknowledgment that the packet was received at the destination.

control workstation. A single point of control allowing the administrator or operator to monitor and manage the SP system using the IBM AIX Parallel System Support Programs.

css. Communication subsystem.

D

daemon. A process, not associated with a particular user, that performs system-wide functions such as administration and control of networks, execution of time-dependent activities, line printer spooling and so forth.

DASD. Direct Access Storage Device. Storage for input/output data.

DCE. Distributed Computing Environment.

DFS. distributed file system. A subset of the IBM Distributed Computing Environment.

DNS. Domain Name Service. A hierarchical name service which maps high level machine names to IP addresses.

E

Error Notification Object. An object in the SDR that is matched with an error log entry. When an error log entry occurs that matches the Notification Object, a user-specified action is taken.

ESCON. Enterprise Systems Connection. The ESCON channel connection allows the RS/6000 to communicate directly with a host System/390; the host operating system views the system unit as a control unit.

Ethernet. (1) Ethernet is the standard hardware for TCP/IP local area networks in the UNIX marketplace. It is a 10-megabit per second baseband type LAN that allows multiple stations to access the transmission medium at will without prior coordination, avoids contention by using carrier sense and deference, and resolves contention by collision detection (CSMA/CD). (2) A passive coaxial cable whose interconnections contain devices or components, or both, that are all active. It uses CSMA/CD technology to provide a best-effort delivery system.

Ethernet network. A baseband LAN with a bus topology in which messages are broadcast on a coaxial cabling using the carrier sense multiple access/collision detection (CSMA/CD) transmission method.

event. In Event Management, the notification that an expression evaluated to true. This evaluation occurs each time an instance of a resource variable is observed.

expect. Programmed dialogue with interactive programs.

expression. In Event Management, the relational expression between a resource variable and other elements (such as constants or the previous value of an instance of the variable) that, when true, generates an event. An example of an expression is `X < 10` where X represents the resource variable `IBM.PSSP.aixos.PagSp.%totalfree` (the percentage of total free paging space). When the expression is true, that is, when the total free paging space is observed to be less than 10%, the Event Management subsystem generates an event to notify the appropriate application.

F

failover. Also called fallover, the sequence of events when a primary or server machine fails and a secondary or backup machine assumes the primary workload. This is a disruptive failure with a short recovery time.

fall back. Also called fallback, the sequence of events when a primary or server machine takes back control of its workload from a secondary or backup machine.

FDDI. Fiber Distributed Data Interface.

Fiber Distributed Data Interface (FDDI). An American National Standards Institute (ANSI) standard for 100-megabit-per-second LAN using optical fiber cables. An FDDI local area network (LAN) can be up to 100 km (62 miles) and can include up to 500 system units. There can be up to 2 km (1.24 miles) between system units and/or concentrators.

File Transfer Protocol (FTP). The Internet protocol (and program) used to transfer files between hosts. It is an application layer protocol in TCP/IP that uses TELNET and TCP protocols to transfer bulk-data files between machines or hosts.

file. * A set of related records treated as a unit, for example, in stock control, a file could consist of a set of invoices.

file name. A CMS file identifier in the form of 'filename filetype filemode' (like: TEXT DATA A).

file server. A centrally located computer that acts as a storehouse of data and applications for numerous users of a local area network.

File Transfer Protocol (FTP). The Internet protocol (and program) used to transfer files between hosts. It is an application layer protocol in TCP/IP that uses TELNET and TCP protocols to transfer bulk-data files between machines or hosts.

foreign host. Any host on the network other than the local host.

FTP. File transfer protocol.

G

gateway. An intelligent electronic device interconnecting dissimilar networks and providing protocol conversion for network compatibility. A gateway provides transparent access to dissimilar networks for nodes on either network. It operates at the session presentation and application layers.

H

HACMP. High Availability Cluster Multi-Processing for AIX.

HACWS. High Availability Control Workstation function, based on HACMP, provides for a backup control workstation for the SP system.

Hashed Shared Disk (HSD). The data striping device for the IBM Virtual Shared Disk. The device driver lets application programs stripe data across physical disks in multiple IBM Virtual Shared Disks, thus reducing I/O bottlenecks.

help key. In the SP graphical interface, the key that gives you access to the SP graphical interface help facility.

High Availability Cluster Multi-Processing. An IBM facility to cluster nodes or components to provide high availability by eliminating single points of failure.

HiPPI. High Performance Parallel Interface. RS/6000 units can attach to a HiPPI network as defined by the ANSI specifications. The HiPPI channel supports burst rates of 100 Mbps over dual simplex cables; connections can be up to 25 km in length as defined by the standard and can be extended using third-party HiPPI switches and fiber optic extenders.

home directory. The directory associated with an individual user.

host. A computer connected to a network, and providing an access method to that network. A host provides end-user services.

I

instance vector. Obsolete term for resource identifier.

Intermediate Switch Board. Switches mounted in the Sp Switch expansion frame.

Internet. A specific inter-network consisting of large national backbone networks such as APARANET, MILNET, and NSFnet, and a myriad of regional and campus networks all over the world. The network uses the TCP/IP protocol suite.

Internet Protocol (IP). (1) A protocol that routes data through a network or interconnected networks. IP acts as an interface between the higher logical layers and the physical network. This protocol, however, does not provide error recovery, flow control, or guarantee the reliability of the physical network. IP is a connectionless protocol. (2) A protocol used to route data from its source to it destination in an Internet environment.

IP address. A 32-bit address assigned to devices or hosts in an IP internet that maps to a physical address. The IP address is composed of a network and host portion.

ISB. Intermediate Switch Board.

K

Kerberos. A service for authenticating users in a network environment.

kernel. The core portion of the UNIX operating system which controls the resources of the CPU and allocates them to the users. The kernel is memory-resident, is said to run in “kernel mode” and is protected from user tampering by the hardware.

L

LAN. (1) Acronym for Local Area Network, a data network located on the user's premises in which serial transmission is used for direct data communication among data stations. (2) Physical network technology that transfers data a high speed over short distances. (3) A network in which a set of devices is connected to another for communication and that can be connected to a larger network.

local host. The computer to which a user's terminal is directly connected.

log database. A persistent storage location for the logged information.

log event. The recording of an event.

log event type. A particular kind of log event that has a hierarchy associated with it.

logging. The writing of information to persistent storage for subsequent analysis by humans or programs.

M

mask. To use a pattern of characters to control retention or elimination of portions of another pattern of characters.

menu. A display of a list of available functions for selection by the user.

Motif. The graphical user interface for OSF, incorporating the X Window System. Also called OSF/Motif.

MTBF. Mean time between failure. This is a measure of reliability.

MTTR. Mean time to repair. This is a measure of serviceability.

N

naive application. An application with no knowledge of a server that fails over to another server. Client to server retry methods are used to reconnect.

network. An interconnected group of nodes, lines, and terminals. A network provides the ability to transmit data to and receive data from other systems and users.

NFS. Network File System. NFS allows different systems (UNIX or non-UNIX), different architectures, or vendors connected to the same network, to access remote files in a LAN environment as though they were local files.

NIM. Network Installation Management is provided with AIX to install AIX on the nodes.

NIM client. An AIX system installed and managed by a NIM master. NIM supports three types of clients:

- Standalone
- Diskless
- Dataless

NIM master. An AIX system that can install one or more NIM clients. An AIX system must be defined as a NIM master before defining any NIM clients on that system. A NIM master managers the configuration database containing the information for the NIM clients.

NIM object. A representation of information about the NIM environment. NIM stores this information as objects in the NIM database. The types of objects are:

- Network
- Machine
- Resource

NIS. Network Information System.

node. In a network, the point where one or more functional units interconnect transmission lines. A computer location defined in a network. The SP system can house several different types of nodes for both serial and parallel processing. These node types can include thin nodes, wide nodes, 604 high nodes, as well as other types of nodes both internal and external to the SP frame.

Node Switch Board. Switches mounted on frames that contain nodes.

NSB. Node Switch Board.

NTP. Network Time Protocol.

O

ODM. Object Data Manager. In AIX, a hierarchical object-oriented database for configuration data.

P

parallel environment. A system environment where message passing or SP resource manager services are used by the application.

Parallel Environment. A licensed IBM program used for message passing applications on the SP or RS/6000 platforms.

parallel processing. A multiprocessor architecture which allows processes to be allocated to tightly coupled multiple processors in a cooperative processing environment, allowing concurrent execution of tasks.

parameter. * (1) A variable that is given a constant value for a specified application and that may denote the application. * (2) An item in a menu for which the operator specifies a value or for which the system provides a value when the menu is interpreted. * (3) A name in a procedure that is used to refer to an argument that is passed to the procedure. * (4) A particular piece of information that a system or application program needs to process a request.

partition. See system partition.

Perl. Practical Extraction and Report Language.

perspective. The primary window for each SP Perspectives application, so called because it provides a unique view of an SP system.

pipe. A UNIX utility allowing the output of one command to be the input of another. Represented by the | symbol. It is also referred to as filtering output.

PMR. Problem Management Report.

POE. Formerly Parallel Operating Environment, now Parallel Environment for AIX.

port. (1) An end point for communication between devices, generally referring to physical connection. (2) A 16-bit number identifying a particular TCP or UDP resource within a given TCP/IP node.

predicate. Obsolete term for expression.

Primary node or machine. (1) A device that runs a workload and has a standby device ready to assume the primary workload if that primary node fails or is taken out of service. (2) A node on the SP Switch that initializes, provides diagnosis and recovery services, and performs other operations to the switch network. (3) In IBM Virtual Shared Disk function, when physical disks are connected to two nodes (twin-tailed), one node is designated as the primary node for each disk and the other is designated the secondary, or backup, node. The primary node is the server node for IBM Virtual Shared Disks defined on the physical disks under normal conditions. The secondary node can become the server node for the disks if the primary node is unavailable (off-line or down).

Problem Management Report. The number in the IBM support mechanism that represents a service incident with a customer.

process. * (1) A unique, finite course of events defined by its purpose or by its effect, achieved under defined conditions. * (2) Any operation or combination of operations on data. * (3) A function being performed or waiting to be performed. * (4) A program in operation. For example, a daemon is a system process that is always running on the system.

protocol. A set of semantic and syntactic rules that defines the behavior of functional units in achieving communication.

R

RAID. Redundant array of independent disks.

rearm expression. In Event Management, an expression used to generate an event that alternates with an original event expression in the following way: the event expression is used until it is true, then the rearm expression is used until it is true, then the event expression is used, and so on. The rearm expression is commonly the inverse of the event expression (for example, a resource variable is on or off). It can also be used with the event expression to define an upper and lower boundary for a condition of interest.

rearm predicate. Obsolete term for rearm expression

remote host. *See foreign host*.

resource. In Event Management, an entity in the system that provides a set of services. Examples of resources include hardware entities such as processors, disk drives, memory, and adapters, and software entities such as database applications, processes, and file systems. Each resource in the system has one or more attributes that define the state of the resource.

resource identifier. In Event Management, a set of elements, where each element is a name/value pair of the form `name=value`, whose values uniquely identify the copy of the resource (and by extension, the copy of the resource variable) in the system.

resource monitor. A program that supplies information about resources in the system. It can be a command, a daemon, or part of an application or subsystem that manages any type of system resource.

resource variable. In Event Management, the representation of an attribute of a resource. An example of a resource variable is `IBM.AIX.PagSp.%totalfree`, which represents the percentage of total free paging space. `IBM.AIX.PagSp` specifies the resource name and `%totalfree` specifies the resource attribute.

RISC. Reduced Instruction Set Computing (RISC), the technology for today's high performance personal computers and workstations, was invented in 1975. Uses a small simplified set of frequently used instructions for rapid execution.

rlogin (remote LOGIN). A service offered by Berkeley UNIX systems that allows authorized users of one machine to connect to other UNIX systems across a network and interact as if their terminals were connected directly. The rlogin software passes information about the user's environment (for example, terminal type) to the remote machine.

RPC. Acronym for Remote Procedure Call, a facility that a client uses to have a server execute a procedure call. This facility is composed of a library of procedures plus an XDR.

RSH. A variant of RLOGIN command that invokes a command interpreter on a remote UNIX machine and passes the command line arguments to the command interpreter, skipping the LOGIN step completely. See also *rlogin*.

S

SCSI. Small Computer System Interface.

Secondary node. In IBM Virtual Shared Disk function, when physical disks are connected to two nodes (twin-tailed), one node is designated as the primary node for each disk and the other is designated as the secondary, or backup, node. The secondary node acts as the server node for the IBM Virtual Shared disks defined on the physical disks if the primary node is unavailable (off-line or down).

server. (1) A function that provides services for users. A machine may run client and server processes at the same time. (2) A machine that provides resources to the network. It provides a network service, such as disk storage and file transfer, or a program that uses such a service. (3) A device, program, or code module on a network dedicated to providing a specific service to a network. (4) On a LAN, a data station that provides facilities to other data stations. Examples are file server, print server, and mail server.

shell. The shell is the primary user interface for the UNIX operating system. It serves as command language interpreter, programming language, and allows foreground and background processing. There are three different implementations of the shell concept: Bourne, C and Korn.

Small Computer System Interface (SCSI). An input and output bus that provides a standard interface for the attachment of various direct access storage devices (DASD) and tape drives to the RS/6000.

Small Computer Systems Interface Adapter (SCSI Adapter). An adapter that supports the attachment of various direct-access storage devices (DASD) and tape drives to the RS/6000.

SMIT. The System Management Interface Toolkit is a set of menu driven utilities for AIX that provides functions such as transaction login, shell script creation, automatic updates of object database, and so forth.

SNMP. Simple Network Management Protocol. (1) An IP network management protocol that is used to monitor attached networks and routers. (2) A TCP/IP-based protocol for exchanging network management information and outlining the structure for communications among network devices.

socket. (1) An abstraction used by Berkeley UNIX that allows an application to access TCP/IP protocol functions. (2) An IP address and port number pairing. (3) In TCP/IP, the Internet address of the host computer on which the application runs, and the port number it uses. A TCP/IP application is identified by its socket.

standby node or machine. A device that waits for a failure of a primary node in order to assume the identity of the primary node. The standby machine then runs the primary's workload until the primary is back in service.

subnet. Shortened form of subnetwork.

subnet mask. A bit template that identifies to the TCP/IP protocol code the bits of the host address that are to be used for routing for specific subnetworks.

subnetwork. Any group of nodes that have a set of common characteristics, such as the same network ID.

subsystem. A software component that is not usually associated with a user command. It is usually a daemon process. A subsystem will perform work or provide services on behalf of a user request or operating system request.

SUP. Software Update Protocol.

Sysctl. Secure System Command Execution Tool. An authenticated client/server system for running commands remotely and in parallel.

syslog. A BSD logging system used to collect and manage other subsystem's logging data.

System Administrator. The user who is responsible for setting up, modifying, and maintaining the SP system.

system partition. A group of nonoverlapping nodes on a switch chip boundary that act as a logical SP system.

T

tar. Tape ARchive, is a standard UNIX data archive utility for storing data on tape media.

Tcl. Tool Command Language.

TclX. Tool Command Language Extended.

TCP. Acronym for Transmission Control Protocol, a stream communication protocol that includes error recovery and flow control.

TCP/IP. Acronym for Transmission Control Protocol/Internet Protocol, a suite of protocols designed to allow communication between networks regardless of the technologies implemented in each network. TCP provides a reliable host-to-host protocol between hosts in packet-switched communications networks and in interconnected systems of such networks. It assumes that the underlying protocol is the Internet Protocol.

Telnet. Terminal Emulation Protocol, a TCP/IP application protocol that allows interactive access to foreign hosts.

Tk. Tcl-based Tool Kit for X Windows.

TMPCP. Tape Management Program Control Point.

token-ring. (1) Network technology that controls media access by passing a token (special packet or frame) between media-attached machines. (2) A network with a ring topology that passes tokens from one attaching device (node) to another. (3) The IBM Token-Ring LAN connection allows the RS/6000 system unit to participate in a LAN adhering to the IEEE 802.5 Token-Passing Ring standard or the ECMA standard 89 for Token-Ring, baseband LANs.

transaction. An exchange between the user and the system. Each activity the system performs for the user is considered a transaction.

transceiver (transmitter-receiver). A physical device that connects a host interface to a local area network, such as Ethernet. Ethernet transceivers contain electronics that apply signals to the cable and sense collisions.

transfer. To send data from one place and to receive the data at another place. Synonymous with move.

transmission. * The sending of data from one place for reception elsewhere.

TURBOWAYS 100 ATM Adapter. An IBM high-performance, high-function intelligent adapter that provides dedicated 100 Mbps ATM (asynchronous transfer mode) connection for high-performance servers and workstations.

U

UDP. User Datagram Protocol.

UNIX operating system. An operating system developed by Bell Laboratories that features multiprogramming in a multiuser environment. The UNIX operating system was originally developed for use on minicomputers, but has been adapted for mainframes and microcomputers. **Note:** The AIX operating system is IBM's implementation of the UNIX operating system.

user. Anyone who requires the services of a computing system.

User Datagram Protocol (UDP). (1) In TCP/IP, a packet-level protocol built directly on the Internet Protocol layer. UDP is used for application-to-application programs between TCP/IP host systems. (2) A transport protocol in the Internet suite of protocols that provides unreliable, connectionless datagram service. (3) The Internet Protocol that enables an application programmer on one machine or process to send a datagram to an application program on another machine or process.

user ID. A nonnegative integer, contained in an object of type *uid_t*, that is used to uniquely identify a system user.

V

Virtual Shared Disk, IBM. The function that allows application programs executing at different nodes of a system partition to access a raw logical volume as if it were local at each of the nodes. In actuality, the logical volume is local at only one of the nodes (the server node).

W

workstation. * (1) A configuration of input/output equipment at which an operator works. * (2) A terminal or microcomputer, usually one that is connected to a mainframe or to a network, at which a user can perform applications.

X

X Window System. A graphical user interface product.

Index

Communicating Your Comments to IBM

IBM Parallel System Support
Programs for AIX
Performance Monitoring Guide and Reference
Version 3 Release 1

Publication No. SA22-7353-00

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

If you are mailing a reader's comment form (RCF) from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:
  - FAX: (International Access Code)+1+914+432-9405
- If you prefer to send comments electronically, use this network ID:
  - IBM Mail Exchange: USIB6TC9 at IBMMAIL
  - Internet e-mail: mhvrcfs@us.ibm.com
  - World Wide Web: http://www.s390.ibm.com/os390

Make sure to include the following in your note:

- Title and publication number of this book
- Page number or topic to which your comment applies

Optionally, if you include your telephone number, we will be able to respond to your comments by phone.

Reader's Comments — We'd Like to Hear from You

IBM Parallel System Support Programs for AIX
Performance Monitoring Guide and Reference
Version 3 Release 1

Publication No. SA22-7353-00

You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Today's date: ____________________________________

What is your occupation?

Newsletter number of latest Technical Newsletter (if any) concerning this publication:

How did you use this publication?

[ ]	As an introduction	[ ]	As a text (student)
[ ]	As a reference manual	[ ]	As a text (instructor)
[ ]	For another purpose (explain)		

Is there anything you especially like or dislike about the organization, presentation, or writing in this manual? Helpful comments include general usefulness of the book; possible additions, deletions, and clarifications; specific errors and omissions.

Page Number: Comment:

Name	Address
Company or Organization	
Phone No.	

Reader's Comments — We'd Like to Hear from You
SA22-7353-00

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Department 55JA, Mail Station P384
522 South Road
Poughkeepsie NY 12601-5400

Fold and Tape **Please do not staple** Fold and Tape

SA22-7353-00

Cut or Fold
Along Line

Program Number: 5765-D51

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SA22-7353-00