Altair®

PBS Professional™

9.1

External Reference Specification

UNIX®, Linux, and Windows®

Portable Batch System™ External Reference Specification

Altair® PBS Professional™ 9.1, Updated: October 24, 2007

Edited by: Anne Urban

For more information, copies of documentation, and sales, contact Altair at:

Web:	www.altair.com, www.pbspro.com
Email:	sales@pbspro.com.

Location	Telephone	e-mail
North America	+1 248 614 2425	pbssupport@altair.com
China	+86 (0)21 5393 0011	support@altair.com.cn
France	+33 (0)1 4133 0990	francesupport@altair.com
Germany	+49 (0)7031 6208 22	hwsupport@altair.de
India	+91 80 658 8540 +91 80 658 8542	support@altair-eng.soft.net
Italy	+39 0832 315573 +39 800 905595	support@altairtorino.it
Japan	+81 3 5396 1341	pbs@altairjp.co.jp
Korea	+82 31 728 8600	support@altair.co.kr
Scandinavia	+46 (0)46 286 2050	support@altair.se
United Kingdom	+44 (0) 2476 323 600	support@uk.altair.com

Table of Contents

Acknowledgements

PBS Professional is the enhanced commercial version of the PBS software originally developed for NASA. The NASA version had a number of corporate and individual contributors over the years, for which the PBS developers and PBS community are most grateful. Below we provide formal legal acknowledgements to corporate and government entities, then special thanks to individuals.

The NASA version of PBS contained software developed by NASA Ames Research Center, Lawrence Livermore National Laboratory, and MRJ Technology Solutions. In addition, it included software developed by the NetBSD Foundation, Inc., and its contributors, as well as software developed by the University of California, Berkeley and its contributors.

Other contributors to the NASA version of PBS include Bruce Kelly and Clark Streeter of NERSC; Kent Crispin and Terry Heidelberg of LLNL; John Kochmar and Rob Pennington of *Pittsburgh Supercomputing Center*; and Dirk Grunwald of *University of Colorado, Boulder*. The ports of PBS to the Cray T3e and the IBM SP SMP were funded by *DoD USAERDC*, Major Shared Research Center; the port of PBS to the Cray SV1 was funded by DoD MSIC.

No list of acknowledgements for PBS would possibly be complete without special recognition of the first two beta test sites. Thomas Milliman of the *Space Sciences Center* of the *University of New Hampshire* was the first beta tester. Wendy Lin of *Purdue University* was the second beta tester and holds the honor of submitting more problem reports than anyone else outside of NASA.

Preface

Intended Audience

This document provides a computer programmer with the information required to write an application using the Portable Batch System (PBS) external API. PBS is a workload management system that provides a unified batch queuing and job management interface to a set of computing resources.

Related Documents

The following publications contain information that may also be useful in the management and administration of PBS.

PBS Professional Quick Start Guide: Provides a quick overview of PBS Professional installation and license key generation.

PBS Professional Administrator's Guide: provides the system administrator with information required to install, configure, and manage PBS, as well as a thorough discussion of how the various components of PBS interoperate.

PBS Professional User's Guide: Provides an overview of PBS Professional and serves as an introduction to the software, explaining how to use the user commands and graphical user interface to submit, monitor, track, delete, and manipulate jobs.

Ordering Software and Publications

To order additional copies of this manual and other PBS publications, or to purchase additional software licenses, contact your reseller or the PBS Products Department. Contact information is included on the copyright page of this document.

Document Conventions

PBS documentation uses the following typographic conventions.

abbreviation	If a PBS command can be abbreviated (such as subcommands to `qmgr`) the shortest acceptable abbreviation is underlined.
`command`	This fixed width font is used to denote literal commands, filenames, error messages, and program output.
input	Literal user input is shown in this bold, fixed-width font.
`manpage`(x)	Following UNIX tradition, manual page references include the corresponding section number in parentheses appended to the manual page name.
terms	Words or terms being defined, as well as variable names, are in italics.

Chapter 1
Introduction

This book, the **External Reference Specification** for the Portable Batch System, Professional Edition (PBS Professional) is intended provided to document the external application programming interfaces to the PBS Professional software.

1.1 Book Organization

This book is organized into seven chapters:

Chapter 1	**Introduction**: Gives an overview of this book, PBS, and the PBS team.
Chapter 2	**Concepts and Terms**: Discusses the components of PBS and how they interact.
Chapter 3	**Server Functions**: Describes, from a programmer's perspective, the various functions of the PBS Server module.
Chapter 4	**Batch Interface Library (IFL)**: Documents the main external API.
Chapter 5	**RPP Library**: Documents the Reliable Packet Protocol API.
Chapter 6	**RM Library**: Documents the Resource Monitor API.
Chapter 7	**RM Library**: Documents the Task Management API.
Chapter 8	**TCL/tk Interface**: Documents the TCL-TK wrapped PBS API.

1.2 What is PBS Pro?

PBS Professional is the professional version of the Portable Batch System (PBS), a flexible resource and workload management system, originally developed to manage aerospace computing resources at NASA. PBS has since become the leader in supercomputer workload management and the *de facto* standard on Linux clusters.

Today, growing enterprises often support hundreds of users running thousands of jobs across different types of machines in different geographical locations. In this distributed heterogeneous environment, it can be extremely difficult for administrators to collect detailed, accurate usage data or to set system-wide resource priorities. As a result, many computing resource are left under-utilized, while others are over-utilized. At the same time, users are confronted with an ever expanding array of operating systems and platforms. Each year, scientists, engineers, designers, and analysts must waste countless hours learning the nuances of different computing environments, rather than being able to focus on their core priorities. PBS Professional addresses these problems for computing-intensive enterprises such as science, engineering, finance, and entertainment.
Now you can use the power of PBS Professional to better control your computing resources. This product enables you to unlock the potential in the valuable assets you already have. By reducing dependency on system administrators and operators, you will free them to focus on other actives. PBS Professional can also help you to efficiently manage growth by tracking real usage levels across your systems and by enhancing effective utilization of future purchases.

1.2.1 History of PBS

In the past, UNIX systems were used in a completely interactive manner. Background jobs were just processes with their input disconnected from the terminal. However, as UNIX moved onto larger and larger processors, the need to be able to schedule tasks based on available resources increased in importance. The advent of networked compute servers, smaller general systems, and workstations led to the requirement of a networked batch scheduling capability. The first such UNIX-based system was the Network Queueing System (NQS) funded by NASA Ames Research Center in 1986. NQS quickly became the *de facto* standard for batch queueing.

Over time, distributed parallel systems began to emerge, and NQS was inadequate to handle the complex scheduling requirements presented by such systems. In addition, computer system managers wanted greater control over their compute resources, and users wanted a single interface to the systems. In the early 1990's NASA needed a solution to this problem, but found nothing on the market that adequately addressed their needs. So NASA led an international effort to gather requirements for a next-generation resource management system. The requirements and functional specification were later adopted as an IEEE POSIX standard (1003.2d). Next, NASA funded the development of a new resource management system compliant with the standard. Thus the Portable Batch System (PBS) was born.

PBS was quickly adopted on distributed parallel systems and replaced NQS on traditional supercomputers and server systems. Eventually the entire industry evolved toward distributed parallel systems, taking the form of both special purpose and commodity clusters. Managers of such systems found that the capabilities of PBS mapped well onto cluster computers. The PBS story continued when Veridian (the R&D contractor that developed PBS for NASA) released the Portable Batch System Professional Edition (PBS Pro), a commercial, enterprise-ready, workload management solution. Three years later, the Veridian PBS Products business unit was acquired by Altair Engineering, Inc. Altair set up the PBS Products unit as a subsidiary company named Altair Grid Technologies focused on PBS Professional and related Grid software. This unit then became part of Altair Engineering.

1.3 About the PBS Team

The PBS Professional product is being developed by the same team that originally designed PBS for NASA. In addition to the core engineering team, Altair Engineering includes individuals who have supported PBS on computers all around the world, including some of the largest supercomputers in existence. The staff includes internationally-recognized experts in resource- and job-scheduling, supercomputer optimization, message-passing programming, parallel computation, and distributed high-performance computing. In addition, the PBS team includes co-architects of the NASA Metacenter (the first full-production geographically distributed meta-computing environment), co-architects of the Department of Defense MetaQueueing (prototype Grid) Project, co-architects of the NASA Information Power Grid, and co-chair of the Global Grid Forum’s Scheduling Group.

1.4 About Altair Engineering

Through engineering, consulting and high performance computing technologies, Altair Engineering increases innovation for more than 1,500 clients around the globe. Founded in 1985, Altair's unparalleled knowledge and expertise in product development and manufacturing extend throughout North America, Europe and Asia. Altair specializes in the development of high-end, open CAE software solutions for modeling, visualization, optimization and process automation.

Chapter 2

Concepts and Terms

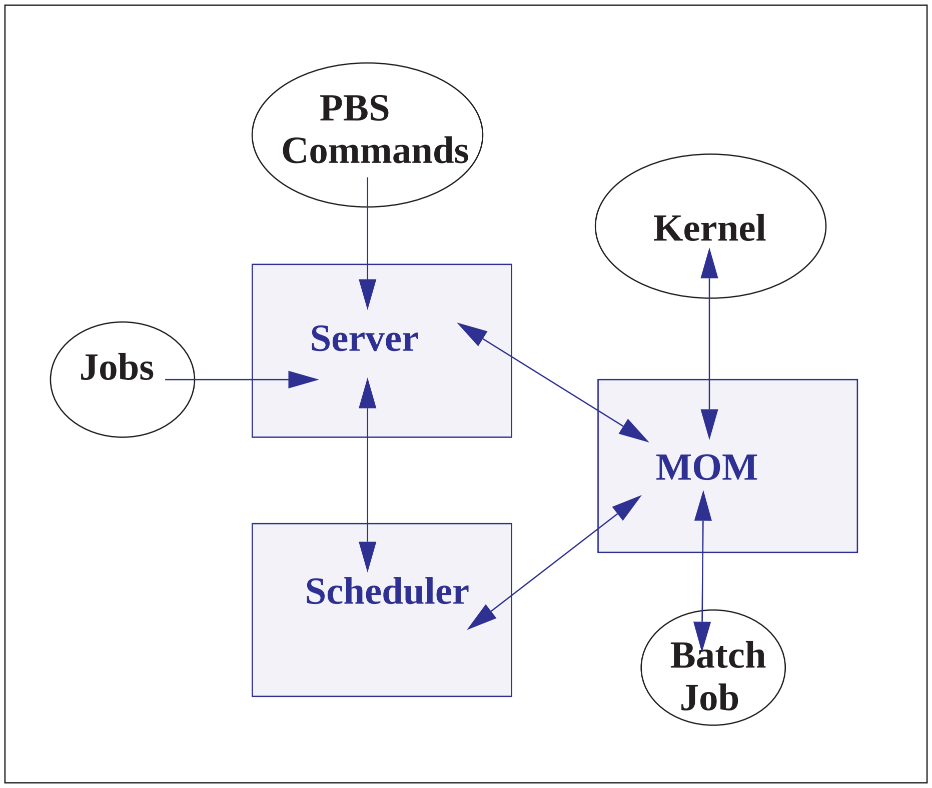
PBS is a distributed workload management system. As such, PBS handles the management and monitoring of the computational workload on a set of one or more computers. Modern workload/resource management solutions like PBS include the features of traditional batch queueing but offer greater flexibility and control than first generation batch systems (such as the original batch system NQS).

Workload management systems have three primary roles:

Queuing	The collecting together of work or tasks to be run on a computer. Users submit tasks or “jobs” to the resource management system where they are held until the system is ready to run them.
Scheduling	The process of selecting which jobs to run when and where, according to a predetermined policy. Sites balance competing needs and goals on the system(s) to maximize efficient use of resources (both computer time and people time).
Monitoring	The act of tracking and reserving system resources and enforcing usage policy. This covers both user-level and system-level monitoring as well as monitoring of the scheduling algorithms to see how well they are meeting the stated goals

2.1 PBS Components

PBS consist of two major component types: system daemons and user-level commands. A brief description of each is given here to help you make decisions during the installation process.



Job Server The *Job Server* daemon process is the central focus for PBS. Within this document, it is generally referred to as *the Server* or by the execution name *pbs_server.* All commands and daemons communicate with the Server via an *Internet Protocol* (IP) network. The Server's main function is to provide the basic batch services such as receiving/creating a batch job, modifying the job, protecting the job against system crashes, and running the job. Typically there is one Server managing a given set of resources.

Job Executor (MOM) The *Job Executor* is the daemon that actually places the job into execution. This daemon, *pbs_mom*, is informally called

MOM as it is the mother of all executing jobs. (MOM is a reverse-engineered acronym that stands for Machine Oriented Miniserver.) MOM places a job into execution when it receives a copy of the job from a Server. MOM creates a new session that is as identical to a user login session as is possible. For example, if the user's login shell is `csh`, then MOM creates a session in which `.login` is run as well as `.cshrc`. MOM also has the responsibility for returning the job's output to the user when directed to do so by the Server. One MOM daemon runs on each computer which will execute PBS jobs.

A special version of MOM, called the *Globus MOM*, is available if it is enabled during the installation of PBS. It handles submission of jobs to the Globus environment. Globus is a software infrastructure that integrates geographically distributed computational and information resources. Globus is discussed in more detail in the "Globus Support" section of the **PBS Professional User's Guide**.

Job Scheduler

The *Job Scheduler* daemon, *pbs_sched*, implements the site's policy controlling when each job is run and on which resources. The Scheduler communicates with the various MOMs to query the state of system resources and with the Server to learn about the availability of jobs to execute. The interface to the Server is through the same API as used by the client commands. Note that the Scheduler communicates with the Server with the same privilege as the PBS Manager.

Commands

PBS supplies both command line programs that are POSIX 1003.2d conforming and a graphical interface. These are used to submit, monitor, modify, and delete jobs. These *client commands* can be installed on any system type supported by PBS and do not require the local presence of any of the other components of PBS.

There are three classifications of commands: user commands (which any authorized user can use), operator commands, and manager (or administrator) commands. Operator and Manager commands require specific access privileges, as discussed in the **PBS Professional Administrator's Guide**.

Chapter 3
Server Functions

This chapter presents formal definitions for identifiers and names to be used throughout the remainder of this document, followed by detailed discussion of the various functions of the PBS Professional Server process.

3.1 General Identifiers

The following identifiers or names are referenced throughout this document. Unless otherwise noted, their usage will conform to the definition and syntax described in the following subsections and to the general rules described in the next paragraph. If allowed as part of the identifier, when entering the identifier string on the command line or in a PBS job script directive, embedded single or double quote marks must be escaped by enclosing the string in the other type of quote mark. Therefore, the string may not contain both types of quote marks. If white space is allowed in the identifier string, the string must be quoted when it is entered on the command line or in a PBS job directive.

3.1.1 Account String

An Account String is a string of characters that some Server implementations may use to provide addition accounting or charge information. The syntax is unspecified except that it must be a single string. When provided on the command line to a PBS utility or in a directive in a PBS job script, any embedded white space must be escaped by enclosing the string in quotes.

3.1.2 Attribute Name

An Attribute Name identifies an attribute or data item that is part of the information that makes up a job, queue, or Server. The name must consist of alphanumeric characters plus the underscore, '_', character. It should start with an alphanumeric character. The length is not limited. The names recognized by PBS are listed in sections 2.2, 2.3, and 2.4.

3.1.3 Destination Identifiers

A destination identifier is a string used to specify a particular destination. The identifier may be specified in one of three forms:

```
queue@server_name
queue
@server_name
```

where `queue` is an ASCII character string of up to 15 characters. Valid characters are alphanumerics, the hyphen and the underscore. The string must begin with a letter. Queue is the name of a queue at the batch Server specified by `server_name`. That Server will interpret the `queue` string. If `queue` is omitted, a null string is assumed. `server_name` is a string identifying a Server; see `server_name`, below. If `server_name` is omitted, the default Server is assumed.

3.1.4 Default Server

When a Server is not specified to a client, the client will send batch requests to the Server identified as the default Server. A client identifies the default Server by (a) the setting of the environment variable `PBS_DEFAULT` which contains a Server name, or (b) by editing the `PBS_SERVER` variable in the `/etc/pbs.conf` file on the local host. Note that if both are present, `PBS_DEFAULT` overrides the `PBS_SERVER` specification.

3.1.5 Host Name

A Host Name is a string that identifies a host or system on the network. The syntax of the string must follow the rules established by the network. For IP, a host name is of the form name.domain, where domain is a hierarchical, dot-separated List of subdomains. Therefore, a host name cannot contain a dot, "." as a legal character other than as a subdomain separator. The name must not contain the commercial at sign, "@", as this is often used to separate a file from the host in a remote file name. Also, to prevent confusion with port numbers (see section 2.7.9) a host name cannot contain a colon, ":". The maximum length of a host name supported by PBS is defined by `PBS_MAXHOSTNAME`, currently set to 64.

3.1.6 Job Identifiers

When the term job identifier is used, the identifier is specified as: `sequence_number[.server_name][@server]` The `sequence_number` is the number supplied by the Server when the job was submitted. The `server_name` component is the name of the Server which created the job. If it is missing, the name of the default Server will be assumed. `@server` specifies the current location of the job. When the term fully qualified job identifier is used, the identifier is specified as:

```
sequence_number.server[@server]
```

The `@server` suffix is not required if the job is still resides at the original Server which created the job. The qsub command will return a fully qualified job identifier.

3.1.7 Job Name

A Job Name is a string assigned by the user to provide a meaningful label to identify the job. The job name is up to and including 15 characters in length and may contain any printable characters other than white space. It must start with an alphanumeric character. If the user does not assign a name, PBS will assign a default name as described under the `-N` option of the `qsub(1)` command.

3.1.8 Resource Name

A Resource Name identifies a job resource requirement and may also identify a resource usage limit. The name must consist of alphanumeric characters plus the underscore, “_”, character. It should start with an alphanumeric character. The length is not limited. Certain resource names are identified and reserved by POSIX 1003.2d and by PBS. They are listed below in section “Types of Resources”.

3.1.9 Server Name.

Server Name is an ASCII character string of the form: `basic_server_name[:port]` The string identifies a batch Server. Basic Server names are identical to host names. The network routine `gethostbyname` will be used to translate to a network address. The network routine `getservbyname` will be used to determine the port number. An alternate port number may be specified by appending a colon, “:”, and the port number to the host name. This provides the means of specifying an alternate (test) Server on a host

3.1.10 User Name

A User Name is a string which identifies a user on the system under PBS. It is also known as the login name. PBS will accept names up to and including 16 characters. The name may contain any printable, non white space character excluding the commercial at sign, “@”. The various systems on which PBS is executing may place additional limitations on the user name.

3.2 Batch Server Functions

A batch Server provides services in one of two ways, (1) the Server provides a service at the request of a client; or (2) the Server provides a deferred service as a result of a change in conditions monitored by the Server. The Server also performs a number of internal bookkeeping functions that are described in this major section.

3.2.1 Client Service Requests

By definition, clients are processes that make requests of a batch Server. The requests may ask for an action to be performed on one or more jobs, one or more queues, or the Server itself. Those requests that cannot be successfully completed, are rejected. The reason for the rejection is returned in the reply to the client.

3.2.2 Deferred Services

The Server may, depending on conditions being monitored, defer a client service request until a later time. (Deferred services include file staging, job scheduling, etc.) Detailed discussion of the deferred services provided by the Server is given in section 3.7 “Deferred Services” on page 24 below.

3.3 Server Management

The following sections describe the services provided by a batch Server in response to a request from a client. The requests are grouped in the following subsections by the type of object affected by the request: Server, queue, job, or resource. The batch requests described in this section control the functioning of the batch Server. The control is either direct as in the Shut Down request, or indirect as when Server attributes are modified. The following table provides the numeric value of each of the batch request codes.

.

0	PBS_BATCH_Connect	24	PBS_BATCH_Rescq
1	PBS_BATCH_QueueJob	25	PBS_BATCH_ReserveResc
2	UNUSED	26	PBS_BATCH_ReleaseResc
3	PBS_BATCH_jobscript	27	PBS_BATCH_FailOver
4	PBS_BATCH_RdytoCommit	48	PBS_BATCH_StageIn
5	PBS_BATCH_Commit	49	PBS_BATCH_AuthenUser
6	PBS_BATCH_DeleteJob	50	PBS_BATCH_OrderJob
7	PBS_BATCH_HoldJob	51	PBS_BATCH_SelStat
8	PBS_BATCH_LocateJob	52	PBS_BATCH_RegistDep
9	PBS_BATCH_Manager	54	PBS_BATCH_CopyFiles
10	PBS_BATCH_MessJob	55	PBS_BATCH_DelFiles
11	PBS_BATCH_ModifyJob	56	PBS_BATCH_JobObit
12	PBS_BATCH_MoveJob	57	PBS_BATCH_MvJobFile
13	PBS_BATCH_ReleaseJob	58	PBS_BATCH_StatusNode
14	PBS_BATCH_Rerun	59	PBS_BATCH_Disconnect
15	PBS_BATCH_RunJob	60-61	UNUSED
16	PBS_BATCH_SelectJobs	62	PBS_BATCH_JobCred
17	PBS_BATCH_Shutdown	63	PBS_BATCH_CopyFiles_Cred
18	PBS_BATCH_SignalJob	64	PBS_BATCH_DelFiles_Cred
19	PBS_BATCH_StatusJob	65	PBS_BATCH_GSS_Context
20	PBS_BATCH_StatusQue	66-69	UNUSED
21	PBS_BATCH_StatusSvr	70	PBS_BATCH_SubmitResv
22	PBS_BATCH_TrackJob	71	PBS_BATCH_StatusResv
23	PBS_BATCH_AsyrunJob	72	PBS_BATCH_DeleteResv

3.3.1 Manage Request

The Manage request supports the `qmgr(8)` command and several of the operator commands. The command directs the Server to create, alter, or delete an object managed by the Server or one of its attributes. For more information, see the `qmgr` command.

3.3.2 Server Status Request

The status of the Server may be requested with a Server Status request. The batch Server will reject the request if the user of the client is not authorized to query the status of the Server. If the request is accepted, the Server will return a Server Status Reply. See the `qstat` command and the Data Exchange Format description for details of which Server attributes are returned to the client.

3.3.3 Start Up

A batch request to start a Server cannot be sent to a Server since the Server is not running. Therefore a batch Server must be started by a process local to the host on which the Server is to run. The Server is started by a `pbs_server` command. The Server recovers the state of managed objects, such as queues and jobs, from the information last recorded by the Server. The.treatment of jobs which were in the running state when the Server previously shut down is dictated by the start up mode, see the description of the `pbs_server(8)` command.

3.3.4 Shut Down

The batch Server is "shut down" when it no longer responds to requests from clients and does not perform deferred services. The batch Server is requested to shut down by sending it a Server Shutdown request. The Server will reject the request from a client not authorized to shut down the Server. When the Server accepts a shut down request, it will terminate in the manner described under the `qterm` command. When shutting down, the Server must record the state of all managed objects (jobs, queues, etc.) in non-volatile memory. Jobs which were running will be marked in the secondary state field for possible special treatment when the Server is restarted. If checkpoint is supported, any job running at the time of the shut down request whose Checkpoint attribute is not n, will be checkpointed. This includes jobs whose Checkpoint attribute value is “unspecified”, a value of u. If the Server receives either a `SIGTERM` or a `SIGSHUTDN` signal, the Server will act as if it had received a shut down immediate request.

3.4 Queue Management

The following client requests effect one or more queues managed by the Server. These requests require a privilege level generally assigned to operators and administrators.

3.4.1 Queue Status Request

The status of a queue at the Server may be requested with a Queue Status request. The batch Server will reject the request if any of the following conditions are true:

- The user of the client is not authorized to query the status of the designated queue.
- The designated queue does not exist on the Server.

If the request does not specify a queue, status of all the queues at the Server will be returned. When the request is accepted, the Server will return a Queue Status Reply. See the `qstat` command and the Data Exchange Format description for details of which queue attributes are returned to the client.

3.5 Job Management

The following client requests effect one or more jobs managed by the Server. These requests do not require any special privilege except when the job for which the request is issued is not owned by the user making the request.

3.5.1 Queue Job Request

A Queue Job request is a complex request consisting of several subrequests: Initiate Job Transfer, Job Data, Job Script, and Commit. The end result of a successful Queue Job request is an additional job being managed by the Server. The job may have been created by the request or it may have been moved from another Server. The job resides in a queue managed by the Server. When a queue is not specified in the request, the job is placed in a queue selected by the Server. This queue is known as the default queue. The default queue is an attribute of the Server that is settable by the administrator. The queue, whether specified or defaulted, is called the target queue. The batch Server will reject a Queue Job Request if any of the following conditions are true:

- The client is not authorized to create a job in the target queue.
- The target queue does not exist at the Server.
- The target queue is not enabled.
- The target queue is an execution queue and a resource requirement of the job exceeds the limits set upon the queue.
- The target queue is an execution queue and an unrecognized resource is requested by the job.
- The job requires access to a user identifier that the client is not authorized to access.

When a job is placed in a execution queue, it is placed in the queued state unless one of the following conditions applies:

- The job has an `execution_time` attribute that specifies a time in the future and the `Hold_Types` attribute has value of {NONE}; in which case the job is placed in the waiting state.
- The job has a `Hold_Types` attribute with a value other than {NONE}, wherein the job is placed in the held state.

When a job is placed in a routing queue, its state may change based on the conditions described in section 3.7.4 “Job Routing” on page 26.

A Server that accepts a Queue Job Request for a new job will: (1) add the `PBS_O_QUEUE` variable to the `Variable_List` attribute of the job and set the value to the name of the target queue; (2) add the `PBS_JOBID` variable to the `Variable_List` attribute of the job and set the value to the job identifier assigned to the job; (3) add the `PBS_JOBNAME` variable to the `Variable_List` attribute of the job and set the value to the value of the `Job_Name` attribute of the job. When the Server accepts a Queue Job request for an existing job, the Server will send a Track Job request to the Server which created the job.

3.5.2 Job Credential Request

The Job Credential sub-request is part of the Queue Job complex request. This sub-request transfers a copy of the credential provided by the authentication facility explained below.

3.5.3 Job Script Request

The Job Script sub-request is part of the Queue Job complex request. This sub-request passes a block of the job script file to the receiving Server. The script is broken into 8 kilobyte blocks to prevent having to hold the entire script in memory. One or more Job Script sub-requests may be required to transfer the script file.

3.5.4 Commit Request

The Commit sub-request is part of the Queue Job request. The Commit notifies the receiving Server that all parts of the job have been transferred and the receiving Server should now assume ownership of the job. Prior to sending the Commit, the sending client, command or another Server, is the owner.

3.5.5 Message Job Request

A batch Server can be requested to write a string of characters to one or both output streams of an executing job. This request is primarily used by an operator to record a message for the user. The batch Server will reject a Message Job request if any of the following conditions are true:

- The designated job is not in the running state.
- The user of the client is not authorized to post a message to the designated job.
- The designated job is not owned by the Server.

When the Server accepts the Message Job request, it will forward the request to the primary MOM daemon for the job. (Upon receipt of the Message Job request from the Server, the MOM will append the message string, followed by a new line character, to the file or files indicated. If no file is indicated, the message will be written to the standard error of the job.)

3.5.6 Locate Job Request

A client may ask a Server to respond with the location of a job that was created or is owned by the Server. When the Server accepts the Locate Job request, it returns a Locate Reply. The request will be rejected if any of the following conditions are true:

- The Server does not own (manage) the job, and
- The Server did not create the job.
- The Server is not maintaining a record of the current location of the job.

3.5.7 Delete Job Request

A Delete Job request asks a Server to remove a job from the queue in which it exists and not place it elsewhere. The batch Server will reject a Delete Job Request if any of the following conditions are true:

- The user of the client is not authorized to delete the designated job.
- The designated job is not owned by the Server.
- The designated job is not in an eligible state. Eligible states are queued, held, waiting, running, and transiting.

If the job is in the running state, the Server will forward the Delete Job request to the primary MOM daemon responsible for the job. (Upon receipt, the MOM daemon will first send a SIGTERM signal to the job process group. After a delay specified by the delete request or if not specified, the `kill_delay` queue attribute, the MOM will send a SIGKILL signal to the job process group. The job is then placed into the exiting state.) Option arguments exist to specify the “delay” time (seconds) between the `SIGTERM` and `SIGKILL` signals, as well as to “force” the deletion of the job even if the node on it is running is not responding.

3.5.8 Modify Job Request

A batch client makes a Modify Job request to the Server to alter the attributes of a job. The batch Server will reject a Modify Job Request if any of the following conditions are true:

- The user of the client is not authorized to make the requested modification to the job.
- The designated job is not owned by the Server.
- The requested modification is inconsistent with the state of the job.
- A requested resource change would exceed the limits of the queue or Server.
- An unrecognized resource is requested for a job in an execution queue.

When the batch Server accepts a Modify Job Request, it will modify all the specified attributes of the job. When the batch Server rejects a Modify Job Request, it will modify none of the attributes of the job.

3.5.9 Run Job

The "Run Job" request directs the Server to place the specified job into immediate execution. The request is issued by a `qrun` operator command and by the PBS Job Scheduler.

3.5.9.1 Rerun Job Request

To rerun a job is to kill the members of the session (process) group of the job and leave the job in the execution queue. If the `Hold_Types` attribute is not {NONE }, the job is eligible to be re-scheduled for execution. The Server will reject the Rerun Job request if any of the following conditions are true:

- The user of the client is not authorized to rerun the designated job.
- The Rerunnable attribute of the job has the value {FALSE}.
- The job is not in the running state.
- The Server does not own the job.

When the Server accepts the Rerun Job request, the request will be forwarded to the primary MOM responsible for the job, who will then perform the following actions:

- Send a SIGKILL signal to the session (process) group of the job.
  Send an OBIT notice to the Server with resource usage information
- The Server will then requeue the job in the execution queue in which it was executing.

If the `Hold_Types` attribute is not {NONE}, the job will be placed in the held state. If the `execution_time` attribute is a future time, the job will be placed in the waiting state. Otherwise, the job is.placed in the queued state.

3.5.10 Hold Job Request

A client can request that one or more holds be applied to a job. The batch Server will reject a Hold Job request if any of the following conditions are true:

- The user of the client is not authorized to add any of the specified holds.
- The batch Server does not manage the specified job.

When the Server accepts the Hold Job Request, it will add each type of hold listed which is not already present to the value of the `Hold_Types` attribute of the job. If the job is in the queued or waiting state, it is placed in the held state. If the job is in running state, then the following additional actions are taken: If check-point / restart is supported by the host system, placing a hold on a running job will cause the job (1) to be checkpointed, (2) the resources assigned to the job will be released, and (3) the job is placed in the held state in the execution queue. If checkpoint / restart is not supported, the Server will only set the requested hold attribute. This will have no effect unless the job is rerun or restarted.

3.5.11 Release Job Request

A client can request that one or more holds be removed from a job. A batch Server rejects a Release Job request if any of the following conditions are true:

- The user of the client is not authorized to add (remove) any of the specified holds.
- The batch Server does not manage the specified job.

When the Server accepts the Release Job Request, it will remove each type of hold listed from the value of the `Hold_Types` attribute of the job. Normally, the job will then be placed in the queued state, unless another hold type is remaining on the job. However, if the job is in the held state and all holds have been removed, the job is placed in the waiting state if the `Execution_Time` attribute specifies a time in the future.

3.5.12 Move Job Request

A client can request a Server to move a job to a new destination. The batch Server will reject a Move Job Request if any of the following conditions are true:

- The user of the client is not authorized to remove the designated job from the queue in which the job resides.
- The user of the client is not authorized to submit a job to the new destination.
- The designated job is not owned by the Server.
- The designated job is not in the queued, held, or waiting state.
- The new destination is disabled.
- The new destination is inaccessible. When the Server accepts a Move Job request, it will
    - Queue the designated job at the new destination.
    - Remove the job from the current queue.

If the destination exists at a different Server, the current Server will transfer the job to the new Server by sending a Queue Job request sequence to the target Server. The Server will insure that a job is neither lost nor duplicated.

3.5.13 Select Jobs Request

A client is able to request from the Server a list of jobs owned by that Server that match a list of selection criteria. The request is a Select Jobs request. All the jobs owned by the Server and which the user is authorized to query are initially eligible for selection. Job attributes and resources relationships listed in the request restrict the selection of jobs. Only jobs which have attributes and resources that meet the specified relations will be selected. The Server will reject the request if the queue portion of a specified destination does not exist on the Server. When the request is accepted, the Server will return a Select Reply containing a list of zero or more jobs that met the selection criteria.

3.5.14 Signal Job Request

A batch client is able to request that the Server signal the session (process) group of a job. Such a request is called a Signal Job request. The batch Server will reject a Signal Job Request if any of the following conditions are true:

- The user of the client is not authorized to signal the job.
- The job is not in the running state, except for the special signal "resume" when the job must be in the Suspended state.
- The Server does not own the designated job.
- The requested signal is not supported by the host operating system. (The kill system call returns [EINVAL].)

When the Server accepts a request to signal a job, it will forward the request to the primary MOM daemon responsible for the job, who will then send the signal requested by the client to the all processes in the job's session.

3.5.15 Status Job Request

The status of a job or set of jobs at a destination may be requested with a Status Job request. The batch Server will reject a Status Job Request if any of the following conditions are true:

- The user of the client is not authorized to query the status of the designated job.
- The designated job is not owned by the Server.

When the Server accepts the request, it will return a Job Status Message to the client. See the qstat command and the Data Exchange Format description for details of which job attributes are returned to the client. If the request specifies a job identifier, status will be returned only for that job. If the request specifies a destination identifier, status will be returned for all jobs residing within the specified queue that the user is authorized to query.

3.6 Server to Server Requests.

Server to Server requests are a special category of client requests. They are only issued to a Server by another Server.

3.6.1 Track Job Request

A client that wishes to request an action be performed on a job must send a batch request to the Server that currently manages the job. As jobs are routed or moved through the batch network, finding the location of the job can be difficult without a tracking service. The Track Job request forms the basis for this service. A Server that queues a job sends a track job request to the Server which created the job. Additional backup location Servers may be defined. A Server that receives a track job request records the information contained therein. This information is made available in response to a Locate Job request.

3.6.2 Synchronize Job Starts

PBS provides for synchronizing the initiation of separate jobs. This is done to support distributing processing. Job start synchronization is requested through a special dependency attribute. The first job in the set, the “master”, specifies the dependency attribute as:

```
-W synccount=count
```

where count is an integer which is the number of other jobs to be synchronized with this job. This job is the master only in the sense that it defines the rendezvous point for the semaphore messages and that it must be submitted first so the identifier is known for the other jobs in the set. The other jobs in the sync set specify the dependency attribute as:

```
-W syncwith=job_identifier
```

where `job_identifier` is the job identifier assigned to the job which contained the sync-count resource, the master job. When the Server queues a job in an execution queue and the job is a member of a sync set, including the “master”, the Server places a system hold on the job. The secondary state is set to indicate the system hold is for sync. The Server managing the non master jobs will register the job with the Server managing the master by sending a Register Dependent request with a "Register" operation. When all jobs have registered, as determined by the count on the master, the Server managing the master job will send a Register Dependent request, with a "Release" operation, request to each job in turn in the set to remove the system hold. The released jobs may now vie for resources. The jobs are released in order of the “cheapest” resources first; the concept of “Resource Costs” will be explained shortly. When the resources required by a released job are available, as determined by the Scheduler, A run Job Request will be issued for that job. The Server which manages the job will send a Register Dependent request with a “Ready” operation to the Server that owns the master job. This request indicates that the dependent job is ready and the job with the next cheapest resources can be released.

If the master of a sync set is aborted before all jobs in the set begin execution, an Abort Job request is sent to all jobs in the set. This is done because the synchronous feature is intended for a set jobs which need communication amount themselves during execution. If the master is gone, (1) the rendezvous point for Server messages is lost, and (2) the job set is unlikely to be able to establish the inter job communications required.

3.6.3 Job Dependency

PBS provides support for job dependency. A job, the “child”, can be declared to be dependent on one or more jobs, the “parents”. A parent may have any number of children. The dependency is specified as an attribute on the `qsub` command with the `-W` option The general specification is of the form:

```
    -W type=argument[,type=argument,...]
```

See the `qalter`(1B) or `qsub`(1B) man pages for the complete specification of the dependency list, and the **PBS Professional User’s Guide** for detailed discussion of use.

When a Server queues a job with a dependency type of `syncwith`, `after`, `afterok`, `after notok`, or `after-any` in an execution queue, the Server will send a Register Dependent Job request to the Server managing the job specified by the associated `job_identifier`. The request will specify that the Server is to register the dependency. This actually creates a corresponding `before` type dependency attribute entry on the parent (e.g. run job X *before* job Y). If the request is rejected because the parent job does not exist, the child job is aborted. If the request is accepted, a system hold is placed on the child job. When a parent job, with any of the `before...` types of dependency, reaches the required state, started or terminated, the Server executing the parent job sends a Register Dependent Job request to the Server managing the child job directing it to release the child job. If there are no other dependencies on other jobs, the system hold on the child job is removed. When a child job is submitted with an on dependency and the parent is submitted with any of the `before...` types of dependencies, the parent will register with the child. This causes the `on` dependency count to be reduced and a corresponding `after...` dependency to be created for the child job. The result is a pairing between corresponding `before...` and `after...` dependency types. If the parent job terminates in a manner that the child is not released, it is up to the user to correct the situation by either deleting the child job or by correcting the problem with the parent job and resubmitting it. If the parent job is resubmitted, it must have a dependency type of `before`, `beforeok`, `beforenotok`, or `beforeany` specified to connect it to the waiting child job.

3.7 Deferred Services

This section describes the deferred services performed by batch Servers: file staging, job selection, job initiation, job routing, job exit, job abort, and the rerunning of jobs after a restart of the Server. The following rules apply to deferred services on behalf of jobs:

- If the Server cannot complete a deferred service for a reason which is permanent, then the job is aborted.
- If the service cannot be completed at the current time but may be later, the service is retried a finite number of times.

3.7.1 Job Scheduling

If the Server attribute scheduling is set true, the Server will immediately request a scheduling cycle of the PBS Job Scheduler. While it remains true, the Scheduler will be cycled when any of four events occur:

- Enqueuing of a job in an execution queue or the change of state of a job in an execution queue to Queued from Waiting or Held.
- Termination of a running job. The termination may be normal execution completion, or because the job was deleted by request.
- Elapse of a specified cycle time as established by the administrator.
- The completion of a scheduling cycle in which one and only one job was scheduled for execution. This provides for the implementation of scheduling scripts that must see the impact of the new job on system resources before picking a second job.

While a request for a scheduling cycle is outstanding, the connection to the Scheduler is open, the Server will not make another request of the Scheduler. If the Server attribute scheduling is set false, the Server will not contact the scheduler. This condition is indicated by the `server_state` attribute as Idle.

3.7.2 File Staging

Two types of file staging services exist, in-staging before execution and out-staging after execution. These services are requested by an attribute (via the `-W` option) which specifies the files to be staged:

```
-Wstagein=local_file@host:remote_path[,local_file@host:remote_path,...]

-Wstage-
out=local_file@host:remote_path[,local_file@host:remote_path,...]
```

A request to stage in a file directs the Server to direct MOM to copy a file from a remote host to the local host. The user must have authority to access the file under the same user name under which the job will be run. The remote file is not modified or destroyed. The file will be available before the job is initiated. If a file cannot be staged in for any reason, any files which were staged-in are deleted and the job is placed into wait state and mail is sent to the job owner.

A request to stage out a file directs the Server to direct MOM to move a file from the local host to a remote host. This service is performed after the job has completed execution and regardless of its exit status. If a file cannot be moved, mail is sent to the job owner. If a file is successfully staged out, the local file is deleted. A version of the BSD 4.4-Lite system utility, `rcp`(1), will be used to move files over the network. This version of `rcp` has been modified to always return a non-zero exit status on any failure.

3.7.3 Job Initiation

Job initiation is to place a job into execution. The Server may receive a Run Job request from the `qrun` command, or the PBS Job Scheduler. If the request is authenticated, then the Server forwards the Run Job request to the appropriate MOM (as either specified in the Run Job request, or as selected by the Server itself if unspecified).

The receiving MOM daemon will then create a session leader that runs the shell program indicated by the `Shell_Path_List` attribute of the job. The pathname of the script and any script arguments are passed as parameters to the shell. If the path name of the shell is a relative name, the MOM will search its execution path, $PATH, for the shell. If the path name of the shell is omitted or is the null string, the MOM uses the login shell for the user under whose name the job is to be run. The MOM will determine the user name under which the job is to be run by the following rules:

1. Select the user identifier from the `User_List` job attribute which has a host name that matches the execution host.

2. Select the user identifier from the `User_List` job attribute which has no associated host name.

3. Use the user name from the `job_owner` attribute of the job.

The MOM will create, in the environment of the session leader of the job, the environment variables named: `PBS_ENVIRONMENT`, the value of which is the string “PBS_BATCH”. `PBS_QUEUE` has the value of the name of the execution queue. The MOM will also place in the environment of the session leader of the job, all of the variables and their corresponding values found in the variables attribute of the job. The MOM will place the required limits on the resources for which the host system supports resource limits. If the job had been run before and is now being rerun, the MOM will insure that the standard output and standard error streams of the job are appended to the prior streams, if any. If the MOM and host system support accounting, the MOM will use the value of the `Account_Name` job attribute as required by the host system. If the MOM and host system support checkpoint, the MOM will set up checkpointing of the job according to the value of the Checkpoint job attribute. If checkpoint is supported and the Checkpoint attribute requests checkpointing at the minimum interval or a interval less than the minimum interval for the queue, then checkpoint will be set for an interval given by the queue attribute `minimum_interval`. The MOM will set up the standard output stream and the standard error stream of the job according to the following rules:

- The stream will be located in a temporary file in the MOM’s `spool` directory.

- If the job attribute `Join_Path` has the value `eo` or the value `oe`, the MOM connects the standard error stream of the job to the same file as the standard output stream.

3.7.4 Job Routing

Job routing is moving a job from a routing queue to one of the destinations associated with the queue. If the `started` queue attribute is {TRUE}, the Server will route all eligible jobs which reside in the queue. All jobs in the queued state are eligible. If the queue attribute `route_held_jobs` is {TRUE}, jobs in the held state are eligible for routing. If the queue attribute `route_waiting_jobs` is {TRUE}, jobs in the waiting state are eligible. The Server will execute the function specified by the queue attribute `route_function` to select a destination for the job. Possible destinations are listed in the queue attribute `route_destinations`. If the destination to which the job is to be routed is at another Server, the current Server will use a Queue Job request sequence to move the job to the new destination. If the Server is unable to route a job to a chosen destination, the Server will select another destination from the list and retry the route. If the Server is unable to route a job to any destination because of a temporary condition, such as being unable to connect with the Server at the destination, the Server will retry the route after a delay specified by the queue attribute `route_retry_time`. The Server will proceed to route other jobs in the queue. The Server will retry the route up to the (queue attribute) `number_retries` times. If the Server is unable to route a job to any destination and all failures are permanent (non-temporary), the Server will abort the job.

3.7.5 Job Exit

When the session leader of a batch job exits, the MOM will perform the following actions in the order listed.

- Place the job in the exiting state.
- “Free” the resources allocated to the job. The actual releasing of resources assigned to the processes of the job is performed by the kernel. PBS will free the resources which it “reserved” for the job by decrementing the `resources_used` generic data item for the queue and Server.
- Return the standard output and standard error streams of the job to the user. If the `Keep_Files` attribute of the job contains {`KEEP_OUTPUT`}, the Server copies the spooled file holding the standard output steam of the job to the home directory of the user under whose name the job executed. The file name for the output is `job_name.oseq_number`. See the `qsub`(1B) command description. If the `Keep_Files` attribute of the job contains {`KEEP_ERROR`} and the `Join_Path` attribute does not contain ’e’, the Server copies the spooled file holding the standard error stream of the job to the home directory of the user under whose name the job executed. The file name for the

error file is `job_name.eseq_number`.

If the files are not to be kept on the execution host as described above, the temporary file holding the standard output is copied or renamed to the host and path name specified by the job attribute `Output_Path`. If the path name is relative, the file will be located relative to home directory of the user on the receiving host.

- If the `Join_Path` attribute does not contain the value `e`, the standard error of the job is delivered according to the same rules as the standard output described above. If either output file cannot be copied to its specified destination, the Server will send mail to the job owner specifying the current location of the output.

- If the `Mail_Points` job attribute contains the value {EXIT}, the Server will send mail to the users listed in the job attribute `Mail_List`.

- If out staging of files is supported, the files listed in the outfile resource will be copied to the specified destination.

- The job will be removed from the execution queue.

3.7.6 Job Aborts

If the Server aborts a job and the `Mail_Points` job attribute contains the value {ABORT}, the Server will send mail to the users listed in the job attribute `Mail_List`. The mail message will contain the reason the job was aborted. In addition, the `stdout` and `stderr` files specified for the job, if they exist, will be copied back to the specified location.

3.7.7 Timed Events

The Server performs certain events at a specified time or after a specified time delay. A job may have an `execution_time` attribute set to a time in the future. When that time is reached, the job state is updated. If the Server is unable to make connection with another Server, it is to retry after a time specified by the routing queue attribute `route_retry_time`.

3.7.8 Event Logging

The PBS Server maintains an event logfile, the format and contents of which are documented in the **PBS Professional Administrator's Guide**.

3.7.9 Accounting.

The PBS Server maintains an accounting file, the format and contents of which are documented in the **PBS Professional Administrator's Guide**.

3.8 Resource Management

PBS performs resource allocation at job initiation in two ways depending on the support provided by the host system. Resources are either reservable or non reservable.

3.8.1 Resource Limits

When submitting a job, a user may specify the hard limit of usage for resources known to the system on which the job will run. If the executing job usage of resources exceed the specified limit, the job is aborted. If the user does not specify a limit for a resource type, the limit may be set to a default established by the PBS administrator. The default limit is taken from the first of the following attributes which is set:

1. The current queue's attribute `resources_default`.
2. The Server's attribute `resources_default`.
3. The current queue's attribute `resources_max`.
4. The Server's attribute `resources_max`.

If the user does not specify a limit for a resource and a default is not established via one of the above attributes, the usage of the resource is unlimited.

3.8.2 Resource Names

For additional information, see the **PBS Professional User's Guide** where all resource names are documented.

3.9 Network Protocol

The PBS system fits into a client - Server model, with a batch client making a request of a batch Server and the Server replying. This client - Server communication necessitates an interprocess communication method and a data exchange (data encoding) format. Since the client and Server may reside on different systems, the interprocess communication must be supportable over a network.

While the basic PBS system fits nicely into the client - Server model, it also has aspects of a transaction system. When jobs are being moved between Servers, it is critical that the jobs are not lost or replicated. Updates to a batch job must be applied once and only once. Thus the operation must be atomic. Most of the client to Server requests consist of a single message. Treating these requests as an atomic operation is simple. One request, "Queue Job", is more complex and involves several messages, or subrequests, between the client and the Server. Any of these subrequests might be rejected by the Server. It is important that either side of the connection be able to abort the request (transaction) without losing or replicating the job. The network connection also might be lost during the request. Recovery from a partially transmitted request sequence is critical. The sequence of recovery from lost connections is discussed in the Queue Job Request description.

The batch system data exchange protocol must be built on top of a reliable stream connection protocol. PBS uses TCP/IP and the socket interface to the network. Either the Simple Network Interface, SNI, or the Detailed Network Interface, DNI, as specified by POSIX.12, Protocol Independent Interfaces, could be used as a replacement.

3.9.1 General DIS Data Encoding

The purpose of the “Data is Strings” encoding is to provide a simple, fast, small, machine independent form for encoding data to a character string and back again. Because data can be decoded directly into the final internal data structures, the number of data copy operations are reduced. Data items are represented as people think of them, but preceded with a count of the length of each data item. For small positive integers, it is impossible to tell from the encoded data whether they came from `signed` or `unsigned chars`, `shorts`, `ints`, or `longs`. Similarly, for small negative numbers, the only thing that can be determined from the encoded data is that the source datum was not unsigned. It is impossible to tell the word size of the encoding machine, or whether it uses 2’s complement, one’s complement or sign - magnitude representation, or.even if it uses binary arithmetic. All of the basic C data types are handled. Signed and unsigned chars, shorts, ints, longs produce integers. NULL terminated and counted strings produce counted strings (with the terminating NULL removed). Floats, doubles, and long doubles produce real numbers. Complex data must be built up from the basic types. Note that there is no type tagging, so the type and sequence of data to be decoded must be known in advance.

Chapter 4

Batch Interface Library (IFL)

The primary external application programming interface to PBS is the Batch Interface Library, or IFL. This library provides a means of building new batch clients. Any batch service request can be invoked through calls to the batch interface library. Users may wish to build a job which could status itself or spawn off new jobs. Or they may wish to customize the job status display rather than use qstat. Administrators may use the interface library to build new control commands.

4.1 Interface Library Overview

The IFL provides a user-callable function corresponding to each batch client command. There is (approximately) a one to one correlation between commands and batch service requests. Additional routines are provided for network connection management. The user callable routines are declared in the header file `PBS_ifl.h`. Users open a connection with a batch Server via a call to `pbs_connect`(). Multiple connections are supported. Before a connection is established, `pbs_connect`() will fork and exec an `pbs_iff` process, as shown in figure 4-1 below. The purpose of `pbs_iff` is to provide the user a credential which validates the user's identity. This credential is included in each batch request. The provided credential prevents a user from spoofing another user's identity.

The credential that is sent to the server consists of: a) user's name from the password file based on running pbs_iff's "real uid" value, and b) unprivileged, client-side port value associated with the original pbs_connect request message to the server. The server looks

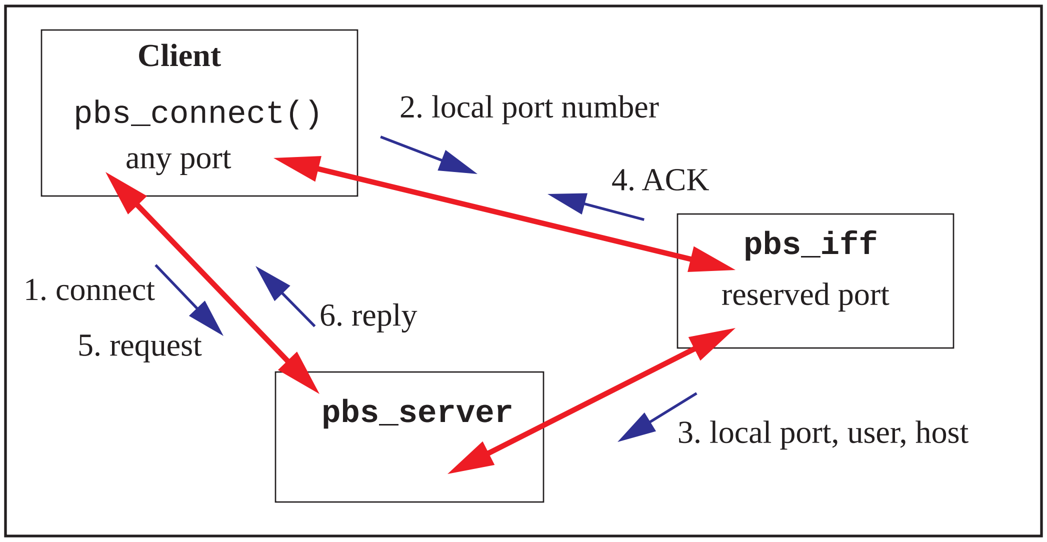


Figure 4-1: Interface Between Client, IFF, and Server

at the entries in its connection table to try and find the entry having these two pieces of information, and which is not yet marked authenticated. To be believed, this information must be gotten from a connection having a privileged, remote-end, port value.

After all requests have been made to a Server, its connection is closed via a call to `pbs_disconnect`().

Users request service of a batch Server by calling the appropriate library routine and passing it the required parameters. The parameters correspond to the options and operands on the commands. It is the user’s responsibility to ensure the parameters have correct syntax. Each function will return zero upon success and a non-zero error code on failure. These error codes are available in the header file `PBS_error.h`. The library routine will accept the parameters and build the corresponding batch request, then pass it to the Server.

To use pbs_connect with Windows, initialize the network library and link with winsock2. Call winsock_init() before calling pbs_connect(), and link against the ws2_32.lib library.

4.2 Interface Library Routines

The following manual pages describe the user-callable functions in the IFL. These functions are found in the files `src/lib/Libifl/pbsD_*.c`

NAME

pbs_alterjob - alter pbs batch job

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

int pbs_alterjob(int connect, char *job_id, struct attrl *attrib,
char *extend)
```

DESCRIPTION

Issue a batch request to alter a batch job.

A Modify Job batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

The argument, job_id, identifies which job is to be altered, it is specified in the form:

sequence_number.server

The parameter, attrib, is a pointer to an attrl structure which is defined in pbs_ifl.h as:

```
struct attrl {
    char *name;
    char *resource;
    char *value
    struct attrl *next;
};
```

The attrib list is terminated by the first entry where next is a null pointer.

The name member points to a string which is the name of the attribute. The value member points to a string which is the value of the attribute. The attribute names are defined in pbs_ifl.h:

```
#define ATTR_a “Execution_Time”
```

Alter the job’s execution time.

#define ATTR_A “Account_Name”
Alter the account string.

#define ATTR_c “Checkpoint”
Alter the checkpoint interval.

#define ATTR_e “Error_Path”
Alter the path name for the standard error of the job.

#define ATTR_g “Group_List”
Alter the list of group names under which the job may execute.

#define ATTR_h “Hold_Types”
Alter the hold types.

#define ATTR_j “Join_Path”
Alter if standard error and standard output are joined (merged).

#define ATTR_k “Keep_Files”
Alter which output of the job is kept on the execution host.

#define ATTR_l “Resource_List”
Alter the value of a named resource.

#define ATTR_m “Mail_Points”
Alter the points at which the server will send mail about the job.

#define ATTR_M “Mail_Users”
Alter the list of users who would receive mail about the job.

#define ATTR_N “Job_Name”
Alter the job name.

#define ATTR_o “Output_Path”
Alter the path name for the standard output of the job.

#define ATTR_p “Priority”
Alter the priority of the job.

#define ATTR_r “Rerunable”
Alter the rerunable flag.

#define ATTR_S “Shell_Path_List”
Alter the path to the shell which will interprets the job script.

#define ATTR_u “User_List”
Alter the list of user names under which the job may execute.

#define ATTR_v “Variable_List”
Alter the list of environmental variables which are to be exported to the job.

#define ATTR_depend “depend”
Alter the inter-job dependencies.

#define ATTR_stagein “stagein”
Alter the list of files to be staged-in before job execution.

#define ATTR_stageout “stageout”
Alter the list of files to be staged-out after job execution.

If attrib itself is a null pointer, then no attributes are altered.

Associated with an attribute of type ATTR_l (the letter ell) is a resource name indicated by resource in the attrl structure. All other attribute types should have a pointer to a null string (““) for resource.

If the resource of the specified resource name is already present in the job’s Resource_List attribute, it will be altered to the specified value. If the resource is not present in the attribute, it is added.

Certain attributes of a job may or may not be alterable depending on the state of the job; see qalter(1B).

The parameter, extend, is reserved for implementation defined extensions.

SEE ALSO

qalter(1B), qhold(1B), qrls(1B), qsub(1B), pbs_connect(3B), pbs_holdjob(3B), and pbs_rlsjob(3B)

DIAGNOSTICS

When the batch request generated by pbs_alterjob() function has been completed successfully by a batch server, the routine will return 0 (zero). Otherwise, a non zero error is returned. The error number is also set in pbs_errno.

NAME

pbs_connect - connect to a pbs batch server

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

int pbs_connect(char *server)

extern char *pbs_server;
```

DESCRIPTION

A virtual stream (TCP/IP) connection is established with the server specified by server.

This function must be called before any of the other pbs_ functions. They will transmit their batch requests over the connection established by this function. Multiple requests may be issued over the connection before it is closed.

The connection should be closed by a call to pbs_disconnect() when all requests have been sent to the server.

The parameter, server, is of the form host_name[:port], see section 2.7.9. If port is not specified, the standard PBS port number will be used.

If the parameter, server, is either the null string or a null pointer, a connection will be opened to the default server. The default server is defined by (a) the setting of the environment variable PBS_DEFAULT which contains a destination, or (b) by adding the parameter PBS_SERVER to the global configuration file /etc/pbs.conf.

The variable pbs_server, declared in pbs_ifl.h, is set on return to point to the server name to which pbs_connect() connected or attempted to connect.

In order to use pbs_connect with Windows, initialize the network library and link with winsock2. To do this, call winsock_init() before calling pbs_connect(), and link against the ws2_32.lib library.

SEE ALSO

qsub(1B), pbs_alterjob(3B), pbs_deljob(3B), pbs_disconnect(3B), pbs_geterrmsg(3B), pbs_holdjob(3B), pbs_locate(3B), pbs_manager(3B), pbs_movejob(3B), pbs_msgjob(3B), pbs_rerunjob(3B), pbs_rlsjob(3B), pbs_runjob(3B), pbs_selectjob(3B), pbs_selstat(3B), pbs_sigjob(3B), pbs_statjob(3B), pbs_statque(3B), pbs_statserver(3B), pbs_submit(3B), pbs_terminate(3B), pbs_server(8B), and the PBS External Reference Specification

DIAGNOSTICS

When the connection to batch server has been successfully created, the routine will return a connection identifier which is positive. Otherwise, a negative value is returned. The error number is set in pbs_errno.

NAME

pbs_default - return the name of the default PBS server

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

char *pbs_default()
```

DESCRIPTION

A character string is returned containing the name of the default PBS server. The default server is defined by (a) the setting of the environment variable PBS_DEFAULT which contains a destination, or (b) by adding the parameter PBS_SERVER to the global configuration file /etc/pbs.conf.

DIAGNOSTICS

If the default server cannot be determined, a NULL value is returned.

SEE ALSO

qsub(1B), pbs_connect(3B), pbs_disconnect(3B), and the PBS External Reference Specification

NAME
pbs_deljob - delete a PBS batch job

SYNOPSIS
#include <pbs_error.h>
#include <pbs_ifl.h>

int pbs_deljob(int connect, char *job_id, char *extend)

DESCRIPTION
Issue a batch request to delete a batch job. If the batch job is running, the execution server will send the SIGTERM signal followed by SIGKILL .

A Delete Job batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

The argument, job_id, identifies which job is to be deleted. It is specified in the form: sequence_number.server

The argument, extend, is overloaded to serve more than one purpose. If the pointer extend points to a string of the form:
deldelay=nnnn,
it is used to provide control over the delay between sending SIGTERM and SIGKILL signals to a running job. The characters nnnn specify an unsigned decimal integer time delay in seconds. If extend is the null pointer or points to a null string, the administrator-established default time delay is used.

If extend points to a string other than the above, it is taken as text to be appended to the message mailed to the job owner. This mailing occurs if the job is deleted by a user other than the job owner.

SEE ALSO
qdel(1B) and pbs_connect(3B)

DIAGNOSTICS
When the batch request generated by the pbs_deljob() function has been completed successfully by a batch server, the routine will return 0 (zero). Otherwise, a non zero error is returned. The error number is also set in pbs_errno.

NAME

pbs_delresv - delete a reservation

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

int pbs_delresv(int connect, char *resv_id, char *extend)
```

DESCRIPTION

Issue a batch request to delete a reservation. If the reservation is in state RESV_RUNNING, and there are jobs remaining in the reservation queue, the jobs will be deleted before the reservation is deleted.

A Delete Reservation batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

The argument, resv_id, identifies which reservation is to be deleted, it is specified in the form: ‘R’sequence_number.server

The argument, extend is currently unused.

SEE ALSO

pbs_rdel(1B) and pbs_connect(3B)

DIAGNOSTICS

When the batch request generated by the pbs_delresv() function has been completed successfully by a batch server, the routine will return 0 (zero). Otherwise, a non zero error is returned. The error number is also set in pbs_errno.

NAME
pbs_disconnect - disconnect from a pbs batch server

SYNOPSIS
```
#include <pbs_error.h>
#include <pbs_ifl.h>

int pbs_disconnect(int connect)
```

DESCRIPTION
The virtual stream connection specified by connect, which was established with a server by a call to pbs_connect(), is closed.

SEE ALSO
pbs_connect(3B)

DIAGNOSTICS
When the connection to batch server has been successfully closed, the routine will return zero. Otherwise, a non zero error is returned. The error number is also set in pbs_errno.

NAME

pbs_geterrmsg - get error message for last pbs batch operation

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

char *pbs_geterrmsg(int connect)
```

DESCRIPTION

Return the error message text associated with a batch server request.

If the preceding batch interface library call over the connection specified by connect resulted in an error return from the server, there may be an associated text message. If it exists, this function will return a pointer to the null terminated text string.

SEE ALSO

pbs_connect(3B)

DIAGNOSTICS

If an error text message was returned by a server in reply to the previous call to a batch interface library function, pbs_geterrmsg() will return a pointer to it. Otherwise, pbs_geterrmsg() returns the null pointer.

NAME

pbs_holdjob - place a hold on a pbs batch job

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

int pbs_holdjob(int connect, char *job_id, char *hold_type,
char *extend)
```

DESCRIPTION

Issue a batch request to place a hold upon a job.

A Hold Job batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

The argument, job_id, identifies which job is to be held, it is specified in the form: sequence_number.server

The parameter, hold_type, contains the type of hold to be applied. The possible values are defined in pbs_ifl.h as:

#define USER_HOLD “u”
Available to the owner of the job, the batch operator, and the batch administrator.

#define OTHER_HOLD “o”
Available to the batch operator and the batch administrator.

#define SYSTEM_HOLD “s”
Available only to the batch administrator.

If hold_type is either a null pointer or points to a null string, USER_HOLD will be applied.

The parameter, extend, is reserved for implementation defined extensions.

SEE ALSO

qhold(1B), pbs_connect(3B), pbs_alterjob(3B), and pbs_rlsjob(3B)

DIAGNOSTICS

When the batch request generated by pbs_holdjob () function has been completed successfully by a batch server, the routine will return 0 (zero). Otherwise, a non zero error is returned. The error number is also set in pbs_errno.

NAME

pbs_locjob - locate current location of a pbs batch job

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

char *pbs_locjob(int connect, char *job_id, char *extend)
```

DESCRIPTION

Issue a batch request to locate a batch job. If the server currently manages the batch job, or knows which server does currently manage the job, it will reply with the location of the job.

A Locate Job batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

The argument, job_id, identifies which job is to be located, it is specified in the form: sequence_number.server

The argument, extend, is reserved for implementation defined extensions. It is not currently used by this function.

The return value is a pointer to a character sting which contains the current location if known. The syntax of the location string is: queue@server_name. If the location of the job is not known, the return value is the NULL pointer.

SEE ALSO

qsub(1B) and pbs_connect(3B)

DIAGNOSTICS

When the batch request generated by the pbs_locjob() function has been completed successfully by a batch server, the routine will return a non null pointer to the destination. Otherwise, a null pointer is returned. The error number is set in pbs_errno.

NAME

pbs_manager - modifies a PBS batch object

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

int pbs_manager(int connect, int command, int obj_type, char *obj_name,
struct attropl *attrib, char *extend)
```

DESCRIPTION

Issue a batch request to perform administration functions at a server. With this request server objects such as queues can be created and deleted, and have their attributes set and unset.

A Manage batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect(). This request requires full batch administrator privilege.

The parameter, command, specifies the operation to be performed, see pbs_ifl.h:

```
#define MGR_CMD_CREATE 0
#define MGR_CMD_DELETE 1
#define MGR_CMD_SET    2
#define MGR_CMD_UNSET  3
```

The parameter, obj_type, declares the type of object upon which the command operates, see pbs_ifl.h:

```
#define MGR_OBJ_SERVER  0
#define MGR_OBJ_QUEUE   1
#define MGR_OBJ_NODE    3
```

The parameter, obj_name, is the name of the specific object.

The parameter, attrib, is a pointer to an attropl structure which are defined in pbs_ifl.h as:

```
struct attropl {
    char   *name;
    char   *resource;
    char   *value;
    enum batch_op op;
    struct attropl *next;
 };
```

The attrib list is terminated by the first entry where next is a null
pointer.

The name member points to a string which is the name of the attribute.

If the attribute is one which contains a set of resources, the specific
resource is specified in the structure member resource. Otherwise, the
member resource is pointer to a null string.

The value member points to a string which is the new value of the
attribute.

The op member defines the manner in which the new value is assigned to
the attribute. The operators are: enum batch_op { ..., SET, UNSET,
INCR, DECR };

The parameter extend is reserved for implementation defined extensions.

Functions MGR_CMD_CREATE and MGR_CMD_DELETE require PBS Manager
privi-
lege.   Functions MGR_CMD_SET and MGR_CMD_UNSET require PBS Manager
or
Operator privilege.

DIAGNOSTICS
When the batch request generated by pbs_manager() function has been
completed successfully by a batch server, the routine will return 0
(zero). Otherwise, a non zero error is returned. The error number is
also set in pbs_errno.

SEE ALSO
qmgr(8B), pbs_connect(3B)

Local   23 June 2005   pbs_manager(3B)

NAME

pbs_movejob - move a pbs batch job to a new destination

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

int pbs_movejob(int connect, char *job_id, char *destination,
char *extend)
```

DESCRIPTION

Issue a batch request to move a job to a new destination. The job is removed from the present queue and instantiated in a new queue.

A Move Job batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

The job_id parameter identifies which job is to be moved; it is specified in the form: sequence_number.server

The destination parameter specifies the new destination for the job. It is specified as: [queue][@server]. If destination is a null pointer or a null string, the destination will be the default queue at the current server. If destination specifies a queue but not a server, the destination will be the named queue at the current server. If destination specifies a server but not a queue, the destination will be the default queue at the named server. If destination specifies both a queue and a server, the destination is that queue at that server.

A job in the Running , Transiting , or Exiting state cannot be moved.

The parameter, extend, is reserved for implementation defined extensions.

SEE ALSO

qmove(1B), qsub(1B), and pbs_connect(3B)

DIAGNOSTICS

When the batch request generated by pbs_movejob() function has been completed successfully by a batch server, the routine will return 0 (zero). Otherwise, a non zero error is returned. The error number is also set in pbs_errno.

NAME

pbs_msgjob - record a message for a running pbs batch job

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

int pbs_msgjob(int connect, char *job_id, int file, char *message,
char *extend)
```

DESCRIPTION

Issue a batch request to write a message in an output file of a batch job.

A Message Job batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

The argument, job_id, identifies the job to which the message is to be sent; it is specified in the form: sequence_number.server

The parameter, file, indicates the file or files to which the message string is to be written. The following values are defined in pbs_ifl.h:

#define MSG_ERR 2
: directs the message to the standard error stream of the job.

#define MSG_OUT 1
: directs the message to the standard output stream of the job.

The parameter, message, is the message string to be written.

The parameter, extend, is reserved for implementation defined extensions.

SEE ALSO

qmsg(1B) and pbs_connect(3B)

DIAGNOSTICS

When the batch request generated by pbs_msgjob() function has been completed successfully by a batch server, the routine will return 0 (zero). Otherwise, a non zero error is returned. The error number is also set in pbs_errno.

NAME

pbs_orderjob - reorder pbs batch jobs in a queue

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

int pbs_orderjob(int connect, char *job_id1, char *job_id2,
char *extend)
```

DESCRIPTION

Issue a batch request to swap the order of two jobs with in a single queue.

An Order Job batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

The parameters job_id1 and job_id2 identify which jobs are to be swapped. They are specified in the form: sequence_number.server.

The parameter, extend, is reserved for implementation defined extensions.

SEE ALSO

qorder(1B), qmove(1B), qsub(1M), and pbs_connect(3B)

DIAGNOSTICS

When the batch request generated by pbs_orderjob() function has been completed successfully by a batch server, the routine will return 0 (zero). Otherwise, a non zero error is returned. The error number is also set in pbs_errno.

NAME

pbs_rerunjob - rerun a pbs batch job

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

int pbs_rerunjob(int connect, char *job_id, char *extend)
```

DESCRIPTION

Issue a batch request to rerun a batch job.

A Rerun Job batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

If the job is marked as being not rerunable, the request will fail and an error will be returned.

The argument, job_id, identifies which job is to be rerun it is specified in the form: sequence_number.server

The parameter, extend, is reserved for implementation defined extensions.

SEE ALSO

qrerun(1B), qsub(1B), and pbs_connect(3B)

DIAGNOSTICS

When the batch request generated by pbs_rerunjob() function has been completed successfully by a batch server, the routine will return 0 (zero). Otherwise, a non zero error is returned. The error number is also set in pbs_errno.

NAME

pbs_rescreserve, pbs_rescrelease - reserve/free batch resources

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

int pbs_rescreserve(int connect, char **resourcelist, int arraysize,
resource_t *resource_id)

int pbs_rescrelease(int connect, resource_t resource_id)
```

DESCRIPTION

pbs_rescreserver

Issue a request to the batch server to reserve specified resources. connect is the connection returned by pbs_connect(). resourcelist is an array of one or more strings specifying the resources to be queried. arraysize is the is the number of strings in resourcelist. resource_id is a pointer to a resource handle. The pointer cannot be null. If the present value of the resource handle is RESOURCE_T_NULL , this request is for a new reservation and if successful, a resource handle will be returned in resource_id.

If the value of resource_id as supplied by the caller is not RESOURCE_T_NULL , this is a existing (partial) reservation. Resources currently reserved for this handle will be released and the full reservation will be attempted again. If the caller wishes to release the resources allocated to a partial reservation, the caller should pass the resource handle to pbs_rescrelease().

At the present time the only resources which may be specified are “nodes”. It should be specified as nodes=specification where specification is what a user specifies in the -l option arguement list for nodes, see qsub (1B).

pbs_rescrelease

The pbs_rescrelease() call releases or frees resources reserved with the resource handle of resource_id returned from a prior pbs_rescreserve() call. connect is the connection returned by pbs_connect().

Both functions require that the issuing user have operator or administrator privilege.

SEE ALSO

qsub(1B), pbs_connect(3B), pbs_disconnect(3B) and pbs_resources(7B)

DIAGNOSTICS

pbs_rescreserve() and pbs_rescrelease() return zero on success. Otherwise, a non zero error is returned. The error number is also set in pbs_errno.

PBSE_RMPART
: is a special case indicating that some but not all of the requested resources could be reserved; a partial reservation was made. The reservation request should either be rerequested with the returned handle or the partial resources released.

PBSE_RMBADPARAM
: a parameter is incorrect, such as a null for the pointer to the resource_id.

PBSE_RMNOPARAM
: a parameter is missing, such as a null resoruce list.

NAME

pbs_rlsjob - release a hold on a pbs batch job

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

int pbs_rlsjob(int connect, char *job_id, char *hold_type, char *extend)
```

DESCRIPTION

Issue a batch request to release a hold from a job.

A Release Job batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

The argument, job_id, identifies the job from which the hold is to be released, it is specified in the form: sequence_number.server

The parameter, hold_type, contains the type of hold to be released. The possible values are defined in pbs_ifl.h as:

#define USER_HOLD “u”
Available to the owner of the job, the batch operator, and the batch administrator.

#define OTHER_HOLD “o”
Available to the batch operator and the batch administrator.

#define SYSTEM_HOLD “s”
Available only to the batch administrator.

If hold_type is either a null pointer or points to a null string, USER_HOLD will be released.

The parameter, extend, is reserved for implementation defined extensions.

SEE ALSO

qrls(1B), qhold(1B), qalter(1B), pbs_alterjob(3B), pbs_connect(3B), and pbs_holdjob(3B)

DIAGNOSTICS

When the batch request generated by pbs_rlsjob() function has been completed successfully by a batch server, the routine will return 0 (zero). Otherwise, a non zero error is returned. The error number is also set in pbs_errno.

NAME

pbs_runjob - run a pbs batch job

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

int pbs_runjob(int connect, char *job_id, char *location, char *extend)

int    pbs_asyrunjob(int connect,     char *job_id,     char *location,
char *extend)
```

DESCRIPTION

Issue a batch request to run a batch job.

For pbs_runjob() a “Run Job” batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect(). The server will reply when the job has started execution unless file in-staging is required. In that case, the server will reply when the staging operations are started.

For pbs_asyrunjob() an “Asynchronous Run Job” request is generated and set to the server over the connection. The server will validate the request and reply before initiating the execution of the job. This version of the call can be used to reduce latency in scheduling, especially when the scheduler must start a large number of jobs.

These requests requires that the issuing user have operator or administrator privilege.

The argument, job_id, identifies which job is to be run it is specified in the form: sequence_number.server

The argument, location, if not the null pointer or null string, specifies the location where the job should be run. The location is the name of a host in the the cluster managed by the server.

The argument, extend, is reserved for implementation defined extensions.

SEE ALSO

qrun(8B), qsub(1B), and pbs_connect(3B)

DIAGNOSTICS

When the batch request generated by the pbs_runjob() or pbs_asyrunjob() functions has been completed successfully by a batch server, the routines will return 0 (zero). Otherwise, a non zero error is returned. The error number is also set in pbs_errno.

NAME

pbs_selectjob - select pbs batch jobs

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

char **pbs_selectjob(int connect, struct attropl *attrib, char *extend)
```

DESCRIPTION

Issue a batch request to select jobs which meet certain criteria. pbs_selectjob() returns a array of job identifiers which met the criteria.

Initially all batch jobs are selected for which the user is authorized to query status. This set may be reduced or filtered by specifying certain attributes of the jobs.

A Select Jobs batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

The argument, attrib, is a pointer to an attropl structure which is defined in pbs_ifl.h as:

```
struct attropl {
    struct attropl *next;
    char          *name;
    char          *resource;
    char          *value;
    enum batch_op   op;
};
```

The attrib list is terminated by the first entry where next is a null pointer.

The name member points to a string which is the name of the attribute. Not all of the job attributes may be used as a selection criteria. The resource member points to a string which is the name of a resource. This member is only used when name is set to ATTR_l. Otherwise, resource should be a pointer to a null string. The value member points to a string which is the value of the attribute or resource. The attribute names are defined in pbs_ifl.h:

#define ATTR_a “Execution_Time”
Select based upon the job’s execution time.

#define ATTR_A “Account_Name”
Select (E) based upon the account string.

#define ATTR_c “Checkpoint”
Select based upon the checkpoint interval.

#define ATTR_e “Error_Path”
Select (E) based upon the name of the standard error file.

#define ATTR_g “Group_List”
Select (E) based upon the list of group names under which the job may execute.

#define ATTR_h “Hold_Types”
Select (E) based upon the hold types.

#define ATTR_j “Join_Paths”
Select (E) based upon the value of the join list.

#define ATTR_k “Keep_Files”
Select (E) based upon the value of the keep files list.

#define ATTR_l “Resource_List”
Select based upon the value of the resource named in resource.

#define ATTR_m “Mail_Points”
Select (E) based upon the setting of the mail points attribute.

#define ATTR_M “Mail_Users”
Select (E) based upon the list of user names to which mail will be sent.

#define ATTR_N “Job_Name”
Select (E) based upon the job name.

#define ATTR_o “Output_Path”
Select (E) based upon the name of the standard output file.

#define ATTR_p “Priority”
Select based upon the priority of the job.

#define ATTR_q “destination”
Select based upon the specified destination. Jobs selected are restricted to those residing in the named queue. If destination is the null string, the default queue at the server is assumed.

#define ATTR_r “Rerunnable”
Select (E) based upon the rerunnable flag.

#define ATTR_session “session_id”
Select based upon the session id assigned to running jobs.

#define ATTR_S “Shell_Path_List”
Select (E) based upon the execution shell list.

#define ATTR_u “User_List”
Select (E) based upon the owner of the jobs.

#define ATTR_v “Variable_List”
Select (E) based upon the list of environment variables.

#define ATTR_ctime “ctime”
Select based upon the creation time of the job.

#define ATTR_depend “depend”
Select based upon the list of job dependencies.

#define ATTR_mtime “mtime”

Select based upon the last modification time of the job.

#define ATTR_qtime “qtime”
Select based upon the time of the job was placed into the current queue.

#define ATTR_qtype “queue_type”
Select (E) base on the type of queue in which the job resides.

#define ATTR_stagein “stagein”
Select based upon the list of files to be staged-in.

#define ATTR_stageout “stageout”
Select based upon the list of files to be staged-out.

#define ATTR_state “job_state”
Select based upon the state of the jobs. State is not a job attribute, but is included here to allow selection.

The op member defines the operator in the logical expression:
value operator current_value
The logical expression must evaluate as true for the job to be selected. The permissible values of op are defined in pbs_ifl.h as: enum batch_op { ..., EQ, NE, GE, GT, LE, LT, ... };. The attributes marked with (E) in the description above may only be selected with the equal, EQ, or not equal, NE, operators.

If attrib itself is a null pointer, then no selection is done on the basis of attributes.

The return value is a pointer to a null terminated array of character pointers. Each character pointer in the array points to a character string which is a job_identifier in the form: sequence_number.server@server

The array is allocated by pbs_selectjob via malloc(). When the array is no longer needed, the user is responsible for freeing it by a call to free().

The parameter, extend, is reserved for implementation defined extensions.

SEE ALSO

qselect(1B), pbs_alterjob(3B), and pbs_connect(3B)

DIAGNOSTICS

When the batch request generated by pbs_selectjob() function has been completed successfully by a batch server, the routine will return a pointer to the array of job identifiers. If no jobs met the criteria, the first pointer in the array will be the null pointer.

If an error occurred, a null pointer is returned and the error is available in the global integer pbs_errno.

NAME

pbs_selstat - obtain status of selected pbs batch jobs

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

struct batch_status *pbs_selstat(int connect, struct attropl *sel_list,
struct attrl *rattrib char *extend)

void pbs_statfree(struct batch_status *psj)
```

DESCRIPTION

Issue a batch request to examine the status of jobs which meet certain criteria. pbs_selstat() returns a list of batch_status structures for those jobs which met the selection criteria.

This function is a combination of pbs_selectjobs() and pbs_statjob(). It is an extension to the POSIX Batch standard.

Initially all batch jobs are selected for which the user is authorized to query status. This set may be reduced or filtered by specifying certain attributes of the jobs.

A Select Status batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

The parameter, sel_list, is a pointer to an attropl structure which is defined in pbs_ifl.h as:

```
struct attropl {
    struct attropl *next;
    char          *name;
    char          *resource;
    char          *value;
    enum batch_op   op;
};
```

The sel_list list is terminated by the first entry where next is a null pointer.

The name member points to a string which is the name of the attribute. Not all of the job attributes may be used as a selection criteria. The resource member points to a string which is the name of a resource. This member is only used when name is set to ATTR_l, otherwise it should be a pointer to a null string. The value member points to a string which is the value of the attribute or resource. The attribute names are defined in pbs_ifl.h:

#define ATTR_a "Execution_Time"
  Select based upon the job's execution time.

#define ATTR_A "Account_Name"
  Select (E) based upon the account string.

#define ATTR_c "Checkpoint"
  Select based upon the checkpoint interval.

#define ATTR_e "Error_Path"
  Select (E) based upon the name of the standard error file.

#define ATTR_g "Group_List"
  Select (E) based upon the list of group names under which the job may execute.

#define ATTR_h "Hold_Types"
  Select (E) based upon the hold types.

#define ATTR_j "Join_Paths"
  Select (E) based upon the value of the join list.

#define ATTR_k "Keep_Files"
  Select (E) based upon the value of the keep files list.

#define ATTR_l "Resource_List"
  Select based upon the value of the resource named in resource.

#define ATTR_m “Mail_Points”
Select (E) based upon the setting of the mail points attribute.

#define ATTR_M “Mail_Users”
Select (E) based upon the list of user names to which mail will be sent.

#define ATTR_N “Job_Name”
Select (E) based upon the job name.

#define ATTR_o “Output_Path”
Select (E) based upon the name of the standard output file.

#define ATTR_p “Priority”
Select based upon the priority of the job.

#define ATTR_q “destination”
Select based upon the specified destination. Jobs selected are restricted to those residing in the named queue. If destination is the null string, the default queue at the server is assumed.

#define ATTR_r “Rerunable”
Select (E) based upon the rerunable flag.

#define ATTR_session “session_id”
Select based upon the session id assigned to running jobs.

#define ATTR_S “Shell_Path_List”
Select (E) based upon the execution shell list.

#define ATTR_u “User_List”
Select (E) based upon the owner of the jobs.

#define ATTR_v “Variable_List”
Select (E) based upon the list of environment variables.

#define ATTR_ctime “ctime”

Select based upon the creation time of the job.

#define ATTR_depend “depend”
Select based upon the list of job dependencies.

#define ATTR_mtime “mtime”
Select based upon the last modification time of the job.

#define ATTR_qtime “qtime”
Select based upon the time of the job was placed into the current queue.

#define ATTR_qtype “queue_type”
Select (E) base on the type of queue in which the job resides.

#define ATTR_stagein “stagein”
Select based upon the list of files to be staged-in.

#define ATTR_stageout “stageout”
Select based upon the list of files to be staged-out.

#define ATTR_state “job_state”
Select based upon the state of the jobs. State is not a job attribute, but is included here to allow selection.

The op member defines the operator in the logical expression:
value operator current_value
The logical expression must evaluate as true for the job to be selected. The permissible values of op are defined in pbs_ifl.h as: enum batch_op { ..., EQ, NE, GE, GT, LE, LT, ... };. The attributes marked with (E) in the description above may only be selected with the equal, EQ, or not equal, NE, operators.

If sel_list itself is a null pointer, then no selection is done on the basis of attributes.

The parameter, rattrib, is a pointer to an attrl structure which is defined below. The rattrib list is terminated by the first entry where next is a null pointer. If attrib is given, then only the attributes in the list are returned by the server. Otherwise, all the attributes of a job are returned. When an attrib list is specified, the name member is a pointer to a attribute name as listed in pbs_alter(3) and pbs_submit(3). The resource member is only used if the name member is ATTR_l, otherwise it should be a pointer to a null string. The value member should always be a pointer to a null string.

The return value is a pointer to a list of batch_status structures or the null pointer if no jobs can be queried for status. The batch_status structure is defined in pbs_ifl.h as

```
struct batch_status {
    struct batch_status *next;
    char                *name;
    struct attrl         *attribs;
    char                *text;
}
```

The entry, attribs, is a pointer to a list of attrl structures defined in pbs_ifl.h as:

```
struct attrl {
    struct attrl *next;
    char        *name;
    char        *resource;
    char        *value;
};
```

It is up the user to free the list of batch_status structures when no longer needed, by calling pbs_statfree().

The extend parameter is for optional features and or additions. Normally, this should be null pointer.

SEE ALSO
qselect(1B), pbs_alterjob(3B), pbs_connect(3B), pbs_statjob(3B), and pbs_selectjob(3B).

DIAGNOSTICS
When the batch request generated by pbs_selstat() function has been completed successfully by a batch server, the routine will return a

pointer to the list of batch_status structures. If no jobs met the criteria or an error occurred, the return will be the null pointer. If an error occurred, the global integer pbs_errno will be set to a non-zero value.

NAME

pbs_sigjob - send a signal to a pbs batch job

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

int pbs_sigjob(int connect, char *job_id, char *signal, char *extend)
```

DESCRIPTION

Issue a batch request to send a signal to a batch job.

A Signal Job batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect(). If the batch job is in the running state, the batch server will send the job the signal number corresponding to the signal named in signal.

The argument, job_id, identifies which job is to be signaled, it is specified in the form: sequence_number.server

The signal argument is the name of a signal. It may be the alphabetic form with or without the SIG prefix, or it may be a numeric string for the signal number. Two special names are recognized, suspend and resume . If the name of the signal is not a recognized signal name on the execution host, no signal is sent and an error is returned. If the job is not in the running state, no signal is sent and an error is returned, except when the signal is resume and the job is suspended.

The parameter, extend, is reserved for implementation defined extensions.

SEE ALSO

qsig(1B) and pbs_connect(3B)

DIAGNOSTICS

When the batch request generated by pbs_sigjob() function has been completed successfully by a batch server, the routine will return 0 (zero). Otherwise, a non zero error is returned. The error number is also set in pbs_errno.

NAME

pbs_stagein - request that files for a pbs batch job be staged in.

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

int pbs_stagein(int connect, char *job_id, char *location, char *extend)
```

DESCRIPTION

Issue a batch request to start the stage in of files specified in the stagein attribute of a batch job.

A stage in batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

This request directs the server to begin the stage in of files specified in the job's stage in attribute. This request requires that the issuing user have operator or administrator privilege.

The argument, job_id, identifies which job for which file staging is to begin. It is specified in the form: sequence_number.server

The argument, location, if not the null pointer or null string, specifies the location where the job will be run and hence to where the files will be staged. The location is the name of a host in the cluster managed by the server. If the job is then directed to run at different location, the run request will be rejected.

The argument, extend, is reserved for implementation defined extensions.

SEE ALSO

qrun(8B), qsub(1B), and pbs_connect(3B)

DIAGNOSTICS

When the batch request generated by pbs_stagein() function has been completed successfully by a batch server, the routine will return 0 (zero). Otherwise, a non zero error is returned. The error number is also set in pbs_errno.

Local pbs_stagein(3B)

NAME

pbs_statjob - obtain status of pbs batch jobs

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

struct batch_status *pbs_statjob(int connect, char *id,
struct attrl *attrib, char *extend)

void pbs_statfree(struct batch_status *psj)
```

DESCRIPTION

Issue a batch request to obtain the status of a specified batch job or a set of jobs at a destination.

A Status Job batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

The parameter, id, may be either a job identifier or a destination identifier.

If id is a job identifier, it is the identifier of the job for which status is requested. It is specified in the form: sequence_number.server

If id is a destination identifier, it specifies that status of all jobs at the destination (queue) which the user is authorized to see be returned. If id is the null pointer or a null string, the status of each job at the server which the user is authorized to see is returned.

The parameter, attrib, is a pointer to an attrl structure which is defined in pbs_ifl.h as:

```
struct attrl {
    struct attrl *next;
    char        *name;
    char        *resource;
    char        *value;
    };
```

The attrib list is terminated by the first entry where next is a null pointer. If attrib is given, then only the attributes in the list are returned by the server. Otherwise, all the attributes of a job are returned. When an attrib list is specified, the name member is a pointer to a attribute name as listed in pbs_alter(3) and pbs_submit(3). The resource member is only used if the name member is ATTR_l, otherwise it should be a pointer to a null string. The value member should always be a pointer to a null string.

The parameter, extend, is reserved for implementation defined extensions.

The return value is a pointer to a list of batch_status structures or the null pointer if no jobs can be queried for status. The batch_status structure is defined in pbs_ifl.h as

```
    struct batch_status {
        struct batch_status *next;
        char                *name;
        struct attrl         *attribs;
        char                *text;
    }
```

It is up the user to free the structure when no longer needed, by calling pbs_statfree().

SEE ALSO
qstat(1B) and pbs_connect(3B)

DIAGNOSTICS
When the batch request generated by pbs_statjob() function has been completed successfully and the status of each job has been returned by the batch server, the routine will return a pointer to the list of batch_status structures. If no jobs were available to query or an error occurred, a null pointer is returned. The global integer pbs_errno should be examined to determine the cause.

Local pbs_statjob(3B)

NAME

pbs_statnode - obtain status of PBS nodes

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

struct batch_status *pbs_stathost(int connect, char *id,
struct attrl *attrib, char *extend)

struct batch_status *pbs_statnode(int connect, char *id,
struct attrl *attrib, char *extend)

struct batch_status *pbs_statvnode(int connect, char *id,
struct attrl *attrib, char *extend)

void pbs_statfree(struct batch_status *psj)
```

DESCRIPTION

Issue a batch request to obtain the status of PBS execution hosts or vnodes.

pbs_stathost returns information about the single host named in the call or about all hosts known to the PBS Server.

pbs_statnode is identical to pbs_stathost in function. It is retained for backward compatibility.

pbs_statvnode returns information about the single virtual node (vnode) named in the call or about all vnodes known to the PBS Server.

A Status Node batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

The id is the name of a host for pbs_stathost, or a vnode for pbs_statvnode, or the null string. If id specifies a name, the status of that host or vnode will be returned. If the id is a null string (or null pointer), the status of all hosts or vnodes at the server will be returned.

The parameter, attrib, is a pointer to an attrl structure which is defined in pbs_ifl.h as:

```
struct attrl {
    struct attrl *next;
    char        *name;
    char        *resource;
    char        *value;
};
```

The attrib list is terminated by the first entry where next is a null pointer. If attrib is given, then only the attributes in the list are returned by the server. Otherwise, all the attributes of a node are returned. When an attrib list is specified, the name member is a pointer to a attribute name. The resource member is not used and must be a pointer to a null string. The value member should always be a pointer to a null string.

The parameter, extend, is reserved for implementation defined extensions.

The return value is a pointer to a list of batch_status structures, which is defined in pbs_ifl.h as:

```
struct batch_status {
    struct batch_status *next;
    char              *name;
    struct attrl       *attribs;
    char              *text;
}
```

It is up the user to free the structure when no longer needed, by calling pbs_statfree().

DIAGNOSTICS

When the batch request generated by pbs_stathost(), pbs_statnode(), or pbs_statvnode() function has been completed successfully by a batch server, the routine will return a pointer to the batch_status structure. Otherwise, a null pointer is returned and the error code is set in the global integer pbs_errno.

SEE ALSO

qstat(1B), pbs_connect(3B)

NAME

pbs_statque - obtain status of pbs batch queues

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

struct batch_status *pbs_statque(int connect, char *id,
struct attrl *attrib,
char *extend)

void pbs_statfree(struct batch_status *psj)
```

DESCRIPTION

Issue a batch request to obtain the status of a batch queue.

A Status Queue batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

The id is the name of a queue, in the form:

queue_name

or the null string. If queue_name is specified, the status of the queue named queue_name at the server will be returned. If the id is a null string or null pointer, the status of all queues at the server will be returned.

The parameter, attrib, is a pointer to an attrl structure which is defined in pbs_ifl.h as:

```
struct attrl {
    struct attrl *next;
    char        *name;
    char        *resource;
    char        *value;
};
```

The attrib list is terminated by the first entry where next is a null pointer. If attrib is given, then only the attributes in the list are returned by the server. Otherwise, all the attributes of a queue are returned. When an attrib list is specified, the name member is a pointer to a attribute name as listed in pbs_alter(3) and pbs_submit(3). The resource member is only used if the name member is ATTR_l, otherwise it should be a pointer to a null string. The value member should always be a pointer to a null string.

The parameter, extend, is reserved for implementation defined extensions.

The return value is a pointer to a list of batch_status structures, which is defined in pbs_ifl.h as:

```
struct batch_status {
    struct batch_status *next;
    char              *name;
    struct attrl       *attribs;
    char              *text;
}
```

It is up the user to free the structure when no longer needed, by calling pbs_statfree().

SEE ALSO

qstat(1B) and pbs_connect(3B)

DIAGNOSTICS

When the batch request generated by pbs_statque() function has been completed successfully by a batch server, the routine will return a pointer to the batch_status structure. Otherwise, a null pointer is returned and the error code is set in the global integer pbs_errno.

NAME

pbs_statresv - obtain status information about reservations

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

struct batch_status *pbs_statresv(int connect, char *id,
struct attrl *attrib, char *extend)

void pbs_statfree(struct batch_status *psj)
```

DESCRIPTION

Issue a batch request to obtain the status of a specified reservation or a set of reservations at a destination.

A Status Reservation batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

The parameter, id, is a reservation identifier. A reservation identifier is of the form:
‘R’sequence_number.server

If id is the null pointer or a null string, the status of each reservation at the server which the user is authorized to see is returned.

The parameter, attrib, is a pointer to an attrl structure which is defined in pbs_ifl.h as:

```
struct attrl {
    struct attrl *next;
    char        *name;
    char        *resource;
    char        *value;
};
```

The attrib list is terminated by the first entry where next is a null pointer. If attrib is given, then only the attributes in the list are returned by the server. Otherwise, all the attributes of a reservation are returned. When an attrib list is specified, the name member is a pointer to a attribute name as listed in pbs_submitresv(3). The resource member is only used if the name member is ATTR_l, otherwise it should be a pointer to a null string. The value member should always be a pointer to a null string.

The parameter, extend, is reserved for implementation defined extensions.

The return value is a pointer to a list of batch_status structures or the null pointer if no reservations can be queried for status. The batch_status structure is defined in pbs_ifl.h as

```
struct batch_status {
    struct batch_status *next;
    char              *name;
    struct attrl       *attribs;
    char              *text;
}
```

It is up the user to free the structure when no longer needed, by calling pbs_statfree().

SEE ALSO
pbs_rstat(1B) and pbs_connect(3B)

DIAGNOSTICS
When the batch request generated by pbs_statresv() function has been completed successfully and the status of each reservation has been returned by the batch server, the routine will return a pointer to the list of batch_status structures. If no reservations were available to query or an error occurred, a null pointer is returned. The global integer pbs_errno should be examined to determine the cause.

NAME

pbs_statsched - obtain status of PBS scheduler

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

struct batch_status *pbs_statsched(int connect, struct attrl *attrib,
char *extend)

void pbs_statfree(struct batch_status *psj)
```

DESCRIPTION

Issue a batch request to obtain the status of PBS scheduler.

A Status Scheduler batch request is generated and sent to the server. The parameter connect is the return value of pbs_connect().

The parameter, attrib, is a pointer to an attrl structure which is defined in pbs_ifl.h as:

```
struct attrl {
    struct attrl *next;
    char        *name;
    char        *resource;
    char        *value;
};
```

The attrib list is terminated by the first entry where next is a null pointer. If attrib is given, then only the attributes in the list are returned by the server. Otherwise, all the attributes of the scheduler are returned. When an attrib list is specified, the name member is a pointer to an attribute name as listed in pbs_alter(3) and pbs_submit(3). The resource member is only used if the name member is ATTR_l, otherwise it should be a pointer to a null string. The value member should always be a pointer to a null string.

The parameter, extend, is reserved for implementation-qdefined extensions.

The return value of pbs_statsched() is a pointer to a list of batch_status structures, which is defined in pbs_ifl.h as:

```
struct batch_status {
    struct batch_status *next;
    char                *name;
    struct attrl         *attribs;
    char                *text;
}
```

It is up the user to free the batch_status structure when it is no longer needed, by calling pbs_statfree().

SEE ALSO

qstat(1B) and pbs_connect(3B)

DIAGNOSTICS

When the batch request generated by pbs_statsched() has been completed successfully by the PBS server, pbs_statsched() will return a pointer to a batch_status structure. Otherwise, a null pointer is returned and the error code is set in pbs_errno.

NAME

pbs_statserver - obtain status of a pbs batch server

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

struct batch_status *pbs_statserver(int connect, struct attrl *attrib,
char *extend)

void pbs_statfree(struct batch_status *psj)
```

DESCRIPTION

Issue a batch request to obtain the status of a batch server.

A Status Server batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

The parameter, attrib, is a pointer to an attrl structure which is defined in pbs_ifl.h as:

```
struct attrl {
    struct attrl *next;
    char        *name;
    char        *resource;
    char        *value;
};
```

The attrib list is terminated by the first entry where next is a null pointer. If attrib is given, then only the attributes in the list are returned by the server. Otherwise, all the attributes of the server are returned. When an attrib list is specified, the name member is a pointer to a attribute name as listed in pbs_alter(3) and pbs_submit(3). The resource member is only used if the name member is ATTR_l, otherwise it should be a pointer to a null string. The value member should always be a pointer to a null string.

The parameter, extend, is reserved for implementation defined extensions.

The return value is a pointer to a list of batch_status structures, which is defined in pbs_ifl.h as:

```
struct batch_status {
    struct batch_status *next;
    char                *name;
    struct attrl        *attribs;
    char                *text;
}
```

It is up the user to free the space when no longer needed, by calling pbs_statfree().

SEE ALSO
qstat(1B) and pbs_connect(3B)

DIAGNOSTICS
When the batch request generated by pbs_statserver() function has been completed successfully by a batch server, the routine will return a pointer to a batch_status structure. Otherwise, a null pointer is returned and the error code is set in pbs_errno.

NAME

pbs_submit - submit a pbs batch job

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>
size_t cred_len=0;
char* cred_buf;
int cred_type;

char *pbs_submit(int connect, struct attropl *attrib,
char *script, char *destination, char *extend)
```

DESCRIPTION

Issue a batch request to submit a new batch job.

A Queue Job batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect(). The job will be submitted to the queue specified by destination.

The parameter, attrib, is a list of attropl structures which is defined in pbs_ifl.h as:

```
struct attrl {
    char   *name;
    char   *resource;
    char   *value;
    struct attrl *next;
    enum batch_op op;
};
```

The attrib list is terminated by the first entry where next is a null pointer.

The name member points to a string which is the name of the attribute. The value member points to a string which is the value of the attribute. The attribute names are defined in pbs_ifl.h:

```
#define ATTR_a “Execution_Time”
```

Defines the job’s execution time.

#define ATTR_A “Account_Name”
Defines the account string.

#define ATTR_c “Checkpoint”
Defines the checkpoint interval.

#define ATTR_e “Error_Path”
Defines the path name for the standard error of the job.

#define ATTR_g “Group_List”
Defines the list of group names under which the job may execute.

#define ATTR_h “Hold_Types”
Defines the hold types, the only allowable value string is “u”.

#define ATTR_j “Join_Paths”
Defines whether standard error and standard output are joined (merged).

#define ATTR_k “Keep_Files”
Defines which output of the job is kept on the execution host.

#define ATTR_l “Resource_List”
Defines a resource required by the job.

#define ATTR_m “Mail_Points”
Defines the points at which the server will send mail about the job.

#define ATTR_M “Mail_Users”
Defines the list of users who would receive mail about the job.

#define ATTR_N “Job_Name”
Defines the job name.

#define ATTR_o “Output_Path”
Defines the path name for the standard output of the job.

#define ATTR_p “Priority”
Defines the priority of the job.

#define ATTR_r “Rerunable”
Defines the rerunable flag.

#define ATTR_S “Shell_Path_List”
Defines the path to the shell which will interpret the job script.

#define ATTR_u “User_List”
Defines the list of user names under which the job may execute.

#define ATTR_v “Variable_List”
Defines the list of additional environment variables which are exported to the job.

#define ATTR_depend “depend”
Defines the inter-job dependencies.

#define ATTR_stagein “stagein”
Defines the list of files to be staged in prior to job execution.

#define ATTR_stageout “stageout”
Defines the list of files to be staged out after job execution.

If an attribute is not named in the attrib array, the default action will be taken. It will either be assigned the default value or will not be passed with the job. The action depends on the attribute. If attrib itself is a null pointer, then the default action will be taken for each attribute.

Associated with an attribute of type ATTR_l (the letter ell) is a resource name indicated by resource in the attrl structure. All other attribute types should have a pointer to a null string for resource.

The op member is forced to a value of SET by pbs_submit().

The parameter, script, is the path name to the job script. If the path name is relative, it will be expanded to the processes current working directory. If script is a null pointer or the path name pointed to is specified as the null string, no script is passed with the job.

The destination parameter specifies the destination for the job. It is specified as: [queue] If destination is the null string or the queue is not specified, the destination will be the default queue at the connected server.

The parameter, extend, is reserved for implementation defined extensions.

The return value is a character string which is the job_identifier assigned to the job by the server. The space for the job_identifier string is allocated by pbs_submit() and should be released via a call to free() by the user when no longer needed.

SEE ALSO

qsub(1B) and pbs_connect(3B)

DIAGNOSTICS

When the batch request generated by pbs_submit() function has been completed successfully by a batch server, the routine will return a pointer to a character string which is the job identifier of the submitted batch job. Otherwise, a null pointer is returned and the error code is set in pbs_error.

NAME

pbs_submitresv - submit a pbs reservation

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

char *pbs_submitresv(int connect, struct attropl *attrib, char *extend)
```

DESCRIPTION

Issue a batch request to submit a new reservation.

A Submit Reservation batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

The parameter, attrib, is a list of attropl structures which is defined in pbs_ifl.h as:

```
struct attrl {
    char   *name;
    char   *resource;
    char   *value;
    struct attrl *next;
    enum batch_op op;
};
```

The attrib list is terminated by the first entry where next is a null pointer.

The name member points to a string which is the name of the attribute. The value member points to a string which is the value of the attribute. The attribute names are defined in pbs_ifl.h.

If an attribute is not named in the attrib array, the default action will be taken. It will either be assigned the default value or will not be passed with the reservation. The action depends on the attribute. If attrib itself is a null pointer, then the default action will be taken for each attribute.

Associated with an attribute of type ATTR_l (the letter ell) is a resource name indicated by resource in the attrl structure. All other attribute types should have a pointer to a null string for resource.

The op member is forced to a value of SET by pbs_submitresv().

The parameter, extend, is reserved for implementation defined extensions.

The return value is a character string which is the reservation_identifier assigned to the job by the server. The space for the reservation_identifier string is allocated by pbs_submitresv() and should be released via a call to free() by the user when no longer needed.

SEE ALSO

pbs_rsub(1B) and pbs_connect(3B)

DIAGNOSTICS

When the batch request generated by pbs_submitresv() function has been completed successfully by a batch server, the routine will return a pointer to a character string which is the job identifier of the submitted batch job. Otherwise, a null pointer is returned and the error code is set in pbs_error.

Local pbs_submitresv(3B)

NAME

pbs_terminate - terminate a pbs batch server

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

int pbs_terminate(int connect, int manner, char *extend)
```

DESCRIPTION

Issue a batch request to shut down a batch server. This request requires the privilege level usually reserved for batch operators and administrators.

A Server Shutdown batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

The parameter, manner, specifies the manner in which the server is shut down. The available manners are defined in pbs_ifl.h as:

#define SHUT_IMMEDIATE 0

Shutdown is to be immediate, running jobs are checkpointed, requeued, or deleted as required.

#define SHUT_DELAY 1

Jobs which can be checkpointed are checkpointed, terminated, and requeued. Jobs which cannot be checkpointed but are rerunnable are terminated and requeued. Shutdown is delayed until the remaining running jobs complete. No new jobs will be started by the server.

#define SHUT_QUICK 2

Shutdown of the server occurs as soon as the server can record latest state. Jobs which are currently running, are left in the Running state.

The server will not respond to the batch request until the server has completed its termination procedure.

Chapter 5
RPP Library

This chapter discusses the Reliable Packet Protocol (RPP) used by PBS. These functions provide reliable, flow-controlled, two-way transmission of data. Each data path will be called a "stream" in this document. The advantage of RPP over TCP is that many streams can be multiplexed over one socket. This allows simultaneous connections over many streams without regard to the system imposed file descriptor limit.

5.1 RPP Library Routines

The following manual pages document the application programming interface provided by the RPP library.

NAME

rpp_open, rpp_bind, rpp_poll, rpp_io, rpp_read, rpp_write, rpp_close, rpp_getaddr, rpp_flush, rpp_terminate, rpp_shutdown, rpp_rcommit, rpp_wcommit, rpp_eom, rpp_getc, rpp_putc - reliable packet protocol

SYNOPSIS

```
#include <sys/types.h>
#include <netinet/in.h>
#include <rpp.h>

int rpp_open(addr)
struct sockadd_in *addr;

int rpp_bind(port)
int port;

int rpp_poll()

int rpp_io()

int rpp_read(stream, buf, len)
u_int stream;
char *buf;
int len;

int rpp_write(stream, buf, len)
u_int stream;
char *buf;
int len;

int rpp_close(stream)
u_int stream;

struct sockadd_in *rpp_getaddr(stream)
u_int stream;

int rpp_flush(stream)
u_int stream;

int rpp_terminate()
int rpp_shutdown()

int rpp_rcommit(stream, flag)
u_int stream;
int flag;

int rpp_wcommit(stream, flag)
u_int stream;
int flag;

int rpp_eom(stream)
u_int stream;

int rpp_getc(stream)
u_int stream;

int rpp_putc(stream, c)
u_int stream;
int c;
```

DESCRIPTION

These functions provide reliable, flow-controlled, two-way transmission of data. Each data path will be called a "stream" in this document. The advantage of RPP over TCP is that many streams can be multiplexed over one socket. This allows simultaneous connections over many streams without regard to the system imposed file descriptor limit.

Data is sent and received in "messages". A message may be of any length and is either received completely or not at all. Long messages will cause the library to use large amounts of memory in the heap by calling malloc(3V).

In order to use any of the above with Windows, initialize the network library and link with winsock2. To do this, call winsock_init() before calling the function and link against the ws2_32.lib library.

rpp_open() initializes a new stream connection to addr and returns the stream identifier. This is an integer with a value greater than or equal to zero. A negative number indicates an error. In this case, errno will be set.

rpp_bind() is an initialization call which is used to bind the UDP socket used by RPP to a particular port. The file descriptor of the UDP socket used by the library is returned.

rpp_poll() returns the stream identifier of a stream with data to read. If no stream is ready to read, a -2 is returned. A -1 is returned if an error occurs.

rpp_io() processes any packets which are waiting to be sent or received over the UDP socket. This routine should be called if a section of code could be executing for more than a few (~10) seconds without calling any other rpp function. A -1 is returned if an error occurs, 0 otherwise.

rpp_read() transfers up to len characters of a message from stream into buf. If all of a message has been read, the return value will be less than len. The return value could be zero if all of a message had previously been read. A -1 is returned on error. A -2 is returned if the peer has closed its connection. If rpp_poll() is used to determine the stream is ready for reading, the call to rpp_read() will return immediately. Otherwise, the call will block waiting for a message to arrive.

rpp_write() adds information to the current message on a stream. The data in buf numbering len characters is transfered to the stream. The number of characters added to the stream are returned or a -1 on error. In this case, errno will be set. A -2 is returned if the peer has closed its connection.

rpp_close() disconnects the stream from its peer and frees all resources associated with the stream. The return value is -1 on error and 0 otherwise.

rpp_getaddr() returns the address which a stream is connected to. If the stream is not open, a NULL pointer is returned.

rpp_flush() marks the end of a message and commits all the data which has been written to the specified stream. A zero is returned if the message has been successfully committed. A -1 is returned on error.

rpp_terminate() is used to free all memory associated with all streams and close the UDP socket. This is done without attempting to send any final messages that may be waiting. If a process is using rpp and calls fork() , the child must call rpp_terminate() so it will not cause a conflict with the parent's communication.

rpp_shutdown() is used to free all memory associated with all streams and close the UDP socket. An attempt is made to send all outstanding messages before returning.

rpp_rcommit() is used to "commit" or "de-commit" the information read from a message. As calls are made to rpp_read(), the number of characters transfered out of the message are counted. If rpp_rcommit() is called with flag being non-zero (TRUE), the current position in the message is marked as the commit point. If rpp_rcommit() is called with flag being zero (FALSE), a subsequent call to rpp_read() will return characters from the message following the last commit point. If an entire message has been read, rpp_read() will continue to return zero as the number of bytes transfered until rpp_eom() is called to commit the complete message.

rpp_wcommit() is used to "commit" or "de-commit" the information written to a stream. As calls are made to rpp_write(), the number of characters transfered into the message are counted. If rpp_wcommit() is called with flag being non-zero (TRUE), the current position in the message is marked as the commit point. If rpp_wcommit() is called with flag being zero (FALSE), a subsequent call to rpp_write() will transfer characters into the stream following the last commit point. A call to rpp_flush() does an automatic write commit to the current position.

rpp_eom() is called to terminate processing of the current message.

SEE ALSO

tcp(4P), udp(4P)

Chapter 6 TM Library

This chapter describes the PBS Task Management library. The TM library is a set of routines used to manage multi-process, parallel, and distributed applications. The current version is an implementation of the proposed (draft) PSCHED standard sponsored by NASA. Altair has since submitted this draft to the DRAMA working group of the international Global Grid Forum standards body.

6.1 TM Library Routines

The following “manual” pages document the application programming interface provided by the TM library.

NAME

tm_init, tm_nodeinfo, tm_poll, tm_notify, tm_spawn, tm_kill, tm_obit, tm_taskinfo, tm_atnode, tm_rescinfo, tm_publish, tm_subscribe, tm_finalize, tm_attach - task management API

SYNOPSIS

```
#include <tm.h>

int tm_init(info, roots)
      void *info;
      struct tm_roots *roots;

int tm_nodeinfo(list, nnodes)
      tm_node_id **list;
      int *nnodes;

int tm_poll(poll_event, result_event, wait, tm_errno)
      tm_event_t poll_event;
      tm_event_t *result_event;
      int wait;
      int *tm_errno;

int tm_notify(tm_signal)
      int tm_signal;

int tm_spawn(argc, argv, envp, where, tid, event)
      int argc;
      char **argv;
      char **envp;
      tm_node_id where;
      tm_task_id *tid;
      tm_event_t *event;

int tm_kill(tid, sig, event)
      tm_task_id tid;
      int sig;
      tm_event_t *event;

int tm_obit(tid, obitval, event)
      tm_task_id tid;
      int *obitval;
```

```
    tm_event_t *event;

int tm_taskinfo(node, tid_list, list_size, ntasks, event)
    tm_node_id node;
    tm_task_id *tid_list;
    int list_size;
    int *ntasks;
    tm_event_t *event;

int tm_atnode(tid, node)
    tm_task_id tid;
    tm_node_id *node;

int tm_rescinfo(node, resource, len, event)
    tm_node_id node;
    char *resource;
    int len;
    tm_event_t *event;

int tm_publish(name, info, len, event)
    char *name;
    void *info;
    int len;
    tm_event_t *event;

int tm_subscribe(tid, name, info, len, info_len, event)
    tm_task_id tid;
    char *name;
    void *info;
    int len;
    int *info_len;
    tm_event_t *event;

int tm_attach(jobid, cookie, pid, tid, host, port)
    char *jobid;
    char *cookie;
    pid_t pid;
    tm_task_id *tid;
    char *host;
    int port;
int tm_finalize()
```

DESCRIPTION

These functions provide a partial implementation of the task management interface part of the PSCHED API. In PBS, MOM provides the task manager functions. This library opens a tcp socket to the MOM running on the local host and sends and receives messages using the DIS protocol (described in the PBS IDS). The tm interface can only be used by a process within a PBS job.

The PSCHED Task Management API description used to create this library was committed to paper on November 15, 1996 and was given the version number 0.1. Changes may have taken place since that time which are not reflected in this library.

The API description uses several data types that it purposefully does not define. This was done so an implementation would not be confined in the way it was written. For this specific work, the definitions follow:

```
typedef   int           tm_node_id;    /* job-relative node id */
#define   TM_ERROR_NODE  ((tm_node_id)-1)

typedef   int           tm_event_t;    /* > 0 for real events */
#define   TM_NULL_EVENT  ((tm_event_t)0)
#define   TM_ERROR_EVENT ((tm_event_t)-1)

typedef   unsigned long  tm_task_id;
#define   TM_NULL_TASK   (tm_task_id)0
```

There are a number of error values defined as well: TM_SUCCESS, TM_ESYSTEM, TM_ENOEVENT, TM_ENOTCONNECTED, TM_EUNKNOWNCMD, TM_ENOTIMPLEMENTED, TM_EBADENVIRONMENT, TM_ENOTFOUND.

In order to use any of the above with Windows, initialize the network library and link with winsock2. To do this, call winsock_init() before calling the function and link against the ws2_32.lib library.

tm_init() initializes the library by opening a socket to the MOM on the local host and sending a TM_INIT message, then waiting for the reply. The info parameter has no use and is included to conform with the PSCHED document. The roots pointer will contain valid data after the function returns and has the following structure:

```
struct    tm_roots {
    tm_task_id     tm_me;
    tm_task_id     tm_parent;
    int       tm_nnodes;
    int       tm_ntasks;
    int       tm_taskpoolid;
    tm_task_id     *tm_tasklist;
};
```

tm_me	The task id of this calling task.
tm_parent	The task id of the task which spawned this task or TM_NULL_TASK if the calling task is the initial task started by PBS.
tm_nnodes	The number of nodes allocated to the job.
tm_ntasks	This will always be 0 for PBS.
tm_taskpoolid	PBS does not support task pools so this will always be -1.
tm_tasklist	This will be NULL for PBS.

The tm_ntasks, tm_taskpoolid and tm_tasklist fields are not filled with data specified by the PSCHED document. PBS does not support task pools and, at this time, does not return information about current running tasks from tm_init. There is a separate call to get information for current running tasks called tm_taskinfo which is described below. The return value from tm_init is TM_SUCCESS if the library initialization was successful, or an error is returned otherwise.

tm_nodeinfo() places a pointer to a malloc’ed array of tm_node_id’s in the pointer pointed at by list. The order of the tm_node_id's in list is the same as that specified to MOM in the "exec_host" attribute. The int pointed to by nnodes contains the number of nodes allocated to the job. This is information that is returned during initialization and does not require communication with MOM. If tm_init has not been called, TM_ESYSTEM is returned, otherwise TM_SUCCESS is returned.

tm_poll() is the function which will retrieve information about the task management system to locations specified when other routines request an action take place. The bookkeeping for this is done by generating an event for each action. When the task manager (MOM) sends a message that an action is complete, the event is reported by tm_poll and information is placed where the caller requested it. The argument poll_event is meant to be used to request a specific event. This implementation does not use it and it must be set to TM_NULL_EVENT or an error is returned. Upon return, the argument result_event will contain a valid event number or TM_ERROR_EVENT on error. If wait is zero and there are no events to report, result_event is set to TM_NULL_EVENT. If wait is non-zero an there are no events to report, the function will block waiting for an event. If no local error takes place, TM_SUCCESS is returned. If an error is reported by MOM for an event, then the argument tm_errno will be set to an error code.

tm_notify() is described in the PSCHED documentation, but is not implemented for PBS yet. It will return TM_ENOTIMPLEMENTED.

tm_spawn() sends a message to MOM to start a new task. The node id of the host to run the task is given by where. The parameters argc, argv and envp specify the program to run and its arguments and environment very much like exec(). The full path of the program executable must be given by argv[0] and the number of elements in the argv array is given by argc. The array envp is NULL terminated. The argument event points to a tm_event_t variable which is filled in with an event number. When this event is returned by tm_poll , the tm_task_id pointed to by tid will contain the task id of the newly created task.

tm_kill() sends a signal specified by sig to the task tid and puts an event number in the tm_event_t pointed to by event.

tm_obit() creates an event which will be reported when the task tid exits. The int pointed to by obitval will contain the exit value of the task when the event is reported.

tm_taskinfo() returns the list of tasks running on the node specified by node. The PSCHED documentation mentions a special ability to retrieve all tasks running in the job. This is not supported by PBS. The argument tid_list points to an array of tm_task_id's which contains list_size elements. Upon return, event will contain an event number. When this event is polled, the int pointed to by ntasks will contain the number of tasks running on the node and the array will be filled in with tm_task_id's. If ntasks is greater than list_size, only list_size tasks will be returned.

tm_atnode() will place the node id where the task tid exists in the tm_node_id pointed to by node.

tm_rescinfo() makes a request for a string specifying the resources available on a node given by the argument node. The string is returned in the buffer pointed to by resource and is terminated by a NUL character unless the number of characters of information is greater than specified by len. The resource string PBS returns is formated as follows:

A space separated set of strings from the uname system call. The order of the strings is sysname, nodename, release, version, machine.

A comma separated set of strings giving the components of the "Resource_List" attribute of the job, preceded by a colon (:). Each component has the resource name, an equal sign, and the limit value.

For example, a return for a task running on an SGI workstation might look like:

IRIX golum 6.2 03131015 IP22:cput=20:00,mem=400kb

tm_publish() causes len bytes of information pointed at by info to be sent to the local MOM to be saved under the name given by name.

tm_subscribe() returns a copy of the information named by name for the task given by tid. The argument info points to a buffer of size len where the information will be returned. The argument info_len will be set with the size of the published data. If this is larger than the supplied buffer, the data will have been truncated.

tm_attach() commands MOM to create a new PBS “attached task” out of a session running on MOM’s host. The jobid parameter specifies the job which is to have a new task attached. If it is NULL, the system will try to determine the correct jobid. The cookie parameter must be NULL. The pid parameter must be a non-zero process id for the process which is to be added to the job specified by jobid. If tid is non-NULL, it will be used to store the task id of the new task. The host and port parameters specify where to contact MOM. host should be NULL. The return value will be 0 if a new task has been successfully created and non-zero on error. The return value will be one of the TM error numbers defined in tm.h as follows:

TM_ESYSTEM	MOM cannot be contacted
TM_ENOTFOUND	No matching job was found
TM_ENOTIMPLEMENTED	The call is not implemented/supported
TM_ESESSION	The session specified is already attached
TM_EUSER	The calling user is not permitted to attach
TM_EOWNER	The process owner does not match the job
TM_ENOPROC	The process does not exist

tm_finalize() may be called to free any memory in use by the library and close the connection to MOM.

SEE ALSO
pbs_mom(8B), pbs_sched(8B)

Chapter 7
RM Library

This chapter describes the PBS Resource Monitor library. The RM library contains functions to facilitate communication with the PBS Professional resource monitor. It is set up to make it easy to connect to several resource monitors and handle the network communication efficiently.

7.1 RM Library Routines

The following “manual” pages document the application programming interface provided by the RM library.

NAME

openrm, closerm, downrm, configrm, addreq, allreq, getreq, flushreq, activereq, fullresp - resource monitor API

SYNOPSIS

```
#include <sys/types.h>
#include <netinet/in.h>
#include <rm.h>

int openrm (host, port)
char *host;
unsigned int port;

int closerm (stream)
int stream;

int downrm (stream)
int stream;

int configrm (stream, file)
int stream;
char *file;

int addreq (stream, line)
int stream;
char *line;

int allreq (line)
char *line;

char *getreq(stream)
int stream;

int flushreq()

int activereq()

void fullresp(flag)
int flag;
```

DESCRIPTION

The resource monitor library contains functions to facilitate communication with the PBS Professional resource monitor. It is set up to make it easy to connect to several resource monitors and handle the network communication efficiently.

In all these routines, the variable pbs_errno will be set when an error is indicated. The lower levels of network protocol are handled by the "Data Is Strings" DIS library and the "Reliable Packet Protocol" RPP library.

configrm() causes the resource monitor to read the file named.

addreq() begins a new message to the resource monitor if necessary. Then adds a line to the body of an outstanding command to the resource monitor.

allreq() begins, for each stream, a new message to the resource monitor if necessary. Then adds a line to the body of an outstanding command to the resource monitor.

getreq() finishes and sends any outstanding message to the resource monitor. If fullresp() has been called to turn off "full response" mode, the routine searches down the line to find the equal sign just before the response value. The returned string (if it is not NULL) has been allocated by malloc and thus free must be called when it is no longer needed to prevent memory leaks.

flushreq() finishes and sends any outstanding messages to all resource monitors. For each active resource monitor structure, it checks if any outstanding data is waiting to be sent. If there is, it is sent and the internal structure is marked to show "waiting for response".

fullresp() turns on, if flag is true, "full response" mode where getreq() returns a pointer to the beginning of a line of response.

This is the default. If flag is false, the line returned by getreq() is just the answer following the equal sign.

activereq() Returns the stream number of the next stream with something to read or a negative number (the return from rpp_poll) if there is no stream to read.

In order to use any of the above with Windows, initialize the network library and link with winsock2. To do this, call winsock_init() before calling the function and link against the ws2_32.lib library.

SEE ALSO
rpp(3B), tcp(4P), udp(4P)

Local 12 July 2006 RM(3)

Chapter 8

TCL/tk Interface

The PBS Professional software includes a TCL/tk interface to PBS. Wrapped versions of many of the API calls are compiled into a special version of the TCL shell, called `pbs_tclsh`. (A special version of the tk window shell is also provided, called `pbs_wish`.). This chapter documents the TCL/tk interface to PBS.

The `pbs_tclapi` is a subset of the PBS external API wrapped in a TCL library. This functionality allows the creation of scripts that query the PBS system. Specifically, it permits the user to query the `pbs_server` about the state of PBS, jobs, queues, and nodes, and communicate with `pbs_mom` to get information about the status of running jobs, available resources on nodes, etc.

8.1 TCL/tk API Functions

A set of functions to communicate with the PBS Server and resource monitor have been added to those normally available with Tcl. All these calls will set the Tcl variable `pbs_errno` to a value to indicate if an error occurred. In all cases, the value "0" means no error. If a call to a Resource Monitor function is made, any error value will come from the system supplied `errno` variable. If the function call communicates with the PBS Server, any error value will come from the error number returned by the Server. This is the same TCL interface used by the `pbs_tclsh` and `pbs_wish` commands.

Note that the pbs_tclapi pbsrescquery command, which calls the C API pbs_rescquery, is deprecated. Any attempt to use it will result in a PBSE_NOSUPPORT error being returned.

NAME

pbs_tclapi - PBS TCL Application Programming Interface

DESCRIPTION

The pbs_tclapi is a subset of the PBS external API wrapped in a TCL library. This functionality allows the creation of scripts that query the PBS system. Specifically, it permits the user to query the pbs_server about the state of PBS, jobs, queues, and nodes, and communicate with pbs_mom to get information about the status of running jobs, available resources on nodes, etc.

USAGE

A set of functions to communicate with the PBS server and resource monitor have been added to those normally available with Tcl. All these calls will set the Tcl variable "pbs_errno" to a value to indicate if an error occured. In all cases, the value "0" means no error. If a call to a Resource Monitor function is made, any error value will come from the system supplied errno variable. If the function call communicates with the PBS Server, any error value will come from the error number returned by the server. This is the same TCL interface used by the pbs_tclsh and pbs_wish commands.

openrm host ?port?

Creates a connection to the PBS Resource Monitor on host using port as the port number or the standard port for the resource monitor if it is not given. A connection handle is returned. If the open is successful, this will be a non-negative integer. If not, an error occurred.

closerm connection

The parameter connection is a handle to a resource monitor which was previously returned from openrm. This connection is closed. Nothing is returned.

downrm connection

Sends a command to the connected resource monitor to shutdown. Nothing is returned.

configrm connection filename

Sends a command to the connected resource monitor to read the configuration file given by filename. If this is successful, a "0" is returned, otherwise, "-1" is returned.

addreq connection request

A resource request is sent to the connected resource monitor. If this is successful, a "0" is returned, otherwise, "-1" is returned.

getreq connection

One resource request response from the connected resource monitor is returned. If an error occurred or there are no more responses, an empty string is returned.

allreq request

A resource request is sent to all connected resource monitors. The number of streams acted upon is returned.

flushreq

All resource requests previously sent to all connected resource monitors are flushed out to the network. Nothing is returned.

activereq

The connection number of the next stream with something to read is returned. If there is nothing to read from any of the connections, a negative number is returned.

fullresp flag

Evaluates flag as a boolean value and sets the response mode used by getreq to full if flag evaluates to "true". The full return from a resource monitor includes the original request followed by an equal sign followed by the response. The default situation is only to return the response following the equal sign. If a script needs to "see" the entire line, this function may be used.

pbsstatserv

The server is sent a status request for information about the server itself. If the request succeeds, a list with three elements is returned, otherwise an empty string is returned. The first element is the server's name. The second is a list of attributes. The third is the "text" associated with the server (usually blank).

pbsstatjob

The server is sent a status request for information about the all jobs resident within the server. If the request succeeds, a list is returned, otherwise an empty string is returned. The list contains an entry for each job. Each element is a list with three elements. The first is the job's jobid. The second is a list of attributes. The attribute names which specify resources will have a name of the form "Resource_List:name" where "name" is the resource name. The third is the "text" associated with the job (usually blank).

pbsstatque

The server is sent a status request for information about all queues resident within the server. If the request succeeds, a list is returned, otherwise an empty string is returned. The list contains an entry for each queue. Each element is a list with three elements. This first is the queue's name. The second is a list of attributes similar to pbsstatjob. The third is the "text" associated with the queue (usually blank).

pbsstatnode

The server is sent a status request for information about all nodes defined within the server. If the request succeeds, a list is returned, otherwise an empty string is returned. The list contains an entry for each node. Each element is a list with three elements. This first is the node's name. The second is a list of attributes similar to pbsstatjob. The third is the "text" associated with the node (usually blank).

pbsselstat

The server is sent a status request for information about the all runnable jobs resident within the server. If the request succeeds, a list similar to pbsstatjob is returned, otherwise an empty string is returned.

pbsrunjob jobid ?location?

Run the job given by jobid at the location given by location. If location is not given, the default location is used. If this is successful, a "0" is returned, otherwise, "-1" is returned.

pbsasyrunjob jobid ?location?

Run the job given by jobid at the location given by location without waiting for a positive response that the job has actually started. If location is not given, the default location is used. If this is successful, a "0" is returned, otherwise, "-1" is returned.

pbsrerunjob jobid

Re-runs the job given by jobid. If this is successful, a "0" is returned, otherwise, "-1" is returned.

pbsdeljob jobid

Delete the job given by jobid. If this is successful, a "0" is returned, otherwise, "-1" is returned.

pbsholdjob jobid

Place a hold on the job given by jobid. If this is successful, a "0" is returned, otherwise, "-1" is returned.

pbsmovejob jobid ?location?

Move the job given by jobid to the location given by location. If location is not given, the default location is used. If this is successful, a “0” is returned, otherwise, “-1” is returned.

pbsqenable queue

Set the “enabled” attribute for the queue given by queue to true. If this is successful, a “0” is returned, otherwise, “-1” is returned.

pbsqdisable queue

Set the “enabled” attribute for the queue given by queue to false. If this is successful, a “0” is returned, otherwise, “-1” is returned.

pbsqstart queue

Set the “started” attribute for the queue given by queue to true. If this is successful, a “0” is returned, otherwise, “-1” is returned.

pbsqstop queue

Set the “started” attribute for the queue given by queue to false. If this is successful, a “0” is returned, otherwise, “-1” is returned.

pbsalterjob jobid attribute_list

Alter the attributes for a job specified by jobid. The parameter attribute_list is the list of attributes to be altered. There can be more than one. Each attribute consists of a list of three elements. The first is the name, the second the resource and the third is the new value. If the alter is successful, a “0” is returned, otherwise, “-1” is returned.

pbsrescquery resource_list

Deprecated. Obtain information about the resources specified by resource_list. This will be a list of strings. If the request succeeds, a list with the same number of elements as resource_list is returned. Each element in this list will be a list with four numbers. The numbers specify available, allocated, reserved, and down in that order.

pbsrescreserve resource_id resource_list

Make (or extend) a reservation for the resources specified by resource_list which will be given as a list of strings. The parameter resource_id is a number which provides a unique identifier for a reservation being tracked by the server. If resource_id is given as “0”, a new reservation is created. In this case, a new identifier is generated and returned by the function. If an old identifier is used, that same number will be returned. The Tcl variable “pbs_errno” will be set to indicate the success or failure of the reservation.

pbsrescrelease resource_id

The reservation specified by resource_id is released.

The two following commands are not normally used by the scheduler. They are included here because there could be a need for a scheduler to contact a server other than the one which it normally communicates with. Also, these commands are used by the Tcl tools.

pbsconnect ?server?

Make a connection to the named server or the default server if a parameter is not given. Only one connection to a server is allowed at any one time.

pbsdisconnect

Disconnect from the currently connected server.

The above Tcl functions use PBS interface library calls for communication with the server and the PBS resource monitor library to communicate with pbs_mom.

datetime ?day? ?time?

The number of arguments used determine the type of date to be calculated. With no arguments, the current POSIX date is returned. This is an integer in seconds.

With one argument there are two possible formats. The first is a 12 (or more) character string specifying a complete date in the following format:
YYMMDDhhmmss

All characters must be digits. The year (YY) is given by the first two (or more) characters and is the number of years since 1900. The month (MM) is the number of the month [01-12]. The day (DD) is the day of the month [01-32]. The hour (hh) is the hour of the day [00-23]. The minute (mm) is minutes after the hour [00-59]. The second (ss) is seconds after the minute [00-59]. The POSIX date for the given date/time is returned.

The second option with one argument is a relative time. The format for this is
HH:MM:SS

With hours (HH), minutes (MM) and seconds (SS) being separated by colons ":". The number returned in this case will be the number of seconds in the interval specified, not an absolute POSIX date.

With two arguments a relative date is calculated. The first argument specifies a day of the week and must be one of the following strings: "Sun", "Mon", "Tue", "Wed", "Thr", "Fri", or "Sat". The second argument is a relative time as given above. The POSIX date calculated will be the day of the week given which follows the current day, and the time given in the second argument. For example, if the current day was Monday, and the two arguments were "Fri" and "04:30:00", the date calculated would be the POSIX date for the Friday following the current Monday, at four-thirty in the morning. If the day specified and the current day are the same, the current day is used, not the day one week later.

strftime format time

This function calls the POSIX function strftime(). It requires two arguments. The first is a format string. The format conventions are the same as those for the POSIX function strftime(). The second argument is POSIX calendar time in second as returned by datetime. It returns a string based on the format given. This gives the ability to extract information about a time, or format it for printing.

logmsg tag message

This function calls the internal PBS function log_err(). It will cause a log message to be written to the scheduler's log file. The tag specifies a function name or other word used to identify the area where the message is generated. The message is the string to be logged.

SEE ALSO

pbs_tclsh(8B), pbs_wish(8B), pbs_mom(8B), pbs_server(8B), pbs_sched(8B)

Chapter 9
User Commands

Man pages for PBS Professional user commands are listed below.

NAME

nqs2pbs - convert NQS job scripts to PBS

SYNOPSIS

nqs2pbs nqs_script [pbs_script]
nqs2pbs --version

DESCRIPTION

This utility converts an existing NQS job script to work with PBS and NQS. The existing script is copied and PBS directives, #PBS , are inserted prior to each NQS directive #QSUB or #@$, in the original script.

Certain NQS date specification and options are not supported by PBS. A warning message will be displayed indicating the problem and the line of the script on which it occurred.

If any unrecognizable NQS directives are encountered, an error message is displayed. The new PBS script will be deleted if any errors occur.

OPTIONS

--version The nqs2pbs command returns its PBS version information and exits. This option must be used alone.

OPERANDS

nqs_script

Specifies the file name of the NQS script to convert. This file is not changed.

pbs_script

If specified, it is the name of the new PBS script. If not specified, the new file name is nqs_script.new .

NOTES

Converting NQS date specifications to the PBS form may result in a warning message and an incompleted converted date. PBS does not support date specifications of “today”, “tomorrow”, or the name of the days of the week such as “Monday”. If any of these are encountered in a script, the PBS specification will contain only the time portion of the NQS specification, i.e. #PBS -a hhmm[.ss]. It is suggested that you specify the execution time on the qsub command line rather than in the script.

Note that PBS will interpret a time specification without a date in the following way:

- If the time specified has not yet been reached, the job will become eligible to run at that time today.

- If the specified time has already passed when the job is submitted, the job will become eligible to run at that time tomorrow.

PBS does not support time zone identifiers. All times are taken as local time.

SEE ALSO

qsub(1B)

NAME

pbs - about the Portable Batch System

DESCRIPTION

PBS stands for "Portable Batch System." It is a networked subsystem for submitting, monitoring, and controlling a workload of batch jobs on one or more systems. More information about PBS is available in the PBS Professional User's Guide and PBS Professional Administrator's Guide.

Batch means that the job will be scheduled for execution at a time chosen by the subsystem according to a defined policy and the availability of resources. For a normal batch job, the standard output and standard error of the job will be returned to files available to the user when the job is complete. This differs from an interactive session where commands are executed when entered via the terminal and output is returned directly to the terminal. PBS also supports an interactive batch mode where the input and output is connected to the user's terminal, but the scheduling of the job is still under control of the batch system.

A job is typically run by submitting a shell script which specifies resources to be used and attributes for the job. A job does not have to be submitted on the system where it will run. It can be submitted on any system with the PBS commands and access to the execution system; see qsub(1B). Output will be returned to the system from which the job was submitted unless directed otherwise.

Attributes offer control over when a job is eligible to be run, what happens to the output when it is completed and how the user is notified when it completes. The attributes of the job may be specified on the command line or in the job script when the job is submitted. For information about job attributes, see qsub(1B) and pbs_job_attributes(7B).

One important attribute is the resource list. The resource_list specifies the amount and type of resources needed by the job in order to execute. The list also implies a hard upper limit on usage of those resources. When the limit is reached, the job is terminated. The types of resources available to a job vary with the system architecture. For a list of resources supported on the default system, see pbs_resources(7B).

Once a job has been submitted, it may be monitored by use of the qstat(1B) command. Two forms of output are available with the qstat command. The default form is the short display. Information about a job is limited to a single line. Complete information about the job or jobs is available through qstat with the -f option. Information will be given about all jobs in the system, all jobs in specified queues, or only specified jobs.

When displaying status of jobs, you will see in which queue the job resides. In PBS a queue is just a collection point for jobs, it does not imply any execution ordering. That ordering is determined by a scheduling policy implemented by the system administration.

Other commands of interest which have man pages of their own are:

qalter	Alter a job’s attributes.
qdel	Delete a job.
qhold	Place a hold on a job to keep it from being scheduled for running.
qmove	Move a job to a different queue or server.
qmsg	Append a message to the output of an executing job.
qrerun	Terminate an executing job and return it to a queue.
qrls	Remove a hold from a job.
qselect	Obtain a list of jobs that met certain criteria.
qsig	Send a signal to an executing job.

SEE ALSO

The PBS Professional User’s Guide, PBS Professional Administrator’s Guide, qalter(1B), qdel(1B), qhold(1B), qmove(1B), qmsg(1B), qrerun(1B), qrls(1B), qselect(1B), qsig(1B), qsub(1B), qstat(1B), pbs_resources(7B), pbs_job_attributes(7B)

Local 23 June 2005 pbs(1B)

NAME

pbs_rdel - delete a PBS advance reservation

SYNOPSIS

pbs_rdel reservation_identifier ...
pbs_rdel --version

DESCRIPTION

The pbs_rdel command deletes reservations in the order in which their reservation identifiers are presented to the command.

A reservation may be deleted by its owner, the batch operator, or the batch administrator.

OPTIONS

--version

The pbs_rdel command returns its PBS version information and exits. This option can only be used alone.

OPERANDS

The pbs_rdel command accepts one or more reservation_identifier operands of the form:

[R]sequence_number[.server_name][@server]

EXIT STATUS

Upon successful processing of all the operands presented to the the pbs_rdel command, the exit status will be a value of zero.

If the pbs_rdel command fails to process any operand, the command exits with a value greater than zero.

SEE ALSO

The PBS Professional User's Guide, the PBS Professional Administrator's Guide, pbs_rsub(1B) pbs_rstat(1B) pbs_resv_attributes(7B)

NAME

pbs_renew - renew Kerberos credential

SYNOPSIS

pbs_renew [-d] program [arg(s)]
pbs_renew --version

DESCRIPTION

The pbs_renew command is used internally by PBS when a job has a Kerberos credential. The program is run as a child process with any arguments passed to the command line of program. The pbs_renew process runs periodicals to renew any Kerberos credential. It will wait for the child process to return, clean up any Kerberos credential and exit when the child process is done.

OPTIONS

-d Debug messages are printed to stderr.

--version
The pbs_renew command returns its PBS version information and exits. This option can only be used alone.

SEE ALSO

The PBS Professional Administrator's Guide, qsub(1B)

Local 12 April 2007 pbs_renew(1B)

NAME

pbs_rstat - show status of PBS advance reservations

SYNOPSIS

pbs_rstat [-F][-B][-S] [reservation_id...]
pbs_rstat --version

DESCRIPTION

The pbs_rstat command is used to show the status of all the reservations on the PBS Server. There are three different output formats. The brief form just shows the identifiers of all the reservations. The short form (default) shows the status of the reservations in a short concise form. Lastly there is the long form which prints out all the reservations and all of their attributes. See the pbs_resv_attributes(7B) man page for attribute information.

OPTIONS

-B The brief option will only show the identifiers of all the reservations

-S This short option will show all the reservations in a short concise form. The information provided is the identifier of the reservation, name of the queue belonging to the reservation, user who owns the reservation, the state, the start time, duration in seconds, and the end time.

-F The full option will print out the name of the reservation followed by all the attributes of the reservation.

--version The pbs_rstat command returns its PBS version information and exits. This option can only be used alone.

OPERANDS

The pbs_rstat command accepts one or more reservation_identifier operands of the form:

[R]sequence_number[.server_name][@server]

SEE ALSO

The PBS Professional User’s Guide, the PBS Professional Administrator’s Guide,
pbs_rsub(1B), pbs_rdel(1B), pbs_resv_attributes(7B)

NAME

pbs_rsub - create a PBS advance reservation

SYNOPSIS

```
pbs_rsub [-D duration] [-E end_time] [-g group_list]
        [-G auth_group_list] [-H auth_host_list] [-I seconds]
        [-m mail_points] [-M mail_list] [-N reservation_name]
        [-q destination] [-R start_time] [-u user_list]
        [-U auth_user_list] [-W attribute_value_list]
        -l resource_request [-l placement]
pbs_rsub --version
```

DESCRIPTION

The pbs_rsub command is used to create an advance reservation, which reserves specific resources for the requested time period. The reservation must be confirmed by PBS to be usable. PBS creates the reservation and its associated queue. Then users who are allowed to use this reservation can submit jobs to the queue via qsub and qmove. Although a confirmed reservation will accept jobs at any time, jobs in its queue can run only during the reservation period. When the reservation period ends, all of the jobs in its queue are deleted regardless of their state. To check whether a reservation is confirmed, use pbs_rstat.

The pbs_rsub command returns the reservation name, which is in the form

RNNNN.server,

where NNNN is an integer. The associated queue's name is the prefix,

RNNNN.

When using pbs_rsub to request a reservation, the user must specify two of the following options: -R, -E, and -D. The resource request -l walltime can be used instead of the -D option. The pbs_rdel command is used to delete a reservation. Do not use qdel.

OPTIONS

-D duration

Specifies reservation duration. Duration can be expressed either as a total number of seconds of walltime, or as a colon-delimited timestring, e.g. HH:MM:SS or MM:SS. If the start time and end time are the only times specified, this duration time is calculated. Format: integer or string.

-E end_time

Specifies the reservation end time. If start time and duration are the only times specified, the end time value is calculated. See Datetime Format for a description of the datetime string. Format: datetime.

-g group_list

The group_list is a comma-separated list of entries of the form: group@host names. Entries on this list are used by the server in conjunction with an ordered set of rules to associate a group name with the reservation. Refer to the attribute, Group_List, on the pbs_resv_attributes man page.

-G auth_group_list

auth_group_list is a comma-separated list of entries of the form: [+|-]group_name. Entries on this list help control the enqueuing of jobs into the reservation's queue. Jobs owned by members belonging to these groups are either allowed or denied entry into the queue. Any Group on the list is to be interpreted in the context of the server's host not the context of the host from which qsub was submitted. This list becomes the acl_groups list for the reservation's queue. Refer to the attribute, Authorized_Groups, on the pbs_resv_attributes man page.

-H auth_host_list

auth_host_list is a comma-separated list of entries of the form: [+|-]hostname. These entries help control the enqueuing of jobs into the reservation's queue by allowing or denying jobs submitted from these hosts. This list becomes the acl_hosts list for the reservation's queue. Refer to the Authorized_Hosts attribute on the pbs_resv_attributes man page.

-I block_time

Specifies interactive mode. The pbs_rsub command will block, up to block_time seconds, while waiting for the scheduler to either confirm or deny the reservation request.

If block_time is positive, and the scheduler doesn't confirm or deny the reservation in the specified time, the ID string for the reservation is returned with the status "UNCONFIRMED". The requester may periodically issue the pbs_rstat command with ID string as an argument, to monitor the reservation's status.

If block_time is negative, and the scheduler doesn't confirm or deny the reservation in the specified time, the reservation is deleted.

Type: integer.

-m mail_points

Specifies the set of events that cause the server to send mail messages to the specified list of users. This option takes a string consisting of any combination of "a", "b","c" or "e".

a	notify if the reservation is terminated for whatever reason
b	notify when the reservation period begins
e	notify when the reservation period ends
c	notify when the reservation is confirmed

Default: “ac”

-M mail_list

The list of users to whom the server will attempt to send a mail message whenever the reservation transitions to one of the mail states specified in the -m option. Default: reservation’s owner

-N reservation_name

This will declare a name for the reservation. The name specified may be up to 15 characters in length. It must consist of printable, non-white space characters with the first character alphabetic.

-q destination

Specifies the destination server to which to submit the reservation. The default server is used if this option is not selected.

-R start_time

Specifies reservation starting time. If the reservation’s end time and duration are the only times specified, this start time is calculated. Format: datetime.

If the day, DD , is not specified, it will default to today if the time hhmm is in the future. Otherwise, the day will be set to tomorrow. For example, if you submit a reservation having a specification -R 1110 at 11:15am, it will be interpreted as being for 11:10am tomorrow. If the month portion, MM , is not specified, it defaults to the current month provided that the specified day DD , is in the future. Otherwise, the month will be set to next month. Similarly comments apply to the two other optional, left hand components.

-u user_list

Comma-separated list of entries of the form: user@host. Not used. Refer to the attribute, User_List, on the pbs_resv_attributes man page.

-U auth_user_list

Comma-separated list of entries of the form: [+|-]user@host. These are the users who are allowed (denied) permission to submit jobs to the queue associated with this reservation. This list becomes the acl_users attribute for the reservation's queue. Refer to the attribute, Authorized_Users, on the pbs_resv_attributes man page.

-W attribute_value_list

This allows you to define other attributes for the reservation. Supported attributes:

qmove=jobid

Converts a normal job designated by jobid into a reservation job that will run as soon as possible. Creates the reservation with its queue and moves the job into the reservation's queue. Uses the resources requested by the job to create the reservation.

In creating the reservation, resources requested through the pbs_rsub command override existing job resources. Therefore, if the existing job resources are greater than those requested for the reservation, the job will be rejected by the reservation.

If the qmove option is used and the scheduler does not confirm the reservation within 10 seconds, the reservation is deleted. The qmove option is the same as using -I -10.

The -R and -E options to pbs_rsub are disabled when using the qmove=jobid attribute.

Note that some shells require that you enclose a job array ID in double quotes.

-l resource_request

The resource_request specifies the resources required for the reservation. These resources will be used for the limits on the queue that is dynamically created for the reservation. The aggregate amount of resources for currently running jobs from this queue will not exceed these resource limits. Jobs in the queue that request more of a resource than the queue limit for that resource are not allowed to run. Also, the queue inherits the value of any resource limit set on the server, and these are used for the job if the reservation request itself is silent about that resource.

Resources are requested by using the -l option, either in chunks inside of selection statements, or in job-wide requests using resource_name=value pairs. The selection statement is of the form:

```
-l select=[N:]chunk[+[N:]chunk ...]
```

where N specifies how many of that chunk, and a chunk is of the form:

```
resource_name=value[:resource_name=value ...]
```

Job-wide resource_name=value requests are of the form:

```
-l resource_name=value[,resource_name=value ...]
```

-l placement

The placement specifies how a job will be placed on vnodes.

The place statement has this form:

-l place=[arrangement][: sharing][: grouping]

where

arrangement is one of free | pack | scatter
sharing is one of excl | share
grouping can have only one instance of group=resource

and where

free: Place job on any vnode(s).
pack: All chunks will be taken from one host.
scatter: Only one chunk with any MPI processes will be taken from a host. A chunk with no MPI processes may be taken from the same node as another chunk.
excl: Only this job uses the vnodes chosen.
share: This job can share the vnodes chosen.
group=resource: Chunks will be grouped according to a resource. All nodes in the group must have a common value for the resource, which can be either the built-in resource host or a site-defined node-level resource.

Note that nodes can have sharing attributes that override job placement requests. See the pbs_node_attributes(7B) man page.

For more on job placement, see The PBS Professional User's Guide.

--version
The pbs_rsub command returns its PBS version information and exits. This option can only be used alone.

FORMAT
Datetime Format The datetime format adheres to the POSIX time specifi-

cation: [[[[CC]YY]MM]DD]hhmm[.SS] See the date_time operand for the touch(1) command defined by POSIX.2.

SEE ALSO

The PBS Professional User's Guide, the PBS Professional Administrator's Guide,
pbs_rstat(1B), pbs_rdel(1B), qsub(1B), qmove(1B), pbs_resources(7B), pbs_resv_attributes(7B), pbs_queue_attributes(7B), pbs_server_attributes(7B)

NAME

pbsdsh - distribute task(s) to nodes under PBS

SYNOPSIS

```
pbsdsh [-c copies] [-s] [-v] [-o] -- program [program_args]
pbsdsh [-n node_index] [-s] [-v] [-o] -- program [program_args]
pbsdsh --version
```

DESCRIPTION

The pbsdsh command executes (spawns) a normal application program on one or more nodes under control of the PBS. pbsdsh uses the Task Manager API to distribute the program on the allocated nodes.

When run without the -c or the -n option, pbsdsh will spawn the program on all nodes allocated to the PBS job. The spawns take place concurrently - all execute at (about) the same time.

Note that the double dash must come after the options and before the program and arguments. The double dash is only required for Linux.

OPTIONS

-c copies

The program is spawned copies times on the nodes allocated, one per node, unless copies is greater than the number of nodes. If this is true, it will wrap around, running multiple instances on some nodes. This option is mutually exclusive with -n.

-n node_index

The program is spawned only on the node_index -th node allocated. This option is mutually exclusive with -c.

-s The program is run in turn on each node, one after the other.

-v Produces verbose output about error conditions and task exit status.

-o No obit request is made for spawned tasks. The program will not wait for the tasks to finish.

--version
The pbsdsh command returns its PBS version information and exits. This option can only be used alone

OPERANDS
The first operand, program , is the program to execute. The double dash must precede the program under Linux.

Additional operands, program_args , are passed as arguments to the program.

STANDARD ERROR
The pbsdsh command will write a diagnostic message to standard error for each error occurrence.

SEE ALSO
The PBS Professional User’s Guide, the PBS Professional Administrator’s Guide, qsub(1B), tm(3).

Local 12 April 2007 pbsdsh(1B)

NAME

qalter - alter PBS job

SYNOPSIS

```
qalter [-a date_time] [-A account_string] [-c interval] [-e path]
    [-h hold_list] [-j join] [-k keep] [-l resource_list]
    [-m mail_options] [-M user_list] [-N name] [-o path]
    [-p priority] [-q destination] [-r c] [-S path] [-u user_list]
    [-W additional_attributes] job_identifier_list

qalter --version
```

DESCRIPTION

The qalter command is used to alter one or more PBS batch jobs. The attributes listed as options to the qalter command can be modified. If any of the modifications of a job fails, none of the job's attributes is modified.

Modifying resources and job placement

If a job is running, the only resources that can be modified are cputime and walltime. These can only be reduced.

If a job is queued, requested modifications must still fit within the queue's and server's job resource limits. If a requested modification to a resource would exceed the queue's or server's job resource limits, the resource request will be rejected.

Resources are modified by using the -l option, either in chunks inside of selection statements, or in job-wide modifications using resource_name=value pairs. The selection statement is of the form:

```
-l select=[N:]chunk[+[N:]chunk ...]
```

where N specifies how many of that chunk, and a chunk is of the form:

```
resource_name=value[:resource_name=value ...]
```

Job-wide resource_name=value modifications are of the form:

    -l resource_name=value[,resource_name=value ...]

Placement of jobs on nodes is changed using the place statement:

    -l place=modifier[:modifier]

where modifier is any combination of group, excl, and/or one of free|pack|scatter.

For more on resource requests, usage limits and job placement, see pbs_resources(7B).

Modifying attributes
The user alters job attributes by giving options to the qalter command. Each qalter option changes a job attribute.

See the PBS Professional User's Guide, pbs_job_attributes(7B).

OPTIONS

-a date_time

Changes the point in time after which the job is eligible for execution. Given in pairs of digits. Sets job's Execution_Time attribute to date_time. Format:

    [[[[CC]YY]MM]DD]hhmm[.SS]

where CC is the century, YY is the year, MM is the month, DD is the day of the month, hh is the hour, mm is the minute, and SS is the seconds.

Each portion of the date defaults to the current date, as long as the next-smaller portion is in the future. For example, if today is the 3rd of the month and the specified day DD is the 5th, the month MM will be set to the current month.

If a specified portion has already passed, the next-larger portion will be set to one after the current date. For example, if the day DD is not specified, but the hour hh is specified to be 10:00 a.m. and the current time is 11:00 a.m., the day DD will be set to tomorrow.

The job's Execution_Time attribute can be altered after the job has begun execution, in which case it will not take effect until the job is rerun.

-A account_string

Replaces the accounting string associated with the job. Used for labeling accounting data. Sets job's Account_Name attribute to account_string. Format: string.

This attribute cannot be altered once the job has begun execution.

-c interval

Changes the interval at which the job will be checkpointed. Sets job's Checkpoint attribute. Ignored if checkpointing is not supported on the execution host. Default value is -u.

The argument interval can take on one of the following values:

n No checkpointing is to be performed.

s Checkpointing is to be performed only when the server is shut down.

c Checkpointing is to be performed according to the time interval set on the server on which the job resides.

c=minutes
Checkpointing is to be performed at an interval of minutes minutes, which is the number of minutes of CPU time used by the job. Must be greater than zero. Format: integer.

u Checkpointing is to be performed only when the server is shut down.

This attribute can be altered after the job has begun execution, in which case the new value will not take effect until the job is rerun.

-e path Replaces the path to be used for the job's standard error stream. Sets job's Error_Path attribute to path. The path argument is of the form:

[hostname:]path_name

The path will be interpreted as follows:

path_name

If path_name is a relative path, then it is taken to be relative to the current working directory of the qalter command, where it is executing on the current host.

If path_name is an absolute path, then it is taken to be an absolute path on the current host where the qalter command is executing.

hostname:path_name

If path_name is a relative path, then it is taken to be relative to the user's home directory on the host named hostname.

If path_name is an absolute path, then it is the absolute path on the host named hostname.

If path_name does not include a filename, the default filename will be

jobid.ER

If the -e option is not specified, the default filename for the standard error stream is used. It has this form:

job_name.esequence_number

This attribute can be altered after the job has begun execution, in which case the new value will not take effect until the job is rerun.

-h hold_list
Updates the job's hold list. Adds hold_list to the job's Hold_Types attribute. The hold_list is a string of one or more of the following:

u Add a USER hold.

o Add OTHER hold. Requires operator privilege.

n Clear the holds for which the user has privilege.

This attribute can be altered after the job has begun execution, in which case the new value will not take effect until the job is rerun.

-j join Changes whether and how to join the job's standard error and standard output streams. Sets job's Join_Path attribute to join. Default: not merged. Possible values of join:

oe Standard error and standard output are merged into standard output.

eo Standard error and standard output are merged into standard error.

n Standard error and standard output are not merged.

This attribute can be altered after the job has begun execution, in which case the new value will not take effect until the job is rerun.

-k keep Changes whether and which of the standard output and standard error streams will be retained on the execution host. Overrides default path names for these streams. Sets the job's Keep_Files attribute to keep. Cannot be altered once job has begun execution. Default: neither is retained. The keep argument can take on the following values:

e The standard error stream is retained on the execution host, in the home directory of the job's owner. The filename will be:
    job_name.esequence_number

o The standard output stream is retained on the execution host, in the home directory of the job's owner. The filename will be:
    job_name.osequence_number
  This attribute cannot be altered once the job has begun execution.

eo, oe
  Both standard output and standard error streams are retained on the execution host, in the home directory of the job's owner.

n Neither stream is retained.

-l resource_arg

Allows the user to change requested resources and job placement. Sets job's Resource_list attribute to resource_arg. Uses resource request syntax. Requesting a resource places a limit on its usage.

Requesting resources in chunks:
  -l select=[N:]chunk[+[N:]chunk ...] where N specifies how many of that chunk, and a chunk is:
    resource_name=value[:resource_name=value ...]
Requesting job-wide resources:
  -l resource_name=value[,resource_name=value ...]
Specifying placement of jobs:
  -l place=[arrangement][: sharing][: grouping]
where
  arrangement is one of free | pack | scatter
  sharing is one of excl | shared
  grouping can have only one instance of group=resource

and where

free: Place job on any vnode(s).
pack: All chunks will be taken from one host.
scatter: Only one chunk with any MPI processes will be taken from a host. A chunk with no MPI processes may be taken from the same node as another chunk.
excl: Only this job uses the vnodes chosen.
shared: This job can share the vnodes chosen.
group=resource: Chunks will be grouped according to a resource. All nodes in the group must have a common value for the resource, which can be either the built-in resource host or a site-defined node-level resource.

If a requested modification to a resource would exceed the job's queue's limits, the resource request will be rejected. For a running job, resources may only be reduced. Which resources can be altered is system-dependent.

If the job was submitted with an explicit "-l select=", then node level resources must be qaltered using the "-l select=" form. In this case a node level resource RES cannot be qaltered with the "-l RES" form.

For example:

```
Submit the job:
% qsub -l select=1:ncpus=2:mem=512mb jobscript
Job's ID is 230

qalter the job using "-l RES" form:
% qalter -l ncpus=4 230

Error reported by qalter:
qalter: Resource must only appear in "select" specifica-
tion when select is used: ncpus 230

qalter the job using the "-l select=" form:
% qalter -l select=1:ncpus=4:mem=512mb 230

No error reported by qalter:
%
```

For more on resource requests, usage limits and job placement, see pbs_resources(7B).

-m mail_options
Changes the set of conditions under which mail about the job is sent. Format: string. Default value: "a". Sets job's Mail_Points attribute to mail_options. The mail_options argument can be either "n" or any combination of "a", "b", and "e".

n No mail will be sent.

a Mail is sent when the job is aborted by the batch system.

b Mail is sent when the job begins execution.

e Mail is sent when the job terminates.

-M user_list
Alters list of users to whom mail about the job is sent. Sets job's Mail_Users attribute to user_list. Default: job owner. The user_list argument is of the form:

```
user[@host][,user[@host],...]
```

-N name Renames the job. Sets job's Job_Name attribute to name. Format: string, up to 15 characters in length. It must consist of an alphabetic character followed by printable, non-white-space characters. Default: if a script is used to submit the job, the job's name is the name of the script. If no script is used, the job's name is "STDIN".

-o path Alters path to be used for the job's standard output stream. Sets job's Output_Path attribute to path. The path argument is of the form:

```
[hostname:]path_name
```

The path will be interpreted as follows:

path_name

If path_name is a relative path, then it is taken to be relative to the current working directory of the command, where it is executing on the current host.

If path_name is an absolute path, then it is taken to be an absolute path on the current host where the command is executing.

hostname:path_name

If path_name is a relative path, then it is taken to be relative to the user's home directory on the host named hostname.

If path_name is an absolute path, then it is the absolute path on the host named hostname.

If path_name does not include a filename, the default filename will be

jobid.OU

If the -o option is not specified, the default filename for the standard output stream is used. It has this form:

job_name.osequence_number

This attribute can be altered after the job has begun execution, in which case the new value will not take effect until the job is rerun.

-p priority

Alters priority of the job. Format: host-dependent integer. Default: zero. Range: [-1024, +1023] inclusive. Sets job's Priority attribute to priority.

This attribute can be altered after the job has begun execution, in which case the new value will not take effect until the job is rerun.

-r y|n Changes whether the job is rerunnable. See the qrerun(1B) command. Default: “y”. Sets job’s Rerunnable attribute to the argument. Format: single character, “y” or “n”.

y Job is rerunnable.

n Job is not rerunnable.

-S path_list

Specifies the interpreter for the job script. Sets job’s Shell_Path_List attribute to path_list. Default: user’s login shell on execution node. The path_list argument is the full path to the interpreter including the executable name. Format:

```
path[@host][,path@host ...]
```

Only one path may be specified without a host name. Only one path may be specified per named host. The path selected is the one whose host name is that of the server on which the job resides.

This attribute can be altered after the job has begun execution, in which case the new value will not take effect until the job is rerun.

-u user_list

Alters list of usernames. Job will be run under a username from this list. Sets job’s User_List attribute to user_list. Default: job owner (username on submit host.) Format of user_list:

```
user[@host][,user@host ...]
```

Only one username may be specified without a host name. Only one username may be specified per named host. The server on which the job resides will select first the username whose host name is the same as the server name. Failing that, the next selection will be the username with no specified hostname. The usernames on the server and execution hosts must be the same.

The job owner must have authorization to run as the specified user.

This attribute cannot be altered once the job has begun execution.

-W additional_attributes

The -W option allows change in specification of additional job attributes. Format:

-W attribute_name=value[,attribute_name=value...]

If white space occurs within the additional_attributes argument, or the equal sign "=" occurs within an attribute_value string, then that must be enclosed with single- or double-quotes. PBS supports the following attributes within the -W option:

depend=dependency_list

Defines dependencies between this and other jobs. Sets the job's depend attribute to dependency_list. The dependency_list has the form:
type:arg_list[,type:arg_list ...]
where except for the on type, the arg_list is one or more PBS job IDs in the form:
jobid[:jobid ...]
The type can be:

after: arg_list

This job may be scheduled for execution at any point after all jobs in arg_list have started execution.

afterok: arg_list

This job may be scheduled for execution only after all jobs in arg_list have terminated with no errors. See "Warning about exit status with csh" in EXIT

STATUS.

afternotok: arg_list
This job may be scheduled for execution only after all jobs in arg_list have terminated with errors. See “Warning about exit status with csh” in EXIT STATUS.

afterany: arg_list
This job may be scheduled for execution after all jobs in arg_list have terminated, with or without errors.

before: arg_list
Jobs in arg_list may begin execution once this job has begun execution.

beforeok: arg_list
Jobs in arg_list may begin execution once this job terminates without errors. See “Warning about exit status with csh” in EXIT STATUS.

beforenotok: arg_list
If this job terminates execution with errors, then jobs in arg_list may begin. See “Warning about exit status with csh” in EXIT STATUS.

beforeany: arg_list
Jobs in arg_list may begin execution once this job terminates execution, with or without errors.

on: count
This job may be scheduled for execution after count dependencies on other jobs have been satisfied.

This type is used in conjunction with one of the before types listed. Count is an integer greater than 0.

Job IDs in the arg_list of before types must have been submitted with a type of on.

To use the before types, the user must have the authority to alter the jobs in arg_list. Otherwise, the dependency is rejected and the new job aborted.

Error processing of the existence, state, or condition of the job on which the newly submitted job is a deferred service, i.e. the check is performed after the job is queued. If an error is detected, the new job will be deleted by the server. Mail will be sent to the job submitter stating the error.

Dependency examples:

```
qalter -W depend=afterok:123.host1.domain.com
/tmp/script
qalter -W depend=before:234.host1.com:235.host1.com
/tmp/script
```

group_list=g_list

Alters list of group names. Job will be run under a group name from this list. Sets job's group_List attribute to g_list. Default: login group name of job owner. Format of g_list:

```
group[@host][,group@host ...]
```

Only one group name may be specified without a host name. Only one group name may be specified per named host. The server on which the job resides will select first the group name whose host name is the same as the server name. Failing that, the next selection will be the group name with no specified hostname. The group names on the server and execution hosts must be the same.

stagein=file_list
stageout=file_list

Changes files to be staged-in before execution or staged-out after execution is complete. Sets the job's stagein and stageout attributes the the specified file_list. On completion of the job, all staged-in and staged-out files are removed from the execution host(s). The file_list has the form:

```
filespec[,filespec]
```

where filespec is

```
local_file@hostname:remote_file
```

regardless of the direction of the copy. The name local_file is the name of the file on the execution host(s). It may be an absolute path or relative to the user's home directory on the execution host. The name remote_file is the path on hostname. The name may be absolute or relative to the user's home directory on the hostname. If file_list has more than one filespec, i.e. it contains commas, it must be enclosed in double-quotes. The use of wildcards in the file name is not supported.

umask=NNNN

Alters the umask with which the job will be started. Default value: 077. Can be used with one to four digits; typically two. Sets job's umask attribute to NNNN. Controls umask of job's standard output and standard error. Example: -W umask=33 allows group and world read on the job's output.

--version

The qalter command returns its PBS version information and exits. This option can only be used alone.

OPERANDS

The qalter command accepts a job_identifier_list as its operand. The job_identifier_list is one or more jobids for normal jobs or array jobs. Individual subjobs of an array job are not alterable. For a job, this is:

```
sequence_number[.server_name][@server]
```

and for an array job, it is:

```
sequence_number[][.server_name][@server]
```

Note that some shells require that you enclose a job array ID in double quotes.

STANDARD ERROR

The qalter command will write a diagnostic message to standard error for each error occurrence.

EXIT STATUS

Zero upon successful processing of input. Exit value will be greater than zero upon failure of qalter.

Warning about exit status with csh:
If a job is run in csh and a .logout file exists in the home directory in which the job executes, the exit status of the job is that of the .logout script, not the job script. This may impact any inter-job dependencies.

SEE ALSO

The PBS Professional User's Guide, the PBS Professional Administrator's Guide,
pbs_job_attributes(7B), pbs_resources(7B), qdel(1B), qhold(1B), qmove(1B), qmsg(1B), qrerun(1B), qrls(1B), qselect(1B), qstat(1B), qsub(1B)

NAME

qdel - deletes PBS jobs

SYNOPSIS

```
qdel [-W delay <delay>|force|suppress_email=<N>]
    job_identifier [job_identifier ...]
qdel --version
```

DESCRIPTION

The qdel command deletes jobs in the order given.

A PBS job may be deleted by its owner, an operator, or the administrator. The server deletes a PBS job by sending a SIGTERM signal, then, if there are remaining processes, a SIGKILL signal.

The server's default_qdel_arguments attribute may affect the behavior of the qdel command. This attribute is settable by the administrator via the qmgr command. The attribute may be set to "-Wsuppress_email=<N>". The server attribute is overridden by command line arguments. See the pbs_server_attributes(1B) man page.

If someone other than the job's owner deletes the job, mail is sent to the job's owner, or to a list of mail recipients if specified during qsub. See the qsub(1B) man page.

What Happens:

The job's running processes are killed.

The epilogue runs.

Files that were staged in are staged out. This includes standard out (.o) and standard error (.e) files.

Files that were staged in or out are deleted.

The job's temp directory is removed.

The job is removed from the MOM(s) and the server.

OPTIONS

-W <delay>

Overrides the default delay of 2 seconds between the SIGTERM and SIGKILL signals.

The <delay> argument is an integer number of seconds.

The default delay between the signals is given in the queue's kill_delay attribute, settable by the administrator.

-W force Deletes the job whether or not the job's execution host is reachable.

-Wsuppress_email=<N>

No mail is sent if more than N job_identifiers are given. The <N> argument is an integer. Note that there is no space between "W" and "suppress_email".

--version The qdel command returns its PBS version information and exits. This option can only be used alone.

OPERANDS

The qdel command accepts one or more job_identifier operands. Square brackets in the following description have two different meanings. Job array identifiers have square brackets, and the brackets indicate the contents are optional for [.server_name] and [@server] . The format of job_identifier is:

Jobs sequence_number[.server_name][@server]

Job arrays

sequence_number[][.server_name][@server]

Array range
sequence_number[<first>-<last>][.server_name][@server]
first and last are the first and last indices of the subjobs
to be deleted.

Subjob sequence_number[<index>][.server_name][@server]
index is the index of the subjob to be deleted.

Job array identifiers must be enclosed in double quotes for
some shells.

STANDARD ERROR
The qdel command will write a diagnostic messages to standard error for each error occurrence.

EXIT STATUS
Zero upon successful processing of input.

Greater than zero upon error.

SEE ALSO
The PBS Professional User's Guide, the PBS Professional Administrator's Guide,
pbs_queue_attributes(7B), pbs_server_attributes(1B), qsub(1B), qsig(1B), pbs_deljob(3B)

NAME

qhold - hold PBS batch jobs

SYNOPSIS

qhold [-h hold_list] job_identifier_list
qhold --version

DESCRIPTION

The qhold command requests that a server place one or more holds on a job. A job that has a hold is not eligible for execution. Supported holds: USER , OTHER (also known as operator), SYSTEM, and bad password.

A user may place a USER hold upon any job the user owns. An operator, who is a user with operator privilege, may place ether an USER or an OTHER hold on any job. The batch administrator may place any hold on any job.

The p option can only be set by root or admin user via qhold -h p. The owning user can release with qrls -h p or query by qselect -h p.

If no -h option is given, the USER hold will be applied to the jobs described by the job_identifier_list operand list.

If the job identified by job_identifier_list is in the queued, held, or waiting states, then all that occurs is that the hold type is added to the job. The job is then placed into the held state if it resides in an execution queue.

If the job is running, then the following additional action is taken to interrupt the execution of the job. If checkpoint / restart is supported by the host system, requesting a hold on a running job will (1) cause the job to be checkpointed, (2) the resources assigned to the job will be released, and (3) the job is placed in the held state in the execution queue.

If checkpoint / restart is not supported, qhold will only set the requested hold attribute. This will have no effect unless the job is rerun with the qrerun command.

The qhold command can be used on job arrays, but not on subjobs or ranges of subjobs.

OPTIONS

-h hold_list Defines the types of holds to be placed on the job.

The hold_list argument is a string consisting of one or more of the letters “u”, “o”, or “s” in any combination or the character “n” or “p”. The hold type associated with each letter is:

u - USER

o - OTHER

s - SYSTEM

n - None

p - Bad password

--version The qhold command returns its PBS version information and exits. This option can only be used alone.

OPERANDS

The qhold command accepts a job_identifier_list which is one or more space-separated jobids in the form: sequence_number[.server_name][@server]

Note that some shells require that you enclose a job array identifier in double quotes.

STANDARD ERROR

The qhold command will write a diagnostic message to standard error for each error occurrence.

EXIT STATUS

Zero upon successful processing of all the operands.

Greater than zero if the qhold command fails to process any operand.

SEE ALSO

The PBS Professional User's Guide, the PBS Professional Administrator's Guide, qrls(1B), qalter(1B), qsub(1B), pbs_alterjob(3B), pbs_holdjob(3B), pbs_rlsjob(3B), pbs_job_attributes(7B), pbs_resources(7B)

Local 12 April 2007 qhold(1B)

NAME

qmove - move PBS batch job

SYNOPSIS

qmove destination job_identifier ...
qmove --version

DESCRIPTION

To move a job is to remove the job from the queue in which it resides and place the job in another queue.

The qmove command can be used on job arrays, but not on subjobs or ranges of subjobs.

Note that job arrays can only be moved from one server to another if they are in the 'Q', 'H', or 'W' states, and only if there are no running subjobs. The state of the job array is preserved, and the job array will run to completion on the new server.

A job in the Running , Transiting , or Exiting state cannot be moved.

OPTIONS

--version The qmove command returns its PBS version information and exits. This option can only be used alone.

OPERANDS

The first operand is the new destination for the jobs. It will be accepted in the syntax:

queue
@server
queue@server

See the PBS ERS section , "Destination Identifiers".

If the destination operand describes only a queue, then qmove will move jobs into the queue of the specified name at the job's current server.

If the destination operand describes only a batch server, then qmove will move jobs into the default queue at that batch server.

If the destination operand describes both a queue and a batch server, then qmove will move the jobs into the specified queue at the specified server.

All following operands are job_identifiers which specify the jobs to be moved to the new destination . The qmove command accepts one or more job_identifier operands of the form:
    sequence_number[.server_name][@server]

Note that some shells require that you enclose a job array identifier in double quotes.

STANDARD ERROR

The qmove command will write a diagnostic messages to standard error for each error occurrence.

EXIT STATUS

Upon successful processing of all the operands presented to the qmove command, the exit status will be a value of zero.

If the qmove command fails to process any operand, the command exits with a value greater than zero.

SEE ALSO

The PBS Pro User’s Guide, the PBS Pro Administrator’s Guide, qsub(1B), pbs_movejob(3B)

NAME

qmsg - send message to PBS batch jobs

SYNOPSIS

qmsg [-E] [-O] message_string job_identifier ...
qmsg --version

DESCRIPTION

To send a message to a job is to write a message string into one or more output files of the job. Typically this is done to leave an informative message in the output of the job.

The qmsg command writes messages into the files of jobs by sending a Message Job batch request to the batch server that owns the job. The qmsg command does not directly write the message into the files of the job.

The qmsg command cannot be used on job arrays, subjobs or ranges of subjobs.

OPTIONS

-E Specifies that the message is written to the standard error of each job.

-O Specifies that the message is written to the standard output of each job.

--version The qmsg command returns its PBS version information and exits. This option can only be used alone.

If no option is specified, the message will be written to the standard error of the job.

OPERANDS

The first operand, message_string , is the message to be written. If the string contains blanks, the string must be quoted. If the final character of the string is not a newline, a newline character will be added when written to the job's file.

All following operands are job_identifiers which specify the jobs to receive the message string. The qmsg command accepts one or more job_identifier operands of the form:

sequence_number[.server_name][@server]

STANDARD ERROR

The qmsg command will write a diagnostic message to standard error for each error occurrence.

EXIT STATUS

Upon successful processing of all the operands presented to the qmsg command, the exit status will be a value of zero.

If the qmsg command fails to process any operand, the command exits with a value greater than zero.

SEE ALSO

The PBS Professional User's Guide, the PBS Professional Administrator's Guide,
qsub(1B), pbs_msgjob(3B)

NAME

qorder - exchange order of two PBS batch jobs.

SYNOPSIS

```
qorder job_identifier job_identifier
qorder --version
```

DESCRIPTION

Allows the exchange of two jobs' positions in the queue or queues in which the jobs reside. The two jobs must be located at the same server. No attribute of the job, e.g. priority, is changed. The impact of interchanging the order within or between queues is dependent on local job scheduling policy; contact your systems administrator.

A job in the running state cannot be reordered.

The qorder command can be used on job arrays, but not on subjobs or ranges of subjobs.

OPTIONS

--version

The qorder command returns its PBS version information and exits. This option can only be used alone.

OPERANDS

Both operands are job_identifiers which specify the jobs to be exchanged. The qorder command accepts two job_identifier operands of the form:

```
sequence_number[.server_name][@server]
```

The server specification for the two jobs must agree as to the current location of the two job ids.

Note that some shells require that you enclose a job array identifier in double quotes.

STANDARD ERROR

The qorder command will write diagnostic messages to standard error for each error occurrence.

EXIT STATUS

Upon successful processing of all the operands presented to the qorder command, the exit status will be a value of zero.

If the qorder command fails to process any operand, the command exits with a value greater than zero.

SEE ALSO

The PBS Professional User’s Guide, the PBS Professional Administrator’s Guide,
qsub(1B), qmove(1B), pbs_orderjob(3B), pbs_movejob(3B)

Local 12 April 2007 qorder(1B)

NAME

qrerun - rerun a PBS batch job

SYNOPSIS

```
qrerun [-W force] job_identifier [job_identifier ...]
qrerun --version
```

DESCRIPTION

The qrerun command reruns the specified jobs if possible.

To rerun a job is to kill it and requeue it in the execution queue from which it was run.

If a job is marked as not rerunnable then qrerun will fail. See the -r option on the qsub and qalter commands.

The qrerun command can be used on job arrays, subjobs, and ranges of subjobs. It cannot rerun a subjob which is not running.

OPTIONS

-W force   The job is to be requeued even if the node on which the job is executing is unreachable.

--version   The qrerun command returns its PBS version information and exits. This option can only be used alone.

OPERANDS

The qrerun command accepts one or more job_identifier operands of the form:

sequence_number[.server_name][@server]

Note that some shells require that you enclose a job array identifier in double quotes.

STANDARD ERROR

The qrerun command will write a diagnostic message to standard error for each error occurrence.

EXIT STATUS

Zero upon successful processing of all operands.

Greater than zero upon failure to process any operand.

SEE ALSO

The PBS Professional User's Guide, the PBS Professional Administrator's Guide,
qsub(1B), qalter(1B), pbs_alterjob(3B), pbs_rerunjob(3B)

Local 12 April 2007 qrerun(1B)

NAME

qrls - release hold on PBS batch jobs

SYNOPSIS

qrls [-h hold_list] job_identifier ...
qrls --version

DESCRIPTION

The qrls command removes or releases holds which exist on batch jobs.

A job may have one or more types of holds which make the job ineligible for execution. The types of holds are USER , OTHER , SYSTEM, and bad password. The different types of holds may require that the user issuing the qrls command have special privilege. Typically, the owner of the job will be able to remove a USER hold, but not an OTHER or SYSTEM hold. An Attempt to release a hold for which the user does not have the correct privilege is an error and no holds will be released for that job.

If no -h option is specified, the USER hold will be released.

Only root or admin can set a bad password hold via qhold -h p. The owner of the job can qrls -h p a hold set with qhold -h p.

If the job has no execution_time pending, the job will change to the queued state. If an execution_time is still pending, the job will change to the waiting state.

OPTIONS

-h hold_list   Defines the types of hold to be released from the jobs. The hold_list option argument is a string consisting of one or more of the letters u , o , or s in any combination, or one or more of the letters n or p. The hold type associated with each letter is:

u - USER

o - OTHER

s - SYSTEM

n - None

p - Bad password

--version The qrls command returns its PBS version information and exits. This option can only be used alone.

OPERANDS

The qrls command accepts one or more job_identifier operands of the form:

sequence_number[.server_name][@server]

Note that some shells require that you enclose a job array identifier in double quotes.

STANDARD ERROR

The qrls command will write a diagnostic message to standard error for each error occurrence.

EXIT STATUS

Upon successful processing of all the operands presented to the qrls command, the exit status will be a value of zero.

If the qrls command fails to process any operand, the command exits with a value greater than zero.

SEE ALSO

The PBS Professional User’s Guide, the PBS Professional Administrator’s Guide,
qsub(1B), qalter(1B), qhold(1B), pbs_alterjob(3B), pbs_holdjob(3B), and pbs_rlsjob(3B).

Local 12 April 2007 qrls(1B)

NAME

qselect - select PBS batch jobs

SYNOPSIS

```
qselect [-a [op]date_time] [-A account_string] [-c [op]interval]
        [-h hold_list] [-l resource_list] [-N name] [-p [op] priority]
        [-q destination] [-r rerun] [-s states] [-u user_list]
        [-J] [-T]
qselect --version
```

DESCRIPTION

The qselect command lists those jobs that meet the specified selection criteria. Jobs are selected from a single server.

Each option acts as a filter restricting which jobs are listed. With no options, the qselect command will list all jobs at the server which the user is authorized to list (query status of).

OPTIONS

When an option is specified with a optional op component to the option argument, then op specifies a relation between the value of a certain job attribute and the value component of the option argument. If an op is allowable on an option, then the description of the option letter will indicate the op is allowable. The only acceptable strings for the op component, and the relation the string indicates, are shown in the following list:

.eq. the value represented by the attribute of the job is equal to the value represented by the option argument.

.ne. the value represented by the attribute of the job is not equal to the value represented by the option argument.

.ge. the value represented by the attribute of the job is greater than or equal to the value represented by the option argument.

.gt. the value represented by the attribute of the job is greater than the value represented by the option argument.

.le. the value represented by the attribute of the job is less than or equal to the value represented by the option argument.

.lt. the value represented by the attribute of the job is less than the value represented by the option argument.

-a [op]date_time
Restricts selection to a specific time, or a range of times.

The qselect command selects only jobs for which the value of the Execution_Time attribute is related to the date_time argument by the optional op operator. The date_time argument is in the form of the date_time operand of the touch(1) command: [[CC]YY]MMDDhhmm[.SS]
where the MM is the two digits for the month, DD is the day of the month, hh is the hour, mm is the minute, and the optional SS is the seconds. CC is the century and YY the year.

If op is not specified, jobs will be selected for which the Execution_Time and date_time values are equal. If op is specified, jobs will be selected according to the following definitions:

.eq. Execution_Time attribute is equal to the date_time argument.

.ne. Execution_Time attribute is not equal to the date_time argument.

.ge. execution_Time attribute is greater than (after) or equal to the date_time argument.

.gt. Execution_Time attribute is greater than (after) the date_time argument.

.le. Execution_Time attribute is less than (before) or equal to the date_time argument.

.lt. Execution_Time attribute is less than (before) the date_time argument.

-A account_string

Restricts selection to jobs whose Account_Name attribute matches the specified account_string .

-c [op]interval

Restricts selection to jobs whose Checkpoint interval attribute matches the specified relationship.

The values of the Checkpoint attribute are defined to have the following ordered relationship:

n > s > c=minutes > c > u

If the optional op is not specified, jobs will be selected whose Checkpoint attribute is equal to the interval argument. If op is specified, jobs will be selected according to:

.eq. Checkpoint attribute of the job is equal to the interval argument.

.ne. Checkpoint attribute of the job is not equal to the interval argument.

.ge. Checkpoint attribute of the job is greater than or equal to the interval argument.

.gt. Checkpoint attribute of the job is greater than the interval argument.

.le. Checkpoint attribute of the job is less than or equal to the interval argument.

.lt. Checkpoint attribute of the job is less than the interval argument.

For an interval value of “u”, only “.eq.” and “.ne.” are valid.

-h hold_list

Restricts the selection of jobs to those with a specific set of hold types. Only those jobs will be selected whose Hold_Types attribute exactly match the value of the hold_list argument.

The hold_list argument is a string consisting of the single letter n , or one or more of the letters u, o, p, or s in any combination. If letters are duplicated, they are treated as if they occurred once. The letters represent the hold types:

n - none
u - user
o - other
p - bad password
s - system

-l resource_list

Restricts selection of jobs to those with specified resource amounts.

The resource_list is in the following format:
resource_name op value[,resource_name op val,...]
The relation operator op must be present.

For job-wide resources, all operators are useful. However, resource specifications for chunks using the select statement, or placement using the place statement are stored as strings. Therefore the only useful operators for these are .eq. and .ne.

When comparing the values of resources, the following definitions for the operator apply:

.eq. the resource value in the Resource_List attribute of the job equals the value specified in resource_list .

.ne. the resource value in the Resource_List attribute of the job is not equal to the value specified in resource_list .

.ge. the resource value in the Resource_List attribute of the job is greater than or equal to the value specified in resource_list .

.gt. the resource value in the Resource_List attribute of the job is greater than the value specified in resource_list .

.le. the resource value in the Resource_List attribute of the job is less than or equal to the value specified in resource_list .

.lt. the resource value in the Resource_List attribute of the job is less than the value specified in resource_list .

-N name Restricts selection of jobs to those with a specific name.

-p [op]priority

Restricts selection of jobs to those with a priority that matches the specified relationship. If op is not specified, jobs are selected for which the job Priority attribute is equal to the priority

If the op is specified, the relationship is defined as:

.eq. Priority attribute is equal to the value of the priority argument.

.ne. Priority attribute is not equal to the value of the priority argument.

.ge. Priority attribute is greater than or equal to the value of the priority argument.

.gt. Priority attribute is greater than the value of the priority argument.

.le. Priority attribute is less than or equal to the value of the priority argument.

.lt. Priority attribute is less than the value of the priority argument.

-q destination
Restricts selection to those jobs residing at the specified destination.

The destination may be of one of the following three forms:
queue
@server
queue@server

If the -q option is not specified, jobs will be selected from the default server.

If the destination describes only a queue, only jobs in that queue on the default batch server will be selected.

If the destination describes only a server, then jobs in all queues on that server will be selected.

If the destination describes both a queue and a server, then only jobs in the named queue on the named server will be selected.

-r rerun Restricts selection of jobs to those with the specified Rerunable attribute. The option argument must be a single character. The following two characters are supported by PBS: y and n .

-s states Restricts job selection to those in the specified states.

The states argument is a character string which consists of any combination of the characters: B , E , H , Q , R , S , T , U , and W . [A repeated character will be accepted, but no additional meaning is assigned to it.]

Job states:

B Job array has started execution.

E The Exiting state.

H The Held state.

Q The Queued state.

R The Running state.

S The Suspended state.

T The Transiting state.

U Job suspended due to workstation user activity.

W The Waiting state.

X Subjob has completed execution or been deleted.

Jobs will be selected which are in any of the specified states. Since array jobs are never in states R, S, T, or U, if those states are specified, no array job will be selected. Subjobs of the array in those states may be selected if -t is specified.

-u user_list
Restricts selection to jobs owned by the specified user names.

This provides a means of limiting the selection to jobs owned by one or more users.

The syntax of the user_list is:
user_name[@host][,user_name[@host],...]
Host names may be wild carded on the left end, e.g. “*.nasa.gov”. User_name without a “@host” is equivalent to “user_name@*”, that is at any host. Jobs will be selected which are owned by the listed users at the corresponding hosts.

-J Limits the selection to jobs that are array jobs.

-T Causes the subjobs which meet the selection criteria of a
array job to be selected.

--version The qselect command returns its PBS version information and
exits. This option can only be used alone.

STANDARD OUTPUT

The list of job identifiers of selected jobs is written to standard output. Each job identifier is separated by white space. Each job identifier is of the form:

sequence_number.server_name@server

Where sequence_number.server is the identifier assigned at submission time, see qsub. @server identifies the server which currently owns the job.

STANDARD ERROR

The qselect command will write a diagnostic message to standard error for each error occurrence.

EXIT STATUS

Upon successful processing of all options presented to the qselect command, the exit status will be a value of zero.

If the qselect command fails to process any option, the command exits with a value greater than zero.

SEE ALSO

The PBS Professional User's Guide, the PBS Professional Administrator's Guide,
qalter(1B), qdel(1B), qhold(1B), qmove(1B), qrls(1B), qstat(1B), qsub(1B), pbs_job_attributes(7B), pbs_resources(7B)

Local 12 April 2007 qselect(1B)

NAME

qsig - signal PBS batch job

SYNOPSIS

qsig [-s signal] job_identifier ...
qsig --version

DESCRIPTION

The qsig command requests that a signal be sent to the specified executing batch jobs. The signal is sent to the session leader of the job.

If the -s option is not specified, `SIGTERM' is sent. The request to signal a batch job will be rejected if:

- The user is not authorized to signal the job.
- The job is not in the running state.
- The requested signal is not supported by the system upon which the job is executing.

The qsig command sends a Signal Job batch request to the server which owns the job.

The qsig command can be used for job arrays, ranges of subjobs, and subjobs. If it is used on a range of subjobs, the subjobs in the range which are running will be signaled.

OPTIONS

-s signal Declares which signal is sent to the job.

The signal argument is either a signal name, e.g. SIGKILL, the signal name without the SIG prefix, e.g. KILL, or an unsigned signal number, e.g. 9. The signal name SIGNULL is allowed; the server will send the signal 0 to the job which will have no effect. Not all signal names will be recognized by qsig signal name, try issuing the signal number instead.

Two special signal names, “suspend” and “resume”, [note, all lower case], are used to suspend and resume jobs. When suspended, a job continues to occupy system resources but is not executing and is not charged for walltime. Manager or operator privilege is required to suspend or resume a job.

If qsig -s resume is used on a job that was suspended using qsig -s suspend, the job will be resumed when there are sufficient resources.

--version The qsig command returns its PBS version information and exits. This option can only be used alone.

OPERANDS

The qsig command accepts one or more job_identifier operands. For a job, this has the form:

sequence_number[.server_name][@server]

and for a job array, it is:

sequence_number[][.server_name][@server]

Note that some shells require that you enclose a job array identifier in double quotes.

STANDARD ERROR

The qsig command will write a diagnostic messages to standard error for each error occurrence.

EXIT STATUS

Upon successful processing of all the operands presented to the qsig command, the exit status will be a value of zero.

If the qsig command fails to process any operand, the command exits with a value greater than zero.

SEE ALSO

The PBS Professional User's Guide, the PBS Professional Administrator's Guide,
qsub(1B), pbs_sigjob(3B), pbs_resources(7B)

NAME

qstat - display status of PBS batch jobs, queues, or servers

SYNOPSIS

Displaying Job Status

Default format:

```
qstat [-p] [-J] [-t]
    [ [job_identifier | destination] ...]
```

Long format:

```
qstat -f [-p] [-J] [-t]
    [ [job_identifier | destination] ...]
```

Alternate format:

```
qstat [-a [-w]| -i | -r] [-n [-1][-w]] [-s [-1][-w]] [-G | -M]
    [-u user_list] [-J] [-t] [ [job_identifier | destination] ...]
```

Displaying Queue Status

Default format:

```
qstat -Q [destination ...]
```

Long format:

```
qstat -Q -f [destination ...]
```

Alternate format:

```
qstat -q [-G | -M] [destination ...]
```

Displaying Server Status

Default format:

```
qstat -B [server_name ...]
```

Long format:

```
qstat -B -f [server_name ...]
```

Version Information

```
qstat --version
```

DESCRIPTION

The qstat command is used to display the status of jobs, queues, and batch servers. The status information is written to standard output.

Status information can be displayed in a default format, an alternate format, or a long format, depending upon the options given. Default and alternate formats display all status information for a job, queue or server on one line, in columns. Long formats display status information one attribute to a line.

When displaying job status information, the qstat command will display status information about all job_identifiers and destinations specified.

If your job has been moved to another server through peer scheduling, give the job ID as an argument to qstat. If you only give the qstat command, your job will not appear to exist. For example, your job 123.ServerA is moved to ServerB. In this case, use

```
qstat 123
```

or

```
qstat 123.ServerA
```

To list all jobs at ServerB, you can use:

```
qstat @ServerB
```

JOB STATUS DISPLAY

Job Status in Default Format

The qstat command will display job status in default format when the options given are among -p, -J or -t, regardless of operands. Jobs are displayed one to a line, with these column headers:

```
Job id   Name       User      Time Use S Queue
-------- ---------- --------- -------- - -----
```

Description of columns:

Job id	The job_identifier assigned by PBS.
Name	Job name assigned by submitter.

User   Username of job owner.

Time Use The CPU time used by the job.

S   The job's state:

B Array job has at least one subjob running.

E Job is exiting after having run.

H Job is held.

Q Job is queued.

R Job is running.

S Job is suspended.

T Job is being moved to new location.

U Cycle-harvesting job is suspended due to keyboard activity.

W Job is waiting for its submitter-assigned start time to be reached.

X Subjob has completed execution or has been deleted.

Queue   The queue in which the job resides.

Job Status in Long Format
If the -f (full) option is given, full job status information for each job is displayed starting with the Job Id, followed by each attribute, one to a line, as name = value pairs. This includes the exec_host string and the exec_vnode string. The full output output can be very large.

The exec_host string has the format:

```
    hosta/J1+hostb/J2*P+...
```

where J1 and J2 are an index of the job on the named host and P is the number of processors allocated from that host to this job. P does not appear if it is 1.

The exec_vnode string has the format:

```
    (vnodeA:ncpus=N1:mem=M1)+(vnodeB:ncpus=N2:mem=M2)+...
```

where N1 and N2 are the number of CPUs allocated to that job on that vnode, and M1 and M2 are the amount of memory allocated to that job on that vnode.

Job Status in Alternate Format

The qstat command will display job status in the alternate format if any of the -a, -i, -r, -n, -s, -G, -M, or -u user_list options is given. Jobs are displayed one to a line. If jobs are running and the -n option is specified, there is a second line for the exec_host string.

Column headers:

```
                                             Req'd  Req'd   Elap
Job ID Username Queue Jobname SessID NDS TSK Memory Time  S Time
------ -------- ----- ------- ------ --- --- ------ ----- - ----

  exec_host string (if -n is specified)
```

Description of columns:

Job ID	The job_identifier assigned by PBS.
Username	Username of job owner.
Queue	Queue in which the job resides.

Jobname	Job name assigned by submitter.
SessID	Session ID. Only appears if the job is running.
NDS	Number of chunks or nodes requested by the job.
TSK	Number of CPUs requested by the job.
Req'd Memory	Amount of memory requested by the job.
Req'd Time	CPU time or walltime requested by the job, depending upon which was specified by the submitter.
S	The job's state. (See listing above.)
Elap Time	CPU time or walltime used by the job, depending upon which was specified by the submitter.

QUEUE STATUS DISPLAY

Queue Status in Default Format

The qstat command will display queue status in the default format if the only option is -Q, regardless of operands. Queue status is displayed one queue to a line, with these column headers:

```
Queue      Max  Tot  Ena  Str Que  Run  Hld  Wat  Trn  Ext  Type
---------- ---- ---- ---- --- ---- ---- ---- ---- ---- ---- ----
```

Description of columns:

Queue	Queue name.
Max	Maximum number of jobs allowed to run concurrently in the queue.
Tot	Total number of jobs in the queue.
Ena	Whether the queue is enabled or disabled.

Str	Whether the queue is started or stopped.
Que	Number of queued jobs.
Run	Number of running jobs.
Hld	Number of held jobs.
Wat	Number of waiting jobs.
Trn	Number of jobs being moved (transiting.)
Ext	Number of exiting jobs.
Type	Type of queue: execution or routing.

Queue Status in Long Format If the -f (full) option is given, full queue status information for each queue is displayed starting with the queue name, followed by each attribute, one to a line, as name = value pairs.

Queue Status: Alternate Format
The qstat command will display queue status in the alternate format if any of the -q, -G or -M options is given. Queue status is displayed one queue to a line, with these column headers:

```
Queue   Memory CPU Time Walltime Node Run Que Lm State
------- ------ -------- -------- ---- --- --- -- -----
```

Description of columns:

Queue	Queue name.
Memory	Maximum amount of memory that can be requested by a job in the queue.
CPU Time	Maximum amount of CPU time that can be requested by a job in the queue.
Walltime	Maximum amount of wall time that can be requested by a job in the queue.
Node	Maximum number of nodes that can be requested by a job in the queue.
Run	Number of running jobs. Lowest row is total number of running jobs in all the queues shown.
Que	Number of queued jobs. Lowest row is total number of queued jobs in all the queues shown.
Lm	Maximum number of jobs allowed to run concurrently in the queue.
State	State of the queue: E (enabled) or D (disabled), and R (running) or S (stopped).

SERVER STATUS DISPLAY

Server Status in Default Format:
The qstat command will display server status if the only option given is -B, regardless of operands.

Column headers for default server status:

```
Server           Max   Tot   Que   Run   Hld   Wat   Trn   Ext   Status
---------------- ----- ----- ----- ----- ----- ----- ----- ----- ------
```

Description of columns:

Server	Name of the server.
Max	Maxiumum number of jobs allowed concurrently running on the server.
Tot	Total number of jobs currently managed by the server.
Que	Number of queued jobs.
Run	Number of running jobs.
Hld	Number of held jobs.
Wat	Number of waiting jobs.
Trn	Number of transiting jobs.
Ext	Number of exiting jobs.
Status	Status of the server.

Server Status in Long Format: If the -f (full) option is given, full server status information is displayed starting with the server name, followed by each attribute, one to a line, as name = value pairs. PBS version information is listed.

OPTIONS

Job Status

-J	Limits status information to job arrays.
-t	When used with -J option, limits status information to subjobs. When used alone, adds subjob information.

-p The Time Use column is replaced with the percentage completed for the job. For an array job this is the percentage of subjobs completed. For a normal job, it is the larger of percentage used walltime or percentage used CPU time. Default format used.

The following options will cause the alternate job status format to be used:

-a All jobs are displayed. If a destination is given, information for all jobs at that destination is displayed. If a job_identifier is given, information about that job is displayed. Always specify this option before the -n or -s options, otherwise they will not take effect.

-i If a destination is given, information for queued, held or waiting jobs at that destination is displayed. If a job_identifier is given, information about that job is displayed regardless of its state.

-r If a destination is given, information for running or suspended jobs at that destination is displayed. If a job_identifier is given, information about that job is displayed regardless of its state.

-u user_list
If a destination is given, status for jobs at that destination owned by users in user_list is displayed. If a job_identifier is given, status information for that job is displayed regardless of the job's ownership.

Format: username[@host] in comma-separated list. Hostnames may be wildcarded, but not domain names. When no hostname is specified, username is for any host.

-n The exec_host string is listed on the line below the basic information. If the -1 option is given, the exec_host string is listed on the end of the same line. If using the -a option, always specify the -n option after -a otherwise the -n option will not take effect.

-s Any comment added by the administrator or scheduler is shown on the line below the basic information. If the -1 option is given, the comment string is listed on the end of the same line. If using the -a option, always specify the -s option after -a otherwise the -s option will not take effect.

-w Allows display of wider fields. User name, Queue and Job name can be up to 15 characters wide. Session ID can be up to 8 characters wide and NDS can be up to 4 characters wide. Can only be used with -a, -n or -s.

-1 Reformats qstat output to a single line. Can only be used in conjunction with the -n and/or -s options.

Queue Status

-Q Display queue status in default format. Operands must be destinations.

-q Display queue status in alternate format. Operands must be destinations.

Server Status

-B Display server status. Operands must be names of servers.

Job, Queue, Server Status

-f Full display. Job, queue or server attributes displayed one to a line.

-G Show size in gigabytes. Alternate format is used.

-M Show size in megawords. A word is considered to be 8 bytes. Alternate format is used.

Version Information

--version
The qstat command returns its PBS version information and exits. This option can only be used alone.

OPERANDS

job_identifier
Job identifier assigned by PBS at submission. Only used with job status requests. Status information for this job is displayed.
Formats:

```
  Job:       sequence_number[.server_name][@server]
  Job Array: sequence_number[][.server_name][@server]
  Subjob:    sequence_number[index][.server_name][@server]
```

Note that job array identifiers are a sequence number followed by square brackets, e.g.:

```
1234[]
```

and subjob identifiers are a sequence number followed by square brackets enclosing the subjob's index, e.g.:

```
1234[99]
```

Note that some shells require that you enclose a job array identifier in double quotes.

If .server_name is omitted, the default server is queried.
If @server is given, that server is queried.

destination
Name of queue, name of queue at a specific server, or specification of server.

Formats:

Name of queue:	queue_name
Name of queue at server:	queue_name@server
Server:	@server

When displaying job status:
If queue_name is given, status is displayed for all jobs in the named queue at the default server.
If queue_name@server is given, status is displayed for all jobs in queue_name at server.
If @server is given, status is displayed for all jobs at that server.

When displaying queue status:
If queue_name is given, status is displayed for that queue at the default server.
If queue_name@server is given, status is displayed for the named queue at the named server.
If @server is given, status is displayed for all queues at that server.

server_name
Name of server. Used with the -B option to display status for that server.

STANDARD ERROR

The qstat command will write a diagnostic message to standard error for each error occurrence.

EXIT STATUS

Zero upon successful processing of all the operands.

Greater than zero if any operands could not be processed.

SEE ALSO

The PBS Professional User's Guide, the PBS Professional Administrator's Guide,
qalter(1B), qsub(1B), pbs_alterjob(3B), pbs_statjob(3B), pbs_statque(3B), pbs_statserver(3B), pbs_submit(3B), pbs_job_attributes(7B), pbs_queue_attributes(7B), pbs_server_attributes(7B), pbs_resources(7B)

NAME
qsub - submit PBS job

SYNOPSIS
qsub [-a date_time] [-A account_string] [-c interval] [-C directive_prefix] [-e path] [-h] [-I] [-j join] [-J range] [-k keep] [-l resource_list] [-m mail_options] [-M user_list] [-N name] [-o path] [-p priority] [-q destination] [-r c] [-S path_list] [-u user_list] [-v variable_list] [-V] [-W additional_attributes] [-z] [script]

qsub --version

DESCRIPTION
The qsub command is used to submit a batch job to PBS. Submitting a PBS job specifies a task, requests resources and sets job attributes.

The qsub command can read either from a job script or from standard input. When the user has submitted the job, PBS returns the job identifier for that job. For a job, this is of the form:

sequence_number.servername

For an array job, this is of the form:

sequence_number[].servername

During execution, jobs can be interactive or non-interactive.

Where PBS puts job files

By default, PBS copies the stdout and stderr files from the job back to the current working directory where the qsub command is executed. See the -o and -e options.

Submitting jobs by using scripts
To submit a PBS job script, the user types

```
qsub [options] scriptname
```

Scripts can be written in UNIX shells such as csh and sh, as well as Perl, etc. A PBS job script consists of

shell specification (for UNIX)
Any PBS directives
The user’s tasks: programs, commands or applications

UNIX:
Example of a script named “weatherscript” for a job named “Weather1” which will run the executable “weathersim”:

```
#!/bin/sh
#PBS -N Weather1
#PBS -l walltime=1:00:00
/usr/local/weathersim
```

To submit the job, the user types:

```
qsub weatherscript
```

Windows:
Example of a script named “weather.exe” for a job named “Weather1” run under Windows:

```
#PBS -N Weather1
#PBS -l walltime=1:00:00
weathersim.exe
```

To submit the job, the user types:

```
qsub weather.exe <return>
```

Scripts can contain comments. Under Windows, comments can contain only ASCII characters. See the PBS Professional User’s Guide.

Submitting jobs from standard input
To submit a PBS job by typing job specifications at the command line, the user types

```
qsub [options] <return>
```

then types any directives, then any tasks, followed by

```
(in UNIX)     CTRL-D on a line by itself
(in Windows)  CTRL-Z <return>
```

to terminate the input.

Requesting resources and placing jobs
Requesting resources includes setting limits on resource usage and controlling how the job is placed on nodes.

Resources are requested by using the -l option, either in chunks inside of selection statements, or in job-wide requests using resource_name=value pairs. See the pbs_resources(7B) man page. The selection statement is of the form:

```
-l select=[N:]chunk[+[N:]chunk ...]
```

where N specifies how many of that chunk, and a chunk is of the form:

```
resource_name=value[:resource_name=value ...]
```

Job-wide resource_name=value requests are of the form:

```
-l resource_name=value[,resource_name=value ...]
```

The place statement has this form:

```
-l place=[ arrangement ][: sharing ][: grouping]
```

where

arrangement is one of free | pack | scatter
sharing is one of excl | shared

grouping can have only one instance of group=resource

and where

free: Place job on any vnode(s).
pack: All chunks will be taken from one host.
scatter: Only one chunk with any MPI processes will be taken from a host. A chunk with no MPI processes may be taken from the same node as another chunk.
excl: Only this job uses the vnodes chosen.
shared: This job can share the vnodes chosen.
group=resource: Chunks will be grouped according to a resource. All nodes in the group must have a common value for the resource, which can be either the built-in resource host or a site-defined node-level resource.

Note that nodes can have sharing attributes that override job placement requests. See the pbs_node_attributes(7B) man page.

Do not mix old style resource or node specifications with the new select and place statements. Do not use one in a job script and the other on the command line. Mixing the two will result in an error.

For more on resource requests, usage limits and job placement, see pbs_resources(7B).

Setting attributes
The user sets job attributes by giving options to the qsub command and by using PBS directives. Each qsub option except -C, -q, and -z sets a job attribute, and has a corresponding PBS directive with the same syntax as the option. Attributes set via command-line options take precedence over those set using PBS directives. See the PBS Professional User’s Guide, pbs_job_attributes(7B).

The server’s default_qsub_arguments attribute may affect the behavior of the qsub command. The attribute is a string containing “-r y|n” and/or “-m <mail_options>”. This attribute is settable by the administrator, and is overridden by command-line arguments and script directives. See the pbs_server_attributes(1B) man page.

OPTIONS

-a date_time

Point in time after which the job is eligible for execution. Given in pairs of digits. Sets job's Execution_Time attribute to date_time. Format:

[[[[CC]YY]MM]DD]hhmm[.SS]

where CC is the century, YY is the year, MM is the month, DD is the day of the month, hh is the hour, mm is the minute, and SS is the seconds.

Each portion of the date defaults to the current date, as long as the next-smaller portion is in the future. For example, if today is the 3rd of the month and the specified day DD is the 5th, the month MM will be set to the current month.

If a specified portion has already passed, the next-larger portion will be set to one after the current date. For example, if the day DD is not specified, but the hour hh is specified to be 10:00 a.m. and the current time is 11:00 a.m., the day DD will be set to tomorrow.

-A account_string

Accounting string associated with the job. Used for labeling accounting data. Sets job's Account_Name attribute to account_string. Format: string.

-c interval

Interval at which the job will be checkpointed. Sets job's Checkpoint attribute. Ignored if checkpointing is not supported on the execution host. Default value is u.

The argument interval can take on one of the following values:

n No checkpointing is to be performed.

s   Checkpointing is to be performed only when the server is shut down.

c   Checkpointing is to be performed according to the time interval set on the server on which the job resides.

c=minutes
Checkpointing is to be performed at an interval of minutes minutes, which is the number of minutes of CPU time used by the job. Must be greater than zero. Format: integer.

u   Checkpointing is to be performed only when the server is shut down.

-C directive_prefix
Defines the prefix identifying a PBS directive. Default prefix is “#PBS”.

If the directive_prefix argument is a null string, qsub will not scan the script file for directives. Overrides the PBS_DPREFIX environment variable and the default. Cannot be used as a PBS directive.

-e path Path to be used for the job’s standard error stream. Sets job’s Error_Path attribute to path. The path argument is of the form:
[hostname:]path_name
The path will be interpreted as follows:

path_name
If path_name is a relative path, then it is taken to be relative to the current working directory of the qsub command, where it is executing on the current host.

If path_name is an absolute path, then it is taken to be an absolute path on the current host where the qsub command is executing.

hostname:path_name

If path_name is a relative path, then it is taken to be relative to the user's home directory on the host named hostname.

If path_name is an absolute path, then it is the absolute path on the host named hostname.

If path_name does not include a filename, the default filename will be

jobid.ER

If the -e option is not specified, PBS copies the standard error to the current working directory where the qsub command was executed. The default filename for the standard error stream is used. It has this form:

job_name.e<sequence number>

-h Applies a user hold to the job. Sets the job's Hold_Types attribute to "u".

-I Job is to be run interactively. Sets job's interactive attribute to TRUE. The job will be queued and scheduled as any PBS batch job, but when executed, the standard input, output, and error streams of the job are connected to the terminal session in which qsub is running. If a job script is given, only its directives are processed. When the job begins execution, all input to the job is taken from the terminal session. See the PBS Professional User's Guide for additional information on interactive jobs.

-j join Whether and how to join the job's standard error and standard output streams. Sets job's Join_Path attribute to join. Default: not merged. Possible values of join:

oe Standard error and standard output are merged into standard output.

eo Standard error and standard output are merged into standard error.

n Standard error and standard output are not merged.

-J range

Declares that this job is an array job. Sets job's array attribute to TRUE. The argument range identifies the integers greater than or equal to zero that are associated with the subjobs of the array. range is specified in the form X-Y[:Z] where X is the first index, Y is the upper bound on the indices and Z is the stepping factor. For example, 2-7:2 will produce indices of 2, 4, and 6. If Z is not specified, it is taken to be 1.

-k keep Specifies whether and which of the standard output and standard error streams will be retained on the execution host. Overrides default path names for these streams. Sets the job's Keep_Files attribute to keep. Default: neither is retained. The keep argument can take on the following values:

e The standard error stream is retained on the execution host, in the home directory of the job's owner. The filename will be:

job_name.e<sequence number>

o The standard output stream is retained on the execution host, in the home directory of the job's owner. The filename will be:

job_name.o<sequence number>

eo, oe

Both standard output and standard error streams are retained on the execution host, in the home directory of the job's owner.

n   Neither stream is retained.

-l resource_list

Allows the user to request resources and specify job placement. Sets job's Resource_list attribute to resource_list. Requesting a resource places a limit on its usage.

Requesting resources in chunks:

```
    -l select=[N:]chunk[+[N:]chunk ...]

    where N specifies how many of that chunk, and a chunk is:
        resource_name=value[:resource_name=value ...]
```

Requesting job-wide resources:

```
    -l resource_name=value[,resource_name=value ...]
```

Specifying placement of jobs:

```
    -l place=modifier[:modifier]
```

where modifier is any combination of group, excl, and/or one of free|pack|scatter.

For more on resource requests, usage limits and job placement, see pbs_resources(7B).

-m mail_options

The set of conditions under which mail about the job is sent. Format: string. Default value: "a". Sets job's Mail_Points attribute to mail_options. The mail_options argument can be either "n" or any combination of "a", "b", and "e".

n   No mail will be sent.

a   Mail is sent when the job is aborted by the batch system.

b   Mail is sent when the job begins execution.

e Mail is sent when the job terminates.

-M user_list
List of users to whom mail about the job is sent. Sets job's Mail_Users attribute to user_list. Default: job owner.
The user_list argument is of the form:
user[@host][,user[@host],...]

-N name Sets job's name to name. Sets job's Job_Name attribute to name. Format: string, up to 15 characters in length. It must consist of an alphabetic character followed by printable, non-white-space characters. Default: if a script is used to submit the job, the job's name is the name of the script. If no script is used, the job's name is “STDIN”.

-o path Path to be used for the job's standard output stream. Sets job's Output_Path attribute to path. The path argument is of the form:
[hostname:]path_name
The path will be interpreted as follows:

path_name
If path_name is a relative path, then it is taken to be relative to the current working directory of the command, where it is executing on the current host.

If path_name is an absolute path, then it is taken to be an absolute path on the current host where the command is executing.

hostname:path_name
If path_name is a relative path, then it is taken to be relative to the user's home directory on the host named hostname.

If path_name is an absolute path, then it is the absolute path on the host named hostname.

If path_name does not include a filename, the default filename will be

jobid.OU

If the -o option is not specified, PBS copies the standard output to the current working directory where the qsub command was executed. The default filename for the standard output stream is used. It has this form:

job_name.o<sequence number>

-p priority

Priority of the job. Format: host-dependent integer. Default: zero. Range: [-1024, +1023] inclusive. Sets job's Priority attribute to priority.

-q destination

Where the job is sent upon submission. Default: default queue at default server. Specifies a queue, a server, or a queue at a server. The destination argument can have one of these formats:

queue

Job is submitted to the named queue at the default server.

@server

Job is submitted to the default queue at the named server.

queue@server

Job is submitted to the named queue at the named server.

-r y|n Declares whether the job is rerunnable. See the qrerun(1B) command. Default: "y". Sets job's Rerunnable attribute to the argument. Format: single character, "y" or "n".

y Job is rerunnable.

n Job is not rerunnable.

-S path_list

Specifies the interpreter for the job script. Sets job’s Shell_Path_List attribute to path_list. Default: user’s login shell on execution node. The path_list argument is the full path to the interpreter including the executable name. Format:

```
path[@host][,path@host ...]
```

Only one path may be specified without a host name. Only one path may be specified per named host. The path selected is the one whose host name is that of the server on which the job resides.

-u user_list

List of usernames. Job will be run under a username from this list. Sets job’s User_List attribute to user_list. Default: job owner (username on submit host.) Format of user_list:

```
user[@host][,user@host ...]
```

Only one username may be specified without a host name. Only one username may be specified per named host. The server on which the job resides will select first the username whose host name is the same as the server name. Failing that, the next selection will be the username with no specified hostname. The usernames on the server and execution hosts must be the same. The job owner must have authorization to run as the specified user.

-v variable_list

Lists environment variables to be exported to the job. This is the list of environment variables which will be added to those already automatically exported. These variables exist in the user’s login environment from which qsub is run. The job’s Variable_List attribute is appended with the variables in user_list and their values. See ENVIRONMENT section of this man page. Default: no environment variables are added to job’s variable list. Format: comma-separated list of strings in the form:

    variable

or

    variable=value

-V Declares that all environment variables in the user's login environment where qsub is run are to be exported to the job. The job's Variable_List attribute is appended with all of these environment variables and their values.

-W additional_attributes

The -W option allows specification of additional job attributes. Format:

    -W attribute_name=value[,attribute_name=value...]

If white space occurs within the additional_attributes argument, or the equal sign "=" occurs within an attribute_value string, then that must be enclosed with single- or double-quotes. PBS supports the following attributes within the -W option:

depend=dependency_list

Defines dependencies between this and other jobs. Sets the job's depend attribute to dependency_list. The dependency_list has the form:

    type:arg_list[,type:arg_list ...]

where except for the on type, the arg_list is one or more PBS job IDs in the form:

    jobid[:jobid ...]

The type can be:

after: arg_list
  This job may be scheduled for execution at any point after all jobs in arg_list have started execution.

afterok: arg_list
  This job may be scheduled for execution only after all jobs in arg_list have terminated with no errors. See "Warning about exit status with csh" in EXIT STATUS.

afternotok: arg_list
  This job may be scheduled for execution only after all jobs in arg_list have terminated with errors. See "Warning about exit status with csh" in EXIT STATUS.

afterany: arg_list
  This job may be scheduled for execution after all jobs in arg_list have finished execution, with any exit status (with or without errors.) This job will not run if a job in the arg_list was killed.

before: arg_list
  Jobs in arg_list may begin execution once this job has begun execution.

beforeok: arg_list
  Jobs in arg_list may begin execution once this job terminates without errors. See "Warning about exit status with csh" in EXIT STATUS.

beforenotok: arg_list
  If this job terminates execution with errors, then jobs in arg_list may begin. See "Warning about exit status with csh" in EXIT STATUS.

beforeany: arg_list
  Jobs in arg_list may begin execution once this job terminates execution, with or without errors.

on: count
  This job may be scheduled for execution after count dependencies on other jobs have been satisfied. This type is used in conjunction with one of the before types listed. Count is an integer greater than 0.

Job IDs in the arg_list of before types must have been submitted with a type of on.

To use the before types, the user must have the authority to alter the jobs in arg_list. Otherwise, the dependency is rejected and the new job aborted.

Error processing of the existence, state, or condition of the job on which the newly submitted job is a deferred service, i.e. the check is performed after the job is queued. If an error is detected, the new job will be deleted by the server. Mail will be sent to the job submitter stating the error.

Dependency examples:
```
qsub -W depend=afterok:123.host1.domain.com /tmp/script
qsub -W depend=before:234.host1.com:235.host1.com
/tmp/script
```

group_list=g_list

List of group names. Job will be run under a group name from this list. Sets job’s group_List attribute to g_list. Default: login group name of job owner. Format of g_list:

group[@host][,group@host ...]

Only one group name may be specified without a host name. Only one group name may be specified per named host. The server on which the job resides will select first the group name whose host name is the same as the server name. Failing that, the next selection will be the group name with no specified hostname. The group names on the server and execution hosts must be the same.

block=true

Specifies that qsub waits for the job to terminate, then returns the job's exit value. Sets job's block attribute to TRUE. Cannot be used with interactive jobs. See EXIT VALUES section.

stagein=file_list
stageout=file_list

Specifies files to be staged-in before execution or staged-out after execution is complete. Sets the job's stagein and stageout attributes the the specified file_list. On completion of the job, all staged-in and staged-out files are removed from the execution host(s). The file_list has the form:

filespec[,filespec]

where filespec is

local_file@hostname:remote_file

regardless of the direction of the copy. The name local_file is the name of the file on the execution host(s). It may be an absolute path or relative to the user's home directory on the execution host. The name remote_file is the path on hostname. The name may be absolute or relative to the user's home directory on the hostname. If file_list has more than one filespec, i.e. it contains commas, it must be enclosed in double-quotes. The use of wildcards in the file name is not supported.

umask=NNNN

The umask with which the job will be started. Default value: 077. Can be used with one to four digits; typically two. Sets job's umask attribute to NNNN. Controls umask of job's standard output and standard error. Example: -W umask=33 allows group and world read on the job's output.

-z Job identifier is not written to standard output.

--version
The qsub command returns its PBS version information and exits. This option can only be used alone.

OPERANDS
The qsub command accepts a script or a dash "-" as operands.

script

Path to script. Can be absolute or relative to current directory where qsub is run.

-

Any PBS directives and user tasks are read from the command line. Same as for no operands.

STANDARD OUTPUT

Unless the -z option is set, the job identifier assigned to the job will be written to standard output if the job is successfully created.

STANDARD ERROR

The qsub command will write a diagnostic message to standard error for each error occurrence.

ENVIRONMENT VARIABLES

The qsub command uses the following:

PBS_DEFAULT

Name of default server.

PBS_DPREFIX

Prefix string which identifies PBS directives.

Environment variables beginning with “PBS_O_” are created by qsub. PBS automatically exports the following environment variables to the job, and the job’s Variable_List attribute is set to this list:

PBS_O_HOME

User’s home directory. Value of HOME taken from user’s submission environment.

PBS_O_LANG

Value of LANG taken from user’s submission environment.

PBS_O_LOGNAME

User’s login name. Value of LOGNAME taken from user’s submission environment.

PBS_O_PATH

User's PATH. Value of PATH taken from user's submission environment.

PBS_O_MAIL

Value of MAIL taken from user's submission environment.

PBS_O_SHELL

Value taken from user's submission environment.

PBS_O_TZ

Value taken from user's submission environment.

PBS_O_HOST

Name of submit host. Value taken from user's submission environment.

PBS_O_QUEUE

Name of the queue to which the job was submitted. Value taken from user's submission environment.

PBS_O_SYSTEM

Operating system, from uname -s, on submit host. Value taken from user's submission environment.

PBS_O_WORKDIR

Absolute path to directory where qsub is run. Value taken from user's submission environment.

PBS_ENVIRONMENT

Set to PBS_BATCH for a batch job. Set to PBS_INTERACTIVE for an interactive job. Created upon execution.

PBS_JOBID
Job identifier given by PBS when the job is submitted. Created upon execution.

PBS_JOBNAME
Job name given by user. Created upon execution.

PBS_NODEFILE
Name of file containing the list of nodes assigned to the job. Created upon execution.

PBS_QUEUE
Name of the queue from which the job is executed. Created upon execution.

EXIT STATUS

Zero upon successful processing of input. Exit value will be greater than zero upon failure of qsub.

For blocking jobs, qsub will exit and return the exit value of the job. If the job is deleted without being run, qsub returns an exit value of 3.

Warning about exit status with csh:
If a job is run in csh and a .logout file exists in the home directory in which the job executes, the exit status of the job is that of the .logout script, not the job script. This may impact any inter-job dependencies.

SEE ALSO

The PBS Professional User's Guide, the PBS Professional Administrator's Guide, pbs_job_attributes(7B), pbs_server_attributes(7B), pbs_resources(7B), qalter(1B), qhold(1B), qmove(1B), qmsg(1B), qrerun(1B), qrls(1B), qselect(1B), qstat(1B)

NAME

xpbs - GUI front end to PBS commands

SYNOPSIS

xpbs [-admin]

xpbs --version

DESCRIPTION

The xpbs command provides a user-friendly point-and-click interface to PBS commands. Please see the sections below for a tour and tutorials. Also, within every dialog box, a Help button can be found for assistance.

OPTIONS

-admin A mode where additional buttons are made available for terminating PBS servers, starting/stopping/disabling/enabling queues, and running/rerunning jobs.

--version
The xpbs command returns its PBS version information and exits. This option can only be used alone.

GETTING STARTED

Running xpbs will initialize the X resource database from various sources in the following order:

1. The RESOURCE_MANAGER property on the root window (updated via xrdb) with settings usually defined in the .Xdefaults file

2. Preference settings defined by the system administrator in the global xpbsrc file

3. User’s ~/.xpbsrc file - this file defines various X resources

like fonts, colors, list of PBS hosts to query, criteria for listing queues and jobs, and various view states. See PREFERENCES section below for a list of resources that can be set.

RUNNING XPBS

To run xpbs as a regular, non-privileged user, type:

```
setenv DISPLAY <display_host>:0
xpbs
```

To run xpbs with the additional purpose of terminating PBS servers, stopping and starting queues, or running/rerunning jobs, then run:

```
xpbs -admin
```

NOTE: Be sure to appropriately set ~/.rhosts file if you're planning to submit jobs to some remote server, and expecting output files to be returned to the local host (where xpbs was run). Usually, adding the PBS hostname running the server to your .rhosts file locally, and adding the name of the local machine to the .rhosts file at remote host, should be sufficient.

Also, be sure that the PBS client commands are in the default PATH because xpbs will call these commands.

THE XPBS DISPLAY

This section describes the main parts of the xpbs display. The main window is composed of 5 distinct areas (subwindows) arranged vertically (one on top of another) in the following order:

1) Menu
2) Hosts
3) Queues
4) Jobs
5) Info

Menu. The Menu area is composed of a row of command buttons that signal some action with a click of the left mouse button. The buttons are:

Manual Update	to update the information on hosts, queues, and jobs.
Auto Update	same as Manual Update except updating is done automatically every <some specified> number of minutes.
Track Job	for periodically checking for returned output files of jobs.
Preferences	for setting certain parameters such as the list of server host(s) to query.
Help	contains some help information.
About	tells of the author and who to send comments, bugs, suggestions to.
Close	for exiting xpbs plus saving the current setup information (if anything had changed) in the user's $HOME/.xpbsrc file. Information saved include the selected host(s), queue(s), job(s), the different jobs listing criteria, the view states (i.e. minimized/maximized) of the Hosts, Queues, Jobs, and INFO regions, and anything in the Preferences section.

Hosts. The Hosts area is composed of a leading horizontal HOSTS bar, a listbox, and a set of command buttons. The HOSTS bar contains a minimize/maximize button, identified by a dot or a rectangular image, for displaying or iconizing the Hosts region. The listbox displays information about favorite server host(s), and each entry is meant to be selected via a single left mouse button click, shift key + mouse button 1 click for contiguous selection, or cntrl key + mouse button 1 click for non-contiguous selection. The command buttons represent actions on selected host(s), and commonly found buttons are:

detail for obtaining detailed information about selected server host(s). This functionality can also be achieved by double clicking on an entry in the Hosts listbox.

Submit for submitting a job to any of the queues managed by the selected host(s).

terminate for terminating PBS servers on selected host(s). (-admin only)

The server hosts can be chosen by specifying in the ~/.xpbsrc file (or .Xdefaults) the resource:

```
*serverHosts:  hostname1 hostname2 ...
```

Another way of specifying the host is to click on the Preferences button in the Menu region, and manipulate the server Hosts entry widget from the preferences dialog box.

Queues. The Queues area is composed of a leading horizontal QUEUES bar, a listbox, and a set of command buttons. The QUEUES bar lists the hosts that are consulted when listing queues; the bar also contains a minimize/maximize button for displaying or iconizing the Queues region. The listbox displays information about queues managed by the server host(s) selected from the Hosts listbox; each listbox entry is meant to be selected (highlighted) via a single left mouse button click, shift key + mouse button 1 click for contiguous selection, or cntrl key + mouse button 1 click for non-contiguous selection. The command buttons represent actions for operating on selected queue(s), and commonly found buttons are:

detail for obtaining detailed information about selected queue(s). This functionality can also be achieved by double clicking on a Queues listbox entry.

stop for stopping the selected queue(s). (-admin only)

start for starting the selected queue(s). (-admin only) disable for disabling the selected queue(s). (-admin only)

enable for enabling the selected queue(s). (-admin only)

Jobs. The Jobs area is composed of a leading horizontal JOBS bar, a listbox, and a set of command buttons. The JOBS bar lists the queues that are consulted when listing jobs; the bar also contains a minimize/maximize button for displaying or iconizing the Jobs region. The listbox displays information about jobs that are found in the queue(s) selected from the Queues listbox; each listbox entry is meant to be selected (highlighted) via a single left mouse button click, shift key + mouse button 1 click for contiguous selection, or cntrl key + mouse button 1 click for non-contiguous selection. The region just above the Jobs listbox shows a collection of command buttons whose labels describe criteria used for filtering the Jobs listbox contents. The list of jobs can be selected according to the owner of jobs (Owners), job state (Job_States), name of the job (Job_Name), type of hold placed on the job (Hold_Types), the account name associated with the job (Account_Name), checkpoint attribute (Checkpoint), time the job is eligible for queueing/execution (Queue_Time), resources requested by the job (Resources), priority attached to the job (Priority), and whether or not the job is rerunnable (Rerunnable). The selection criteria can be modified by clicking on any of the appropriate command buttons to bring up a selection box. The criteria command buttons are accompanied by a Select Jobs button, which when clicked, will update the contents of the Jobs listbox based on the new selection criteria. Please see qselect(1B) for more details on how the jobs are filtered.

Finally, to the right of the listbox, the Jobs region is accompanied by the following command buttons, for operating on selected job(s):

detail for obtaining detailed information about selected job(s). This functionality can also be achieved by double clicking on a Jobs listbox entry.

modify for modifying attributes of the selected job(s).

delete for deleting the selected job(s).

hold for placing some type of hold on selected job(s).

release for releasing held job(s).

signal for sending signals to selected job(s) that are running.

msg for writing a message string into the output streams of the selected job(s).

move for moving selected job(s) into some specified destination queue.

order for exchanging order of two selected jobs in a queue.

run for running selected job(s). (-admin only)

rerun for requeueing selected job(s) that are running. (-admin only)

Info. The Info Area shows the progress of the commands' executed by xpbs. Look into this box for errors. The INFO bar also contains a minimize/maximize button for displaying or iconizing the Info region.

WIDGETS USED IN XPBS

Some of the widgets used in xpbs and how they are manipulated are described in the following:

1. listbox - can be multi-selectable (a number of entries can be selected/highlighted using a mouse click) or single-selectable (one entry can be highlighted at a time). For a multi-selectable listbox, the following operations are allowed:

   a. single click with mouse button 1 to select/highlight an entry.

   b. shift key + mouse button 1 to contiguously select more than one entry.

c. cntrl key + mouse button 1 to non-contiguously select more than one entry. NOTE: For systems running Tk < 4.0, the newly selected item is reshuffled to appear next to already selected items.

d. click the Select All/Deselect All button to select all entries or deselect all entries at once.

e. double clicking an entry usually activates some action that uses the selected entry as a parameter.

2. scrollbar - usually appears either vertically or horizontally and contains 5 distinct areas that are mouse clicked to achieve different effects:

top arrow	Causes the view in the associated widget to shift up by one unit (i.e. the object appears to move down one unit in its window). If the button is held down the action will auto-repeat.
top gap	Causes the view in the associated window to shift up by one less than the number of units in the window (i.e. the portion of the object that used to appear at the very top of the window will now appear at the very bottom). If the button is held down the action will auto-repeat.
slider	Pressing button 1 in this area has no immediate effect except to cause the slider to appear sunken rather than raised. However, if the mouse is moved with the button down then the slider will be dragged, adjusting the view as the mouse is moved.
bottom gap	Causes the view in the associated window to shift down by one less than the number of units in the window (i.e. the portion of the object that used to appear at the very bottom of the window will now appear at the very top). If the button is held down the action will auto-repeat.

bottom arrow Causes the view in the associated window to shift down by one unit (i.e. the object appears to move up one unit in its window). If the button is held down the action will auto-repeat.

3. entry - brought into focus with a click of the left mouse button. To manipulate this widget, simply type in the text value. Use of arrow keys, mouse selection of text for deletion or overwrite, copying and pasting with sole use of mouse buttons are permitted. This widget is usually accompanied by a scrollbar for horizontally scanning a long text entry string.

4. matrix of entry boxes - usually shown as several rows of entry widgets where a number of entries (called fields) can be found per row. The matrix is accompanied by up/down arrow buttons for paging through the rows of data, and each group of fields gets one scrollbar for horizontally scanning long entry strings. Moving from field to field can be done using the <Tab>, <Cntrl-f>, or <Cntrl-b> (move backwards) keys.

5. spinbox - a combination of an entry widget and a horizontal scrollbar. The entry widget will only accept values that fall within a defined list of valid values, and incrementing through the valid values is done by clicking on the up/down arrows.

6. button - a rectangular region appearing either raised or pressed that invokes an action when clicked with the left mouse button. When the button appears pressed, then hitting the <RETURN> key will automatically select the button.

7. text - an editor like widget. This widget is brought into focus with a click of the left mouse button. To manipulate this widget, simply type in the text. Use of arrow keys, backspace/delete key, mouse selection of text for deletion or overwrite, copying and pasting with sole use of mouse buttons are permitted. This widget is usually accompanied by a scrollbar for vertically scanning a long

entry.

SUBMITTING JOBS

Submitting a PBS job requires only to manipulate the widgets found in the Submit window. The submit dialog box is composed of 4 distinct regions:

1) Job Script
2) OPTIONS
3) OTHER OPTIONS
4) Command Buttons

The Job Script file region is at the upper left, the OPTIONS region containing various widgets for setting job attributes is scattered all over the dialog box, the OTHER OPTIONS is located just below the Job Script file region, and Command Buttons region is at the bottom.

The job script region is composed of a header box, the text box, FILE entry box, and a couple of buttons labeled load and save. If you have a script file containing PBS options and executable lines, then type the name of the file on the FILE entry box, and then click on the load button. The various widgets in the Submit window will get loaded with values found in the script file. The script file text box will only be loaded with executable lines (non-PBS) found in the script. The job script header box has a Prefix entry box that can be modified to specify the PBS directive to look for when parsing a script file for PBS options. If you don't have a script file, you can start typing the executable lines of the job in the file text box.

To submit a job, perform the following steps:

1. Select a host from the HOSTS listbox in the main xpbs display.

2. Click on the Submit button located in the Menu bar.

3. Specify the script file containing the job execution lines and job resource and attribute values, or simply type in the execution lines in the FILE textbox.

4. Start manipulating the various widgets in the Submit window. Particularly, pay close attention to the Destination listbox. This box lists all the queues found in the host that you selected. A special entry called "@host" refers to the default queue at host. Select appropriately the destination queue of the job. More options can be found by clicking the OTHER OPTIONS buttons.

5. At the bottom of the Submit window, click confirm submit . You can also click on interactive to run the job interactively. Running a job interactively will open an xterm window to your display host containing the session.

   NOTE: The script FILE entry box is accompanied by a save button that you click to save the current widget values to the specified file in a form that can later be read by xpbs or by the qsub command.

MODIFYING ATTRIBUTES OF JOBS

Modifying a PBS job requires only to manipulate the widgets found in the Modify window. To modify a job or jobs, do the following steps:

1. Select one or more jobs from the JOBS listbox in the main xpbs display.

2. Click on the modify button located to the right of the listbox.

3. The Modify window is structured similarly to the Submit window. Simply manipulate the widgets to specify replacement or additional values of job attributes.

4. Click on the confirm modify button located at the bottom of the dialog box.

DELETING JOBS

Deleting a PBS job requires only to manipulate the widgets found in the Delete window. To delete a job or jobs, do the following steps:

1. Select one or more jobs from the JOBS listbox in the main xpbs display.

2. Click on the delete button located to the right of the list-box.

3. Manipulate the spinbox widget to set the kill delay signal interval.

4. Click on the delete button located at the bottom of the dialog box.

TRACKING RETURNED OUTPUT FILES

If you want to be informed of returned output files of current jobs, and be able to quickly see the contents of those files, then enable the "track job" feature as follows:

1. Submit all the jobs that you want monitored.

2. Click on the Track Job button located in the Menu bar to bring up the Track Job dialog box.

3. Specify the list of user names, whose jobs are to be monitored for returned output files, in the matrix located at the upper left of the dialog box.

4. Manipulate the minutes spinbox, located just below the user names matrix, to specify the interval value when output files will be periodically checked.

5. Specify the location of job output files (whether locally or remotely) by clicking on one of the radio buttons located at the upper right of the dialog box. Returned locally means the output files will be returned back to the host where xpbs was run. If the output files are

returned to some remote host, then xpbs will execute an

```
RSH <remote_host> test -f <output_files>
```

to test the existence of the files. RSH is whatever you set the remote shell command to in the corresponding entry box.

NOTE: Be sure the files are accessible from the host where xpbs was run (i.e. .rhosts appropriately set).

6. Click start/reset tracking button located at the bottom of the dialog box to:

   - cancel any previous tracking

   - build a new list of jobs to be monitored for returned output files based on currently queued jobs.

   - start periodic tracking.

7. Click on close window button.

When an output file for a job being monitored is found, then the Track Job button (the one that originally invoked the Track Job dialog box) will turn into a different color, and the Jobs Found Completed listbox, located in the Track Job dialog box, is then loaded with the corresponding job id(s). Then double click on a job id to see the contents of the output file and the error file. Click stop tracking if you want to cancel tracking.

LEAVING XPBS

Click on the Close button located in the Menu bar to leave xpbs. If anything had changed, it will bring up a dialog box asking for a confirmation in regards to saving state information like the view states (minimize/maximize) of the HOSTS, QUEUES, JOBS, and INFO subwindows, and various criteria for listing queues and jobs. The information is saved in ~/.xpbsrc file.

PREFERENCES

The resources that can be set in the X resources file, ~/.xpbsrc, are:

*serverHosts
list of server hosts (space separated) to query by xpbs keyword PBS_DEFAULT_SERVER can be used which will be used as a place holder for the value obtained from *defServerFile.

*defServerFile
the file containing the name of the default server host. The content of this will be substituted for the PBS_DEFAULT_SERVER keyword in *serverHosts value.

*timeoutSecs
specify the number of seconds before timing out waiting for a connection to a PBS host.

*xtermCmd
the xterm command to run driving an interactive PBS session.

*labelFont
font applied to text appearing in labels.

*fixlabelFont
font applied to text that label fixed-width widgets such as listbox labels. This must be a fixed-width font.

*textFont
font applied to a text widget. Keep this as fixed-width font.

*backgroundColor
the color applied to background of frames, buttons, entries, scrollbar handles.

*foregroundColor
the color applied to text in any context (under selection, insertion, etc...).

*activeColor
the color applied to the background of a selection, a selected

command button, or a selected scroll bar handle.

*disabledColor
color applied to a disabled widget.

*signalColor
color applied to buttons that signal something to the user about a change of state. For example, the color of the Track Job button when returned output files are detected.

*shadingColor
a color shading applied to some of the frames to emphasize focus as well as decoration.

*selectorColor
the color applied to the selector box of a radiobutton or check-button.

*selectHosts
list of hosts (space separated) to automatically select/high-light in the HOSTS listbox.

*selectQueues
list of queues (space separated) to automatically select/high-light in the QUEUES listbox.

*selectJobs
list of jobs (space separated) to automatically select/highlight in the JOBS listbox.

*selectOwners
list of owners checked when limiting the jobs appearing on the Jobs listbox in the main xpbs window. Specify value as "Owners: <list_of_owners>". See -u option in qselect(1B) for format of <list_of_owners>.

*selectStates
list of job states to look for (do not space separate) when limiting the jobs appearing on the Jobs listbox in the main xpbs window. Specify value as "Job_States: <states_string>". See -s

option in qselect(1B) for format of <states_string>.

*selectRes
list of resource amounts (space separated) to consult when limiting the jobs appearing on the Jobs listbox in the main xpbs window. Specify value as "Resources: <res_string>". See -l option in qselect(1B) for format of <res_string>.

*selectExecTime
the Execution Time attribute to consult when limiting the list of jobs appearing on the Jobs listbox in the main xpbs window. Specify value as "Queue_Time: <exec_time>". See -a option in qselect(1B) for format of <exec_time>.

*selectAcctName
the name of the account that will be checked when limiting the jobs appearing on the Jobs listbox in the main xpbs window. Specify value as "Account_Name: <account_name>". See -A option in qselect(1B) for format of <account_name>.

*selectCheckpoint
the checkpoint attribute relationship (including the logical operator) to consult when limiting the list of jobs appearing on the Jobs listbox in the main xpbs window. Specify value as "Checkpoint: <checkpoint_arg>". See -c option in qselect(1B) for format of <checkpoint_arg>.

*selectHold
the hold types string to look for in a job when limiting the jobs appearing on the Jobs listbox in the main xpbs window. Specify value as "Hold_Types: <hold_string>". See -h option in qselect(1B) for format of <hold_string>.

*selectPriority
the priority relationship (including the logical operator) to consult when limiting the list of jobs appearing on the Jobs listbox in the main xpbs window. Specify value as "Priority: <priority_value>". See -p option in qselect(1B) for format of <priority_value>.

*selectRerun

the rerunnable attribute to consult when limiting the list of jobs appearing on the Jobs listbox in the main xpbs window. Specify value as “Rerunnable: <rerun_val>”. See -r option in qselect(1B) for format of <rerun_val>.

*selectJobName
name of the job that will be checked when limiting the jobs appearing on the Jobs listbox in the main xpbs window. Specify value as “Job_Name: <jobname>”. See -N option in qselect(1B) for format of <jobname>.

*iconizeHostsView
a boolean value (true or false) indicating whether or not to iconize the HOSTS region.

*iconizeQueuesView
a boolean value (true or false) indicating whether or not to iconize the QUEUES region.

*iconizeJobsView
a boolean value (true or false) indicating whether or not to iconize the JOBS region.

*iconizeInfoView
a boolean value (true or false) indicating whether or not to iconize the INFO region.

*jobResourceList
a curly-braced list of resource names as according to architecture known to xpbs. The format is as follows:

```
{ <arch-type1> resname1 resname2 ... resnameN }
{ <arch-type2> resname1 resname2 ... resnameN }
 . . .
{ <arch-typeN> resname1 resname2 ... resnameN }
```

XPBS AND PBS COMMANDS
xpbs calls PBS commands as follows:

Command Button “ 22	PBS Command
detail (Hosts)	qstat -B -f <selected server_host(s)>
terminate	qterm <selected server_host(s)>
detail (Queues)	qstat -Q -f <selected queue(s)>
stop	qstop <selected queue(s)>
start	qstart <selected queue(s)>
enable	qenable <selected queue(s)>
disable	qdisable <selected queue(s)>
detail (Jobs)	qstat -f <selected job(s)>
modify	qalter <selected job(s)>
delete	qdel <selected job(s)>
hold	qhold <selected job(s)>
release	qrls <selected job(s)>
run	qrun <selected job(s)>
rerun	qrerun <selected job(s)>
signal	qsig <selected job(s)>
msg	qmsg <selected job(s)>
move	qmove <selected job(s)>
order	qorder <selected job(s)>

EXIT STATUS

Upon successful processing, the xpbs exit status will be a value of zero.

If the xpbs command fails, the command exits with a value greater than zero.

SEE ALSO

The PBS Professional User's Guide, the PBS Professional Administrator's Guide,
qalter(1B), qdel(1B), qhold(1B), qmove(1B), qmsg(1B), qrerun(1B), qrls(1B), qselect(1B), qsig(1B), qstat(1B), qorder(1B), qsub(1B), qdisable(8B), qenable(8B), qrun(8B), qstart(8B), qstop(8B), qterm(8B).

NAME

xpbsmon - GUI for displaying, monitoring execution hosts under PBS

SYNOPSIS

xpbsmon

xpbsmon --version

DESCRIPTION

The xpbsmon command provides a way to graphically display the various nodes that run jobs. A node or execution host can be running a pbs_mom daemon, or not running the daemon. For the latter case, it could just be a nodename that appears in a nodes file that is managed by a main pbs_server running on another host. This utility also provides the ability to monitor values of certain system resources by posting queries to the pbs_mom of a node. With this utility, you can see what job is running on what node, who owns the job, how many nodes assigned to a job, status of each node (color-coded and the colors are user-modifiable), how many nodes are available, free, down, reserved, offline, of unknown status, in use running multiple jobs or executing only 1 job. Please see the sections below for a tour and tutorials of xpbsmon. Also, within every dialog box, a Help button can be found for assistance.

GETTING STARTED

Running xpbsmon will initialize the X resource database from various sources in the following order:

1. The RESOURCE_MANAGER property on the root window (updated via xrdb) with settings usually defined in the .Xdefaults file

2. Preference settings defined by the system administrator in the global xpbsmonrc file

3. User's ~/.xpbsmonrc file - this file defines various X resources like fonts, colors, list of colors to use to represent the various status of the nodes, list of PBS sites to query, list of server hosts on each site, list of nodes/execution hosts on each server host, list of system resource queries to send to the nodes' pbs_mom, and various view states. See PREFERENCES section below for a list of resources that can be set.

RUNNING XPBSMON

xpbsmon can be run either as a regular user or superuser. If you run it with less privilege, you may not be able to see all the information for a node. If it is executed as a regular user, you should still be able to see what jobs are running on what nodes, possibly state, as this information are obtained by xpbsmon talking directly to the specified server. If you want other system resource values, it may require special privilege since xpbsmon will have to talk directly to the pbs_mom of a node. In addition, the host where xpbsmon was running must also have been given explicit access permission by the mom (unless the GUI is running on the same host where mom is running). This is done done by updating the $clienthost and/or the $restricted parameter on the mom's configuration file.

To run xpbsmon, type:

```
setenv DISPLAY <display_host>:0
xpbsmon
```

If you are running the GUI and only interested in jobs data, then be sure to set all the nodes' type to NOMOM in the Pref dialog box.

OPTIONS

--version

The xpbsmon command returns its PBS version information and exits. This option can only be used alone.

THE XPBSMON DISPLAY

This section describes the main parts of the xpbsmon display. The main window is composed of 3 distinct areas (subwindows) arranged vertically (one on top of another) in the following order:

1) Menu
2) Site Information
3) Info

Menu. The Menu area is composed of a row of command buttons that signal some action with a click of the left mouse button. The buttons are:

Site.. displays a popup menu containing the list of PBS sites that have been added using the Sites Preferences window. Simply drag your mouse and release to the site name whose servers/nodes information you would like to see.

Pref.. brings up various dialog boxes for specifying the list of sites, servers on each site, nodes that are known to a server, and the system resource queries to be sent to a node's pbs_mom daemon.

Auto Update.. brings up another window for specifying whether or not to do auto updates of nodes information, and also for specifying the interval number of minutes between updates.

Help contains some help information.

About tells who the author is and who to send comments, bugs, suggestions to.

Close for exiting xpbsmon plus saving the current setup information (if anything had changed) in the user's $HOME/.xpbsmonrc file. Information saved include the specified list of sites, servers on each site, nodes known to each server, and system resource queries to send to node's pbs_mom.

Minimize Button
shows the iconized view of Site Information where nodes are represented as tiny boxes, where each box is colored according to status. In order to get more information about a node, you need to double click on the colored box.

Maximize button
shows the full view of Site Information where nodes are represented in bigger boxes, still colored depending on the status, and some informa-

tion on it is displayed.

Site Information. Only one site at a time can be displayed. This area (shown as one huge box referred to as the site box) can be further subdivided into 3 areas: the site name label at the top, server boxes in the middle, and the color status bar at the bottom. The site name label shows the name of the site as specified in the Pref.. window. At the middle of the site box shows a row of big boxes housing smaller boxes.

The big box is an abstraction of a server host (called a server box), showing its server display label at the top of the box, a grid of smaller boxes representing the nodes that the server knows about (where jobs are run), and summary status for the nodes under the server. Status information will show counters for the number of nodes used, available, reserved, offline, or of unknown status and even # of cpus assigned. For a cleaner display, some counters with a value of zero are not displayed. The server boxes are placed in a grid, with a new row being started when either *siteBoxMaxNumServerBoxesPerRow or *siteBoxMaxWidth limit has been reached.

The smaller boxes represent the nodes/execution hosts where jobs are run (referred to as node boxes). Each node box shows the name at the top, and a sub-box (a smaller square) that is colored according to the status of the node that it represents, and if the view type is FULL, it will display some node information according to the system resource queries specified on the Pref.. window. Clicking on the sub-box will show a much bigger box (called the MIRROR view) with bigger fonts containing nodes information. Another view is called ICON and this shows a tiny box with a colored area. The node boxes are arranged in a grid, where a new row is created if either the *serverBoxMaxNumNodeBoxesPerRow or *serverBoxMaxWidth limit has been reached. ICON view of the node boxes will be constrained by the *nodeBoxIconMaxHeight and *nodeBoxIconMaxWidth pixel values; FULL view of the node boxes will be bounded by *nodeBoxFullMaxWidth and *nodeBoxFullMaxHeight; the mirror view of the node boxes has its size be *nodeBoxMirrorMaxWidth, and *nodeBoxMirrorMaxHeight.

Horizontal and vertical scrollbars for the site box, server box, and node box will be displayed as needed.

Finally, the color bar information shows a color chart displaying what the various colors mean in terms of node status. The color-to-status mapping can be modified by setting the X resources: *nodeColorNOINFO, *nodeColorFREE, *nodeColorINUSEshared, *nodeColorINUSEexclusive, *node-ColorDOWN, *nodeColorRSVD, *nodeColorOFFL, *nodeColorBUSY.

Info. The Info Area shows the progress of some of the background actions performed by xpbsmon. Look into this box for errors.

WIDGETS USED IN XPBSMON

Some of the widgets used in xpbsmon and how they are manipulated are described in the following:

1. listbox - the ones found in this GUI are only single-selectable (one entry can be highlighted/selected at a time via a mouse click).

2. scrollbar - usually appears either vertically or horizontally and contains 5 distinct areas that are mouse clicked to achieve different effects:

   top arrow — Causes the view in the associated widget to shift up by one unit (i.e. the object appears to move down one unit in its window). If the button is held down the action will auto-repeat.

   top gap — Causes the view in the associated window to shift up by one less than the number of units in the window (i.e. the portion of the object that used to appear at the very top of the window will now appear at the very bottom). If the button is held down the action will auto-repeat.

   slider — Pressing button 1 in this area has no immediate effect except to cause the slider to appear sunken rather than raised. However, if the mouse is moved with the button down then the slider will be dragged, adjusting the view as the mouse is moved.

bottom gap Causes the view in the associated window to shift down by one less than the number of units in the window (i.e. the portion of the object that used to appear at the very bottom of the window will now appear at the very top). If the button is held down the action will auto-repeat.

bottom arrow Causes the view in the associated window to shift down by one unit (i.e. the object appears to move up one unit in its window). If the button is held down the action will auto-repeat.

3. entry - brought into focus with a click of the left mouse button. To manipulate this widget, simply type in the text value. Use of arrow keys, mouse selection of text for deletion or overwrite, copying and pasting with sole use of mouse buttons are permitted. This widget is usually accompanied by a scrollbar for horizontally scanning a long text entry string.

4. box - made up of 1 or more listboxes displayed adjacent to each other giving the effect of a “matrix”. Each row from the listboxes makes up an element of the box. In order to add items to the box, you need to manipulate the accompanying entry widgets, one for each listbox, and then clicking the add button. Removing items from the box is done by selecting an element, and then clicking delete.

5. spinbox - a combination of an entry widget and a horizontal scrollbar. The entry widget will only accept values that fall within a defined list of valid values, and incrementing through the valid values is done by clicking on the up/down arrows.

6. button - a rectangular region appearing either raised or pressed that invokes an action when clicked with the left mouse button. When the button appears pressed, then hitting the <RETURN> key will automatically select the button.

UPDATING PREFERENCES

CASE 1: Time Sharing

Suppose you have a time-sharing environment where the front-end is called bower and you have 4 nodes: bower1, bower2, bower3, bower4. bower is the host that runs the server; jobs are submitted to host bower where it enqueues it for future execution. Also, a pbs_mom daemon is running on each of the execution hosts. If the server bower also maintains a nodes list containing information like state for the 4 nodes, then this will also be reported. Then to setup xpbsmon, do the following:

1. Click the Pref.. button on the Menu section.

2. On the Sites Preference dialog, enter any arbitrary site name, for example “Local”. Then click the add button.

3. On the Server_Host entry box, enter “bower”, and on the DisplayLabel entry box, put an arbitrary label (as it would appear on the header of the server box) like “Bower”, and then click add.

4. Click the nodes.. button that is accompanying the Servers box. This would bring up the Server Preference dialog.

5. Now add the entries “bower1”, “bower2”, “bower3”, “bower4” specifying type MOM for each on the Nodes box.

6. If you need to monitor certain system resource parameters for each of the nodes, you need to specify query expressions containing resource queries to be sent to the individual PBS moms. For example, if you want to obtain memory usage, then select a node from the Nodes list, click on the query.. button that accompanies the Nodes list, and this would bring up the Query Table dialog. Specify the following input:

   ```
   Query_Expr:    (availmem/totmem) * 100
   Display_Info:  Memory Usage:
   Display_Type:  SCALE
   ```

The above says to display the result of the "Query_Expr" in a scale widget calibrated over 100. The queries "availmem" and "totmem" will be sent to the PBS mom, and the expression is evaluated upon receiving all results from the mom. If you want to display the result of another query, say "loadave", directly, then specify the following:

Query_Expr: loadave
Display_Info: Load Average:
Display_Type: TEXT

NOTE: For a list of queries that can be sent to a pbs_mom, please click on the Help button on the Query table window.

CASE 2: Jobs Exclusive Environment

Supposing you have a "space non-sharing" environment where the server maintains a list of nodes that it runs jobs on exclusively (one job at a time outstanding per node). Let's call this server b1. Simply update Preferences information as follows:

1. Click the Pref.. button on the Menu section.

2. On the Sites Preference dialog, enter a site name, for example "B System". Then click the add button.

3. On the Server_Host entry box, enter "b1", DisplayLabel entry box type "B1" (or whatever label that you would like to appear on the header of the server box), and then click add.

CASE 3: Hybrid Time Sharing/Space Sharing Environment

A cluster of heterogeneous machines, time-sharing or jobs exclusive, could easily be represented in xpbsmon by combining steps in CASE 1 and CASE 2.

LEAVING XPBSMON

Click on the Close button located in the Menu bar to leave xpbsmon. If anything had changed, it will bring up a dialog box asking for a confirmation in regards to saving preferences information about list of sites, their view types, list of servers on each site, the list of nodes known to each server, and the list of queries to be sent to the pbs_mom of each node. The information is saved in ~/.xpbsmonrc file.

PREFERENCES

The resources that can be set in the X resources file, ~/.xpbsmonrc, are described in the following:

Node Box Properties

Resource names beginning with "*small" or "*node" apply to the properties of the node boxes. A node box is made of an outer frame where the node label sits on top, the canvas (smaller box) is on the middle, and possibly some horizontal/ vertical scrollbars.

nodeColorNOINFO
: color of node box when information for the node it represents could not be obtained.

*nodeColorFREE
: color of canvas when node it represents is up.

*nodeColorINUSEshared
: color when node it represents has more than 1 job running on it, or when node has been marked by the server that manages it as "job-sharing".

*nodeColorINUSEexclusive
: list of colors to assign to a node box when host it represents is running only 1 job, or when node has been marked by the server that manages it as "time-sharing". xpbsmon will use this list to assign 1 distinct color per job unless all the colors have been exhausted, in which case, colors will start getting assigned more than once in a round-robin fashion.

*nodeColorDOWN

color when node it represents is down.

*nodeColorRSVD
color when node it represents is reserved.

*nodeColorOFFL
color when node it represents is offline.

*nodeColorBUSY
color when node it represents is busy (high load average).

*smallForeground
applies to the color of text inside the canvas.

*smallBackground
applies to the color of the frame.

*smallBorderWidth
distance (in pixels) from other node boxes.

*smallRelief
how node box will visually appear (style).

*smallScrollBorderWidth
significant only in FULL mode, this is the distance of the horizontal/vertical scrollbars from the canvas and lower edge of the frame.

*smallScrollBackground
background color of the scrollbars

*smallScrollRelief
how scrollbars would visually appear (style).

*smallCanvasBackground
color of the canvas (later overridden depending on status of the node it represents)

*smallCanvasBorderWidth
distance of the canvas from the frame and possibly the scroll-

bars.

*smallCanvasRelief
how the canvas is visually represented (style).

*smallLabelBorderWidth
the distance of the node label from the canvas and the topmost edge of the frame.

*smallLabelBackground
the background of the area of the node label that is not filled.

*smallLabelRelief
how the label would appear visually (style).

*smallLabelForeground
the color of node label text.

*smallLabelFont
the font to use for the node label text.

*smallLabelFontWidth
font width (in pixels) of *smallLabelFont

*smallLabelFontHeight
font height (in pixels) of *smallLabelFont

*smallTextFont
font to use for the text that appear inside a canvas.

*smallTextFontWidth
font width (in pixels) of *smallTextFont.

*smallTextFontHeight
font height (in pixels) of *smallTextFont.

*nodeColorTrough
color of trough part (the /100 portion) of a canvas scale item.

*nodeColorSlider
color of slider part (value portion) of a canvas scale item.

*nodeColorExtendedTrough
color of extended trough (over 100 portion when value exceeds max) of a canvas scale item.

*nodeScaleFactor
tells how much bigger you want the scale item on the canvas to appear. (1 means to keep size as is)

*nodeBoxFullMaxWidth

*nodeBoxFullMaxHeight
maximum width and height (in pixels) of a node box in FULL mode.

*nodeBoxIconMaxWidth

*nodeBoxIconMaxHeight
maximum width and height (in pixels) of a node box in ICON mode.

*nodeBoxMirrorMaxWidth

*nodeBoxMirrorMaxHeight
maximum width and height (in pixels) of a node box displayed on a separate window (after it has been clicked with the mouse to obtain a bigger view)

*nodeBoxMirrorScaleFactor
tells how much bigger you want the scale item on the canvas to appear while the node box is displayed on a separate window (1 means to keep size as is)

Server Box Properties

Resource names beginning with “*medium” apply to the properties of the server boxes. A server box is made of an outer frame where the server display label sits on top, a canvas filled with node boxes is on the middle, possibly some horizontal/vertical scrollbars, and a status label at the bottom.

*mediumLabelForeground

color of text applied to the server display label and status label.

*mediumLabelBackground
background color of the unfilled portions of the server display label and status label.

*mediumLabelBorderWidth
distance of the server display label and status label from other parts of the server box.

*mediumLabelRelief
how the server display label and status label appear visually (style).

*mediumLabelFont"
font used for the text of the server display label and status label.

*mediumLabelFontWidth
font width (in pixels) of *mediumLabelFont.

*mediumLabelFontHeight
font height (in pixels) of *mediumLabelFont.

*mediumCanvasBorderWidth
the distance of the server box's canvas from the label widgets.

*mediumCanvasBackground
the background color of the canvas.

*mediumCanvasRelief
how the canvas appear visually (style).

*mediumScrollBorderWidth
distance of the scrollbars from the other parts of the server box.

*mediumScrollBackground
the background color of the scrollbars

*mediumScrollRelief
    how the scrollbars appear visually.

*mediumBackground
    the color of the server box frame.

*mediumBorderWidth
    the distance of the server box from other boxes.

*mediumRelief
    how the server box appears visually (style).

*serverBoxMaxWidth

*serverBoxMaxHeight
    maximum width and height (in pixels) of a server box.

*serverBoxMaxNumNodeBoxesPerRow
    maximum # of node boxes to appear in a row within a canvas.

Miscellaneous Properties

Resource names beginning with "*big" apply to the properties of a site box, as well as to widgets found outside of the server box and node box. This includes the dialog boxes that appear when the menu buttons of the main window are manipulated. The site box is the one that appears on the main region of xpbsmon.

*bigBackground
    background color of the outer layer of the main window.

*bigForeground
    color applied to regular text that appear outside of the node box and server box.

*bigBorderWidth
    distance of the site box from the menu area and the color information area.

*bigRelief

how the site box is visually represented (style)

*bigActiveColor
the color applied to the background of a selection, a selected command button, or a selected scroll bar handle.

*bigShadingColor
a color shading applied to some of the frames to emphasize focus as well as decoration.

*bigSelectorColor
the color applied to the selector box of a radiobutton or checkbutton.

*bigDisabledColor
color applied to a disabled widget.

*bigLabelBackground
color applied to the unfilled portions of label widgets.

*bigLabelBorderWidth
distance from other widgets of a label widget.

*bigLabelRelief
how label widgets appear visually (style)

*bigLabelFont
font to use for labels.

*bigLabelFontWidth
font width (in pixels) of *bigLabelFont.

*bigLabelFontHeight
font height (in pixels) of *bigLabelFont.

*bigLabelForeground
color applied to text that function as labels.

*bigCanvasBackground
the color of the main region.

*bigCanvasRelief
how the main region looks like visually (style)

*bigCanvasBorderWidth:
distance of the main region from the menu and info regions.

*bigScrollBorderWidth
if the main region has a scrollbar, this is its distance from other widgets appearing on the region.

*bigScrollBackground
background color of the scrollbar appearing outside a server box and node box.

*bigScrollRelief
how the scrollbar that appears outside a server box and node box looks like visually (style)

*bigTextFontWidth
the font width (in pixels) of *bigTextFont

*bigTextFontHeight
the font height (in pixels) of *bigTextFont

*siteBoxMaxWidth
maximum width (in pixels) of the site box.

*siteBoxMaxHeight
maximum height (in pixels) of the site box.

*siteBoxMaxNumServerBoxesPerRow
maximum number of server boxes to appear in a row inside the site box.

*autoUpdate
if set to true, then information about nodes is periodically gathered.

*autoUpdateMins
the # of minutes between polling for data regarding nodes when

*autoUpdate is set.

*siteInView
the name of the site that should be in view

*rcSiteInfoDelimeterChar
the separator character for each input within a curly-bracketed line of input of *siteInfo.

*sitesInfo
{<site1name><sep><site1-display-type><sep> <server-name><sep>
<server-display-label><sep><nodename><sep> <nodetype><sep>
<node-query-expr>}
. . .
{<site2name><sep><site2-display-type> <sep><server-name><sep>
<server-display-label><sep><nodename> <sep><nodetype><sep>
<node-query-expr>}

information about a site where <site1-display-type> can be either {FULL,ICON}, <nodetype> can be {MOM, NOMOM}, and <node-query-expr> has the format:

{ {<expr>} {expr-label} <output-format>}

where <output-format> could be {TEXT, SCALE}. It's probably better to use the Pref dialog boxes in order to specify a value for this.

Example:

*rcSiteInfoDelimeterChar ;
*sitesInfo: {NAS;ICON;newton;Newton; newton3;NOMOM;} {Langley;FULL;db;DB;db.nas.nasa.gov;MOM; {{ (availmem / totmem) * 100} {Memory Usage:} SCALE} {{ (loadave / ncpus) * 100} {Cpu Usage:} SCALE} {ncpus {Number of Cpus:} TEXT} {physmem {Physical Memory:} TEXT} {idletime {Idle Time (s):} TEXT} {loadave {Load Avg:} TEXT}} {NAS;ICON;newton;Newton;newton4; NOMOM;} {NAS;ICON;newton;Newton; newton1;NOMOM;} {NAS;ICON;newton;Newton; newton2;NOMOM;} {NAS;ICON;b0101;DB;aspasia.nas.nasa.gov; MOM;{{ (availmem / totmem) * 100} {Memory Usage:} SCALE} {{ (loadave / ncpus) * 100} {Cpu Usage:} SCALE} {ncpus {Number of

Cpus:} TEXT} {physmem {Physical Memory:} TEXT} {idletime {Idle Time (s):} TEXT} {loadave {Load Avg:} TEXT}} {NAS;ICON;newton;Newton;newton7;NOMOM;}

EXIT STATUS

Upon successful processing, the xpbsmon exit status will be a value of zero.

If the xpbsmon command fails, the command exits with a value greater than zero.

If xpbsmon is querying a host running a server with an incompatible version, you may see the following messages:

```
Internal error: pbsstatnode: End of File (15031)
```

The above message can be safely ignored.

SEE ALSO

The PBS Professional User's Guide, the PBS Professional Administrator's Guide,
pbs_sched(8B), pbs_mom(8B), pbs_tclapi(3B).

Chapter 10
Administrator Commands

Man pages for PBS Professional administrator commands are listed below.

NAME

mpiexec - run MPI programs under PBS on Linux on IA64

SYNOPSIS

mpiexec

mpiexec --version

DESCRIPTION

The PBS mpiexec command provides the standard mpiexec interface on the Altix running ProPack 4 or greater. If executed on a non-Altix system, it will assume it was invoked by mistake. In this case it will use the value of PBS_O_PATH to search for the correct mpiexec. If one is found, the PBS mpiexec will exec it.

The PBS mpiexec calls the SGI mpirun(1). The name of the array to use when invoking mpirun is user-specifiable via the PBS_MPI_SGIARRAY environment variable.

It is transparent to the user; MPI jobs submitted outside of PBS will run as they would normally. MPI jobs can be launched across multiple Altixes. PBS will manage, track, and cleanly terminate multi-host MPI jobs. PBS users can run MPI jobs within specific partitions.

If CSA has been configured and enabled, PBS will collect accounting information on all tasks launched by an MPI job. CSA information will be associated with the PBS job ID that invoked it, on each execution host.

If the PBS_MPI_DEBUG environment variable's value has a nonzero length, PBS will write debugging information to standard output.

USAGE

The PBS mpiexec command presents the mpiexec interface described in section "4.1 Portable MPI Process Startup" of the "MPI-2: Extensions to the Message-Passing Interface" document in http://www.mpi-forum.org/docs/mpi-20-html/node42.htm

OPTIONS

--version

The mpiexec command returns its PBS version information and exits. This option can only be used alone.

REQUIREMENTS

Altix running ProPack 4 or greater.

PBS uses SGI's mpirun(1) command to launch MPI jobs. SGI's mpirun must be in the standard location.

The location of pbs_attach(8B) on each node of a multinode MPI job must be the same as it is on the mother superior node.

In order to run multihost jobs, the SGI Array Services must be correctly configured. Altix systems communicating via SGI's Array Services must all use the same version of the sgi-arraysvcs package. Altix systems communicating via SGI's Array Services must have been configured to interoperate with each other using the default array. See SGI's array_services(5) man page.

ENVIRONMENT VARIABLES

The PBS mpiexec script sets the PBS_CPUSET_DEDICATED environment variable to assert exclusive use of the resources in the assigned cpuset.

The PBS mpiexec checks the PBS_MPI_DEBUG environment variable. If this variable has a nonzero length, debugging information is written.

If the PBS_MPI_SGIARRAY environment variable is present, the PBS mpiexec will use its value as the name of the array to use when invoking mpirun.

The PBS_ENVIRONMENT environment variable is used to determine whether mpiexec is being called from within a PBS job.

The PBS mpiexec uses the value of PBS_O_PATH to search for the correct mpiexec if it was invoked by mistake.

PATH

PBS' mpiexec is located in PBS_EXEC/bin/mpiexec.

SEE ALSO

The PBS Professional Administrator's Guide

SGI man pages: SGI's mpirun(1), SGI's array_services(5)

PBS man pages: pbs_attach(8B)

Local 14 June 2006 mpiexec(8B)

NAME

pbs-report - print PBS job statistics

SYNOPSIS

```
pbs-report [--age seconds[:offset]] [--account account]
        [--begin -b yyyymmdd[:hhmm[ss]]] [--count -c]
        [--cpumax seconds] [ --cpumin seconds] [--csv character]
        [--dept department] [--end -e yyyymmdd[:hhmm[ss]]]
        [--exit -x integer] [--explainwait] [--group UNIX group]
        [--help] [--host hostname] [--inclusive] [--index key]
        [--man] [--negate option] [--package solver]
        [--point yyyymmdd[:hhmm[ss]]] [--queue PBS queue]
        [--range span] [--reslist] [--sched] [--sort field]
        [--time option] [--user username] [--verbose]
        [--vsort field] [--waitmax seconds] [--waitmin seconds]
        [--wall] [--wallmax seconds] [--wallmin seconds]

pbs-report --version
```

DESCRIPTION

Allows the PBS Administrator to generate a report of job statistics from the PBS accounting logfiles. Options to the pbs-report command control how jobs are selected for reporting and how reports are generated.

The pbs-report command is not available on Windows.

Before first using pbs-report, the Administrator is advised to tune the pbs-report configuration to match the local site by editing the file PBS_EXEC/lib/pm/PBS.pm.

Selecting Jobs For Reporting

Filtering Jobs by Dates or Times: --begin, --end, --range, --age, --point

--begin and --end work from hard date limits. Omitting either will cause the report to contain all data to either the beginning or the end of the accounting data. Unbounded date reports may take several minutes to run, depending on the volume of work logged.

--range is a short-hand way of selecting a prior date range and will supersede --begin and --end.

--age allows the user to select an arbitrary period going back a specified number of seconds from the time the report is run. --age will silently supersede all other date options.

--point displays all jobs which were running at the specified point in time, and is incompatible with the other options. --point will produce an error if specified with any other date-related option.

Filtering Jobs by Attribute: --cpumax, --cpumin, --waitmax, --waitmin, --wallmax, --wallmin

A maximum value will cause any jobs with more than the specified amount to be ignored. A minimum value will cause any jobs with less than the specified amount to be ignored. All six options may be combined, though doing so will often restrict the filter such that no jobs can meet the requested criteria. Combine time filters for different time with caution.

Filtering Jobs by User or Department: --dept, --group, --user

--dept allows for integration with an LDAP server and will generate reports based on department codes as queried from that server. If no LDAP server is available, department-based filter- ing and sorting will not function.

--group allows for filtering of jobs by primary group ownership of the submitting user, as defined by the operating system on which the PBS server runs.

--user allows for explicit naming of users to be included.

It is possible to specify a list of values for these filters, by pro- viding a single colon-concatenated argument or using the option multi- ple times, each with a single value.

Filtering Jobs by Job Property: --host, --exit, --package, --queue

--host allows for filtering of jobs based on the host on which the job was executed.

--exit allows for filtering of jobs based on the job exit code.

--package allows for filtering of jobs based on the software package used in the job. This option will only function when a package-specific custom resource is defined for the PBS server and requested by the jobs as they are submitted.

--queue allows for filtering of jobs based on the queue in which the job finally executed. With the exception of --exit, it is possible to specify a list of values for these filters, by providing a single colon-concatenated argument or using the option multiple times, each with a single value.

Filtering Jobs by Account String: --account

This option allows the user to filter jobs based on an arbitrary, user-specified job account string. The content and format of these strings is site-defined and unrestricted; it may be used by a custom job front-end which enforces permissible account strings, which are passed to qsub with qsub's -A option.

Negating Filters:

The --negate option allows for logical negation of one or more specified filters. Only the account, dept, exit, group, host, package, queue, and user filters may be negated. If a user is specified with --user, and the '--negate user' option is used, only jobs not belonging to that user will be included in the report. Multiple report filters may be negated by providing a single colon-concatenated argument or using --negate multiple times, each with a single value.

Generating Reports

Several report types can be generated, each indexed and sorted according to the user's needs.

--verbose generates a wide tabular output with detail for every job selected. It can be used to generate output for import to a spreadsheet. Verbose reports may be sorted on any field using the --vsort option. Default: summary report only.

--reslist generates a tabular output with detail on resources requested for every job selected. Resource list reports may be sorted on any field using the --vsort option. Default: summary report only.

--inclusive allows a user to require that the job's start time also falls within the date range. By default, all date selections are bounds around a job's end time.

--index allows specification of the field on which data in the summary should be grouped. Fields listed in the option description are mutually exclusive. The selected field will be the left-most column of the summary report output. One value may be selected as an index while another is selected for sorting. However, since index values are mutually exclusive, the only sort options which may be used (other than the index itself) are account, cpu, jobs, sus- pend, wait, and wall. If no sort order is selected, the index is used as the sort key for the summary.

--sort allows the user to specify a field on which to sort the summary report. It operates independently of the sort field for ver- bose reports (see --vsort). See the description for --index for how the two options interact.

--vsort allows the user to specify a field on which to sort the verbose report. It operates independently of the sort field for sum- mary reports (see --sort).

OPTIONS

--age -a seconds[:offset]

Report age in seconds. If an offset is specified, the age range is taken from that offset backward in time, otherwise a zero offset is assumed. The time span is from (now - age - offset) to (now - offset). This option silently supersedes --begin, --end, and --range.

--account account

Limit results to those jobs with the specified account string. Multiple values may be concatenated with colons or specified with multiple instances of --account.

--begin -b yyyymmdd[:hhmm[ss]]
Report begin date and optional time. Default: most recent log data. --begin and --end work from hard date limits. Omitting either will cause the report to contain all data to either the beginning or the end of the accounting data. Unbounded date reports may take several minutes to run, depending on the volume of work logged.

--count -c
Display a numeric count of matching jobs. Currently only valid with --cpumax for use in monitoring rapidly-exiting jobs.

--cpumax seconds
Filter out any jobs which have more than the specified number of CPU seconds.

--cpumin seconds
Filter out any jobs which have less than the specified number of CPU seconds.

--dept -d department
Limit results to those jobs whose owners are in the indicated department . Default: any. This option only works in conjunction with an LDAP server which supplies department codes. See also the --group option. Multiple values may be concatenated with colons or specified with multiple instances of --dept.

--end -e yyyymmdd[:hhmm[ss]]
Report end date and optional time. Default: most recent log data. --begin and --end work from hard date limits. Omitting either will cause the report to contain all data to either the beginning or the end of the account-

ing data. Unbounded date reports may take several minutes to run, depending on the volume of work logged.

--exit -x integer
Limit results to jobs with the specified exit status. Default: any.

--explainwait Print a reason for why jobs had to wait before running.

--group -g group
Limit results to the specified group name. Group is defined by the operating system. Multiple values may be concatenated with colons or specified with multiple instances of --group.

--help -h Prints a brief help message and exits.

--host -m execution host
Limit results to the specified execution host. Multiple values may be concatenated with colons or specified with multiple instances of --host.

--inclusive key
Limit results to jobs which had both start and end times in the range.

--index -i key Field on which to index the summary report. Default: user. Valid values include: date, dept, host, package, queue, user.

--man Prints the manual page and exits.

--negate -n option

Logically negate the selected options; print all records except those that match the values for the selected criteria. Default: unset. Valid values: account, dept, exit, group, host, package, queue, user. Defaults cannot be negated, only options explicitly specified are negated. Multiple values may be concatenated with colons or specified with multiple instances of --negate.

--package -p package

Limit results to the specified software package. Multiple values may be concatenated with colons or specified with multiple instances of --package. Valid values are can be seen by running a report with the --index package option. This option keys on custom resources requested at job submission time. Sites not using such custom resources will have all jobs reported under the catch-all None package with this option.

--point yyyymmdd[:hhmm[ss]]

Print a report of all jobs which were actively running at the point in time specified. This option cannot be used with any other date or age option.

--queue -q queue

Limit results to the specified queue. Multiple values may be concatenated with colons or specified with multiple instances of --queue. Note that if specific queues are defined via the @QUEUES line in PBS.pm, then only those queues will be displayed. Leaving that parameter blank allows all queues to be displayed.

--range -r period

Time period used is period before now. For example, if the period given is “week”, then pbs-report looks at all jobs which have finished and which were running any time from a week ago to now. Default: all. Valid values for period are today, week, month, quarter, and year. This option silently supersedes --begin and --end, and is superseded by --age.

--reslist Include resource requests for all matching jobs. This option is mutually exclusive with --verbose.

--sched -t Generate a brief statistical analysis of Scheduler cycle times. No other data on jobs is reported.

--sort -s field
Field by which to sort reports. Default: user. Valid values are cpu, date, dept, host, jobs, package, queue, suspend (aka muda), wait, and wall.

--time option Valid values: “full”, “partial”. Used to indicate how time should be accounted. The default of “full” means that entire job’s CPU and wall time is counted in the report if the job ended during the report’s date range. With the “partial” option, only CPU and wall time during the report’s date range are counted.

By default, time is credited at the point when the job ended. This can be changed using the --inclusive option. For a job which ended a few seconds after the report range begins, this can cause significant overlap, which may boost results. During a sufficiently large time frame, this overlap effect is negligible and may be ignored. This value for --time should be used when generating monthly usage reports. With “partial”, any CPU or wall time accumulated prior to the beginning of the report is ignored. “partial” is intended to allow for more accurate calculation of overall cluster efficiency during short time spans during which a significant overlap effect can skew results. See --inclusive.

--user -u username
Limit results to the specified username. Multiple values may be concatenated with colons or specified with multiple instances of --user.

--verbose -v Include attributes for reported jobs. Subjobs are shown, but not job arrays. Default: no attributes.

--version The pbs-report command returns its PBS version information and exits. This option can only be used alone.

--vsort field Field by which to sort the verbose output section reports. Default: jobid. Valid values are cpu, date, exit, host, jobid, jobname, mem, name, package, queue, scratch, suspend, user, vmem, wall, wait. If neither --verbose nor --reslist is specified, --vsort is silently ignored. The scratch sort option is available only for resource reports (--reslist).

--waitmax seconds
Filter out any jobs which have more than the specified wait time in seconds.

--waitmin seconds
Filter out any jobs which have less than the specified wait time in seconds.

--wallmax seconds
Filter out any jobs which have more than the specified wall time in seconds.

--wallmin seconds

Filter out any jobs which have less than the specified wall time in seconds.

--wall -w Use the walltime resource attribute rather than wall time calculated by subtracting the job start time from end time. The walltime resource attribute does not accumulate when a job is suspended for any reason, and thus may not accurately reflect the local interpretation of wall time.

EXAMPLES

"How much in the way of resources did every job this month waiting more than 10 minutes request?"

```
pbs_report --range month --waitmin 600 --reslist
```

This information might be valuable to determine if some simple resource additions (e.g. more memory or more disk) might increase overall throughput of the cluster.

Statistical Analysis

At the bottom of the summary statistics, prior to the job set summary, is a statistical breakdown of the values in each column. Example:

```
             # of      Total       Total            Average
Date         jobs    CPU Time   Wall Time  Efcy.   Wait Time
----------  -----  ----------  ----------  -----  ----------
TOTAL        1900    10482613    17636290  0.594        1270
Minimum         4        4715       13276  0.054         221
Maximum       162     1399894     2370006  1.782       49284
Mean           76      419304      705451  0.645        2943
Deviation      41      369271      616196  0.408        9606
Median         80      242685      436724  0.556         465
```

This summary should be read in column format. The values each represent a statistical data point in the column. For instance, while the minimum number of jobs run in one day was 4 and the maximum 162, these values do not correlate to the 4715 and 1399894 CPU seconds listed as minima and maxima.

In the Job Set Summary section, the values should be read in rows, as shown here:

```
                                        Standard
             Minimum    Maximum      Mean   Deviation     Median
           ----------  ----------  ----------  ----------  ----------
CPU time            0      18730       343        812          0
Wall time           0     208190      8496      19711         93
Wait time           0     266822      4129       9018          3
```

These values represent aggregate statistical analysis for the entire set of jobs included in the report. The values in the prior summary represent values over the set of totals based on the summary index (e.g. Maximum and Minimum are the maximum and minimum totals for a given day/user/department, rather than an individual job. The job set summary represents an analysis of all individual jobs.

Cluster Monitoring

The --count and --cpumax functions are intended to allow an adminstra-tor to periodically run this script to monitor for jobs which are exit-ing rapidly, representing a potential global error condition causing all jobs to fail. It is most useful in conjunction with --age, which allows a report to span an arbitrary number of seconds backward in time from the current moment. A typical set of options would be “--count --cpumax 30 --age 21600”, which would show a total number of jobs which consumed less than 30 seconds of CPU time within the last six hours.

STANDARD ERROR

The pbs-report command will write a diagnostic message to standard error for each error occurrence.

EXIT STATUS

Zero upon successful processing of all operands.

Greater than zero if the pbs-report command fails to process any oper-
and.

SEE ALSO

The PBS Professional Administrator’s Guide,
pbs_server(8B), pbs_sched(8B), pbs_mom(8B)

NAME

pbs_attach - attaches a session ID to a PBS job

SYNOPSIS

pbs_attach [-j jobid] [-m port] -p pid
pbs_attach [-j jobid] [-m port] [-P] [-s] cmd [arg ...]
pbs_attach --version

DESCRIPTION

The pbs_attach command associates the processes in a session with a PBS job by attaching the session ID to the job. This allows PBS MOM to monitor and control those processes.

MOM uses process IDs to determine session IDs, which are put into MOM's task list (attached to the job.) All process IDs in a session are then associated with the job.

When a command cmd is given as an operand, the pbs_attach process becomes the parent process of cmd, and the session ID of pbs_attach is attached to the job.

The -p option cannot be used with the -P or -s options or the cmd operand.

OPTIONS

-j jobid   The job ID to which the session ID is to be attached. If jobid is not specified, a best effort will be made to determine the job to which to attach the session.

-m port   The port at which to contact MOM. Default: value of PBS_MANAGER_SERVICE_PORT from pbs.conf

-p pid   Process ID whose session ID will be attached to the job. Default: process ID of pbs_attach.

-P	Attach sessions of both pbs_attach and the parent of pbs_attach to job. When used with -s option, this means the sessions of the new fork() ed pbs_attach and its parent, which is pbs_attach, are attached to the job.
-s	Starts a new session by fork() ing pbs_attach. The session ID of the new pbs_attach is attached to the job.
--version	The pbs_attach command returns its PBS version information and exits. This option can only be used alone.

OPERANDS

cmd	Name of command whose process ID is to be associated with the job.

EXIT STATUS

0	Success
1	Any error following successful command line processing. A message is printed to standard error.

If cmd is specified, pbs_attach waits for cmd to exit, then exits with the exit value of cmd.

If cmd is not specified, pbs_attach exits after attaching the session ID(s) to the job.

SEE ALSO

The PBS Professional Administrator's Guide
pbs_mom(8B), pbs_tmrsh(8B), tm(3)

NAME

pbs_hostn - report hostname and network address(es)

SYNOPSIS

```
pbs_hostn [ -v ] hostname
pbs_hostn --version
```

DESCRIPTION

The pbs_hostn command takes a hostname, and reports the results of both gethostbyname(3) and gethostbyaddr(3) system calls. Both forward and reverse lookup of hostname and network addresses need to succeed in order for PBS to authenticate a host.

Running this command can assist in troubleshooting problems related to incorrect or non-standard network configuration, especially within clusters.

OPTIONS

-v	Turns on verbose mode.
--version	The pbs_hostn command returns its PBS version information and exits. This option can only be used alone.

OPERANDS

The pbs_hostn command accepts a hostname operand either in short name form, or in fully qualified domain name (FQDN) form.

STANDARD ERROR

The pbs_hostn command will write a diagnostic message to standard error for each error occurrence.

EXIT STATUS

Zero upon successful processing of all the operands presented to the pbs_hostn command.

Greater than zero if the pbs_hostn command fails to process any operand.

SEE ALSO

The PBS Professional Administrator's Guide and the following manual page: pbs_server(8B)

NAME

pbs_idled - PBS daemon to watch the X console and inform pbs_mom of idle time

SYNOPSIS

```
pbs_idled [-w wait_time] [-f idle_file] [-D display]
        [-r reconnect_delay]
pbs_idled --version
```

DESCRIPTION

The pbs_idled program sits and watches an X windows display and communicates the idle time of the display back to PBS. If the mouse is moved or a key is touched, PBS is informed that the node is busy.

This program should be run out of the system-wide Xsession file. It should be run in the background before the window manager is run. If this program is run outside of the Xsession, it will need to be able to make a connection to the X display. See the xhost or xauth man pages for a description of X security.

OPTIONS

-w <wait_time>
: Granularity between when the daemon checks for events or pointer movement.

-f <idle_file>
: Update file times on <file>. PBS will not monitor any other than the default.

-D <display>
: The display to connect to and monitor.

-r <reconnect_delay>
: The amount of time to try and reconnect to the X display if the previous attempt was unsuccessful.

--version The pbs_idled command returns its PBS version information and exits. This option can only be used alone.

SEE ALSO

The PBS Professional Administrator’s Guide and the following manual pages: pbs_mom(8B), xhost(1), xauth(1)

Local 12 April 2007 pbs_idled(8B)

NAME

pbs_lamboot - PBS front end to LAM's lamboot program

SYNOPSIS

pbs_lamboot

pbs_lamboot --version

DESCRIPTION

The PBS command pbs_lamboot replaces the standard lamboot command in a PBS LAM MPI job, for starting LAM software on each of the PBS execution hosts running Linux 2.4 or higher.

Usage is the same as for LAM's lamboot. All arguments except for bhost are passed directly to lamboot. PBS will issue a warning saying that the bhost argument is ignored by PBS since input is taken automatically from $PBS_NODEFILE. The pbs_lamboot program will not redundantly consult the $PBS_NODEFILE if it has been instructed to boot the nodes using the tm module. This instruction happens when an argument is passed to pbs_lamboot containing "-ssi boot tm" or when the LAM_MPI_SSI_boot environment variable exists with the value tm.

OPTIONS

--version

The pbs_lamboot command returns its PBS version information and exits. This option can only be used alone.

OPERANDS

The operands for pbs_lamboot are the same as for lamboot.

ENVIRONMENT VARIABLES

PATH

The PATH on remote machines must contain PBS_EXEC/bin.

SEE ALSO

The PBS Professional Administrator’s Guide

lamboot(1), tm(3)

Local 3 August 2005 pbs_lamboot(8B)

NAME

pbs_migrate_users - transfer per-user or per-server passwords between PBS servers during a migration upgrade

SYNOPSIS

pbs_migrate_users old_server new_server
pbs_migrate_users --version

DESCRIPTION

The pbs_migrate_users command is used to transfer the per-user or per-server password of a PBS user from one server to another during a migration upgrade.

Users' passwords on the old server are not deleted.

OPTIONS

--version	The pbs_migrate_users command returns its PBS version information and exits. This option can only be used alone.

OPERANDS

The format of old_server and new_server is
hostname[:port_number]

EXIT STATUS

0	Success
-1	Writing out passwords to files failed.
-2	Communication failure between old_server and new_server.
-3	Single_signon_password_enable not set in either old_server or new_server
-4	User running pbs_migrate_users not authorized to migrate users.

SEE ALSO

pbs_password(8B)

NAME

pbs_mom - The PBS job monitoring and execution daemon

SYNOPSIS

```
pbs_mom [-a alarm_timeout] [-C checkpoint_directory] [-c config_file]
        [-d home_directory] [-L logfile] [-M TCP_port] [-n nice_val]
        [-N] [-p|-r] [-R UDP_port] [-S server_port]
        [-s script_options] [-x]
pbs_mom --version
```

DESCRIPTION

The pbs_mom command starts the PBS job monitoring and execution daemon, called MOM. The pbs_mom daemon starts jobs on the execution host, monitors and reports resource usage, enforces resource usage limits, and notifies the server when the job is finished. The MOM also runs any prologue scripts before the job runs, and runs any epilogue scripts after the job runs.

The MOM performs any communication with job tasks and with other MOMs. The MOM on the first vnode on which a job is running manages communication with the MOMs on the remaining vnodes on which the job runs.

The MOM manages one or more vnodes. PBS may treat a host such as an Altix as a set of virtual nodes, in which case one MOM would manage all of the host's vnodes. See the PBS Professional Administrator's Guide.

The MOM's log file is in PBS_HOME/mom_logs. The MOM writes an error message in its log file when it encounters any error. If it cannot write to its log file, it writes to standard error. The MOM will write events to its log file. The MOM writes its PBS version and build information to the logfile whenever it starts up or the logfile is rolled to a new file.

The executable for pbs_mom is in PBS_EXEC/sbin, and can be run only by root.

CPUSETS

A cpusetted machine can have a "boot cpuset" defined by the administrator. A boot cpuset contains one or more CPUs and memory boards and is used to restrict the default placement of system processes, including login. If defined, the boot cpuset will contain CPU 0.

Run parallel jobs exclusively within a cpuset for repeatability of performance. SGI states, “Using cpusets on an Altix system improves cached locality and memory access times and can substantially improve an application’s performance and runtime repeatability.”

The CPUSET_CPU_EXCLUSIVE flag will prevent CPU 0 from being used by the MOM in the creation of job cpusets. This flag is set by default, so this is the default behahavior.

To find out which cpuset is assigned to a running job, use qstat -f to see the cpuset field in the job’s altid attribute.

Altix Running ProPack 4 or 5

The cpusets created for jobs are marked cpu-exclusive.

MOM does not use any CPU which was in use at startup.

A PBS job can run across multiple Altixes that run ProPack 4 or 5.

PBS can run using SGI’s MPI (MPT) over InfiniBand. See the PBS Professional Administrator’s Guide.

Altix Running ProPack 2 or 3

MOM does not use the CPUs on any nodeboard containing either CPU 0 or a CPU which was in use at startup.

IRIX The pbs_mom for the irix6_cpuset architecture forks into two pbs_moms: one that services jobs, and one that gathers process information for every process that it tracks. It can fork a MOM for killing off stray or unauthorized processes. This MOM is turned off by default, but can be turned on by adding the “enforce hammer” configuration file option.

The irix6_cpuset pbs_mom classifies jobs as either small or multinode, meaning jobs that use more than one nodeboard. Small jobs use limited CPUs and memory, and run in shared cpusets, which are designated for small jobs. The definition of a small job is set using cpuset_small_ncpus and cpuset_small_mem in MOM’s config file. The default for small jobs is one CPU and the memory size of one nodeboard, which is system-dependent. The number of nodeboards used for shared cpusets is set in max_shared_nodes in MOM’s config file. Multinode jobs use the resources of more than one nodeboard, and run in exclusive cpusets, by themselves. Any job with the “ssinodes” attribute set will run in exclusive cpusets.

Mom will not use any cpuset that is already in use when MOM starts up. This includes the boot cpuset, if it exists.

A cpuset containing CPU 0 will only be created and allocated for a job if there is no boot cpuset and no other CPUs are available to satisfy a request. Use of CPU 0 for jobs can degrade performance, since the kernel uses this CPU heavily for system daemons.

A PBS job cannot run on more than one IRIX machine being managed by a cpuset PBS MOM.

OPTIONS

-a alarm_timeout

Number of seconds before alarm timeout. Whenever a resource request is processed, an alarm is set for the given amount of time. If the request has not completed before alarm_timeout, the OS generates an alarm signal and sends it to MOM. Default: 10 seconds. Format: integer.

-C checkpoint_directory

Specifies the path of the directory used to hold checkpoint files. Only valid on systems supporting checkpoint/restart.

The default directory is PBS_HOME/spool/checkpoint. Any directory specified with the -C option must be owned by root and accessible (rwx) only by root to protect the security of the checkpoint files. See the -d option. Format: string.

-c config_file

MOM will read this alternate default configuration file upon starting. If this is a relative file name it will be relative to PBS_HOME/mom_priv. If the specified file cannot be opened, pbs_mom will abort. See the -d option.

MOM's normal operation, when the -c option is not given, is to attempt to open the default configuration file "config" in PBS_HOME/mom_priv. If this file is not present, pbs_mom will log the fact and continue.

-d home_directory

Specifies the path of the directory to be used in place of PBS_HOME by pbs_mom. The default directory is given by $PBS_HOME. Format: string.

-L logfile

Specifies an absolute path and filename for the log file. The default is a file named for the current date in PBS_HOME/mom_logs/. See the -d option. Format: string.

-M TCP_port

Specifies the number of the TCP port on which MOM will listen for server requests and instructions. Default: 15002. Format: integer port number.

-n nice_val

Specifies the priority for the pbs_mom daemon. Format: integer.

-N
: Specifies that when starting, MOM should not detach from the current session.

-p
: Specifies that when starting, MOM should track any running jobs, and allow them to continue running. Cannot be used with the -r option. MOM's default behavior is to requeue the jobs, and allow job processes to continue to run, but not to track them. MOM is not the parent of these jobs.

  Altix running ProPack 4
  : The Altix ProPack 4 cpuset pbs_mom will, if given the -p flag, use the existing CPU and memory allocations for the /PBSPro cpusets. The default behavior is to remove these cpusets. Should this fail, MOM will exit, asking to be restarted with the -p flag.

-r
: Specifies that when starting, MOM should kill any job processes, mark the jobs as terminated, and notify the server. Cannot be used with the -p option. MOM's default behavior is to allow these jobs to continue to run. MOM is not the parent of these jobs.

  Do not use the -r option after a reboot, because process IDs of new, legitimate tasks may match those MOM was previously tracking. If they match and MOM is started with the -r option, MOM will kill the new tasks.

-R UDP_port
: Specifies the number of the UDP port on which MOM will listen for pings, resource information requests, communication from other MOMs, etc. Default: 15003. Format: integer port number.

-S server_port

Specifies the number of the TCP port on which pbs_mom initially contact the server. Default: 15001. Format: integer port number.

-s script_options

This option provides an interface that allows the administrator to add, delete, and display MOM's configuration files. See CONFIGURATION FILES. script_options are used this way:

-s insert <scriptname> <inputfile>

Reads inputfile and inserts its contents in a new site-defined pbs_mom configuration file with the filename scriptname. If a site-defined configuration file with the name scriptname already exists, the operation fails, a diagnostic is presented, and pbs_mom exits with a nonzero status. Scripts whose names begin with the prefix "PBS" are reserved. An attempt to add a script whose name begins with "PBS" will fail. pbs_mom will print a diagnostic message and exit with a nonzero status.

On **Windows**, the syntax is:

```
pbs_mom -N -s insert <scriptname> <inputfile>
```

This forces pbs_mom to start up as a standalone program.

-s remove <scriptname>

The configuration file named scriptname is removed if it exists. If the given name does not exist or if an attempt is made to remove a script with the reserved "PBS" prefix, the operation fails, a diagnostic is presented, and pbs_mom exits with a nonzero status.

-s show <scriptname>

Causes the contents of the named script to be printed to standard output. If scriptname does not exist, the operation fails, a diagnostic is presented, and pbs_mom exits with a nonzero status

-s list
: Causes pbs_mom to list the set of PBS-prefixed and site-defined configuration files in the order in which they will be executed.

-x
: Disables the check for privileged-port connections.

--version
: The pbs_mom command returns its PBS version information and exits. This option can only be used alone.

CONFIGURATION FILES

MOM's configuration information can be contained in configuration files of three types: default, PBS-prefixed, and site-defined. The default configuration file is usually PBS_HOME/mom_priv/config. The "PBS" prefix is reserved for files created by PBS. Site-defined configuration files are those created by the site administrator. MOM reads the configuration files at startup and reinitialization. The files are processed in this order:

The default configuration file
PBS-prefixed configuration files
Site-defined configuration files

The contents of a file read later override the contents of a file read earlier. For example, to change the cpuset flags, create a script "update_flags" containing only

```
cpuset_create_flags <new flags>
```

then use the -s insert option:

```
pbs_mom -s insert update_script update_flags
```

This adds the configuration file "update_script". Configuration files can be added, deleted and displayed using the -s option. An attempt to create or remove a file with the "PBS" prefix will result in an error.

MOM's configuration files can use either the syntax shown below under Default Syntax and Contents or the syntax for describing vnodes shown in Vnode Syntax.

Location

The default configuration file is in PBS_HOME/mom_priv/.

PBS places PBS-prefixed and site-defined configuration files in an area that is private to each installed instance of PBS. This area is relative to the default PBS_HOME. Note that the -d option changes where MOM looks for PBS_HOME, and using this option will prevent MOM from finding any but the default configuration file.

The -c option will change which default configuration file MOM reads.

Site-defined configuration files can be moved from one installed instance of PBS to another. Do not move PBS-prefixed configuration files. To move a set of site-defined configuration files from one installed instance of PBS to another:

1 Use the -s list directive with the “source” instance of PBS to enumerate the site-defined files.

2 Use the -s show directive with each site-defined file of the “source” instance of PBS to save a copy of that file.

3 Use the -s insert directive with each file at the “target” instance of PBS to create a copy of each site-defined configuration file.

Vnode Configuration File Syntax and Contents
Configuration files with the following syntax describe vnodes and the resources available on them. They do not contain initialization values for MOM. See the PBS Professional Administrator’s Guide for a definition of vnodes. PBS-prefixed configuration files use the following syntax. Other configuration files can use the following syntax.

Any configuration file containing vnode-specific assignments must begin with this line:

```
$configversion 2
```

The format a file containing vnode information is:

```
<ID> : <ATTRNAME> = <ATTRVAL>
```

where

<ID>	sequence of characters not including a colon (":")
<ATTRNAME>	sequence of characters beginning with alphabetics or numerics, which can contain underscore ("_") and dash ("-")
<ATTRVAL>	sequence of characters not including an equal sign ("=")

The colon and equal sign may be surrounded by spaces.

A vnode's ID is an identifier that will be unique across all vnodes known to a given pbs_server and will be be stable across reinitializations or invocations of pbs_mom. ID stability is of importance when a vnode's CPUs or memory might be expected to change over time and PBS is expected to adapt to such changes by resuming suspended jobs on the same vnodes to which they were originally assigned. Vnodes for which this is not a consideration may simply use IDs of the form "0", "1", etc. concatenated with some identifier that ensures uniqueness across the vnodes served by the pbs_server.

A natural vnode does not correspond to any actual hardware. It is used to define any placement set information that is invariant for a given host, such as pnames.

It is defined as follows:

The name of the natural vnode is, by convention, the MOM contact name, which is usually the hostname. The MOM contact name is the vnode's MOM attribute. See the pbs_node_attributes(7B) man page.

An attribute, "pnames", with value set to the list of resource names that define the placement sets' types for this machine.

An attribute, "sharing" is set to the value "force_shared"

The natural vnode is used to define any placement set information that is invariant for a given host (e.g. the placement set resource names themselves).

The order of the pnames attribute follows placement set organization.

If name X appears to the left of name Y in this attribute's value, an entity of type X may be assumed to be smaller (that is, be capable of containing fewer vnodes) than one of type Y. No such guarantee is made for specific instances of the types.

For example, on an Altix named "HostA", with two vnodes, a natural vnode, four processors and two cbricks, the description would look like this:

```
HostA:  pnames = cbrick
HostA:  sharing = force_shared
HostA[001c02#0]:  sharing = default_excl
HostA[001c02#0]:  resources_available.ncpus = 2
HostA[001c02#0]:  resources_available.cbrick = cbrick-0
HostA[001c02#0]:  resources_available.mem = 1968448kb
HostA[001c04#0]:  sharing = default_excl
HostA[001c04#0]:  resources_available.ncpus = 2
HostA[001c04#0]:  resources_available.cbrick = cbrick-1
HostA[001c04#0]:  resources_available.mem = 1961328kb
```

The natural vnode is described in the first two lines. The first vnode uses cbrick-0, and the second one uses cbrick-1.

Default Syntax and Contents
Configuration files with this syntax list local resources and initialization values for MOM. Local resources are either static, listed by name and value, or externally-provided, listed by name and command path. See the -c option.

Each configuration item is listed on a single line, with its parts separated by white space. Comments begin with a hashmark ("#").

The default configuration file must be secure. It must be owned by a user ID and group ID both less than 10 and must not be world-writable.

Externally-provided Resources

Externally-provided resources use a shell escape to run a command. These resources are described with a name and value, where the first character of the value is an exclamation mark ("!"). The remainder of the value is the path and command to execute.

Parameters in the command beginning with a percent sign (“%”) can be replaced when the command is executed. For example, this line in a configuration file describes a resource named “escape”:

escape !echo 0xx %yyy

If a query for the “escape” resource is sent with no parameter replacements, the command executed would be “echo 0xx %yyy”. If one parameter replacement is sent, “escape[xxx=hi there]”, the command executed would be “echo hi there %yyy”. If two parameter replacements are sent, “escape[xxx=hi][yyy=there]”, the command executed would be “echo hi there”. If a parameter replacement is sent with no matching token in the command line, “escape[zzz=snafu]”, an error is reported.

Initialization Values

Initialization value directives have names beginning with a dollar sign (“$”). See The PBS Professional Administrator’s Guide.

$action <default_action> <timeout> <new_action>

Replaces the default_action for an event with the site-specified new_action. timeout is the time allowed for new_action to run. See The PBS Professional Administrator’s Guide. The default_action can be one of:

checkpoint

Run new_action in place of the periodic job checkpoint, after which the job continues to run.

checkpoint_abort

Run new_action to checkpoint the job, after which the job is terminated.

multinodebusy

Used with cycle harvesting and multi-vnode jobs. Changes default action when a vnode becomes busy.

Instead of allowing the job to run, the job is requeued. The new_action is requeue.

restart
Runs new_action in place of restart.

terminate
Runs new_action in place of SIGTERM or SIGKILL when MOM terminates a job.

$checkpoint_path <path>
MOM will write checkpoint files in the directory given by path. This path can be absolute or relative to PBS_HOME/mom_priv.

$clienthost <hostname>
hostname is added to the list of hosts which will be allowed to connect to MOM as long as they are using a privileged port. For example, this will allow the hosts "fred" and "wilma" to connect to MOM:

```
$clienthost      fred
$clienthost      wilma
```

Two hostnames are always allowed to connect to pbs_mom, "localhost" and the name returned to MOM by the system call gethostname(). These hostnames do not need to be listed in the configuration file.

The hosts listed as "clienthosts" make up a "sisterhood" of machines. Any one of the sisterhood will accept connections from within the sisterhood. The sisterhood must all use the same port number.

$cputmult <factor>
This sets a factor used to adjust CPU time used by each job. This allows adjustment of time charged and limits enforced where jobs run on a system with different CPU performance. If MOM's system is faster than the reference system, set factor to a decimal value greater than 1.0. For example:

$cputmult 1.5

If MOM's system is slower, set factor to a value between 1.0 and 0.0. For example:
$cputmult 0.75

$dce_refresh_delta <delta>

Defines the number of seconds between successive refreshings of a job's DCE login context. For example:
$dce_refresh_delta 18000

$enforce <limit>

MOM will enforce the given limit. Some limits have associated values, and appear in the configuration file like this:
$enforce variable_name value
See The PBS Professional Administrator's Guide.

$enforce mem

MOM will enforce each job's memory limit.

$enforce cpuaverage

MOM will enforce ncpus when the average CPU usage over a job's lifetime usage is greater than the job's limit.

$enforce average_trialperiod <seconds>

Modifies cpuaverage. Minimum number of seconds of job walltime before enforcement begins. Default: 120. Integer.

$enforce average_percent_over <percentage>

Modifies cpuaverage. Gives percentage by which a job may exceed its ncpus limit.

Default: 50. Integer.

$enforce average_cpufactor <factor>

Modifies cpuaverage. The ncpus limit is multiplied by factor to produce actual limit. Default: 1.025. Float.

$enforce cpuburst

MOM will enforce the ncpus limit when CPU burst usage exceeds the job’s limit.

$enforce delta_percent_over <percentage>

Modifies cpuburst. Gives percentage over limit to be allowed. Default: 50. Integer.

$enforce delta_cpufactor <factor>

Modifies cpuburst. The ncpus limit is multiplied by factor to produce actual limit. Default: 1.5. Float.

$enforce delta_weightup <factor>

Modifies cpuburst. Weighting factor for smoothing burst usage when average is increasing. Default: 0.4. Float.

$enforce delta_weightdown <factor>

Modifies cpuburst. Weighting factor for smoothing burst usage when average is decreasing. Default: 0.4. Float.

$ideal_load <load>

Defines the load below which the vnode is not con-

sidered to be busy. Used with the $max_load directive. No default. Float. Example:
$ideal_load 1.8

Use of $ideal_load adds a static resource to the vnode called “ideal_load”, which is only internally visible.

$kbd_idle <idle_wait> <min_use> <poll_interval>
Declares that the vnode will be used for batch jobs during periods when the keyboard and mouse are not in use.

The vnode must be idle for a minimum of idle_wait seconds before being considered available for batch jobs. No default. Integer.

The vnode must be in use for a minimum of min_use seconds before it becomes unavailable for batch jobs. Default: 10. Integer.

Mom checks for activity every poll_interval seconds. Default: 1. Integer.

Example:
$kbd_idle 1800 10 5

$logevent <mask>
Sets the mask that determines which event types are logged by pbs_mom. To include all debug events, use 0xffffffff.
Log events:

```
Name           Hex Value  Message Category
--------------------------------------------------
PBSE_ERROR      0001       Internal errors
PBSE_SYSTEM     0002       System errors
PBSE_ADMIN      0004       Administrative events
PBSE_JOB        0008       Job-related events
PBSE_JOB_USAGE  0010       Job accounting info
```

PBSE_SECURITY	0020	Security violations
PBSE_SCHED	0040	Scheduler events
PBSE_DEBUG	0080	Common debug messages
PBSE_DEBUG2	0100	Uncommon debug messages
PBSE_RESV	0200	Reservation-related info
PBSE_DEBUG3	0400	Rare debug messages

$max_check_poll <seconds>

Maximum time between polling cycles, in seconds. Must be greater than zero. Upper limit: 10 seconds. See the PBS Professional Administrator's Guide for usage. Integer.

The interval between each poll starts at $min_check_poll and increases with each cycle until it reaches $max_check_poll, after which it remains the same. The amount by which the cycle increases is 1/20 of the difference between $max_check_poll and $min_check_poll.

$min_check_poll <seconds>

Minimum time between polling cycles, in seconds. Must be greater than zero and less than $max_check_poll. See $max_check_poll. Integer.

$max_load <load> [suspend]

Defines the load above which the vnode is considered to be busy. Used with the $ideal_load directive. No default. Float. Example:
$max_load 3.5

Use of $max_load adds a static resource to the vnode called "max_load", which is only internally visible.

The optional suspend directive tells PBS to suspend jobs running on the node if the load average exceeds the max_load number, regardless of the source of the load (PBS and/or logged-in users). Without this directive, PBS will not suspend jobs due to load.

$prologalarm <timeout>

Defines the maximum number of seconds the prologue and epilogue may run before timing out. Default: 30. Integer. Example:
$prologalarm 30

$restart_background <true|false>

Controls how MOM runs a restart script after check-pointing a job. When this option is set to true, MOM forks a child which runs the restart script. The child returns when all restarts for all the local tasks of the job are done. MOM does not block on the restart. When this option is set to false, MOM runs the restart script and waits for the result. Boolean. Default: false.

$restart_transmogrify <true|false>

Controls how MOM runs a restart script after check-pointing a job. When this option is set to true, MOM runs the restart script, replacing the session ID of the original task's top process with the session ID of the script.

When this option is set to false, MOM runs the restart script and waits for the result. The restart script must restore the original session ID for all the processes of each task so that MOM can continue to track the job.

When this option is set to false and the restart uses an external command, the configuration parameter restart_background is ignored and treated as if it were set to true, preventing MOM from blocking on the restart.

Boolean. Default: false.

$restrict_user <value>

Controls whether users not submitting jobs have access to this machine. If value is “on”, restrictions are applied. See $restrict_user_exceptions and $restrict_user_maxsysid. Boolean. Default: off.

$restrict_user_exceptions <user_list>

Comma-separated list of users who are exempt from access restrictions applied by $restrict_user. Leading spaces within each entry are allowed.

$restrict_user_maxsysid <value>

Any user with a numeric user ID less than or equal to value is exempt from restrictions applied by $restrict_user.

If $restrict_user is on and no value exists for $restrict_user_maxsysid, PBS looks in /etc/login.defs, if it exists, for the value. Otherwise the default is used.

Integer. Default: 999

$restricted <hostname>

The hostname

is added to the list of hosts which will be allowed to connect to MOM without being required to use a privileged port. Hostnames can be wildcarded. For example, to allow queries from any host from the domain “xyz.com”:

$restricted   *.xyz.com

Queries from the hosts in the restricted list are only allowed access to information internal to this host, such as load average, memory available, etc. They may not run shell commands.

$suspendsig <susupend_signal> [resume_signal]

Alternate signal suspend_signal is used to suspend jobs instead of SIGSTOP. Optional resume_signal is used to resume jobs instead of SIGCONT.

$tmpdir <directory>

Location where each job's scratch directory will be created. Default: /tmp. For example:
$tmpdir /memfs

$usecp <hostname:source_prefix> <destination_prefix>

MOM will use /bin/cp to deliver output files when the destination is a network mounted file system, or when the source and destination are both on the local host, or when the source_prefix can be replaced with the destination_prefix on hostname. Both source_prefix and destination_prefix are absolute pathnames of directories, not files. For example:
$usecp   HostA:/users/work/myproj   /shared-work/proj_results

$wallmult <factor>

Each job's walltime usage is multiplied by this factor. For example:
$wallmult 1.5

Altix-only Initialization Values

pbs_accounting_workload_mgmt <value>

Controls whether CSA accouting is enabled. Name does not start with dollar sign. If set to “1”, “on”, or “true”, CSA accounting is enabled. If set to “0”, “off”, or “false”, accounting is disabled. Default: “true”; enabled.

IRIX-only Initialization Values

$checkpoint_upgrade <value>

PBS will pass a special upgrade checkpoint flag to the IRIX checkpoint system to use before upgrading. The value can be “1”, “true”, “on”, “0”, “false”, “off”. Default: false.

$enforce complexmem

Specifies whether memory segments should be shared across jobs, as shown by getmemusage. If not set, shared segments count in their entirety against each job, as shown by ps.

Static Resources

Static resources local to the vnode are described one resource to a line, with a name and value separated by white space. For example, tape drives of different types could be specified by:

```
tape3480    4
tape3420    2
tapedat     1
tape8mm     1
```

cpuset_create_flags <flags>

Lists the flags for when MOM does a cpusetCreate(3) for each job. flags is an or-ed list of flags. The flags are:

IRIX:

CPUSET_CPU_EXCLUSIVE
CPUSET_MEMORY_LOCAL
CPUSET_MEMORY_EXCLUSIVE
CPUSET_MEMORY_MANDATORY
CPUSET_MEMORY_KERNEL_AVOID
CPUSET_POLICY_KILL
CPUSET_POLICY_PAGE
CPUSET_POLICY_SHARE_WARN
CPUSET_POLICY_SHARE_FAIL
See SGI’s documentation on cpusetCreate(3).

Default: CPUSET_CPU_EXCLUSIVE|CPUSET_MEMORY_LOCAL|

CPUSET_MEMORY_EXCLUSIVE|CPUSET_MEMORY_MANDATORY|
CPUSET_POLICY_KILL|CPUSET_EVENT_NOTIFY

Altix ProPack 2, 3

CPUSET_CPU_EXCLUSIVE
CPUSET_MEMORY_LOCAL
CPUSET_MEMORY_EXCLUSIVE
CPUSET_KERNEL_AVOID
CPUSET_MEMORY_MANDATORY
CPUSET_POLICY_KILL
CPUSET_EVENT_NOTIFY
See SGI’s documentation on cpusetCreate(3x).

Default: CPUSET_CPU_EXCLUSIVE|CPUSET_MEMORY_LOCAL|

CPUSET_MEMORY_EXCLUSIVE|CPUSET_MEMORY_MANDATORY|
CPUSET_POLICY_KILL|CPUSET_EVENT_NOTIFY

Altix ProPack 4, 5

CPUSET_CPU_EXCLUSIVE
0

Default: CPUSET_CPU_EXCLUSIVE

cpuset_destroy_delay <delay>
MOM will wait delay seconds before issuing a cpusetDestroy(3) on the cpuset of a just-completed job. This allows processes time to finish. Defaults: Altix: 0; IRIX: 5. Integer. For example,
cpuset_destroy_delay 10

memreserved <megabytes>
The amount of per-vnode memory reserved for system overhead. Default: 0MB.
For example,
memreserved 16

IRIX-only Static Resources
The following resources are IRIX-specific.

alloc_nodes_greedy <0|1>
Determines whether MOM allocates nodeboards that are close together. A value of 1 means that MOM will allocate any nodeboard. See the PBS Professional Administrator’s Guide. Default: 1. For example,
alloc_nodes_greedy 0

cpuset_small_mem <mem>
Defines the maximum amount of memory for a small job. Jobs requesting mem kilobytes of memory will be considered small, and will be assigned a shared cpuset.

Default: the amount of memory on one nodeboard. For example,
cpuset_small_mem 1024

cpuset_small_ncpus <num>
Defines the maximum number of CPUs for a small job. Jobs requesting num or fewer will be considered small, and will be assigned a shared cpuset. Cannot exceed the number of CPUs on a nodeboard. Default: 1. For example,
cpuset_small_ncpus 2

enforce <IRIX_limit>
MOM will enforce the following IRIX-only limits for a job. Note the lack of a dollar-sign ("$") prefix. The following IRIX_limits can be enforced or not enforced.

enforce <mem | !mem>
Enforce or don't enforce each job's mem request. Default: enforced.

enforce <pvmem | !pvmem>
Enforce or don't enforce each job's pvmem request. Default: enforced.

enforce <vmem | !vmem>
Enforce or don't enforce each job's vmem request. Default: enforced.

enforce <walltime | !walltime>
Enforce or don't enforce each job's walltime request. Default: enforced.

enforce <pcput | !pcput>
Enforce or don’t enforce each job’s pcput request. Default: enforced.

enforce <cput | !cput>
Enforce or don’t enforce each job’s cput request. Default: enforced.

enforce <cpupct | !cpupct>
Enforce or don’t enforce each job’s cpupercent request. Default: not enforced.

enforce <file | !file>
Enforce or don’t enforce each job’s file request. Default: enforced.

enforce <hammer | !hammer>
Enforce or don’t enforce the killing of processes of unauthorized users. Default: not enforced.

enforce <nokill | !nokill>
Don’t kill or kill the non-PBS processes if hammer code is enabled. Default: don’t kill.

enforce <cpusets | !cpusets>
Enforce or don’t enforce cpusets. Default: enforced.

max_shared_nodes <vnodes>
The maximum number of nodeboards that are allowed to be assigned to shared cpusets. Default: 2048. For example,
max_shared_nodes 64

minnodecpus <num>
Sets num as the minimum number of working cpus on a vnode to consider it for running jobs. Default: smallest number of CPUs found on any nodeboard. Integer.
For example,
minnodecpus 2

minnodemem <mem>
Sets mem megabytes as the minimum amount of memory on a vnode to consider it for running jobs. MOM calculates that available memory for a job is (minnodemem - memreserved) MB. Default: smallest amount of memory found on any nodeboard.
For example,
minnodemem 512

schd_quantum <num>
Sets num as the minimum number of nodeboards to be assigned to a job. Default: 1. Integer.
For example,
schd_quantum 2

FILES AND DIRECTORIES

$PBS_HOME/mom_priv
Default directory for default configuration files.

$PBS_HOME/mom_priv/config
MOM’s default configuration file.

$PBS_HOME/mom_logs
Default directory for log files written by MOM.

$PBS_HOME/mom_priv/prologue
File containing administrative script to be run before job execution.

$PBS_HOME/mom_priv/epilogue
File containing administrative script to be run after job execution.

SIGNAL HANDLING
pbs_mom handles the following signals:

SIGHUP The pbs_mom daemon will reread its configuration files, close and reopen the log file, and reinitialize resource structures. On Blue Gene, SIGHUP does not do any of these. See the PBS Professional Administrator’s Guide.

SIGALRM MOM writes a log file entry. See the -a alarm_timeout option.

SIGINT The pbs_mom daemon exits, leaving all running jobs still running. See the -p option.

SIGKILL This signal is not caught. The pbs_mom daemon exits immediately.

SIGTERM, SIGXCPU, SIGXFSZ, SIGCPULIM, SIGSHUTDN
The pbs_mom daemon terminates all running children and exits.

SIGPIPE, SIGUSR1, SIGUSR2, SIGINFO
These are ignored.

All other signals have their default behavior installed.

EXIT STATUS

Greater than zero if the pbs_mom daemon fails to start, if the -s insert option is used with an existing scriptname, or if the administrator attempts to add a script whose name begins with “PBS”. Greater than zero if the administrator attempts to use the -s remove option on a nonexistent configuration file, or on a configuration file whose name begins with “PBS”. Greater than zero if the administrator attempts to use the -s show option on a nonexistent script.

SEE ALSO

The PBS Professional Administrator’s Guide, pbs_server(8B), pbs_sched(8B), qstat(1B), SGI’s IRIX documentation, SGI’s Altix documentation

NAME

pbs_mom_globus - start the PBS job monitoring and execution daemon that supports Globus

SYNOPSIS

```
pbs_mom_globus [-a alarm] [-c config] [-d directory] [-L logfile]
          [-M MOM_Globus_port] [-R RPP_Globus_port] [-r] [-x]
```

DESCRIPTION

The pbs_mom_globus command starts the operation of a batch Machine Oriented Mini-server, MOM, supporting Globus, on the local host. Typically, this command will be in a local boot file such as /etc/rc.local . To insure that the pbs_mom_globus command is not runnable by the general user community, the server will only execute if its real and effective uid is zero.

When pbs_mom_globus picks up a job for execution, the globus resource string -l site=globus:<gatekeeper> of the job is consulted and used to open up a connection to Globus. Pbs_mom forks a process for the job, starts up a globus-gass-server on a 1 server per unique username scheme, transforms #PBS directive lines in the user's submission script into an RSL string and submits the job to Globus, and exiting out of the forked process. All Globus job state changes are communicated back to pbs_mom_globus through periodic polling.

When job fails to submit due to globus job initialization failures, or any non GRAM authentication failures, then error message gets dumped into stderr and user is sent email.

When job fails due to no user password, proxy credential from certificate, or credential has expired, or some sort of "handshaking" error, then user is sent email of the error, and job is placed on hold.

Pbs_mom_globus will record a diagnostic message in a log file for any error occurrence. The log files are maintained in the mom_globus_logs directory below the home directory of the server. If the log file cannot be opened, the diagnostic message is written to the system console.

OPTIONS

-a alarm Used to specify the alarm timeout in seconds for computing a resource. Every time a resource request is processed, an alarm is set for the given amount of time. If the request has not completed before the given time, an alarm signal is generated. The default is 5 seconds.

-c config Specify a alternative configuration file, see description below. If this is a relative file name it will be relative to PBS_HOME/mom_globus_priv, see the -d option. If the specified file cannot be opened, pbs_mom_globus will abort. If the -c option is not supplied, pbs_mom_globus will attempt to open the default configuration file “config” in PBS_HOME/mom_globus_priv. If this file is not present, pbs_mom_globus will log the fact and continue.

-d directory Specifies the path of the directory which is the home of the servers working files, PBS_HOME. This option is typically used along with -M when debugging MOM Globus. The default directory is given by $PBS_SERVER_HOME which is typically /usr/spool/PBS.

-L logfile Specify an absolute path name for use as the log file. If not specified, MOM Globus will open a file named for the current date in the PBS_HOME/mom_globus_logs directory, see the -d option.

-M MOM_Globus_port
Specifies the port number on which the mini-server with Globus will listen for batch requests. Default: 15005.

-R RPP_Globus_port

Specifies the port number on which the mini-server with Globus will listen for resource monitor requests. Both a UDP and a TCP port of this number will be used. Default: 15006.

-r Specifies the impact on jobs which were in execution when the mini-server shut down. With the -r option, MOM Globus will cancel submitted Globus jobs, mark the jobs as terminated, and notify the batch server which owns the job.

Normally the mini-server is started from the system boot file without the -r option. The mini-server will make no attempt to signal the former session of any job which may have been running when the mini-server terminated. It is assumed that on reboot, all processes have been killed. It will however attempt to cancel the Globus job.

If the -r option is used following a reboot, process ids (pids) may be reused and MOM may kill a process that is not a batch session.

-x Disables the check for privileged port resource monitor connections. This is used mainly for testing since the privileged port is the only mechanism used to prevent any ordinary user from connecting.

CONFIGURATION FILE

The configuration file may be specified on the command line at program start with the -c flag. The use of this file is to provide several types of run time information to pbs_mom_globus: static resource names and values, external resources provided by a program to be run on request via a shell escape, and values to pass to internal set up functions at initialization (and re-initialization).

Each item type is on a single line with the component parts separated by white space. If the line starts with a hash mark (pound sign, #), the line is considered to be a comment and is skipped.

Static Resources

For static resource names and values, the configuration file contains a list of resource names/values pairs, one pair per line and separated by white space. An Example of static resource names and values could be the number of tape drives of different types and could be specified by

```
tape3480      4
tape3420      2
tapedat      1
tape8mm       1
```

Shell Commands

If the first character of the value is an exclamation mark (!), the entire rest of the line is saved to be executed through the services of the system(3) standard library routine.

The shell escape provides a means for the resource monitor to yield arbitrary information to the scheduler. Parameter substitution is done such that the value of any qualifier sent with the query, as explained below, replaces a token with a percent sign (%) followed by the name of the qualifier. For example, here is a configuration file line which gives a resource name of "escape":

```
escape     !echo 0xx %yyy
```

If a query for "escape" is sent with no qualifiers, the command executed would be "echo 0xx %yyy". If one qualifier is sent, "escape[xxx=hi there]", the command executed would be "echo hi there %yyy". If two qualifiers are sent, "escape[xxx=hi][yyy=there]", the command executed would be "echo hi there". If a qualifier is sent with no matching token in the command line, "escape[zzz=snafu]", an error is reported.

Initialization Value

An initialization value directive has a name which starts with a dollar sign ($) and must be known to MOM via an internal table. The entries in this table now are:

clienthost

which causes a host name to be added to the list of hosts which will be allowed to connect to MOM as long as they are using a privilaged port. For example, here are two configuration file lines which will allow the hosts "fred" and "wilma" to connect:

```
$clienthost     fred
$clienthost     wilma
```

Two host name are always allowed to connection to pbs_mom_globus, "localhost" and the name returned to pbs_mom_globus by the system call gethostname(). These names need not be specified in the configuration file.

restricted

which causes a host name to be added to the list of hosts which will be allowed to connect to MOM Globus without needing to use a privileged port. These names allow for wildcard matching. For example, here is a configuration file line which will allow queries from any host from the domain "ibm.com".

```
$restricted     *.ibm.com
```

The restriction which applies to these connections is that only internal queries may be made. No resources from a config file will be found. This is to prevent any shell commands from being run by a non-root process.

logevent

which sets the mask that determines which event types are logged by pbs_mom_globus. For example:

```
$logevent 0x1fff
$logevent 255
```

The first example would set the log event mask to 0x1ff (511) which enables logging of all events including debug events. The second example would set the mask to 0x0ff (255) which enables all events except debug events.

The configuration file must be “secure”. It must be owned by a user id and group id less than 10 and not be world writable.

FILES

$PBS_SERVER_HOME/mom_globus_priv
the default directory for configuration files, typical (/usr/spool/pbs)/mom_globus_priv.

$PBS_SERVER_HOME/mom_globus_logs
directory for log files recorded by the server.

$PBS_SERVER_HOME/mom_globus_priv/prologue
the administrative script to be run before job execution.

$PBS_SERVER_HOME/mom_globus_priv/epilogue
the administrative script to be run after job execution.

Signal Handling

Pbs_mom_globus handles the following signals:

SIGHUP causes pbs_mom_globus to re-read its configuration file, close and reopen the log file, and reinitialize resource structures.

SIGALRM
results in a log file entry. The signal is used to limit the time taken by certain children processes, such as the prologue and epilogue.

SIGINT and SIGTERM
Result in pbs_mom_globus terminating all running children and exiting. This is the action for the following signals as well: SIGXCPU, SIGXFSZ, SIGCPULIM, and SIGSHUTDN.

SIGPIPE, SIGUSR1, SIGUSR2, SIGINFO

are ignored.

All other signals have their default behavior installed.

EXIT STATUS

If the mini-server command fails to begin operation, the server exits with a value greater than zero.

SEE ALSO

The PBS Professional Administrator's Guide and the following manual pages: pbs_server(8B), pbs_sched(8B)

NAME

pbs_mpihp - run an MPI application in a PBS job with HP MPI

SYNOPSIS

pbs_mpihp [-np #] [-h host] [other HP mpirun options] program [args]

pbs_mpihp [HP mpirun options] -f appfile [-- [<extra_args>]]

pbs_mpihp --version

DESCRIPTION

The PBS command pbs_mpihp replaces the standard mpirun command in a PBS HP MPI job, for executing programs.

pbs_mpihp is a front end to the HP MPI version of mpirun. It is for PBS jobs running under HP-UX and Linux 2.4 and higher. pbs_mpihp has the same usage as mpirun. When pbs_mpihp is invoked from a PBS job, it will process the command line arguments, then call standard HP mpirun to actually start the MPI ranks. The ranks created will be mapped onto cpus on the nodes allocated to the PBS job. The environment variable MPI_REMSH will be set to $PBS_EXEC/bin/pbs_tmrsh. This will cause the processes that are created to become part of the PBS job.

The path to standard HP mpirun is found by checking to see if a link exists with the name PBS_EXEC/etc/pbs_mpihp. If this link exists, it will point to standard HP mpirun. If it does not exist, a call to mpirun -version will be made to determine if it is HP mpirun. If so, the call will be made to “mpirun” without an absolute path. If HP mpirun cannot be found, an error will be output, all temp files will be cleaned up and the script will exit with value 127.

If pbs_mpihp is invoked from outside a PBS job, it will pass all of its arguments directly to standard HP mpirun without further processing.

The first form above allows one executable to be specified. The second form above uses an appfile to list multiple executables. The format is described in the HP mpirun man page. If this form is used from inside a PBS job, the file will be read to determine what executables are to be run and how many processes will be started for each.

When HP MPI is wrapped with pbs_mpihp, “rsh” is the default used to start the mpids. If you wish to use “ssh” or something else, be sure to set the following in $PBS_HOME/pbs_environment:

```
PBS_RSHCOMMAND=ssh
```

or put the following in the job script:

```
exportPBS_RSHCOMMAND=<rsh_cmd>
```

Executing pbs_mpihp with the -client option is not supported under PBS.

USAGE

Usage is the same as for HP mpirun.

OPTIONS

All options except the following are passed directly to HP mpirun with no modification.

-client Not supported.

-np number Specifies the number of processes to run on the PBS nodes.

-h host Ignored by pbs_mpihp.

-l user Ignored by pbs_mpihp.

-f appfile The specified appfile is read by pbs_mpihp.

--version The pbs_mpihp command returns its PBS version information and exits. This option can only be used alone.

ENVIRONMENT VARIABLES

SEE ALSO

The PBS Professional Administrator’s Guide

mpirun(1)

NAME

pbs_mpilam - run MPI programs under PBS with LAM MPI

SYNOPSIS

pbs_mpilam [options]

pbs_mpilam --version

DESCRIPTION

The PBS command pbs_mpilam replaces the standard mpirun command in a PBS LAM MPI job, for executing programs under Linux 2.4 or higher.

Usage is the same as for LAM mpirun. All options are passed directly to mpirun. If used to run a single program, PBS tracks resource usage and controls all user processes spawned by the program. If used to run multiple programs as specified in an application file (no <where> argument and no -np/-c option), then PBS does not manage the spawned user processes of each program.

If the where argument is not specified, then pbs_mpilam will try to run the user's program on all available CPUs using the C keyword.

OPTIONS

options The pbs_mpilam command uses the same options as mpirun.

--version

The pbs_mpilam command returns its PBS version information and exits. This option can only be used alone.

ENVIRONMENT VARIABLES

PATH

The PATH on remote machines must contain PBS_EXEC/bin.

SEE ALSO

The PBS Professional Administrator's Guide

mpirun(1)

NAME

pbs_mpirun - run MPI programs under PBS with MPICH

SYNOPSIS

pbs_mpirun [options]

pbs_mpirun --version

DESCRIPTION

The PBS command pbs_mpirun replaces the standard mpirun command in a PBS MPICH job using P4 running under Linux 2.4 and higher. Usage is the same as for mpirun, except for the -machinefile option. All other options are passed directly to mpirun.

OPTIONS

options The options to pbs_mpirun are the same as for mpirun, except for the -machinefile option. This is generated by pbs_mpirun. The user should not attempt to specify -machinefile.

The value for -machinefile is a temporary file created from PBS_NODEFILE in the format:

```
hostname-1[:number of processors]
hostname-2[:number of processors]
hostname-n[:number of processors]
```

where if the number of processors is not specified, it is 1. An attempt by the user to specify the -machinefile option will result in a warning saying “Warning, -machinefile value replaced by PBS”.

The default value for the -np option is the number of entries in PBS_NODEFILE.

--version

The pbs_mpirun command returns its PBS version information and exits. This option can only be used alone.

ENVIRONMENT VARIABLES

pbs_mpirun modifies P4_RSHCOMMAND and PBS_RSHCOMMAND. Users should not
edit these. pbs_mpirun copies the value of P4_RSHCOMMAND into PBS_RSH-COMMAND.

PATH

The PATH on remote machines must contain PBS_EXEC/bin.

SEE ALSO

The PBS Professional Administrator's Guide

mpirun(1)

NAME

pbs_password - set or update password of a PBS user

SYNOPSIS

pbs_password [-r] [-s server] [-d] [user]
pbs_password --version

DESCRIPTION

The pbs_password command is used to set or update the password of a PBS user. The user does not have to have any jobs on the system.

When no options are given to pbs_password, the password credential on the default PBS server for the current user, i.e. the user who executes the command, is updated to the prompted password. Any user jobs previously held due to an invalid password are not released.

OPTIONS

-r Any user jobs previously held due to an invalid password are released.

-s server Allows user to specify server where password will be changed.

-d Deletes the password.

user The password credential of user user is updated to the prompted password. If user is not the current user, this action is only allowed if:

1. The current user is root or admin.

2. User user has given the current user explicit access via the ruserok() mechanism:

   a. The hostname of the machine from which the current user is logged in appears in the server's hosts.equiv file, or

b. The current user has an entry in user's HOME-DIR\.rhosts file.

--version The pbs_password command returns its PBS version information and exits. This option can only be used alone.

EXIT STATUS

0	Success
-1	single_signon_password_enable not set
-2	Password of user on server failed to be created/updated
-3	Failed to release jobs held due to bad password owned by user on server
-4	Failed to delete password of user on server
-5	Current user not authorized to change password of user

SEE ALSO

qhold(1B), qrls(1B), qselect(1B), ruserok()

NAME

pbs_probe - report PBS diagnostic information

SYNOPSIS

pbs_probe [-f | -v]
pbs_probe --version

DESCRIPTION

The pbs_probe command reports post-installation information that is useful for PBS diagnostics. Aside from the direct information that is supplied on the command line, pbs_probe uses as the source for basic information the file /etc/pbs.conf and the values of any of the following environment variable that may be set in the environment in which pbs_probe is run: PBS_CONF, PBS_HOME, PBS_EXEC, PBS_START_SERVER, PBS_START_MOM, and PBS_START_SCHED.

In order to execute pbs_probe, the user must have PBS Operation or Manager privilege.

Used without options, the pbs_probe runs in “report” mode. In this mode pbs_probe reports on any errors in the PBS infrastructure files that it detects. The problems are categorized, and a list of the problem messages placed in each category are output. Those categories which are empty do not show in the output.

OPTIONS

-f Run in “fix” mode. In this mode pbs_probe will examine each of the relevant infrastructure files and, where possible, fix any errors that it detects, and print a message of what got changed. If it is unable to fix a problem, it will simply print a message regarding what was detected.

-v Run in “verbose” mode. If the verbose option is turned on, pbs_probe will also output a complete list of the infrastructure files that it checked.

--version The pbs_probe command returns its PBS version information and exits. This option can only be used alone.

STANDARD ERROR

The pbs_probe command will write a diagnostic message to standard error for each error occurrence.

FILES

/etc/pbs.conf /etc/init.d/pbs

SEE ALSO

The PBS Professional Administrator’s Guide and the following manual pages: pbs_server(8B), pbs_sched(8B), pbs_mom(8B).

Local 12 April 2007 pbs_probe(8B)

NAME

pbs_sched - run the PBS scheduler

SYNOPSIS

```
pbs_sched [-a alarm] [assign_ssinodes] [-d home] [-L logfile]
          [-p file] [-S port] [-R port] [-n] [-N]
pbs_sched --version
```

DESCRIPTION

pbs_sched is the PBS scheduling daemon. It schedules PBS jobs.

pbs_sched must be executed with root permission.

OPTIONS

-a alarm
: Time in seconds to wait for a scheduling cycle to finish. If this takes too long to finish, an alarm signal is sent, and the scheduler is restarted. If a core file does not exist in the current directory, abort() is called and a core file is generated. The default for alarm is 1000 seconds.

-d home
: The directory in which the scheduler will run. The default is PBS_HOME/sched_priv.

-L logfile
: The absolute path and filename of the log file. If this option is not given, the scheduler will open a file named for the current date in the PBS_HOME/sched_logs directory. The scheduler writes its PBS version and build information to the logfile whenever it starts up or the logfile is rolled to a new file. See the -d option.

-p file
: Any output which is written to standard out or standard error will be written to this file. The pathname can be absolute or relative, in which case it will be relative

to PBS_HOME/sched_priv. If this option is not given, the file used will be PBS_HOME/sched_priv/sched_out . See the -d option.

-S port The port for the scheduler to use. If this option is not given, the default port for the PBS scheduler is taken from PBS_SCHEDULER_SERVICE_PORT, in pbs.conf. Default: 15004.

-R port The port for MOM to use. If this option is not given, the port number is taken from PBS_MANAGER_SERVICE_PORT, in pbs.conf. Default: 15003.

-n This will tell the scheduler to not restart itself if it receives a sigsegv or a sigbus. The scheduler will by default restart itself if it receives either of these two signals. The scheduler will not restart itself if it receives either one within five minutes of starting.

-N Instructs the scheduler not to detach itself from the current session.

--version The pbs_sched command returns its PBS version information and exits. This option can only be used alone.

FILES

$PBS_HOME/sched_priv is the default directory for configuration files.

$PBS_HOME/sched_priv/holidays is the holidays file.

SIGNAL HANDLING

SIGHUP The scheduler will close and reopen its log file and reread the config file if one exists.

SIGALRM
If the scheduler exceeds the time limit, the Alarm will cause the scheduler to attempt to core dump and restart itself.

SIGINT and SIGTERM
Will result in an orderly shutdown of the scheduler.

All other signals have the default action installed.

EXIT STATUS
Zero upon normal termination.

SEE ALSO
The PBS Professional Administrator's Guide, pbs_server(8B), pbs_mom(8B)

NAME

pbs_server - start a PBS batch server

SYNOPSIS

```
pbs_server [-a active] [-A acctfile] [-C] [-d config_path] [-e mask]
        [-F seconds] [-g mom_globus_port] [-G mom_globusRPP_port]
        [-L logfile] [-M mom_port] [-N] [-p port] [-R momRPP_port]
        [-S scheduler_port] [-t type]
pbs_server --version
```

DESCRIPTION

The pbs_server command starts the operation of a batch server on the local host. Typically, this command will be in a local boot file such as /etc/rc.local . If the batch server is already in execution, pbs_server will exit with an error. To insure that the pbs_server command is not runnable by the general user community, the server will only execute if its real and effective uid is zero.

The server will record a diagnostic message in a log file for any error occurrence. The log files are maintained in the server_logs directory below the home directory of the server. If the log file cannot be opened, the diagnostic message is written to the system console. The server writes its PBS version and build information to the logfile whenever it starts up or the logfile is rolled to a new file.

OPTIONS

-a active Specifies if scheduling is active or not. This sets the server attribute scheduling. If the option argument is “true” (“True”, “t”, “T”, or “1”), the server is active and the PBS job scheduler will be called. If the argument is “false” (“False”, “f”, “F”, or “0), the server is idle, and the scheduler will not be called and no jobs will be run. If this option is not specified, the server will retain the prior value of the scheduling attribute.

-A acctfile

Specifies an absolute path name of the file to use as the accounting file. If not specified, the file is named for the current date in the PBS_HOME/server_priv/accounting directory.

-C The server starts up, creates the database, and exits. Windows only.

-d config_path
Specifies the path of the directory which is home to the servers configuration files, PBS_HOME. A host may have multiple servers. Each server must have a different configuration directory. The default configuration directory is given by the symbol $PBS_HOME which is typically /usr/spool/PBS.

-e mask Specifies a log event mask to be used when logging. See "log_events" in the pbs_server_attributes(7B) man page and in the ERS.

-F seconds
Specifies the number of seconds that the secondary server should wait before taking over when it believes the primary server is down. If the number of seconds is specified as -1, the secondary will make one attempt to contact the primary and then become active. Default: 30 seconds.

-g mom_globus_port
Specifies the host name and/or port number on which the server should connect the PBS Mom Globus daemon. The option argument, mom_conn, is one of the forms: host_name, [:]port_number, or host_name:port_number. If host_name not specified, the local host is assumed. If port_number is not specified, the default port is assumed. Default: 15005.

-G mom_globus_RPPport

Specifies the port number on which the server should query the up/down status of PBS Mom Globus daemon. Default: 15006.

-L logfile
Specifies an absolute path name of the file to use as the log file. If not specified, the file is one named for the current date in the PBS_HOME/server_logs directory, see the -d option.

-M mom_port
Specifies the host name and/or port number on which the server should connect the job executor, MOM. The option argument, mom_conn, is one of the forms: host_name, [:]port_number, or host_name:port_number. If host_name not specified, the local host is assumed. If port_number is not specified, the default port is assumed. See the -M option for pbs_mom(8). Default: 15002.

-N The server runs in standalone mode. In Windows, it does not register as a Windows service. On other platforms, MOM will not detach from the current session.

-p port Specifies the port number on which the server will listen for batch requests. If multiple servers are running on a single host, each must have its own unique port number. This option is for use in testing with multiple batch systems on a single host. Default: 15001.

-R mom_RPPport
Specifies the port number on which the server should query the up/down status of Mom. See the -R option for pbs_mom(8). Default: 15003.

-S scheduler_port

Specifies the port number to which the server should connect when contacting the Scheduler. The option argument, scheduler_conn, is of the same syntax as under the -M option. Default: 15004.

-t type Specifies the impact on jobs when the server restarts. type argument is:

hot All jobs in the Running state are retained in that state. Any job that was requeued into the Queued state from the Running state when the server last shut down will be run immediately, assuming the required resources are available. This returns the server to the same state as when it went down. After those jobs are restarted, then normal scheduling takes place for all remaining queued jobs. All other jobs are retained in their current state.

If a job cannot be restarted immediately because of a missing resource, such as a node being down, the server will attempt to restart it periodically for up to 5 minutes. After that period, the server will revert to a normal state, as if warm started, and will no longer attempt to restart any remaining jobs which were running prior to the shutdown.

warm All jobs in the Running state are retained in that state. All other jobs are maintained in their current state. The job scheduler will typically make new selections for which jobs are placed into execution. Warm is the default if -t is not specified.

cold All jobs are purged. Positive confirmation is required before this direction is accepted.

create The server will discard any existing configuration files: server, nodes, queues and jobs, and initialize configuration files to the default values. The server is idled (scheduling set false).

--version The pbs_server command returns its PBS version information and exits. This option can only be used alone.

FILES

$PBS_HOME/server_priv
default directory for configuration files.

$PBS_HOME/server_logs
directory for log files recorded by the server.

Signal Handling

On receipt of the following signals, the server performs the defined action:

SIGHUP The current server log and accounting log are closed and reopened. This allows for the prior log to be renamed and a new log started from the time of the signal.

SIGTERM and SIGINT
Causes a rapid orderly shutdown of pbs_server, identical to “qterm -t quick”.

SIGSHUTDN
On systems (Unicos) where SIGSHUTDN is defined, it also causes an orderly “quick” shutdown of the server.

SIGPIPE, SIGUSR1, SIGUSR2
These signals are ignored.

All other signals have their default behavior installed.

EXIT STATUS

If the server command fails to begin batch operation, the server exits with a value greater than zero.

SEE ALSO

The PBS Professional Administrator's Guide and the following manual pages: qsub (1B), pbs_connect(3B), pbs_mom(8B), pbs_sched(8B), pbsnodes(8B), qdisable(8B), qenable(8B), qmgr(8B), qrun(8B), qstart(8B), qstop(8B), and qterm(8B)

Local 12 April 2007 pbs_server(8B)

NAME

pbs_tclsh - TCL shell with TCL-wrapped PBS API

SYNOPSIS

pbs_tclsh

pbs_tclsh -version

DESCRIPTION

The pbs_tclsh is a version of the TCL shell which includes wrapped versions of the PBS external API. The PBS TCL API is documented in the pbs_tclapi (3B) manual page.

The pbs_tclsh command is used to query MOM. For example:

```
> pbs_tclsh
tclsh> openrm <hostname>
<file descriptor>
tclsh> addreq <file descriptor> "loadave"
tclsh> getreq <file descriptor>
<load average>
tclsh> closereq <file descriptor>
```

OPTIONS

--version

The pbs_tclsh command returns its PBS version information and exits. This option can only be used alone.

STANDARD ERROR

The pbs_tclsh command will write a diagnostic message to standard error for each error occurrence.

SEE ALSO

The PBS Professional Administrator's Guide, the PBS External Reference Specification, and the following manual pages: pbs_wish(8B), pbs_server(8B), pbs_mom(8B), pbs_sched(8B)

NAME

pbs_tmrsh - TM-enabled replacement for rsh/ssh for use by MPI implementations

SYNOPSIS

pbs_tmrsh host [-l username] [-n] command [args ...]
pbs_tmrsh --version

DESCRIPTION

The pbs_tmrsh command attempts to emulate an “rsh” connection to the specified host, via underlying calls to the Task Management (TM) API. The program is intended to be used during MPI integration activities, and not by end-users. The initial version of this program is targeted for use with MPICH and HP-MPI.

Running “pbs_tmrsh host command” will cause a PBS task to be started on “host” running “command”. The “host” may be in IP dot address form.

The environment variables used by the two MPI implementations to point to the rsh work-alike (MPI_REMSH in the case of HP and P4_RSHCOMMAND for MPICH) must be set in the job environment and point to the full path for pbs_tmrsh.

The file $PBS_HOME/pbs_environment will be used to set an environment variable PATH to be used to search for the program executable. This applies to both Windows and UNIX. It is expected that a full path will be specified for the command and the PATH variable will not be needed.

Output and errors are written to the PBS job’s output and error files, not to standard output/error.

OPTIONS

-l username Specifies the username under which to execute the task. If used, username must match the username running the pbs_tmrsh command.

-n Currently a no-op; provided for MPI implementations that expect to call rsh with the “-n” option.

--version The pbs_tmrsh command returns its PBS version information and exits. This option can only be used alone.

STANDARD ERROR

The pbs_tmrsh command will write a diagnostic message to the PBS job's error file for each error occurrence.

EXIT STATUS

The pbs_tmrsh program will exit with the exit status of the remote command or with 255 if an error occurred. This is because ssh works this way.

SEE ALSO

The PBS Professional Administrator's Guide and the following manual pages: pbs_attach(8B), tm(3)

NAME

pbs_wish - TK window shell with TCL-wrapped PBS API

SYNOPSIS

```
pbs_wish

pbs_wish --version
```

DESCRIPTION

The pbs_wish command is a version of the TK window shell which includes wrapped versions of the PBS external API. The PBS TCL API is documented in the pbs_tclapi(3B) manual page.

OPTIONS

--version

The pbs_wish command returns its PBS version information and exits. This option can only be used alone.

STANDARD ERROR

The pbs_wish command will write a diagnostic message to standard error for each error occurrence.

SEE ALSO

The PBS Professional Administrator's Guide and the following manual pages: pbs_tclsh(8B), pbs_mom(8B), pbs_server(8B), pbs_sched(8B)

NAME

pbsfs - show or manipulate PBS fairshare usage data

SYNOPSIS

pbsfs -[t|p]

pbsfs -g entity

pbsfs -s entity usage_value

pbsfs -d

pbsfs -e

pbsfs -c entity1 entity2

pbsfs --version

DESCRIPTION

The pbsfs command is used to print or manipulate the PBS scheduler's fairshare usage data. Some options should only be used when the scheduler is not running. There are multiple parts to a fairshare node and you can print these data in different formats. The pbsfs command must be run by root; otherwise it will print the error message, "Unable to access fairshare data".

The data:

fairshare entity	the entity in the fairshare tree.
group	the group ID the node is in (i.e. the node's parent).
cgroup	the group ID of this group
shares	the number of shares the group has
usage	the amount of usage

percentage
: the percentage the entity has of the tree. Note that only the leaf nodes sum to 100%. If all of the nodes are summed, the result will be greater then 100%. Only the leaf nodes of the tree are fairshare entities.

usage / perc
: The value the scheduler will use to the pick which entity has priority over another. The smaller the number the higher the priority.

Path from root
: The path from the root of the tree to the node. This is useful because the scheduler will look down the path to compare two nodes to see which has the higher priority.

resource The resource for which the scheduler accumulates usage for its fairshare calculations. This defaults to cput (cpu seconds) but can be set in the scheduler's config file.

OPTIONS

Scheduler can be running:

-t print the fairshare tree in a hierarchical format.

-p print the fairshare tree in a flat format with more data.

-g entity print one entry with all data and print the path from the root of the tree to the node.

-c entity1 entity2
: compare two fairshare entities

Scheduler must be down:

-s entity usage_value
: set entity's usage value to usage_value. Please note that

editing a non-leaf node is ignored. All non-leaf node usage values are calculated each time the scheduler is run/HUPed.

-d decay the fairshare tree (divide all values in half)

-e trim fairshare tree to just the entities in the resource_group file

Scheduler can be running or down:

--version The pbsfs command returns its PBS version information and exits. This option can only be used alone.

SEE ALSO

The PBS Professional Administrator’s Guide, pbs_sched(8B)

NAME

pbsnodes - query PBS host status or mark hosts free or offline

SYNOPSIS

```
pbsnodes [ -c | -o | -r ] [-s server] hostname [hostname ...]

pbsnodes [-l] [-s server]

pbsnodes -a [-v] [-s server]

pbsnodes --version
```

DESCRIPTION

The pbsnodes command is used to query the status of hosts, or to mark hosts FREE or OFFLINE. The pbsnodes command obtains host information by sending a request to the PBS server.

To print the status of the specified host or hosts, run pbsnodes with no options (except the -s option) and with a list of hosts.

To print the command usage, run pbsnodes with no options and without a list of hosts.

PBS Manager or Operator privilege is required to execute pbsnodes with the -c , -o , or -r options.

To remove a node from the scheduling pool, mark it OFFLINE. If it is marked DOWN, when the server next queries the MOM, and can connect, the node will be marked FREE.

For hosts with multiple vnodes, pbsnodes operates on a host and all of its vnodes, where the hostname is resources_available.host. See the -v option.

To act on individual vnodes, use the qmgr command.

OPTIONS

(no options) If neither options nor a host list is given, the pbsnodes command prints usage syntax.

-a Lists all hosts and all their attributes (available and used.)

When listing a host with multiple vnodes:

The output for the jobs attribute lists all the jobs on all the vnodes on that host. Jobs that run on more than one vnode will appear once for each vnode they run on.

For consumable resources, the output for each resource is the sum of that resource across all vnodes on that host.

For all other resources, e.g. string and boolean, if the value of that resource is the same on all vnodes on that host, the value is returned. Otherwise the output is the literal string "<various>".

-c host_list Clears OFFLINE and DOWN from listed hosts. The listed hosts will become FREE if they are online, or remain DOWN if they are not (for example, powered down.) Requires PBS Manager or Operator privilege.

-l Lists all hosts marked as DOWN or OFFLINE. Each such host's state and comment attribute (if set) is listed. If a host also has state STATE-UNKNOWN, that will be listed. For hosts with multiple vnodes, only hosts where all vnodes are marked as DOWN or OFFLINE are listed.

-o host_list Marks listed hosts as OFFLINE even if currently in use. This is different from being marked DOWN. A host that is marked OFFLINE will continue to execute the jobs already on it, but will be removed from the scheduling pool (no more jobs will be scheduled on it.) Requires PBS Manager or Operator privilege.

-r host_list Clears OFFLINE from listed hosts.

-s server Specifies the PBS server to which to connect.

-v Can only be used with the -a option. Prints one entry for each vnode in the PBS complex. (Information for all hosts is displayed.)

The output for the jobs attribute for each vnode lists the jobs executing on that vnode. The output for resources and attributes lists that for each vnode.

--version The pbsnodes command returns its PBS version information and exits. This option can only be used alone.

OPERANDS

server Specifies the server to which to connect. Default: default server.

host_list Specifies the host(s) whose status will be returned. Format: hostname [hostname ...]

EXIT STATUS

Zero upon success.

Greater than zero, if:

incorrect operands are given,
pbsnodes cannot connect to the server,
there is an error querying the server for the nodes.

SEE ALSO

The PBS Professional Administrator’s Guide,
pbs_server(8B) and qmgr(8B)

Local 20 Sept 2007 pbsnodes(8B)

NAME

pbsrun - general-purpose wrapper script for mpirun

SYNOPSIS

pbsrun

pbsrun --version

DESCRIPTION

pbsrun is a wrapper script for any of several versions of mpirun. This provides a user-transparent way for PBS to control jobs which call mpirun in their jobscripts. The pbsrun_wrap script instantiates pbsrun so that the wrapper script for the specific version of mpirun being used has the same name as that version of mpirun.

If the mpirun wrapper script is run inside a PBS job, then it will translate any mpirun call of the form:

```
     mpirun [options] <executable> [args]
```

into

```
     mpirun [options] pbs_attach [special_option_to_pbs_attach] \
             <executable> [args]
```

where [special options] refer to any option needed by pbs_attach to do its job (e.g. -j $PBS_JOBID).

If the wrapper script is executed outside of PBS, a warning is issued about “not running under PBS”, but it proceeds as if the actual program had been called in standalone fashion.

The pbsrun wrapper script is not meant to be executed directly but instead it is instantiated by pbsrun_wrap. It is copied to the target directory and renamed “pbsrun.<mpirun version/flavor>” where <mpirun version/flavor> is a string that identifies the mpirun version being wrapped (e.g. ch_gm).

The pbsrun script, if executed inside a PBS job, runs an initialization script, named $PBS_EXEC/lib/MPI/pbsrun.<mpirun version/flavor>.init, then parses mpirun-like arguments from the command line, sorting which options and option values to retain, to ignore, or to transform, before calling the actual mpirun script with a “pbs_attach” prefixed to the executable. The actual mpirun to call is found by tracing the link pointed to by $PBS_EXEC/lib/MPI/pbsrun.<mpirun version/flavor>.link.

For all of the wrapped MPIs, the maximum number of ranks that can be launched is the number of entries in $PBS_NODEFILE.

The wrapped MPIs are:

MPICH-GM's mpirun (mpirun.ch_gm) with rsh/ssh
MPICH-MX's mpirun (mpirun.ch_mx) with rsh/ssh
MPICH-GM's mpirun (mpirun.mpd) with MPD
MPICH-MX's mpirun (mpirun.mpd) with MPD
MPICH2's mpirun
Intel MPI's mpirun
MVAPICH1's mpirun
MVAPICH2's mpiexec
IBM's poe

OPTIONS

--version

The pbsrun command returns its PBS version information and exits. This option can only be used alone.

INITIALIZATION SCRIPT

The initialization script, called $PBS_EXEC/lib/MPI/pbsrun.<mpirun version/flavor>.init, where <mpirun version/flavor> reflects the mpirun flavor/version being wrapped, can be modified by an administrator to customize against the local flavor/version of mpirun being wrapped.

1. Inside this sourced init script, 8 variables are set:

```
options_to_retain=”-optA -optB <val> -optC <val1> val2> ...”
options_to_ignore=”-optD -optE <n> -optF <val1> val2> ...”
options_to_transform=”-optG -optH <val> -optI <val1> val2> ...”
options_to_fail=”-optY -optZ ...”
options_to_configfile=”-optX <val> ...”
options_with_another_form=”-optW <val> ...”
pbs_attach=pbs_attach
options_to_pbs_attach=”-J $PBS_JOBID”
```

options_to_retain

Space-separted list of options and values that pbsrun.<mpirun version/flavor> passes on to the actual mpirun call. options must begin with “-” or “--”, and option arguments must be specified by some arbitrary name with left and right arrows, as in “<val1>”.

options_to_ignore

Space-separated list of options and values that pbsrun.<mpirun version/flavor> does not pass on to the actual mpirun call. Options must begin with “-” or “--”, and option arguments must be specified by arbitrary names with left and right arrows, as in “<n>”.

options_to_transform

Space-separted list of options and values that pbsrun modifies before passing on to the actual mpirun call.

option_to_fail

Space-separated list of options that will cause pbsrun to exit upon encountering a match.

options_to_configfile

Single option and value that refers to the name of the “configfile” containing command line segments found in certain versions of mpirun.

options_with_another_form

Space-separated list of options and values that can be found in options_to_retain, options_to_ignore, or options_to_transform, whose syntax has an alternate, unsupported form.

pbs_attach

Path to pbs_attach, which is called before the <executable>

argument of mpirun.

options_to_pbs_attach
Special options to pass to the pbs_attach call. You may pass variable references (e.g. $PBS_JOBID) and they are substituted by pbsrun to actual values.

If pbsrun encounters any option not found in options_to_retain, options_to_ignore, and options_to_transform, then it is flagged as an error.

2. These functions are created inside the init script.
These can be modified by the PBS administrator.

```
transform_action () {
        # passed actual values of $options_to_transform
        args=$*
    }

boot_action () {
        mpirun_location=$1
    }

evaluate_options_action () {
        # passed actual values of transformed options
        args=$*
    }

configfile_cmdline_action () {
        args=$*
    }

end_action () {
        mpirun_location=$1
    }
```

transform_action()

The pbsrun.<mpirun version/flavor> wrapper script invokes the function transform_action() (called once on each matched item and value) with actual options and values received matching one of the “options_to_transform”. The function returns a string to pass on to the actual mpirun call.

boot_action()

Performs any initialization tasks needed before running the actual mpirun call. For instance, GM’s MPD requires the MPD daemons to be user-started first. This function is called by the pbsrun.<mpirun version/flavor> script with the location of actual mpirun passed as the first argument. Also, the pbsrun.<mpirun version/flavor> checks for the exit value of this function to determine whether or not to progress to the next step.

evaluate_options_action()

Called with the actual options and values that resulted after consulting options_to_retain, options_to_ignore, options_to_transform, and executing transform_action(). This provides one more chance for the script writer to evaluate all the options and values in general, and make any necessary adjustments, before passing them on to the actual mpirun call. For instance, this function can specify what the default value is for a missing -np option.

configfile_cmdline_action()

Returns the actual options and values to be put in before the option_to_configfile parameter.

configfile_firstline_action()
Returns the item that is put in the first line of the configuration file specified in the option_to_configfile parameter.

end_action()
Called by pbsrun.<mpirun version/flavor> at the end of execution. It undoes any action done by transform_action(), like cleanup of temporary files. It is also called when pbsrun.<mpirun version/flavor> is prematurely killed. This function is called with the location of actual mpirun passed as first argument.

3. The actual mpirun program to call is the path pointed to by
$PBS_EXEC/lib/MPI/pbsrun.<mpirun version/flavor>.link."

Modifying *.init scripts
In order for administrators to modify *.init scripts without breaking package verification in RPM, master copies of the initialization scripts are named *.init.in. pbsrun_wrap instantiates the *.init.in files as *.init. For instance, $PBS_EXEC/lib/MPI/pbsrun.mpich2.init.in is the master copy, and pbsrun_wrap instantiates it as $PBS_EXEC/lib/MPI/pbsrun.mpich2.init. pbsrun_unwrap takes care of removing the *.init files.

MPIRUN VERSIONS/FLAVORS

MPICH-GM's mpirun (mpirun.ch_gm) with rsh/ssh: pbsrun.ch_gm

SYNTAX

```
pbsrun.ch_gm <options> <executable> <arg1> <arg2> ... <argn>
```

```
    This is the PBS wrapper script to MPICH-GM's mpirun (mpirun.ch_gm)
    with rsh/ssh process startup method.

    If executed inside a PBS job, this allows for  PBS  to  track  all
    MPICH-GM  processes  started  by  rsh/ssh  so that PBS can perform
    accounting and have complete job control.

    If executed outside of a PBS job, it behaves exactly as  if  stan-
    dard mpirun.ch_gm was used.

OPTIONS HANDLING
    If  executed  inside  a  PBS  job script, all mpirun.ch_gm options
    given are passed on to the actual mpirun call  with  these  excep-
    tions:


    -machinefile <file>
        The  file  argument  contents are ignored and replaced by the
        contents of the $PBS_NODEFILE.


    -np  If  not  specified,  the  number  of  entries  found  in  the
        $PBS_NODEFILE is used.


    -pg  The use of the -pg option, for having multiple executables on
        multiple hosts, is allowed but it is up to user to make  sure
        only  PBS  hosts are specified in the process group file; MPI
        processes spawned are not guaranteed to be under the  control
        of PBS.

WRAP/UNWRAP
    To wrap MPICH-GM's mpirun script:
        # pbsrun_wrap [MPICH-GM_BIN_PATH]/mpirun.ch_gm pbsrun.ch_gm

    To unwrap MPICH-GM's mpirun script:
        # pbsrun_unwrap pbsrun.ch_gm

-----------------------------------------------------------
MPICH-MX's mpirun (mpirun.ch_mx) with rsh/ssh: pbsrun.ch_mx
-----------------------------------------------------------
```

SYNTAX

pbsrun.ch_mx <options> <executable> <arg1> <arg2> ... <argn>

This is the PBS wrapper script to MPICH-MX's mpirun (mpirun.ch_mx) with rsh/ssh process startup method.

If executed inside a PBS job, this allows for PBS to track all MPICH-MX processes started by rsh/ssh so that PBS can perform accounting and has complete job control.

If executed outside of a PBS job, it behaves exactly as if standard mpirun.ch_mx was used.

OPTIONS HANDLING

If executed inside a PBS job script, all mpirun.ch_gm options given are passed on to the actual mpirun call with some exceptions:

-machinefile <file>
The file argument contents is ignored and replaced by the contents of the $PBS_NODEFILE.

-np If not specified, the number of entries found in the $PBS_NODEFILE is used.

-pg The use of the -pg option, for having multiple executables on multiple hosts, is allowed but it is up to user to make sure only PBS hosts are specified in the process group file; MPI processes spawned are not guaranteed to be under the control of PBS.

WRAP/UNWRAP

To wrap MPICH-MX's mpirun script:

```
# pbsrun_wrap [MPICH-MX_BIN_PATH]/mpirun.ch_mx pbsrun.ch_mx
```

To unwrap MPICH-MX's mpirun script:

```
    # pbsrun_unwrap pbsrun.ch_mx
```

```
-------------------------------------------------------
MPICH-GM's mpirun (mpirun.mpd) with MPD: pbsrun.gm_mpd
-------------------------------------------------------

SYNTAX

    pbsrun.gm_mpd <options> <executable> <arg1> <arg2> ... <argn>

    This  is  the PBS wrapper script to MPICH-GM's mpirun (mpirun.mpd)
    with MPD process startup method.

    If executed inside a PBS job, this allows for  PBS  to  track  all
    MPICH-GM processes started by the MPD daemons so that PBS can per-
    form accounting have and complete job control.

    If executed outside of a PBS job, it behaves exactly as  if  stan-
    dard mpirun.ch_gm with MPD was used.

OPTIONS HANDLING
    If  executed  inside  a  PBS job script, all mpirun.ch_gm with MPD
    options given are passed on to the actual mpirun call  with  these
    exceptions:


    -m <file>
        The  file  argument  contents are ignored and replaced by the
        contents of the $PBS_NODEFILE.


    -np  If  not  specified,  the  number  of  entries  found  in  the
        $PBS_NODEFILE is used.


    -pg  The use of the -pg option, for having multiple executables on
        multiple hosts, is allowed but it is up to user to make  sure
        only  PBS  hosts are specified in the process group file; MPI
        processes spawned are not guaranteed to be under the  control
        of PBS.
```

STARTUP/SHUTDOWN

The script starts MPD daemons on each of the unique hosts listed in $PBS_NODEFILE, using either rsh or ssh method based on value of environment variable RSHCOMMAND. The default is rsh.

The script also takes care of shutting down the MPD daemons at the end of a run.

WRAP/UNWRAP

To wrap MPICH-GM's mpirun script with MPD:

```
# pbsrun_wrap [MPICH-GM_BIN_PATH]/mpirun.mpd pbsrun.gm_mpd
```

To unwrap MPICH-GM's mpirun script with MPD:

```
# pbsrun_unwrap pbsrun.gm_mpd
```

MPICH-MX's mpirun (mpirun.mpd) with MPD: pbsrun.mx_mpd

SYNTAX

```
pbsrun.mx_mpd <options> <executable> <arg1> <arg2> ... <argn>
```

This is the PBS wrapper script to MPICH-MX's mpirun (mpirun.mpd) with MPD process startup method.

If executed inside a PBS job, this allows for PBS to track all MPICH-MX processes started by the MPD daemons so that PBS can perform accounting and have complete job control.

If executed outside of a PBS job, it behaves exactly as if standard mpirun.ch_mx with MPD was used.

OPTIONS HANDLING

If executed inside a PBS job script, all mpirun.ch_gm with MPD options given are passed on to the actual mpirun call with these exceptions:

```
    -m <file>
        The file argument contents are ignored and  replaced  by  the
        contents of the $PBS_NODEFILE.


    -np  If  not  specified,  the  number  of  entries  found  in  the
        $PBS_NODEFILE is used.


    -pg  The use of the -pg option, for having multiple executables on
        multiple  hosts, is allowed but it is up to user to make sure
        only PBS hosts are specified in the process group  file;  MPI
        processes  spawned are not guaranteed to be under the control
        of PBS.

STARTUP/SHUTDOWN
    The script starts MPD daemons on each of the unique  hosts  listed
    in  $PBS_NODEFILE,  using either rsh or ssh method, based on value
    of environment variable RSHCOMMAND -- rsh is the default.

    The script also takes care of shutting down the MPD daemons at the
    end of a run.

WRAP/UNWRAP
    To wrap MPICH-MX's mpirun script with MPD:
        # pbsrun_wrap [MPICH-MX_BIN_PATH]/mpirun.mpd pbsrun.mx_mpd

    To unwrap MPICH-MX's mpirun script with MPD:
        # pbsrun_unwrap pbsrun.mx_mpd

-----------------------------
MPICH2's mpirun: pbsrun.mpich2
-----------------------------

SYNTAX

    pbsrun.mpich2 [global args] [local args] executable [args] \
                [: [local args] executable [args]]
    - or -
    pbsrun.mpich2 -configfile <configfile>
```

where <configfile> contains command line segments as lines:

    [local args] executable1 [args]
    [local args] executable2 [args]
    [local args] executable3 [args]

This is the PBS wrapper script to MPICH2's mpirun.

If executed inside a PBS job, this allows for PBS to track all MPICH2 processes so that PBS can perform accounting and have complete job control.

If executed outside of a PBS job, it behaves exactly as if standard MPICH2's mpirun was used.

OPTIONS HANDLING

If executed inside a PBS job script, all MPICH2's mpirun options given are passed on to the actual mpirun call with these exceptions:

-host and -ghost

For specifying the execution host to run on. Not passed on to the actual mpirun call.

-machinefile <file>

The file argument contents are ignored and replaced by the contents of the $PBS_NODEFILE.

MPICH2's mpirun -localonly <x>

For specifying the <x> number of processes to run locally. Not supported. The user is advised instead to use the equivalent arguments: -np <x> -localonly . The reason for this is that the pbsrun wrapper script cannot handle a variable number of arguments to an option (e.g. "-localonly" has 1 argument and "-localonly <x>" has 2 arguments).

-np If user did not specify a -np option, then no default value is provided by the PBS wrapper scripts. It is up to the local mpirun to decide what the reasonable default value should be, which is usually 1.

STARTUP/SHUTDOWN

The script takes care of ensuring that the MPD daemons on each of the hosts listed in the $PBS_NODEFILE are started. It also takes care of ensuring that the MPD daemons have been shut down at the end of MPI job execution.

WRAP/UNWRAP

To wrap MPICH2's mpirun script:

```
# pbsrun_wrap [MPICH2_BIN_PATH]/mpirun pbsrun.mpich2
```

To unwrap MPICH2's mpirun script:

```
# pbsrun_unwrap pbsrun.mpich2
```

Intel MPI's mpirun: pbsrun.intelmpi

SYNTAX

```
pbsrun.intelmpi [mpdboot options] \
            [mpiexec options] executable [prog-args] \
            [: [mpiexec options] executable [prog-args]]
- or -
pbsrun.intelmpi [mpdboot options] -f <configfile>
```

where [mpdboot options] are any options to pass to the mpdboot program, which is automatically called by Intel MPI's mpirun to start MPDs, and <configfile> contains command line segments as lines.

This is the PBS wrapper script to Intel MPI's mpirun.

If executed inside a PBS job, this allows for PBS to track all Intel MPI processes so that PBS can perform accounting and have complete job control.

If executed outside of a PBS job, it behaves exactly as if standard Intel MPI’s mpirun was used.

OPTIONS HANDLING

If executed inside a PBS job script, all of the options to the PBS interface to MPI’s mpirun are passed to the actual mpirun call with these exceptions:

-host and -ghost

For specifying the execution host to run on. Not passed on to the actual mpirun call.

-machinefile <file>

The file argument contents are ignored and replaced by the contents of the $PBS_NODEFILE.

mpdboot options --totalnum=* and --file=*

Ignored and replaced by the number of unique entries in $PBS_NODEFILE and name of $PBS_NODEFILE respectively.

arguments to mpdboot options --file=* and -f <mpd_hosts_file>

Replaced by $PBS_NODEFILE.

-s If pbsrun.intelmpi is called inside a PBS job, Intel MPIs mpirun -s argument to mpdboot are not supported as this closely matches the mpirun option -s <spec> . The user can simply run a separate mpdboot -s before calling mpirun. A warning message is issued by pbsrun.intelmpi upon encountering a -s option telling users of the supported form.

-np If the user does not specify a -np option, then no default value is provided by the PBS wrap scripts. It is up to the local mpirun to decide what the reasonable default value should be, which is usually 1.

STARTUP/SHUTDOWN

Intel MPI's mpirun itself takes care of starting/stopping the MPD daemons. pbsrun.intelmpi always passes the arguments -totalnum=<number of mpds to start> and -file=<mpd_hosts_file> to the actual mpirun, taking its input from unique entries in $PBS_NODEFILE.

WRAP/UNWRAP

To wrap Intel MPI's mpirun script:

```
# pbsrun_wrap [INTEL_MPI_BIN_PATH]/mpirun pbsrun.intelmpi
```

To unwrap Intel MPI's mpirun script:

```
# pbsrun_unwrap pbsrun.intelmpi
```

---

MVAPICH1's mpirun: pbsrun.mvapich1

---

SYNTAX

```
pbsrun.mvapich1 <mpirun options> <executable> <options>
```

Only one executable can be specified. This is the PBS wrapper script to MVAPICH1's mpirun.

If executed inside a PBS job, this allows for PBS to be aware of all MVAPICH1 ranks and track their resources, so that PBS can perform accounting and have complete job control.

If executed outside of a PBS job, it behaves exactly as if standard mpirun was used.

OPTIONS HANDLING

If executed inside a PBS job script, all mpirun options given are passed on to the actual mpirun call with these exceptions:

-map <list>

The map option is ignored.

-exclude <list>
The exclude option is ignored.

-machinefile <file>
The machinefile option is ignored.

-np If not specified, the number of entries found in the $PBS_NODEFILE is used.

WRAP/UNWRAP

To wrap MVAPICH1's mpirun script:
```
# pbsrun_wrap <path-to-actual-mpirun> pbsrun.mvapich1
```

To unwrap MVAPICH1's mpirun script:
```
# pbsrun_unwrap pbsrun.mvapich1
```

MVAPICH2's mpiexec: pbsrun.mvapich2

SYNTAX

```
pbsrun.mvapich2 <mpiexec args> executable <executable's
              args> [: <mpiexec args> executable
              <executable's args>]
```

Multiple executables can be specified using the colon notation. This is the PBS wrapper script to MVAPICH2's mpiexec, which have the same format.

If executed inside a PBS job, this allows for PBS to be aware of all MVAPICH2 ranks and track their resources, so that PBS can perform accounting and have complete job control.

If executed outside of a PBS job, it behaves exactly as if standard mpiexec was used.

OPTIONS HANDLING

If executed inside a PBS job script, all mpiexec options given are passed on to the actual mpiexec call with these exceptions:

-host <host>

The host argument contents are ignored.

-machinefile <file>

The file argument contents are ignored and replaced by the contents of the $PBS_NODEFILE.

WRAP/UNWRAP

To wrap MVAPICH2's mpiexec script:

```
# pbsrun_wrap <path-to-actual-mpiexec> pbsrun.mvapich2
```

To unwrap MVAPICH2's mpiexec script:

```
# pbsrun_unwrap pbsrun.mvapich2
```

IBM's poe: pbsrun.poe

SYNTAX

```
pbsrun.poe <options> <executable> <arg1> <arg2> ... <argn>
```

This is the PBS wrapper script to IBM's poe, allowing poe jobs to use the HPS in US mode.

If executed inside a PBS job, this allows for PBS to track all resources and MPI ranks. PBS can perform accounting and have complete job control.

If executed outside of a PBS job, it behaves exactly as if standard poe was used.

The script will use the -euilib {ip | us} option and the MP_EUILIB environment variable to indicate use of US mode, to maintain compatibility with standard poe.

OPTIONS HANDLING

If executed inside a PBS job script, all pbsrun.poe options given are passed on to the actual mpirun call with these exceptions:

-hostfile <file>

The file argument contents are ignored.

-procs <numranks>

If the -procs option or the MP_PROCS environment variable is not set by the user, a default of the number of entries in the file $PBS_NODEFILE is used.

-euilib {ip | us}

If the command line option -euilib is set, it will take precedence over the MP_EUILIB environment variable. If the -euilib option is set to us , user mode is set for the job. If the option is set to any other value, that value is passed to standard poe.

MP_MSG_API

This option can only take the values “MPI” or “LAPI”.

ENVIRONMENT VARIABLES

MP_EUILIB

If the MP_EUILIB environment variable is set to us , user mode is set for the job. If the variable is set to any other value, that value is passed to standard poe.

MP_HOSTFILE

The MP_HOSTFILE environment variable is excised.

MP_PROCS

If the -procs option or the MP_PROCS environment variable is not set by the user, a default of the number of entries in the file $PBS_NODEFILE is used.

MP_MSG_API

This variable can only take the values “MPI” or “LAPI”.

WRAP/UNWRAP

To wrap IBM’s poe:

```
# pbsrun_wrap <path_to_actual_poe> pbsrun.poe
```

To unwrap the IBM poe mpirun script:

```
# pbsrun_unwrap pbsrun.poe
```

REQUIREMENTS

The mpirun being wrapped must be installed and working on all the nodes in the PBS cluster.

ERRORS

If pbsrun encounters any option not found in options_to_retain, options_to_ignore, and options_to_transform, then it is flagged as an error.

SEE ALSO

The PBS Professional Administrator’s Guide

pbs_attach(8B), pbsrun_wrap(8B), pbsrun_unwrap(8B)

Local 20 Sept 2007 pbsrun(8B)

NAME

pbsrun_unwrap - unwraps mpirun, reversing pbsrun_wrap

SYNOPSIS

pbsrun_unwrap pbsrun.<mpirun version/flavor>

pbsrun_unwrap --version

DESCRIPTION

The pbsrun_unwrap script is used to reverse the actions of the pbsrun_wrap script.

Use pbsrun_wrap to wrap mpirun.

USAGE

Syntax:

```
pbsrun_unwrap pbsrun.<mpirun version/flavor>
```

For example, running the following:

```
pbsrun_unwrap pbsrun.ch_gm
```

causes the following actions:

```
Checks for a link in $PBS_EXEC/lib/MPI/pbsrun.ch_gm.link;
If one exists, get the pathname it points to:
/opt/mpich-gm/bin/mpirun.ch_gm.actual

rm $PBS_EXEC/lib/MPI/pbsrun.mpirun.ch_gm.link

rm /opt/mpich-gm/bin/mpirun.ch_gm

rm $PBS_EXEC/bin/pbsrun.ch_gm

mv /opt/mpich-gm/bin/mpirun.ch_gm.actual \
/opt/mpich-gm/bin/mpirun.ch_gm
```

OPTIONS

--version

The pbsrun_unwrap command returns its PBS version information and exits. This option can only be used alone.

SEE ALSO

The PBS Professional Administrator’s Guide

pbs_attach(8B), pbsrun(8B), pbsrun_wrap(8B)

Local 28 June 2006 pbsrun_unwrap(8B)

NAME

pbsrun_wrap - general-purpose script for wrapping mpirun in pbsrun

SYNOPSIS

pbsrun_wrap [-s] <path_to_actual_mpirun> pbsrun.<mpirun version/flavor>

pbsrun_wrap --version

DESCRIPTION

The pbsrun_wrap script is used to wrap any of several versions of mpirun in pbsrun. The pbsrun_wrap script creates a symbolic link with the same path and name as the mpirun being wrapped. This calls pbsrun, which uses pbs_attach to give MOM control of jobs. The result is transparent to the user; when mpirun is called from inside a PBS job, PBS can monitor and control the job, but when mpirun is called from outside of a PBS job, it behaves as it would normally. See the pbs_attach(8B) and pbsrun(8B) man pages.

Use pbsrun_unwrap to reverse the process.

OPTIONS

-s Sets the "strict_pbs" options in the various initialization scripts (e.g. pbsrun.bgl.init, pbsrun.ch_gm.init, etc...) to 1 from the default 0. This means that the mpirun being wrapped by pbsrun will only be executed if inside a PBS environment. Otherwise, the user will get the error:

> Not running under PBS exiting since strict_pbs is enabled;
> execute only in PBS

For Blue Gene systems, to use strict_pbs with mpirun, wrap the mpiruns on the front-end node and the service node by specifying pbsrun_wrap - s. This will ensure that no Blue Gene partitions are spawned outside of PBS.

--version

The pbsrun_wrap command returns its PBS version information and exits. This option can only be used alone.

USAGE

Syntax:

```
pbsrun_wrap [-s] <path_to_actual_mpirun> pbsrun.<mpirun version/flavor>
```

Any mpirun version/flavor that can be wrapped has an initialization script ending in “.init”, found in $PBS_EXEC/lib/MPI:

```
$PBS_EXEC/lib/MPI/pbsrun.<mpirun version/flavor>.init.
```

The pbsrun_wrap script instantiates the pbsrun wrapper script as pbsrun.<mpirun version/flavor> in the same directory where pbsrun is located, and sets up the link to actual mpirun call via the symbolic link

```
$PBS_EXEC/lib/MPI/pbsrun.<mpirun version/flavor>.link
```

For example, running:

```
pbsrun_wrap /opt/mpich-gm/bin/mpirun.ch_gm pbsrun.ch_gm
```

causes the following actions:

Save original mpirun.ch_gm script:

```
mv /opt/mpich-gm/bin/mpirun.ch_gm \
   /opt/mpich/gm/bin/mpirun.ch_gm.actual
```

Instantiate pbsrun wrapper script as pbsrun.ch_gm:

```
cp $PBS_EXEC/bin/pbsrun $PBS_EXEC/bin/pbsrun.ch_gm
```

Link “mpirun.ch_gm” to actually call “pbsrun.ch_gm”:

```
ln -s $PBS_EXEC/bin/pbsrun.ch_gm /opt/mpich-gm/bin/mpirun.ch_gm
```

Create a link so that “pbsrun.ch_gm” calls “mpirun.ch_gm.actual”:

```
ln -s /opt/mpich-gm/bin/mpirun.ch_gm.actual \
      $PBS_EXEC/lib/MPI/pbsrun.ch_gm.link
```

REQUIREMENTS

The mpirun being wrapped must be installed and working on all the nodes in the PBS cluster.

SEE ALSO

The PBS Professional Administrator's Guide

pbs_attach(8B), pbsrun(8B), pbsrun_unwrap(8B)

NAME
printjob - print job data and attributes from binary files

SYNOPSIS
printjob [-a] file [file...]
printjob --version

DESCRIPTION
The printjob command is used to print the contents of the binary file representing a PBS batch job saved within the PBS system. By default all the job data including job attributes are printed.

This command is useful for troubleshooting, as during normal operation, the qstat(8B) command is the perferred method for displaying job-specific data and attributes.

In order to execute printjob, the user must have PBS Operator or Manager privilege.

OPTIONS

-a	Suppresses the printing of job attributes.
--version	The printjob command returns its PBS version information and exits. This option can only be used alone.

OPERANDS
The printjob command accepts one or more file operands.

STANDARD ERROR
The printjob command will write a diagnostic message to standard error for each error occurrence.

EXIT STATUS
Upon successful processing of all the operands presented to the print-job command, the exit status will be a value of zero.

If the printjob command fails to process any operand, the command exits with a value greater than zero.

SEE ALSO
The PBS Professional Administrator’s Guide, pbs_server(8B), qstat(8B)

Local 31 August 2007 printjob(8B)

NAME

qdisable - disable input to a PBS destination

SYNOPSIS

qdisable destination ...
qdisable --version

DESCRIPTION

The qdisable command directs that a destination should no longer accept batch jobs. If the command is accepted, the destination will no longer accept Queue Job requests which specified the disabled queue. Jobs which already reside in the queue will continue to be processed. This allows a queue to be "drained."

In order to execute qdisable, the user must have PBS Operation or Manager privilege.

OPTIONS

--version

The qdisable command returns its PBS version information and exits. This option can only be used alone.

OPERANDS

The qdisable command accepts one or more destination operands. The operands are one of three forms:

queue
@server
queue@server

If queue is specified, the request is to disable that queue at the default server. If the @server form is given, the request is to disable all the queues at that server. If a full destination identifier, queue@server, is given, the request is to disable the named queue at the named server.

STANDARD ERROR

The qdisable command will write a diagnostic message to standard error for each error occurrence.

EXIT STATUS

Upon successful processing of all the operands presented to the qdisable command, the exit status will be a value of zero.

If the qdisable command fails to process any operand, the command exits with a value greater than zero.

SEE ALSO

The PBS Professional Administrator’s Guide and the following manual pages: pbs_server(8B), qmgr(8B), and qenable(8B)

Local 12 April 2007 qdisable(8B)

NAME

qenable - enable input to a PBS destination

SYNOPSIS

qenable destination ...
qenable --version

DESCRIPTION

The qenable command directs that a destination should accept batch jobs.

The qenable command sends a Manage request to the batch server specified by destination. If the command is accepted, the destination will accept Queue Job requests which specified the queue.

In order to execute qenable, the user must have PBS Operation or Manager privilege.

OPTIONS

--version

The qenable command returns its PBS version information and exits. This option can only be used alone.

OPERANDS

The qenable command accepts one or more destination operands. The operands are one of three forms:

queue
@server
queue@server

If queue is specified, the request is to enable that queue at the default server. If the @server form is given, the request is to enable all the queues at that server. If a full destination identifier, queue@server, is given, the request is to enable the named queue at the named server.

STANDARD ERROR

The qenable command will write a diagnostic message to standard error for each error occurrence.

EXIT STATUS

Upon successful processing of all the operands presented to the qenable command, the exit status will be a value of zero.

If the qenable command fails to process any operand, the command exits with a value greater than zero.

SEE ALSO

The PBS Professional Administrator's Guide and the following manual pages: pbs_server(8B), qdisable(8B), and qmgr(8B)

Local 12 April 2007 qenable(8B)

NAME

qmgr - administrator’s command interface for managing PBS

SYNOPSIS

qmgr [-a] [-c command] [-e] [-n] [-z] [server...]
qmgr --version

DESCRIPTION

The qmgr command is used to create or delete queues and nodes, to set or change node, queue, server or scheduler attributes, including resources, and to view information about queues, nodes, the server, and the scheduler. See the pbs_resources(7B), pbs_queue_attributes(7B), pbs_server_attributes(7B), and pbs_node_attributes(7B) man pages.

The qmgr command provides different services depending on the level of privilege of the user. All users can list or print attributes. Operator privilege is required in order to set or unset attributes. Manager privilege is required in order to create or delete queues or nodes.

The command reads directives from standard input. To save and recreate a configuration, print the configuration information to a file, then read it back in later. See the print command and the STANDARD INPUT section.

Attributes whose values are unset do not appear in the output of the qmgr command.

OPTIONS

-a Abort qmgr on any syntax errors or any requests rejected by a server.

-c command Execute a single command and exit qmgr . The command must be enclosed in double quotes, e.g.
qmgr -c “print server”

-e Echo all commands to standard output.

-n No commands are executed, syntax checking only is performed.

-z No errors are written to standard error.

--version The qmgr command returns its PBS version information and exits. This option can only be used alone.

OPERANDS

The server operands identify the name of the batch server to which the administrator requests are sent. Each server conforms to the following syntax:

host_name[:port]

where host_name is the network name of the host on which the server is running and port is the port number to which to connect. If port is not specified, the default port number is used.

If server is not specified, the administrator requests are sent to the local server.

STANDARD INPUT

The qmgr command reads standard input for directives until end of file is reached, or the exit or quit directive is read. To recreate a configuration from a saved configuration file, use qmgr < savedfile. See the print command.

STANDARD OUTPUT

If Standard Output is connected to a terminal, a command prompt will be written to standard output when qmgr is ready to read a directive.

If the -e option is specified, qmgr will echo the directives read from standard input to standard output.

STANDARD ERROR

If the -z option is not specified, the qmgr command will write a diagnostic message to standard error for each error occurrence.

EXTENDED DESCRIPTION

If qmgr is invoked without the -c option and standard output is connected to a terminal, qmgr will write a prompt to standard output and read a directive from standard input.

Commands can be abbreviated to their minimum unambiguous form. A command is terminated by a new line character or a semicolon (“;”) character. Multiple commands may be entered on a single line. A command may extend across lines by escaping the new line character with a backslash “\”.

Comments begin with the # character and continue to end of the line. Comments and blank lines are ignored by qmgr.

Type “help” at the qmgr prompt for syntax and command information.

DIRECTIVE SYNTAX

A qmgr directive is one of the following forms:

command server [names] [attr OP value[,attr OP value,...]]
command queue [names] [attr OP value[,attr OP value,...]]
command node [names] [attr OP value[,attr OP value,...]]
command sched [names] [attr OP value[,attr OP value,...]]

Where,
command is the command to perform on a object. Commands are:

active sets the active objects. If the active objects are specified, and the name is not given in a qmgr cmd, the active object names will be used.

create is to create a new object, applies to queues and nodes.

delete is to destroy an existing object, applies to queues and nodes.

set is to define or alter attribute values of the object.

unset is to clear the value of attributes of the object. Note, this form does not accept an OP and value, only the attribute name.

list is to list the current attributes and associated values of the object.

print If “print queue QUEUE” is given, qmgr prints the commands used to create the queue and set its attributes.

If “print server” is given, qmgr prints the commands used to create any queues and set their attributes, as well as those used to set server attributes. The file produced by Qmgr: print server > savedfile can be used as input to the qmgr command when recreating a configuration.

names is a list of one or more names of specific objects. The name list is in the form:

[name][@server][,queue_name[@server]...]

with no intervening white space. The name of an object is declared when the object is first created. If the name is @server, then all the objects of specified type at the server will be affected. Node attributes cannot be used as node names.

attr specifies the name of an attribute of the object which is to be set or modified. If the attribute is one which consist of a set of resources, then the attribute is specified in the form:

attribute_name.resource_name

OP operation to be performed with the attribute and its value:

= set the value of the attribute. If the attribute has an existing value, the current value is replaced with the new value.

+= increase the current value of the attribute by the amount in the new value. When used for a string array, adds the new value as another string after a comma.

-= decrease the current value of the attribute by the amount in the new value. When used for a string array, removes the first matching string.

value the value to assign to an attribute, which may be a resource. If the value includes white space, commas or other special characters, such as the # character, the value string must be enclosed in double quotes (“).

Resource values can be any string made up of alphanumeric, comma (“,”), underscore (“_”), dash (“-”), colon (“:”), slash (“/”), backslash (“ space (“ “), and equal sign (“=”) characters.

The following are examples of qmgr directives:

```
list sched @serverA - list serverA's scheduler's attributes
l sched @default - list attributes for default server's scheduler
l sched @default pbs_version
        - list PBS version for default server's scheduler
set node mynode resources_available.software = "myapp=/tmp/foo"
create queue fast priority=10,queue_type=e,enabled = true,max_running=0
set queue fast max_running +=2
create queue little
set queue little resources_max.mem=8mw,resources_max.cput=10
unset queue fast max_running
set node state = "down,offline"
active server s1,s2,s3
list queue @server1
set queue max_running = 10    - uses active queues
```

EXIT STATUS

Upon successful processing of all the operands presented to the qmgr command, the exit status will be a value of zero.

If the qmgr command fails to process any operand, the command exits with a value greater than zero.

SEE ALSO

The PBS Professional Administrator's Guide,
pbs_server(8B), pbs_queue_attributes(7B), pbs_server_attributes(7B), pbs_node_attributes(7B), qstart(8B), qstop(8B), qenable(8B), and qdisable(8).

NAME

qrun - run a PBS batch job now

SYNOPSIS

qrun [-a] [-H vnode-specification] job_identifier_list
qrun --version

DESCRIPTION

The qrun command is used to force a job to run, regardless of scheduling position or resource requirements.

In order to execute qrun, the user must have PBS Operator or Manager privilege, and the job must be in the Queued state and reside in an execution queue.

The qrun command can be used on a subjob or a range of subjobs, but not on a job array. When it is used on a range of subjobs, the non-running subjobs in that range are run.

NOTE: If you use a -H vnode_specification option to run a job, but specify insufficient vnodes or resources, the job may not run correctly. Avoid using this option unless you are sure.

OPTIONS

-a The qrun command exits before the job actually starts execution.

(no -H option) A request will be made of the Scheduler to schedule this job. If the Scheduler is available, the job will run immediately if it is otherwise runnable:

The queue in which the job resides is an execution queue and the queue is started.

Either the resources required by the job are available, or preemption is enabled and the required resources can be made available by preempting jobs that are running.

(with -H option)

With the -H option, all scheduling policies are bypassed and the job is run directly. The job will be run immediately on the named vnodes, regardless of current usage on those vnodes with the excption of vnode state. The job will not be run and the qrun request will be rejected if any named vnode is down, offline, already allocated exclusively or would need to be allocated exclusively and another job is already running on the vnode.

-H vnode_specification, without resources

The vnode_specification without resources has this format:

    (vchunk)[+(vchunk) ...]

where vchunk has the format

    vnode[+vnode ..]

Example: -H (VnodeA+VnodeB)+(VnodeC)

PBS will apply one requested chunk from the job's selection directive in round-robin fashion to each vchunk in the list. Each vchunk must be sufficient to run the job's corresponding chunk, otherwise the job may not execute correctly.

-H vnode_specification, with resources

The vnode_specification with resources has this format:

    (vchunk)[+(vchunk) ...]

where vchunk has the format

    vnode:vnode_resources[+vnode:vnode_resources ...]

and where vnode_resources has the format

    resource=value[:resource=value ...]

```
Example: -H (VnodeA:mem=100kb:ncpus=1) \
         +(VnodeB:mem=100kb:ncpus=2+VnodeC:mem=100kb)
```

PBS creates a new selection directive from the vnode_specification, using it instead of the original specification from the user. Any single resource specification will result in the job's original selection directive being ignored. Each vchunk must be sufficient to run the job's corresponding chunk, otherwise the job may not execute correctly.

--version The qrun command returns its PBS version information and exits. This option can only be used alone.

OPERANDS

The qrun command accepts a job_identifier_list containing one or more job_identifiers of the form:

sequence_number[.server_name][@server]

Note that some shells require that you enclose a job array identifier in double quotes.

STANDARD ERROR

The qrun command will write a diagnostic message to standard error for each error occurrence.

EXIT STATUS

Zero, on success.

Greater than zero, if the qrun command fails to process any operand.

SEE ALSO

The PBS Professional Administrator's Guide,
qsub(1B), qmgr(8B), pbs_runjob(3B)

NAME

qstart - start PBS batch job processing at a destination

SYNOPSIS

qstart destination ...
qstart --version

DESCRIPTION

The qstart command directs that a destination should process batch jobs. If the destination is an execution queue, the server will begin to schedule jobs that reside in the queue for execution. If the destination is a routing queue, the server will begin to route jobs from that queue.

In order to execute qstart, the user must have PBS Operation or Manager privilege.

OPTIONS

--version
: The qstart command returns its PBS version information and exits. This option can only be used alone.

OPERANDS

The qstart command accepts one or more destination operands. The operands are one of three forms:

queue
@server
queue@server

If queue is specified, the request is to start that queue at the default server. If the @server form is given, the request is to start all queues at that server. If a full destination identifier, queue@server, is given, the request is to start the named queue at the named server.

STANDARD ERROR

The qstart command will write a diagnostic message to standard error for each error occurrence.

EXIT STATUS

Upon successful processing of all the operands presented to the qstart command, the exit status will be a value of zero.

If the qstart command fails to process any operand, the command exits with a value greater than zero.

SEE ALSO

The PBS Professional Administrator’s Guide and the following manual pages: pbs_server(8B), qstop(8B), and qmgr(8B)

NAME

qstop - stop PBS batch job processing at a destination

SYNOPSIS

qstop destination ...
qstop --version

DESCRIPTION

The qstop command directs that a destination should stop processing batch jobs. If the destination is a execution queue, the server will cease scheduling jobs that reside in the queue for execution. If the destination is a routing queue, the server will cease routing jobs from that queue.

In order to execute qstop, the user must have PBS Operation or Manager privilege.

OPTIONS

--version

The qstop command returns its PBS version information and exits. This option can only be used alone

OPERANDS

The qstop command accepts one or more destination operands. The operands are one of three forms:

queue
@server
queue@server

If queue is specified, the request is to stop that queue at the default server. If the @server form is given, the request is to stop all the queues at that server. If a full destination identifier, queue@server, is given, the request is to stop the named queue at the named server.

STANDARD ERROR

The qstop command will write a diagnostic message to standard error for each error occurrence.

EXIT STATUS

Upon successful processing of all the operands presented to the qstop command, the exit status will be a value of zero.

If the qstop command fails to process any operand, the command exits with a value greater than zero.

SEE ALSO

The PBS Professional Administrator’s Guide and the following manual pages: pbs_server(8B), qstart(8B), and qmgr(8B)

NAME

tracejob - print log messages for a PBS job

SYNOPSIS

```
tracejob [-a|l|m|s|v] [-c count] [-f filter] [-n days] [-p path]
      [-w cols] jobid
tracejob --version
```

DESCRIPTION

The tracejob command extracts log messages for a given jobid and prints them in chronological order.

Log messages contain server, scheduler, accounting and MOM information. Server logs contain information such as when a job was queued or modified. Scheduler logs contain clues as to why a job is not running. Accounting logs contain accounting records for when a job was queued, started, ended or deleted. MOM logs contain information about what happened to a job while it was running.

To get MOM log messages for a job, tracejob must be run on the machine on which the job ran.

All users have access to server, scheduler and MOM information. Only Administrator or root can access accounting information.

Some log messages appear many times. In order to make the output of tracejob more readable, messages that appear over a certain number of times (see option -c below) are restricted to only the most recent message.

If tracejob is run on a job array, the information returned will be about the job array itself, and not its subjobs. Job arrays do not have associated MOM log messages. If tracejob is run on a subjob, the same types of log messages will be available as for a job. Certain log messages that occur for a regular job will not occur for a subjob.

Note that some shells require that you enclose a job array identifier in double quotes.

OPTIONS

-a Do not report accounting information.

-c <count> Set excessive message limit to count. If a message is logged at least count times, only the most recent message is printed. The default for count is 15.

-f <filter> Do not include logs of type filter. The -f option can be used more than once on the command line.

filter: error, system, admin, job, job_usage, security, sched, debug, debug2

-l Do not report scheduler information.

-m Do not report MOM information.

-n <days> Report information from up to days days in the past. Default is 1 = today.

-p <path> Use path as path to PBS_HOME on machine being queried.

-s Do not report server information.

-w <cols> Width of current terminal. If not specified by the user, tracejob queries OS to get terminal width. If OS doesn't return anything, default is 80.

-v Verbose. Report more of tracejob's errors than default.

-z Disable excessive message limit. Excessive message limit is enabled by default.

--version The tracejob command returns its PBS version information and exits. This option can only be used alone.

EXIT STATUS

Zero upon successful processing of all options.

Exit value is greater than zero if tracejob is unable to process any options.

SEE ALSO

The PBS Professional Administrator’s Guide

pbs_server(8B), pbs_sched(8B), pbs_mom(8B)

Chapter 11

Informational Man Pages

The man pages below give information about PBS Professional.

NAME

pbs_job_attributes - attributes of PBS jobs

DESCRIPTION

A PBS batch job has attributes which control various aspects of the job. If an attribute is unset, the indicated default value is used. Unless otherwise stated, all attributes are readable by an unprivileged user.

User-alterable Attributes

The following attributes are alterable by users:

Account_Name

Used for accounting on some hosts. Format: string; default value: none.

block When true, specifies that qsub will wait for the job to complete, and return the exit value of the job. Default: false. Set via the -W option to qsub. If qsub receives one of the signals: SIGHUP, SIGINT, SIGQUIT or SIGTERM, it will print the following message on stderr:

```
qsub: wait for job <jobid> interrupted by signal <signal>
```

Checkpoint

If supported by the server implementation and the host operating system, the checkpoint attribute determines when checkpointing will be performed by PBS on behalf of the job. The legal values for checkpoint are described under the qalter and qsub commands. Format: the strings “n”, “s”, “c”, “c=mmmm”; default value: “u”, which is unspecified.

depend The type of inter-job dependencies specified by the job owner. Format: “type:jobid[,jobid...]”; default value: no dependencies.

Error_Path

The final path name for the file containing the job’s standard error stream. See the qsub and qalter command description for more detail. Format: “[hostname:]pathname”; default value: (job_name).e(job_number).

Execution_Time

The time after which the job may execute. The time is maintained in seconds since Epoch. If this time has not yet been reached, the job will not be scheduled for execution and the job is said to be in wait state. Format: “[[CCwYY]MMDDhhmm[.ss]”; default value: time 0, no delay.

group_list

A list of group_names@hosts which determines the group under which the job is run on a given host. When a job is to be placed into execution, the server will select a group name according to the following ordered set of rules:

1. Select the group name from the list for which the associated host name matches the name of the execution host.

2. Select the group name which has no associated host name, the wild card name.

3. Use the login group for the user name under which the job will be run.

Format: “group_name[@host][,group_name[@host]...]”.

Hold_Types

The set of holds currently applied to the job. If the set is not null, the job will not be scheduled for execution and is said to be in the hold state. Note, the hold state takes precedence over the wait state. Format: string made up of the letters ‘u’, ‘s’, ‘o’; default value: no hold.

Job_Name

The name assigned to the job by the qsub or qalter command. Format: string up to 15 characters, first character must be alphabetic; default value: the base name of the job script or STDIN.

Join_Path

If the Join_Paths attribute is TRUE , then the job’s standard error stream will be merged, inter-mixed, with the job’s standard output stream and placed in the file determined by the Output_Path attribute. The Error_Path attribute is maintained, but ignored. Format: boolean, values accepted are "True", "TRUE", "true", "Y", "y", "1", "False", "FALSE", "false", "N", "n", "0"; default value: false.

Keep_Files
: If Keep_Files contains the values "o" and/or "e" the corresponding streams of the batch job will be retained on the execution host upon job termination. Keep_Files overrides the Output_Path and Error_Path attributes. Format: "o", "e", "oe" or "eo"; default value: no keep, return files to submission host.

Mail_Points
: Identifies at which state changes the server will send mail about the job. Format: string made up of the letters 'a' for abort, 'b' for beginning, and
'e' for ending; default value: 'a', send on job abort.

Mail_Users
: The set of users to whom mail may be sent when the job makes certain state changes. Format: "user@host[,user@host]"; default value: job owner only.

no_stdio_sockets
: A flag to indicate whether a multi-node job should have the standard output and standard error streams of tasks running on remote (non "mother superior") nodes returned to mother superior via sockets. These sockets may cause a job to not be checkpointable. default value: false (which results in sockets being created).

Output_Path
: The final path name for the file containing the job's standard output stream. See the qsub and qalter command description for more detail. Format: see error_path, default value: (job_name).o(job_number).

Priority
: The job scheduling priority assigned by the user. Format: "[+|-]nnnnn"; range: [-1024, +1023] inclusive; default value:

undefined.

Rerunnable
The rerunnable flag given by the user. Format: “y” or “n”, see Join_Path; default value: y, job is rerunnable.

Resource_List
The list of resources required by the job. The resource list is a set of name=value strings. The meaning of name and value is server-dependent. The value also establishes the limit of usage of that resource. If not set, the value for a resource may be determined by a queue or server default established by the administrator. Default value: no usage or no limit depending on specific resource.

Shell_Path_List
A set of absolute paths of the program to process the job’s script file. The list is in the format: “path[@host][,path[@host]...]”. If this is null, then the user’s login shell on the host of execution will be used. Default value: null, login shell.

stagein
The list of files to be staged in prior to job execution. Format: local_path@remote_host:remote_path

stageout
The list of files to be staged out after job execution. Format: local_path@remote_host:remote_path

umask The initial umask of the job is set to the value of this attribute when the job is created. This may be changed by umask commands in the shell initialization files such as .profile or .cshrc. Default value: 077

User_List
The list of user@hosts which determines the user name under which the job is run on a given host. When a job is to be placed into execution, the server will select a user name from the list according to the following ordered set of rules:

1. Select the user name from the list for which the associated host name matches the name of the execution host.

2. Select the user name which has no associated host name, the wild card name.

3. Use the Job_Owner as the user name.

Default value: job owner name.

Variable_List
This is the list of environment variables passed with the Queue Job batch request. Format: “name=value[,name=value...]”.

Attributes Requiring Privilege to Set
The following attributes require system, manager, or operator privilege to set. They are visible to clients depending on privilege as noted.

comment
An attribute for displaying comments about the job from the system. Visible to any client. Format: any string; default value: none.

sched_hint
This attribute is present when the job is a member of a synchronous dependency set. It is set when the hold is released on the job. The value is SYNC_SCHED_HINT_FIRST (1) when the first job of the set is released for scheduling. This is a hint that may be used by the scheduler to decrease the priority of the job. This keeps a user from attempting to game the scheduler. The attribute is set to SYNC_SCHED_HINT_OTHER (2) for all other jobs in the set as they become schedulable. This should be taken as a hint by the scheduler to increase their priority to insure they will run at the same time as the earlier scheduled jobs in the set. [This attribute is viewable only by the batch administrator.] [type: integer]

Read-only Attributes
The following attributes are read-only. They are established by the server and are visible to the client but cannot be set by a client.

Certain ones are only visible to privileged clients (those run by the batch administrator).

accounting_id
Accounting ID for tracking accounting data not produced by PBS.

alt_id For a few systems, such as Irix 6.x running Array Services, the session id is insufficient to track which processes belong to the job. Where a different identifier is required, it is recorded in this attribute. If set, it will also be recorded in the end-of-job accounting record.

For Irix 6.x running Array Services, the alt_id attribute is set to the Array Session Handle (ASH) assigned to the job.

For jobs running in CPU sets, the alt_id will hold the set name in a form usable by the cpuset(1) command.

array Boolean. True if this is a job array.

array_id
Job array attribute. String. Applies to subjob. Subjob's job array identifier.

array_index
Job array attribute. String. Applies to subjob. Subjob's index number.

array_indices_remaining
Job array attribute. String. List of indices of subjobs still queued. Range or list of ranges, e.g. 500, 552, 596-1000.

array_indices_submitted
Job array attribute. String. Complete list of indices of sub-jobs given at submission time. Given as range, e.g. 1-100.

array_state_count
Job array attribute. String. Similar to state_count attribute for server and queue objects. Lists number of subjobs in each state.

ctime The time that the job was created.

egroup If the job is queued in an execution queue, this attribute is set to the group name under which the job is to be run. This attribute is readable only by the batch administrator.

etime The time that the job became eligible to run, i.e. in a queued state while residing in an execution queue.

euser If the job is queued in an execution queue, this attribute is set to the user name under which the job is to be run. This attribute is readable only by the batch administrator.

exec_host
If the job is running, this is set to the name of the host or hosts on which the job is executing. The format of the string is "host/N[*C][+...]", where "host" is the name of the host, "N" is task slot number, starting with 0, on that node, and "C" is the number of CPUs allocated to the job. "*C" does not appear if C has a value of one.

exec_vnode
If the job is running, this is set to the name of each node used by the job with the node-level, consumable resources allocated from that node. Each chunk's worth of nodes is enclosed in parentheses, and chunks are connected by plus signs. So for a job which requested two chunks that were satisfied by resources from three nodes, exec_vnode could look like

(vnodeA:ncpus=N:mem=X)+(nodeB:ncpus=P:mem=Y+nodeC:mem=Z).

hashname

The name used as a basename for various files, such as the job file, script file, and the standard output and error of the job. No longer used. This attribute is readable only by the batch administrator.

interactive

True if the job is an interactive PBS job. Format: boolean, see Join_Paths; default value: false.

Job_Owner

The login name on the submitting host of the user who submitted the batch job.

job_state

The state of the job.

E for exiting, the job has completed execution, with or without errors, and the batch system is doing post-execution clean-up.

H for Held, one or more holds have been applied to the job.

Q for Queued, the job resides in a execution or routing queue pending execution or routing. It is not in held or waiting state.

R for Running, the job resides in a execution queue and has been placed into execution.

S for Suspend, the job was executing and has been suspended. The job retains its assigned resources but does not use cpu cycle or walltime.

T for Transiting, the job is in process of being routed or moved to a new destination.

U for User suspend, the job was running on a workstation configured for cycle harvesting and the keyboard/mouse is currently busy. The job is suspended until the workstation has been idle for a configured amount of time.

W for Waiting, the job is not held but the Execution_Time attribute contains a time which has not yet been reached.

mtime The time that the job was last modified, changed state, or changed locations.

qtime The time that the job entered the current queue.

queue The name of the queue in which the job currently resides.

queue_rank
An number indicating the job's position with in the queue. Only used internally by PBS. This attribute is readable by the batch manager only.

queue_type
An identification of the the type of queue in which the job is currently residing. This is provided as an aid to the scheduler. This attribute is readable by the batch manager only. Format: The letter E or the letter R.

resources_used
The amount of resources used by the job. This is provided as part of job status information if the job is running.

run_count
The number of times the server has run the job. Can only be read by PBS Manager. Format: integer.

schedselect
This is set to the union of the "select" resource of the job and the queue and server defaults for resources in a chunk. Can only be read by PBS Manager.

server
The name of the server which is currently managing the job.

session_id
If the job is running, this is set to the session id of the first executing task.

stime The time when the job started execution.

substate
A numerical indicator of the substate of the job. The substate is used by the PBS job server internally. The attribute is visible to privileged clients, such as the scheduler. Can only be read by PBS Operator or Manager. Format: integer.

sw_index
Switch index for High Speed Switch on IBM SP systems. Visible to PBS Manager only. Format: string.

SEE ALSO
qsub(1B), qalter(1B), qhold(1B), qrls(1B), pbs_resources(7B)

NAME

pbs_node_attributes - attributes of PBS vnodes

DESCRIPTION

The attributes of PBS vnodes can either be altered by a privileged user, or are read-only. Some of the alterable attributes can be changed by an operator, some only by an administrator (manager).

The following attributes can be altered:

state Shows or sets the state of the vnode. Certain state values, marked with an * in the following list, may be set by the manager or operator, the other states are set internally. Format: string, one of the above states.

free * Node is up and capable of accepting additional job(s).

offline *
Node has been marked by operator or administrator as unusable.

down Node is not responding to queries from the Server.

job-busy
All CPUs on the vnode are allocated to jobs.

job-exclusive
The entire vnode has been exclusively allocated to one job at the job's request.

busy The vnode is reporting a load average greater than the configured high water value.

stale Server can still communicate with MOM, but MOM is not reporting any information.

state-unknown
The Server has never been able to contact the node.

Either pbs_mom is not running on the node, the node hardware is down, or there is a network problem.

comment
This attribute may be set by the administrator to any string to inform the users of any information relating to the node. If this attribute is not explicitly set, the PBS Server will use the attribute to pass information about the node status, specifically why the node is down. If the attribute is explicitly set by the administrator, it will not be modified by the Server.

Format: string.

lictype
Deprecated. No longer used.

max_running
The maximum number of jobs allowed to be run on this vnode at any given time. This attribute is advisory to the Scheduler, it is not enforced by the server. Format: integer.

max_user_run
The maximum number of jobs owned by a single user that are allowed to be run on this vnode at one time. This attribute is advisory to the Scheduler, it is not enforced by the server. Format: integer; default value: none.

max_group_run
The maximum number of jobs owned by any users in a single group that are allowed to be run on this vnode at one time. This attribute is advisory to the Scheduler, it is not enforced by the server. Format: integer; default value: none.

MOM Hostname of host on which MOM daemon will run. Can be explicitly set only via qmgr, and only at vnode creation. Defaults to value of vnode resource (vnode name.)

no_multinode_jobs
If set true, jobs which request more than one vnode are not

allowed to execute on this vnode. Format: boolean; default value: false

Port Port number on which MOM daemon will listen. Can be explicitly set only via qmgr, and only at vnode creation. Integer.

priority
The priority of this vnode compared with other vnodes.

queue The queue with which this vnode is associated. If set, only jobs in that queue may run on this vnode. If not set, any job in a queue without associated vnodes may run on this vnode. Requires full manager privilege to set or alter. Format: "queue name"; default value: none.

resources_available
The list of resource and amounts available on this vnode. If not explicitly set, the amount shown is that reported by the pbs_mom running on the vnode. Currently, only the ncpus number will be retained across Server restarts. Format: "resources_available.resource_name=value", see qmgr(1B).

resv_enable
This attribute is used to give the administrator ultimate say as to whether or not the vnode can be used to satisfy advance reservation requests, including the case where the administrator has configured the vnode to do cycle harvesting. If there is no intervention by the administrator, the vnode is available for advance reservation considerations, except for the special case where the administrator has configured the vnode for cycle harvesting. As is the case for the functioning of resv_enable on the server, any reservations that are already assigned to use this vnode will not be automatically removed if this attribute is subsequently set to false. Requires full manager privilege to set or alter. Format: True/False; default value: True (exception: default value is False if the vnode is marked for cycle harvesting.)
See also, resv_enable on the pbs_server_attributes manpage.

sharing
Defines whether more than one job at a time can use this vnode's

resources. Either a) the vnode is allocated exclusively to one job, or b) the vnode's unused resources are available to other jobs.
Allowable values: default_share | default_excl | force_shared | force_excl
This attribute can be set via the vnode definition entries in MOM's config file.
Example: vnodename: sharing=force_excl
Default value: default_share.

A vnode's behavior is determined by a combination of its sharing attribute and a job's placement directive. The behavior is defined as follows:

	Placement Request (-l place=)		
	Not Set	place=share	place=excl
sharing not set	share	share	excl
sharing=default_share	share	share	excl
sharing=default_excl	excl	share	excl
sharing=force_share	share	share	share
sharing=force_excl	excl	excl	excl

The following attributes are read-only:

jobs List of jobs running on the vnode. This attribute is read-only. Format: "#/jobid,...", where # represents the number of the processor.

license
Deprecated. Indicates whether this vnode is being used for a job. The possible values are

f At least one job is running on this vnode.

u There are no jobs running on this vnode.

ntype This attribute defines the type of the vnode. Format: string, “PBS”, “globus”; default value: PBS. Currently there are two types of vnodes supported:

PBS PBS vnodes are the default type and are assumed to run multiple jobs. The placement of the jobs among them is controlled by the site policy defined in the Job Scheduler.

globus A special pbs_mom is running to hand off jobs to the Globus distributed system. There can be only one globus Mom defined.

pcpus The number of physical CPUs on the vnode.

reservations
List of advance reservations pending on the vnode. This attribute is read-only. Format: “#/reservation id,...”

resources_assigned
The total amount of certain types of resources allocated to jobs running on this vnode. This attribute is read-only.

SEE ALSO
The PBS Professional Administrator’s Guide, pbs_resources(7B), qmgr(1B)

NAME

pbs_queue_attributes - pbs queue attributes

DESCRIPTION

Queue attributes are either set by the server, in which case they are read-only, or can be set by operator or administrator. Queues are either routing queues or execution queues.

The following attributes can be set by operator or administrator, and apply to both routing and execution queues:

acl_group_enable

When true directs the server to use the queue group access control list acl_groups. Format: boolean, “TRUE”, “True”, “true”, “Y”, “y”, “1”, “FALSE”, “False”, “false”, “N”, “n”, “0”; default value: false = disabled.

acl_groups

List of groups which are allowed or denied access to this queue. The groups in the list are groups on the server host, not submitting hosts. See section 10.1, Authorization, in the PBS External Reference Specification. Format: “[+|-]group_name[,...]”; default value: all groups allowed.

acl_host_enable

When true directs the server to use the acl_hosts access list. Format: boolean (see acl_group_enable); default value: disabled.

acl_hosts

List of hosts from which jobs may be submitted to this queue. See section 10.1, Authorization, in the PBS External Reference Specification. Format: “[+|-]hostname[...]”; default value: all hosts allowed.

acl_user_enable

Attribute which when true directs the server to use the the acl_users List of users allowed or denied access to this queue. Format: boolean (see acl_group_enable); default value: disabled.

acl_users

List of users allowed or denied access to this queue. See section 10.1, Authorization, in the PBS External Reference Specification. Format: “[+|-]user[@host][,...]”; default value: all users allowed.

enabled

Determines whether queue will accept new jobs. When false the queue is disabled and will not accept jobs. Format: boolean (see acl_group_enable); default value: disabled.

from_route_only

When true, this queue will only accept jobs from a routing queue. Requires manager privilege to set or alter. Format: boolean; default value: disabled.

max_array_size

The maximum number of subjobs (separate indices) that are allowed in an array job. Format: integer; default value: none, no limit.

max_queuable

The maximum number of jobs allowed to reside in the queue at any given time. Format: integer; default value: infinite.

max_running

The maximum number of jobs allowed to be selected from this queue for routing or execution at any given time. For a routing queue, this is enforced, if set, by the server. Format: integer.

node_group_key

Specifies the resource to use for node grouping. Overrides server’s node_group_key. Format: string. Default value: disabled. Example:

```
Qmgr> set queue QUEUE node_group_key=RESOURCE
```

Priority

The priority of this queue against other queues of the same type on this server. Format: integer.

queue_type
The type of the queue: execution or route. Requires manager privilege to set or alter. Format: “execution”, “e”, “route”, “r”. This attribute must be explicitly set.

require_cred
Specifies the credential type required. All jobs submitted to the named queue without the specified credential will be rejected. Requires manager privilege to set or alter. Not supported under Windows. Format: string: krb5 or dce. Default value: unset

require_cred_enable
Directs the Server to use the credential authentication method specified by require_cred for this queue. Requires manager privilege to set or alter. Not supported under Windows. Format: boolean Default: false = disabled

resources_max
The maximum amount of each resource which can be requested by a single job in this queue. The queue value supersedes any server wide maximum limit. Format: “resources_max.resource_name=value”, see qmgr(1B); default value: infinite usage.

resources_min
The minimum amount of each resource which can be requested by a single job in this queue. Format: see resources_max, default value: zero usage.

resources_default
The list of default resource values which are set as limits for a job residing in this queue and for which the job did not specify a limit. Format: “resources_default.resource_name=value”, see qmgr(1B); default value: none; if not set, the default limit for a job is determined by the first of the following attributes which is set: server’s resources_default, queue’s resources_max, server’s resources_max. If none of these are set, the job

will get unlimited resource usage.

started
Jobs may be scheduled for execution from this queue. When false, the queue is considered stopped. Advisory to the Scheduler, not enforced by the server. [default value: false, but depends on scheduler interpretation] Format: boolean (see acl_group_enable).

The following attributes apply only to execution queues:

checkpoint_min S
Specifies the minimum interval of cpu time, in minutes, which is allowed between checkpoints of a job. If a user specifies a time less than this value, this value is used instead. Format: integer; default value: no minimum.

default_chunk
The list of resources which will be inserted into each chunk of a job's select specification if the corresponding resource is not specified by the user. This provides a means for a site to be sure a given resource is properly accounted for even if not specified by the user.

resources_available
The list of resources and amounts available to jobs running in this queue. The sum of the resource of each type used by all jobs running from this queue cannot exceed the total amount listed here. Format: "resources_available.resource_name=value", see qmgr(1B).

kill_delay
The amount of the time delay between the sending of SIGTERM and SIGKILL when a qdel command is issued against a running job. Format: integer seconds; default value: 2 seconds.

max_user_res
The maximum amount of the specified resource that any single user may consume. The named resource can be any valid PBS resource, e.g. "ncpus", "mem", "pmem", etc. Default value: none. Format: max_user_res.resource_name=value Example: set

server max_user_res.ncpus=6

max_user_res_soft

The soft limit on the amount of the specified resource that any single user may consume. The named resource can be any valid PBS resource, e.g. “ncpus”, “mem”, “pmem”, etc. If a user is consuming more than this amount of the specified resource, their jobs are eligible to be preempted by jobs from users who are not over their soft limit. Default value: none. Format: max_user_res_soft.resource_name=value Example: set server max_user_res_soft.ncpus=3

max_user_run

The maximum number of jobs owned by a single user that are allowed to be running from this queue at one time. Format: integer; default value: none.

max_user_run_soft

The soft limit on the number of jobs owned by a single user that are allowed to be running from this queue at one time. If a user has more than this number of jobs running, their jobs are eligible to be preempted by jobs from users who are not over their soft limit. Format: integer; default value: none.

max_group_res

The maximum amount of the specified resource that any single group may consume in a complex. The named resource can be any valid PBS resource, e.g. “ncpus”, “mem”, “pmem”, etc. Default value: none. Format: max_group_res.resource_name=value Example: set server max_group_res.ncpus=6

max_group_res_soft

The soft limit on the amount of the specified resource that any single group may consume in a complex. The named resource can be any valid PBS resource, e.g. “ncpus”, “mem”, “pmem”, etc. If a group is consuming more than this amount of the specified resource, their jobs are eligible to be preempted by jobs from groups who are not over their soft limit. Default value: none. Format: max_group_res_soft.resource_name=value

Example: set server max_group_res_soft.ncpus=3

max_group_run
The maximum number of jobs owned by a group that are allowed to be running from this queue at one time. Format: integer; default value: none.

max_group_run_soft
The maximum number of jobs owned by users in a single group that are allowed to be running from this queue at one time. If a group has more than this number of jobs running, their jobs are eligible to be preempted by jobs from groups who are not over their soft limit. Format: integer; default value: none.

The following attributes apply only to routing queues:

route_destinations
The list of destinations to which jobs may be routed. Requires manager privilege to set or alter. Format: comma separated strings of the form “queue_name[@server_host[:port]]”. [default value: none, should be set to at least one valid destination] Example: “here,there@remote,test@remote:15501”

alt_router
If true, a site-supplied alternative job routing function is used to determine the destination for routing jobs from this queue. Otherwise the default round-robin router is used. Requires manager privilege to set or alter. Format: boolean (see acl_group_enable); default value: false.

route_held_jobs
If true, jobs with a hold may be routed from this queue. If false, held jobs are not routed. Format: boolean (see acl_group_enable); default value: false.

route_waiting_jobs
If true, jobs with a future execution_time attribute may be routed from this queue. If false, they are not to be routed. Format: boolean (see acl_group_enable); default value: false.

route_retry_time

Time delay between route retries. Typically used when the network between servers is down. Format: integer seconds; default value: PBS_NET_RETRY_TIME (30 seconds).

route_lifetime

The maximum time a job is allowed to exist in a routing queue. If the job cannot be routed in this amount of time, the job is aborted. If unset or set to a value of zero (0), the lifetime is infinite. Format: integer seconds; default infinite.

The following attributes are set by the server and are read-only.

These read-only attributes apply to both execution and routing queues:

total_jobs

The number of jobs currently residing in the queue.

state_count

The total number of jobs currently residing in the queue in each state.

These read-only attributes apply to execution queues:

resources_assigned

The total amount of certain types of resources allocated to jobs running from this queue.

hasnodes

This attribute is set true if there are nodes associated with this queue.

SEE ALSO

the PBS ERS, qmgr(1)

NAME

pbs_resources - computational resources for PBS jobs

DESCRIPTION

PBS provides computational resources for jobs, limits on using resources, and control over how jobs are placed on the vnodes from which resources may be allocated for a job.

PBS provides built-in resources, and allows the administrator to define custom resources. The administrator can specify which resources are available on a given vnode, as well as at the queue or server level (e.g. floating licenses.) Resources can be "stretched" across vnodes. See the qmgr(8B) man page and The PBS Professional Administrator's Guide.

Resources defined at the queue or server level apply to an entire job. If they are defined at the host level, they apply only to the part of the job running on that host.

Jobs request resources, which are allocated to the job, along with any defaults specified by the administrator.

For information on defining resources, see The PBS Professional Administrator's Guide.

Summary

Resources are allocated to jobs both by explicitly requesting them and by applying defaults. Resources are explicitly requested (in order of precedence) through a qalter operation, the qsub command line, and PBS job script directives. Default resources can be specified by the administrator (in order of precedence) for qsub arguments, queues, the server, and vnodes.

Jobs are assigned limits on the amount of resources they can use. These limits apply to how much the job can use on each vnode and to how much the whole job can use. Limits are derived from both requested resources and applied default resources.

Jobs are placed on vnodes according to their explicit placement request, or according to default placement rules. The explicit placement request can be specified (in order of precedence) using qalter, qsub, and PBS job script directives. Default placement rules can be specified for queues and the server, and rules for default placement take effect if no other placement specifications exist.

A job submitted with the old node or resource specification syntax will be converted to the new select and place syntax. If the job is submitted with -lnodes= or -lncpus= it will be converted to -l select= and -l place=. See BACKWARD COMPATIBILITY. Jobs cannot use both new and old syntax for resource requests.

Allocation
Resources are allocated to jobs both by explicitly requesting them and by applying specified defaults. Jobs explicitly request resources either at the host level in chunks defined in a selection statement, or in job-wide resource requests. The only resources that can be requested in chunks are host-level resources, such as mem and ncpus. The only resources that can be in a job-wide request are server-level or queue-level resources, such as walltime. An explicit resource request can appear here, with this order of precedence:

qalter
qsub
PBS job script directives

Requesting Resources in Chunks
A chunk declares the value of each resource in a set of resources which are to be allocated as a unit to a job. All of a chunk must be taken from a single vnode. A chunk request is a host-level request, and it must be for a host-level resource. A chunk is the smallest set of resources that will be allocated to a job. It is one or more resource_name=value statements separated by a colon, e.g.:

```
ncpus=2:mem=10GB:host=Host1
ncpus=1:mem=20GB:arch=linux
```

Chunks are described in a selection statement, which specifies how many of each kind of chunk. A selection statement is of the form:

```
-l select=[N:]chunk[+[N:]chunk ...]
```

If N is not specified, it is taken to be 1.

Example of multiple chunks in a selection statement:

```
-l select=2:ncpus=1:mem=10GB+3:ncpus=2:mem=8GB:arch=solaris
```

Requesting Job-wide Resources
A job-wide resource request is for resource(s) at the server or queue level. This resource must be a server-level or queue-level resource. Job-wide resources are requested outside of a selection statement, in this form:

```
-l keyword=value[,keyword=value ...]
```

where keyword identifies either a consumable resource or a time-based resource such as walltime.

Job-wide resources are used for requesting floating licenses or other resources not tied to specific hosts, such as cput and walltime.

Do not mix old style resource or node specification with the new select and place statements. Do not use one in a job script and the other on the command line. This will result in an error.

Applying Resource Defaults
Jobs get default resources, both job-wide and per- chunk with the following order of precedence, from

Default qsub arguments
Default queue resources
Default server resources

For each chunk in the job’s selection statement, first queue chunk defaults are applied, then server chunk defaults are applied. If the chunk does not contain a resource defined in the defaults, the default is added. The chunk defaults are called “default_chunk.RESOURCE”.

For example, if the queue in which the job is enqueued has the following defaults defined:

```
default_chunk.ncpus=1
default_chunk.mem=2gb
```

a job submitted with this selection statement:

```
select=2:ncpus=4+1:mem=9gb
```

will have this specification after the default_chunk elements are applied:

```
select=2:ncpus=4:mem=2gb+1:ncpus=1:mem=9gb.
```

In the above, mem=2gb and ncpus=1 are inherited from default_chunk.

The job-wide resource request is checked against queue resource defaults, then against server resource defaults. If a default resource is defined which is not specified in the resource request, it is added to the resource request.

Default Resources on Server or Queue

The administrator can specify default resources on the server and queue. These resources can be job-wide or apply to chunks. Specifying a job-wide resource has the same effect as adding -l RESOURCE to the job’s resource request. Specifying a chunk resource is the same as adding :RESOURCE=VALUE to the job’s chunks (for chunks that don’t already specify that resource.) Job-wide resources are specified via resources_default on the server or queue, and chunk resources are specified via default_chunk on the server or queue.

The administrator can also specify default resources to be added to any qsub arguments, as well as default placement of jobs.

See the qmgr(8B) man page for how to set default resources.

How Default Resources Work When Moving Jobs Between Queues
If the job is moved from the current queue to a new queue, any default resources in the job's resource list are removed. This includes a select specification and place directive generated by the rules for conversion from the old syntax. If a job's resource is unset (undefined) and there exists a default value at the new queue or server, that default value is applied to the job's resource list. If either select or place is missing from the job's new resource list, it will be automatically generated, using any newly inherited default values.

Example:

Given the following set of queue and server default values:

```
Server
    resources_default.ncpus=1

Queue QA
    resources_default.ncpus=2
    default_chunk.mem=2gb

Queue QB
    default_chunk.mem=1gb
    no default for ncpus
```

The following illustrate the equivalent select specification for jobs submitted into queue QA and then moved to (or submitted directly to) queue QB:

```
qsub -l ncpus=1 -lmem=4gb
    In QA: select=1:ncpus=1:mem=4gb - no defaults need
    be applied
    In QB: select=1:ncpus=1:mem=4gb - no defaults need
    be applied
```

```
qsub -l ncpus=1
    In QA: select=1:ncpus=1:mem=2gb
    In QB: select=1:ncpus=1:mem=1gb


qsub -lmem=4gb
    In QA: select=1:ncpus=2:mem=4gb
    In QB: select=1:ncpus=1:mem=4gb


qsub -l nodes=4
    In QA: select=4:ncpus=1:mem=2gb
    In QB: select=4:mem=1gb

qsub -l mem=16gb -l nodes=4
    In QA: select=4:ncpus=1:mem=4gb
    In QB: select=4:ncpus=1:mem=4gb
```

Limits on Resource Usage
Each chunk's per-chunk limits determine how much of any resource can be used in that chunk. Per-chunk resource usage limits are established by per-chunk resources, both from explicit requests and from defaults.

Job resource limits set a limit for per-job resource usage. Job resource limits are established both by requesting job-wide resources and when per-chunk consumable resources are summed. Job resource limits from sums of all chunks, including defaults, override those from job-wide defaults and resource requests. Limits include both explicitly requested resources and default resources.

If a job's job resource limit exceeds queue or server restrictions, it will not be put in the queue or accepted by the server. If, while running, a job exceeds its limit for a consumable or time-based resource, it will be terminated. See The PBS Professional Administrator's Guide.

Controlling Placement of Jobs
Jobs are placed on vnodes according to their place statements. The place statement can be specified, in order of precedence, via:

Explicit placement request in qalter
Explicit placement request in qsub
Explicit placement request in PBS job script directives
Default qsub place statement
Queue default placement rules
Server default placement rules
Built-in default conversion and placement rules

The place statement may be not be used without the select statement.

The place statement has this form:

-l place=[arrangement][: sharing][: grouping]

where

arrangement is one of free | pack | scatter
sharing is one of excl | shared
grouping can have only one instance of group=resource

and where

free: Place job on any vnode(s).
pack: All chunks will be taken from one host.
scatter: Only one chunk with any MPI processes will be taken from a host. A chunk with no MPI processes may be taken from the same vnode as another chunk.
excl: Only this job uses the vnodes chosen.
shared: This job can share the vnodes chosen.
group=resource: Chunks will be grouped according to a resource. All vnodes in the group must have a common value for the resource, which can be either the built-in resource host or a site-defined host-level resource.

Note that vnodes can have sharing attributes that override job placement requests. See the pbs_node_attributes(7B) man page.

Default Placement
If, after all defaults have been applied to a resource request that contains a selection statement, there is no place statement, then arrangement is set to free. Default sharing is shared.

If the job’s place statement does not contain group=resource, then a grouping defined at the queue level may be used, or a grouping defined at the server level if there is none at the queue level.

Placement of Jobs Submitted with Old Syntax
A job submitted with a node (-lnodes=) or resource (-lncpus=) specification will be converted to select and place, according to the rules described below in BACKWARD COMPATIBILITY.

Boolean Resources
A boolean resource can be either true or false. A resource request can specify the value a boolean resource should have. For example, if some vnodes have green=true and some have red=true, a selection statement for two vnodes, each with one CPU, all green and no red, would be:

```
-l select=2:green=true:red=false:ncpus=1
```

Consumable Resources
Consumable resources are those whose use by a job reduces the amount available to other concurrent jobs, e.g. memory (mem), CPUs (ncpus) and licenses. Non-consumable resources include time-based resources such as walltime and CPU time (cput), and string-value resources such as architecture (arch).

Custom Resources
Custom resources are defined in PBS_HOME/server_priv/resourcedef. Custom resources are site-defined and site-dependent. Typically used for licenses and scratch space. See The PBS Professional Administrator’s Guide.

A job requesting a floating license must specify it outside of a selection statement, as a job-wide resource limit. A job requesting a node-locked license must specify it inside a selection statement in a chunk. See your system administrator. Refer to The PBS Professional User’s Guide.

Matching Jobs to Resources
For all resources except boolean resources, if a resource is unset (not defined) at a server, queue or vnode, a resource request will behave as

if that resource has zero value. The undefined resource at the server or queue will cause the job to be rejected by the server or queue, and the undefined resource at the vnode will prevent the job from running on that vnode.

For boolean resources, if a resource is unset (undefined) at a server, queue, or vnode, the resource request will behave as if that resource is set to “false”. It will match a resource request for that boolean with a value of “false”, but not “true”.

BUILT-IN RESOURCES

arch System architecture. For use inside chunks only. One architecture can be defined for a vnode. One architecture can be requested per vnode. Allowable values and effect on job placement are site-dependent. Type: string.

cput Amount of CPU time used by the job for all processes on all vnodes. Establishes a job resource limit. Non-consumable. Type: time.

file Size of any single file that may be created by the job. Type: size.

host Name of execution host. For use inside chunks only. Automatically set to the short form of the hostname in the Mom attribute. Cannot be changed. Site-dependent. Type: string.

mem Amount of physical memory i.e. workingset allocated to the job, either job-wide or host-level. Consumable. Type: size.

mpiprocs Number of MPI processes for this chunk. Defaults to 1 if ncpus > 0, 0 otherwise. For use inside chunks only. Type: integer.

The number of lines in PBS_NODEFILE is the sum of the values of mpiprocs for all chunks requested by the job. For each chunk with mpiprocs=P, the host name for that chunk is written to the PBS_NODEFILE P times.

mppwidth Number of processing elements (PEs). Type: integer mppdepth Depth of each processor (number of threads). Default is 1. Specifies the number of processors each processing element will use. Type: integer

mppnppn Number of processing elements (PEs) per node. Type: integer

mppnodes Manual placement list consisting of a comma-separated list of nodes (node,node), a range of nodes (node1-node2...), and a combination of both formats. Node values are expressed in decimal. The first number in a range must be less than the second number (i.e., 8-6 is invalid). A complete node list is required. Type: integer

mpplabels Places the application only nodes with the specified MPP label: soft or hard.

mppmem The per processing element maximum Resident Set Size memory limit in megabytes. K|M|G suffixes are supported (16 = 16M = 16 megabytes). Any truncated or full spelling of unlimited is recognized.

mpphost MPP host.

mpparch MPP compute node system type.

ncpus Number of processors requested. Cannot be shared across vnodes. Consumable. Type: integer.

nice Nice value under which the job is to be run. Host-dependent. Type: integer.

nodect Read-only. Number of chunks in resource request from selection directive, or number of nodes requested from node specification. Otherwise defaults to value of 1. Type: integer.

ompthreads
Number of OpenMP threads for this chunk. Defaults to ncpus if not specified. For use inside chunks only. Type: integer.

For the MPI process with rank 0, the environment variables NCPUS and OMP_NUM_THREADS are set to the value of ompthreads. For other MPI processes, behavior is dependent on MPI implementation. See The PBS Professional Administrator’s Guide.

pcput Amount of CPU time allocated to any single process in the job. Establishes a job resource limit. Non-consumable. Type: time.

pmem Amount of physical memory (workingset) for use by any single process of the job. Establishes a job resource limit. Consumable. Type: size

pvmem Amount of virtual memory for use by any single process in the job. Establishes a job resource limit. Consumable. Type: size.

software Site-specific software specification. For use only in job-wide resource requests. Allowable values and effect on job placement are site-dependent. Type: string.

vmem Amount of virtual memory for use by all concurrent processes in the job. Establishes a job resource limit. Not consumable. Type: size.

vnode Name of virtual node (vnode) on which to execute. For use inside chunks only. Site-dependent. Type: string. See the pbs_node_attributes(7B) man page.

walltime Amount of wall-clock time during which the job can run. Establishes a job resource limit. Non-consumable. Type: time.

RESOURCE TYPES

boolean Boolean-valued resource. For use inside chunks only. Non-consumable. Allowable values (case insensitive): True|T|Y|1|False|F|N|0
Example: To select a vnode with red but not blue,
```
-l select=1:red=true:blue=false
```

float Float. Allowable values: [+-] 0-9 [[0-9] ...][.][[0-9] ...]

long Long integer. Allowable values: 0-9 [[0-9] ...]

size Number of bytes or words. Expressed in the form:
integer[suffix] where suffix can be

b or w	bytes or words.
kb or kw	Kilo (2 to the 10th, or 1024) bytes or words.
mb or mw	Mega (2 to the 20th, or 1,048,576) bytes or words.
gb or gw	Giga (2 to the 30th, or 1,073,741,824) bytes or words.
tb or tw	Tera (2 to the 40th, or 1024 gigabytes) bytes or words.
pb or pw	Peta (2 to the 50th, or 1,048,576 gigabytes) bytes or words.

The size of a word is the word size on the execution host.

string String. Non-consumable.
Allowable values: [_a-zA-Z0-9][[-_a-zA-Z0-9[]#.] ...]
(Leading underscore ("_"), alphabetic or numeric, followed by dash ("-"), underscore ("_"), alphabetic, numeric, left bracket ("[“), right bracket ("]"), hash ("#") or period (".”))

string_array
String-valued resource which can contain multiple values. Comma-separated list of strings. Non-consumable. Resource request will succeed if request matches one of the values. Resource request can contain only one string.

```
time      The  maximum time period the resource can be used.  Expressed
          in seconds as an integer, or in the form:
              [[hours:]minutes:]seconds[.milliseconds]
```

BACKWARD COMPATIBILITY

Conversion to Select and Place

For backward compatibility, a legal node specification or resource specification will be converted into selection and placement directives.

Node Specification Conversion

Node specification format:

```
    -lnodes=[N:spec_list | spec_list]
           [[+N:spec_list | +spec_list] ...]
           [#suffix ...][-lncpus=Z]
```

where:

```
    spec_list has syntax: spec[:spec ...]
    spec is any of: hostname | property | ncpus=X | cpp=X | ppn=P
    suffix is any of: property | excl | shared
    N and P are positive integers
    X and Z are non-negative integers
```

The node specification is converted into selection and placement directives as follows:

Each spec_list is converted into one chunk, so that N:spec_list is converted into N chunks.

If spec is hostname :
The chunk will include host=hostname

If spec matches any vnode’s resources_available.host value:
The chunk will include host=hostname

If spec is property :
The chunk will include property=true

Property must be a site-defined host-level boolean resource.

If spec is ncpus=X or cpp=X :
The chunk will include ncpus=X

If no spec is ncpus=X and no spec is cpp=X :
The chunk will include ncpus=1

If spec is ppn=P :
The chunk will include mpiprocs=P
Example:

```
   -lnodes=4:ppn=2
```

is converted into

```
   -lselect=4:ncpus=2:mpiprocs=2
```

If -lncpus=Z is specified and no spec contains ncpus=X and no spec is ccp=X :
Every chunk will include ncpus=W,
where W is Z divided by the total number of chunks.
(Note: W must be an integer; Z must be evenly divisible by the number of chunks.)

If property is a suffix :
All chunks will include property=true

If excl is a suffix :
The placement directive will be -lplace=scatter:excl

If shared is a suffix :
The placement directive will be -lplace=scatter:shared

If neither excl nor shared is a suffix :
The placement directive will be -lplace=scatter

Example:

```
-l nodes=3:green:ncpus=2:ppn=2+2:red
```

is converted to:

```
-l select=3:green=true:ncpus=4:mpiprocs=2+2:red=true:ncpus=1
-l place=scatter
```

Node specification syntax for requesting properties is deprecated. The new boolean resource syntax “property=true” is only accepted in a selection directive. It is erroneous to mix old and new syntax.

Resource Specification Conversion
The resource specification is converted to select and place statements after any defaults have been applied.

Resource specification format:

```
-lresource=value[:resource=value ...]
```

The resource specification is converted to:

```
select=1[:resource=value ...]
place=pack
```

with one instance of resource=value for each of the following host-level resources in the resource request:

built-in resources: ncpus | mem | vmem | arch | host

site-defined host-level resources listed in the Server’s resourcedef file with flags including “h”

SEE ALSO

The PBS Professional Administrator’s Guide, The PBS Professional User’s Guide, pbs_node_attributes(7B), pbs_rsub(1B), qalter(1B), qmgr(8B), qstat(1B), qsub(1B)

NAME

pbs_resv_attributes - attributes of PBS advance reservations

DESCRIPTION

Listed below are the attributes which can be set by all users. Some are assigned default values when not explicitly set.

Account_Name

Used for accounting on some hosts.
Format: string.
Default value: none.

group_list

A list of group_name@hosts. Used to determine the group name to assign to a resource reservation at a given server.
Format: “group_name[@host][,group_name[@host]...]”.
When a resource reservation request is submitted to a server, the server selects a group name from this list according to an ordered set of rules:

1. Select from the list that group name for which the associated host name matches the name of the submitting host.

2. Select from the list that group name which has no associated host name, i.e. the wild card name.

3. Use the login group for the name in the euser attribute. The name in the euser attribute is itself determined by the server from an ordered set of rules.

Reserve_Name

The name assigned to the resources reservation during submission via the pbs_rsub or the qsub (in the case of a reservation job) command.
Format: string up to 15 characters, where the first character is alphabetic.

Default value: none.

Mail_Points
Determines whether and where mail is sent on various events. Identifies which of the reservation state transition events cause mail notification to be sent by the server to various receiving parties.
Format: string consisting of one of more letters “n”, “a”, “b”, “c”, “e”. The string “n” means do not send mail, “c” means notify when scheduler confirms, “b” notify when reservation period begins, “e” notify when reservation period ends, “a” notify when reservation is terminated.
Default value: “ac”, send mail on confirmation and on abort/delete events only.

Mail_Users
The set of users to whom mail may be sent when the reservation experiences various changes in state.
Format: string of the form, user@host[,user@host].... .
Default value: reservation owner only.

Priority
The reservation scheduling priority assigned by the user. This attribute is yet to be implemented.
Format: string of the form, [+|-]nnnnn .
Default value: none.

Resource_List
The list of resources requested by the reservation. This list is a set of name=value strings. The actual meaning of name and value is server dependent. The values specified for the various requested resources serve as limits on the aggregate of those jobs that are, at any moment, running under the reservation.
Default value: none

User_List
List of user@hosts. Not used. Default value: Reserve_Owner name.

Variable_List
A list of PBS environment variables passed with the SubmitResv

batch request.
Format: string of the form, name=value[,name=value]... .

The following reservation attributes are read-only:

reserve_type
Indicates the type of resources reservation. A value of (2) means this is a general reservation, (3) means this is a reservation job.

reserve_ID
The resources reservation's identifier. This is a string in one of two formats depending on the type of resources reservation. For a reservation job the the format of this id string is the same as that for a job, namely, [sequence_number.server_name@server]. In the case of a general reservation of resources the format is [Rsequence_number.server_name@server]. The pieces of this identifier are as follows: sequence_number is a positive sequential integer assigned by the server to which first created the reservation, server_name is the name of the host where the reservation was created, @server is the name of the server host on which the reservation currently resides.

reserve_start
The time at which resources requested by the reservation can start to be used.

reserve_end
The time at which resources requested by the reservation can no longer be consumed. Also, the time when the resources reservation (reservation job) will begin to be automatically removed from the system.

reserve_duration
A contiguous amount of time. During this period of time resources requested in the reservation (reservation job) are to be available to satisfy jobs submitted to run in that reservation.

resv_nodes
The value of this attribute is the collection of nodes that will be selected by the PBS system to solve the nodes specification that is part of the resources reservation (reservation job) request.
Format: a contiguous '+'-separated string of specific nodes specifications.

Authorized_Users
The list of those users that are given (denied) the right to submit jobs to the queue instantiated to service the confirmed resources reservation request. This list is used as the user_acl (access control list) for the queue. The list is not changeable by the owner of the reservation.
Format: [+|-]user[hostname.domain],....,[+|-]... where, '-' means "deny" and
Default value: only the reservation owner is allowed to submit to the queue.

Authorized_Groups
List of groups which allows (denies) the right of users belonging to those groups to enqueue jobs in the queue associated with the reservation. This list is used as the group_acl (access control list) for the instantiated queue.
Format: [+|-]group_name,...,[+|-]group_name] (see Authorized_Users).
Default value: owner's login group.

ctime The time that the resources reservation object was created.

egroup Attribute egroup is set to a name that gets determined by the server to which the resources reservation is sent. Only the batch administrator has availability to this attribute (see group_list.)

euser Attribute euser is set to a user name that gets determined by the server to which the resources reservation is sent. Only the batch administrator has availability to this attribute.

hashname

The name used as a basename for the resources reservation file for this reservation. This file is part of the server's database and used in recovery of the resources reservation whenever the PBS server is coming back up. This attribute is available only to the batch administrator.

Interactive

This attribute is set to a non-zero value if the client that submitted the resources reservation request is willing to wait for the scheduler to confirm the reservation. The value of this attribute is the number of seconds that the client is willing to wait for confirmation. A positive number is interpreted as being the number of seconds that the client is willing to wait to get back confirmation of the request, after which confirmation of the reservation will be done by querying via pbs_rstat If the value of this attribute is negative, it is interpreted to mean that the client is willing to wait that many seconds (positive) for confirmation by the scheduler and, if confirmation has not occurred in that period of time the submitted resources reservation is to be automatically deleted from the system.

Reserve_Owner

The login name on the submitting host of the user who submitted the resources reservation request.

reserve_state

The state of the resources reservation. The abbreviations and states are:

NO RESV_NONE

No reservation yet.

UN RESV_UNCONFIRMED

Resources reservation (reservation job) request is awaiting scheduler confirmation.

CO RESV_CONFIRMED

Scheduler has confirmed the resources reservation (reservation job) request. For a general resources reservation, a reservation queue has gotten instantiated, jobs and reservation jobs can now be enqueued into this queue and later, during the reservation period, be selected by the scheduler to run.

WT RESV_WAIT
Unused. (Confirmed reservation job is waiting arrival of the reservation window.)

TR RESV_TIME_TO_RUN
Scheduler signaled by server upon the arrival of the reservation period

RN RESV_RUNNING
Resources reservation period has started

FN RESV_FINISHED
Resources reservation period is now finished

BD RESV_BEING_DELETED
Process of deleting the resources reservation (reservation job) commencing.

DE RESV_DELETED
Resources reservation is now being deleted, any jobs (reservation jobs) that belonged to the reservation are already deleted.

DJ RESV_DELETING_JOBS
Jobs belonging to the resources reservation are being deleted prior to deleting the resources reservation itself.

Queue Name of the reservation queue. Jobs that are to use resources belonging to this reservation need to be submitted to this queue.

substate
A numerical indicator of the substate of the resources reservation. The substate is used internally by the PBS server in managing the reservation. The attribute is visible to privileged clients, such as the scheduler.

Format: integer. The values of substate are defined in the header file reservation.h

SEE ALSO
The PBS Professional User’s Guide, the PBS Professional Administrator’s Guide,
pbs_rdel(1B), pbs_rstat(1B), pbs_rsub(1B), qsub(1B), qalter(1B), pbs_resources(7B), pbs_queue_attributes(7B)

NAME

pbs_sched_attributes - pbs scheduler attributes

DESCRIPTION

Scheduler attributes can be read only by the PBS Manager or Operator. All scheduler attributes are read-only.

Read Only Scheduler Attributes

The following attributes are read-only.

pbs_version

The version of PBS for this scheduler. Available only to Manager/Operator.

sched_host

The hostname of the machine on which the scheduler runs. Available only to Manager/Operator.

SEE ALSO

The PBS Professional Administrator’s Guide, The PBS Professional User’s Guide, qmgr(1B)

Local 17 April 2007 pbs_sched_attributes(7B)

NAME

pbs_server_attributes - pbs server attributes

DESCRIPTION

Server attributes can be read by any client; privilege is not required. Most server attributes are alterable by a privileged client, run by a user with administrator or operator privilege. Certain attributes require the user to have full administrator privilege. The following is a list of the server attributes.

acl_host_enable

Attribute which when true directs the server to use the acl_hosts access control lists. Requires full manager privilege to set or alter. Format: boolean, “TRUE”, “True”, “true”, “Y”, “y”, “1”, “FALSE”, “False”, “false”, “N”, “n”, “0”; default value: false = disabled.

acl_hosts

List of hosts which may request services from this server. This list contains the network name of the hosts. Local requests, i.e. from the server’s host itself, are always accepted even if the host is not included in the list. See section 10.1, Authorization, in the PBS External Reference Specification. Requires full manager privilege to set or alter. Format: “[+|-]hostname.domain[,...]”; default value: all hosts.

acl_resv_host_enable

Attribute which when true directs the server to use the acl_resv_hosts access control list. Requires full manager privilege to set or alter. Format: boolean, “TRUE”, “True”, “true”, “Y”, “y”, “1”, “FALSE”, “False”, “false”, “N”, “n”, “0”; default value: false = disabled.

acl_resv_hosts

List of hosts which may request reservation services from this server. This list contains the network name of the hosts. Local requests, i.e. from the server’s host itself, are always accepted even if the host is not included in the list. See section 10.1, Authorization, in the PBS External Reference

Specification. Requires full manager privilege to set or alter. Format: “[+|-]hostname.domain[,...]”; default value: all hosts.

acl_resv_group_enable

Attribute which when true directs the server to use the reservation group access control list acl_resv_groups . Requires full manager privilege to set or alter. Format: boolean, “TRUE”, “True”, “true”, “Y”, “y”, “1”, “FALSE”, “False”, “false”, “N”, “n”, “0”; default value: false = disabled.

acl_resv_groups

List which allows or denies accepting reservations owned by members of the listed groups. The groups in the list are groups on the server host, not submitting hosts. See section 10.1, Authorization, in the PBS External Reference Specification. Format: “[+|-]group_name[,...]”; default value: all groups allowed.

acl_user_enable

Attribute which when true directs the server to use the server level acl_users access list. Requires full manager privilege to set or alter. Format: boolean; default value: disabled.

acl_users

List of users allowed or denied the ability to make any requests of this server. See section 10.1, Authorization, in the PBS External Reference Specification. Requires full manager privilege to set or alter. Format: “[+|-]user[@host][,...]”; default value: all users allowed.

acl_resv_user_enable

Attribute which when true directs the server to use the server level acl_resv_users access list. Requires full manager privilege to set or alter. Format: boolean; default value: disabled.

acl_resv_users

List of users allowed or denied the ability to make any reservation requests of this server. See section 10.1, Authorization, in the PBS External Reference Specification. Requires full manager privilege to set or alter. Format: "[+|-]user[@host][,...]"; default value: all users allowed.

acl_roots

List of super users who may submit to and execute jobs at this server. If the job execution id would be zero (0), then the job owner, root@host, must be listed in this access control list or the job is rejected. Requires full manager privilege to set or alter. Format: "[+|-]user[@host][,...]"; default value: no root jobs allowed.

comment

A text string which may be set by the scheduler or other privileged client to provide information to the batch system users. Format: any string; default value: none.

default_chunk

The list of resources which will be inserted into each chunk of a job's select specification if the corresponding resource is not specified by the user. This provides a means for a site to be sure a given resource is properly accounted for even if not specified by the user.

default_node

No longer used.

default_qdel_arguments

String containing argument to qdel. Argument is "-Wsuppress_mail=<N>". See qdel(1B). Settable by the administrator via the qmgr command. Overrides standard defaults. Overridden by arguments given on the command line.

default_qsub_arguments

String containing any valid arguments to qsub. See qsub(1B). Settable by the administrator via the qmgr command. Override standard defaults. Overridden by arguments given on the command line and in script directives.

default_queue

The queue which is the target queue when a request does not specify a queue name. Format: a queue name; default value: none, must be set to an existing queue.

flatuid

If set true, this boolean indicates that the user execution ID, UID, space is flat, consistent, across all systems from which a user may submit a job to this server. Therefore, if the job will execute under the UID of the job owner, the server will not need to authorize execution with that UID. If the job is to execute under a user name supplied in the job user list, see -u option, then authorization will take place. Requires full manager privilege to set or alter. Default value: unset - authorization is required for all UIDs

job_sort_formula

Formula for computing job priorities. Described in the PBS Professional Administrator's Guide. If the attribute job_sort_formula is set, the scheduler will use the formula in it to compute job priorities. If it is unset, the scheduler computes job priorities according to fairshare, if fairshare is enabled. If neither is defined, the scheduler uses job_sort_key. When the scheduler sorts jobs according to the formula, it computes a priority for each job, where that priority is the value produced by the formula. Jobs with a higher value get higher priority. Can be set by Manager or Operator. Format: String containing mathematical formula. Viewable by users, Manager or Operator. Default: unset.

The formula can be made up of expressions, where expressions contain terms which are added, subtracted or multiplied. For details, see the PBS Professional Administrator's Guide.

log_events

A bit string which specifies the type of events which are logged. See the PBS Professional Administrator's Guide. Format: integer; default value: 511, all events.

mail_from

The username from which server generated mail is sent to users. Requires full manager privilege to set or alter. On Windows, requires fully qualifed mail address. Format: string; default value: “adm”.

managers

List of users granted batch administrator privileges. Format: user@host.sub.domain[,user@host.sub.domain...] . The host, sub-domain, or domain name may be wild carded by the use of an * character, see the description of user access control lists in chapter 10.1.1 of the ERS. Requires full manager privilege to set or alter. Default value: root on the local host.

max_array_size

The maximum number of subjobs (separate indices) that are allowed in an array job. Format: integer; default value: 10000.

max_running

The maximum number of jobs allowed to be selected for execution at any given time, from all possible jobs. Format: integer.

max_user_res

The maximum amount of the specified resource that any single user may consume within a complex. The named resource can be any valid PBS resource, e.g. “ncpus”, “mem”, “pmem”, etc. Default value: none. Format: max_user_res.resource_name=value Example: set server max_user_res.ncpus=6

max_user_res_soft

The soft limit on the amount of the specified resource that any single user may consume within a complex. The named resource can be any valid PBS resource, e.g. “ncpus”, “mem”, “pmem”, etc. If a user is consuming more than this amount of the specified resource, their jobs are eligible to be preempted by jobs from users who are not over their soft limit. Default value: none. Format:

max_user_res_soft.resource_name=value Example: set server max_user_res_soft.ncpus=3

max_user_run
The maximum number of jobs owned by a single user that are allowed to be running within the complex at one time. Format: integer; default value: none.

max_user_run_soft
The soft limit on the number of jobs owned by a single user that are allowed to be running within this complex at one time. If a user has more than this number of jobs running, their jobs are eligible to be preempted by jobs from users who are not over their soft limit. Format: integer; default value: none.

max_group_run
The maximum number of jobs owned by any users in a single group that are allowed to be running within this complex at one time. Format: integer; default value: none.

max_group_run_soft
The maximum number of jobs owned by any users in a single group that are allowed to be running in this complex at one time. If a group has more than this number of jobs running, their jobs are eligible to be preempted by jobs from groups who are not over their soft limit. Format: integer; default value: none.

max_group_res
The maximum amount of the specified resource that any single group may consume in a complex. The named resource can be any valid PBS resource, e.g. “ncpus”, “mem”, “pmem”, etc. Default value: none. Format: max_group_res.resource_name=value Example: set server max_group_res.ncpus=6

max_group_res_soft
The soft limit on the amount of the specified resource that any single group may consume in a complex. The named resource can be any valid PBS resource, e.g. “ncpus”, “mem”, “pmem”, etc. If a group is consuming more than this amount of the

specified resource, their jobs are eligible to be preempted by jobs from groups who are not over their soft limit. Default value: none. Format: max_group_res_soft.resource_name=value Example: set server max_group_res_soft.ncpus=3

node_group_enable=<true|false>

Enables node grouping when true. See node_group_key. Requires manager privilege to set or alter. Boolean. Default: off. Example:
Qmgr> set server node_group_enable=true

node_group_key=RESOURCE

Specifies the resource to use for node grouping. Requires manager privilege to set or alter. See node_group_enable. String. Default: disabled. Example:
Qmgr> set server node_group_key=RESOURCE

node_fail_requeue

Controls if running a jobs are automatically requeued or deleted if the primary execution node fails. If unset or set to zero, jobs are left in the “Running” state until the execution node is recovered. If set to a non-zero value, the jobs are requeued if they are “rerunnable” or deleted when the node has been down for “value” seconds.

operators

List of users granted batch operator privileges. Format of the list is identical with managers above. Requires full manager privilege to set or alter. Default value: root on the local host.

pbs_license_file_location

Hostname of license server, or local pathname to the actual license file(s) associated with a license server.

To set pbs_license_file_location to the hostname of the license server: qmgr> set server pbs_license_file_location=

<port1>@<host1>:<port2>@<host2>:...:<portN>@<hostN>
where <host1>, <host2>, ... <hostN> can be IP addresses.

To set pbs_license_file_location to a local path: qmgr> set server pbs_license_file_location=<path_to_local_license_file>[[:<path_to_local_license_file2>]:...
:<path_to_local_license_fileN>]]

pbs_license_linger_time
The number of seconds to keep an unused CPU license, when the number of licenses is above the value given by pbs_license_min. Time. Set by PBS Manager. Readable by all. Default: 3600 seconds.

For Windows, use semicolons instead of colons, and enclose the pathlist in double quotes setting multiple paths, or if any path contains spaces.

To set pbs_license_linger_time:
Qmgr> set server pbs_license_linger_time=<Z>

To unset pbs_license_linger_time:
Qmgr> unset server pbs_license_linger_time

pbs_license_max
Maximum number of licenses to be checked out at any time, i.e maximum number of CPU licenses to keep in the PBS local license pool. Sets a cap on the number of CPUs that can be licensed at one time. Long. Set by PBS Manager. Readable by all. Default: maximum value for an integer.

To set pbs_license_max:
Qmgr> set server pbs_license_max=<Y>

To unset pbs_license_max:
Qmgr> unset server pbs_license_max

pbs_license_min

Minimum number of CPUs to permanently keep licensed, i.e. the minimum number of CPU licenses to keep in the PBS local license pool. This is the minimum number of licenses to keep checked out. Long. Set by PBS Manager. Readable by all. Default: zero.

To set pbs_license_min:
Qmgr> set server pbs_license_min=<X>

To unset pbs_license_min:
Qmgr> unset server pbs_license_min)

query_other_jobs
The setting of this attribute controls if general users, other than the job owner, are allowed to query the status of or select the job. Format: boolean (see acl_host_enable); Requires full manager privilege to set or alter. Default value: false - users may not query or select jobs owned by other users.

require_cred
Specifies the credential type required. All jobs submitted without the specified credential will be rejected. See also require_cred_enable. Depends on optional kerberos and DCE support. Format: string (krb5 or dce); Requires full manager privilege to set or alter. Not supported under Windows. Default value: unset

require_cred_enable
When true directs the Server to use the credential authentication method specified by require_cred. Depends on optional kerberos and DCE support. Format: boolean Requires full manager privilege to set or alter. Not supported under Windows. Default value: false = disabled

resv_enable
This attribute can be used as a master switch to turn on/off advance reservation capability on the server. If set False, advance reservations are not accepted by the server, however any already existing reservations will not be automatically

removed. If this attribute is set True the server will accept, for the scheduler's subsequent consideration, any reservation submission not otherwise rejected do to the functioning of an administrator established ACL reservation list. Requires full administrator privilege to set or alter. Default value: True. Format: True/False.
See also, resv_enable on the pbs_node_attributes manpage.

resources_cost
The cost factors of various types of resources. These values are used in determining the order of releasing members of synchronous job sets, see the section on Synchronize Job Starts. For the most part, these value are purely arbitrary and have meaning only in the relative values between systems. The cost of the resources requested by a job is the sum of the products of the various resources_cost s and the amount of each resource requested by the job. It is not necessary to assign a cost for each possible resource, only those which the site wishes to be considered in synchronous job scheduling. Requires full manager privilege to set or alter. Format: "resources_cost.resource_name=value[,...]"; default value: none, cost of resource is not computed.

resources_default
The list of default resource values that are set as limits for a job executing on this server when the job does not specify a limit, and there is no queue default. Format: "resources_default.resource_name=value[,...]"; default value: no limit.

resources_max
The maximum amount of each resource which can be requested by a single job executing on this server if there is not a resources_max valued defined for the queue in which the job resides. Format: "resources_max.resource_name=value[,...]"; default value: infinite usage.

rpp_highwater
The maximum number of RPP packets that can be in transit at any time. Acceptable values: Greater than or equal to one. Integer. Default: 64. Settable by Manager. Visible to all.

rpp_retry

The maximum number of times the RPP network library will try to send a UDP packet again before giving up. The number of retries is added to the original try , so if rpp_retry is set to 2, the total number of tries will be 3. Integer. Acceptable values: Greater than or equal to zero. Default: 10. Settable by Manager. Visible to all.

scheduler_iteration

The time, in seconds, between iterations of attempts by the batch server to schedule jobs. On each iteration, the server examines the available resources and runnable jobs to see if a job can be initiated. This examination also occurs whenever a running batch job terminates or a new job is placed in the queued state in an execution queue. Format: integer seconds; default value: 10 minutes, set by PBS_SCHEDULE_CYCLE in server_limits.h.

scheduling

Controls if the server will request job scheduling by the PBS job scheduler. If true, the scheduler will be called as required; if false, the scheduler will not be called and no job will be placed into execution unless the server is directed to do so by an operator or administrator. Setting or resetting this attribute to true results in an immediate call to the scheduler. For more information, see the section Scheduler - Server Interaction in the PBS Administrator's Guide. Format: boolean (see acl_host_enable); default value: value of -a option when server is invoked, if -a is not specified, the value is is recovered from the prior server run. If it has never been set, the value is “false”.

system_cost

An arbitrary value factored into the resource cost of any job managed by this server for the purpose of selecting which member of synchronous set is released first, see resources_cost and section 3.2.2, Synchronize Job Starts. Requires full manager privilege to set or alter. [default value: none, cost of resource is not computed]

Read Only Server Attributes

The following attributes are read-only. They are maintained by the server and cannot be changed by a client.

FLicenses

The number of floating licenses currently available for allocation to unlicensed CPUs. One license is required for each virtual CPU.

license_count

The license_count attribute is Avail_Global, Avail_Local, Used, High_Use.e Avail_Global is the number of PBS CPU licenses still kept by the Altair license server (checked in.) Avail_Local is the number of PBS CPU licenses still kept by PBS (checked out.) Used is the number of PBS CPU licenses currently in use. High_Use is the highest number of PBS CPU licenses checked out and used at any time by the current instance of the PBS server.

pbs_version

The version of PBS for this server. Available only to Manager/Operator.

resources_assigned

The total amount of certain types of resources allocated to running jobs.

server_name

The name of the server which is the same as the host name. If the server is listening to a non-standard port, the port number is appended, with a colon, to the host name. For example: host.domain:9999 .

server_state

The current state of the server:

Active The server is running and will invoke the job scheduler as required to schedule jobs for execution.

Idle The server is running but will not invoke the job scheduler.

server_host
The host name on which this server is running.

Scheduling
The server is running and there is an outstanding request to the job scheduler.

Terminating
The server is terminating. No additional jobs will be scheduled.

Terminating, Delayed
The server is terminating in delayed mode. The server will not run any new jobs and will shutdown when the last currently executing job completes.

state_count
The total number of jobs managed by the server currently in each state.

total_jobs
The total number of jobs currently managed by the server.

SEE ALSO
The PBS Professional Administrator's Guide, The PBS Professional User's Guide, qdel(1B), qmgr(1B), qsub(1b)

Index