PBS Portable Batch System

External Reference Specification

Albeaus Bayucan
Robert L. Henderson
Casimir Lesiak
Bhroam Mann
Tom Proett
Dave Tweten †

MRJ Technology Solutions
2672 Bayshore Parkway
Suite 810
Mountain View, CA 94043
http://pbs.mrj.com

Release: 2.2
Printed: November 30, 1999

† Numerical Aerospace Simulation Systems Division, NASA Ames Research Center, Moffett Field, CA

Portable Batch System (PBS) Software License

Acknowledgment: The Portable Batch System Software was originally developed as a joint project between the Numerical Aerospace Simulation (NAS) Systems Division of NASA Ames Research Center and the National Energy Research Supercomputer Center (NERSC) of Lawrence Livermore National Laboratory.

Redistribution of the Portable Batch System Software and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright and acknowledgment notices, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright and acknowledgment notices, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- All advertising materials mentioning features or use of this software must display the following acknowledgment:

  This product includes software developed by NASA Ames Research Center, Lawrence Livermore National Laboratory, and MRJ Technology Solutions.

DISCLAIMER OF WARRANTY

THIS SOFTWARE IS PROVIDED BY MRJ TECHNOLOGY SOLUTIONS ("MRJ") "AS IS" WITHOUT WARRANTY OF ANY KIND, AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT ARE EXPRESSLY DISCLAIMED.

IN NO EVENT, UNLESS REQUIRED BY APPLICABLE LAW, SHALL MRJ, NASA, NOR THE U.S. GOVERNMENT BE LIABLE FOR ANY DIRECT DAMAGES WHATSOEVER, NOR ANY INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This license will be governed by the laws of the Commonwealth of Virginia, without reference to its choice of law rules.

This product includes software developed by the NetBSD Foundation, Inc. and its contributors.

PBS Revision History

Revision 1.0	June, 1994 — Alpha Test Release
Revision 1.1	March 15, 1995
...	
Revision 1.1.9	December 20, 1996
Revision 1.1.10	July 31, 1997
Revision 1.1.11	December 19, 1997
Revision 1.1.12	July 9, 1998
Revision 2.0	October 14, 1998
Revision 2.1	May 12, 1999
Revision 2.2	November 30, 1999

Table of Contents

1. Introduction

1.1. Purpose

This document is the Portable Batch System External Reference Specification. It describes the overall design of the Portable Batch System, the **PBS**, and details its external behaviors and interfaces. User, Operator, and Administrator commands are described. The interface library which is used by the commands and also may be used to extend the functionality of **PBS** is described, as is the application level data exchange protocol (network protocol).

NOTICE

However, this system is currently in development and the interfaces and functionality described are subject to change during the course of development.

Suggested reading of sections of the ERS is as follows:

General User
: Users who just wish to make use of the common features of PBS to submit, monitor and control jobs are advised to read ERS chapters 1, 2.2, 2.7, 3.2 through 3.4, and 5.

Programmers
: Programmers wishing to add an interface to PBS to their code should read the sections listed for the general user and chapter 4.

Operators
: Batch system operator should read the sections listed under general user above plus chapters 6 and 7.

Administrators
: Batch system administrators or managers are advised to read the entire ERS with the possible exception of chapters 4 and 11.

The requirements for **PBS** are given in the document Portable Batch System Requirement Specifications. The internal design of each component of **PBS** is specified in the document Portable Batch System Internal Design Specification.

1.2. Glossary

Account	is an arbitrary character string which may have meaning to one or more hosts in the batch system. Frequently, account is used as a grouping for charging for the use of resources.
Administrator	See Manager.
Attribute	is an inherent characteristic of the parent object. Typically, this is a data item whose value affects the operation or behavior of the object and is settable by owner of the object. For example, the user may supply values for attributes of a job.
Batch	or batch processing, is the capability of running *jobs* outside of the interactive login session. In this document, batch implies a more complex subsystem which provides for additional control over job scheduling and resource contention.
Batch Server	is a persistent subsystem (daemon) upon a single host which provides batch processing capability.
Batch System	is a set of batch servers that are configured for processing. The system may consist of multiple hosts, each with multiple servers.

Cluster — A complex made up of cluster nodes.

Complex — A collection of nodes managed by one batch system. A complex may be made up of nodes that are allocated to only one job at a time or of nodes that have many jobs executing on each at once or a combination of both. A complex may be made up of a set of workstations, multiple cpu systems, or one or more parallel systems.

A queue complex in NQS was a set of queues within a batch server. The purpose of a complex was to provide additional control over resource usage. The advanced scheduling features of PBS eliminates the requirement for queue complexes.

Cluster Node — A node whose allocation units (Virtual processors) are allocated specifically to one job at a time (see .I "exclusive node"), or a few jobs (see *temporarily-shared nodes*). This type of node may also be called *space shared*. Hosts that are timeshared among many jobs are called “timeshared.”

In prior versions of PBS, the entire node was allocated to a job. In this version, one or more virtual processors (see definition in this glossary) may be declared. Each virtual processor is allocated as an independent unit.

Destination — is the location within the batch system where the job is sent for processing or executing. IN PBS, a destination may uniquely define a single queue at a single batch server or it may map into many locations.

Destination Identifier — is a string which names the destination. It is in two parts and has the format

queue@server

where server is the name of a batch server and queue is the string identifying a queue on that server.

Exclusive Nodes — An exclusive node is one whose virtual processors are used by one and only one job at a time. The processors of a set of nodes is assigned exclusively to a job for the duration of that job. This is typically done to improve the performance of message passing programs.

Temporarily-shared Nodes — A *temporarily-shared node* is one in which one or more of the virtual processors are temporarily shared by multiple jobs. If several jobs request multiple temporarily-shared nodes, some nodes may be allocated commonly to both job and some may be unique to one of the jobs. When a node is allocated as a temporarily-shared node, it remains so until all jobs using it are terminated. Then the node may be next allocated again for temporarily-shared use or for exclusive use.

File Staging — is the movement of files between a specified location and the execution host. See "Stage In" and "Stage Out" .

Group — is a collection of system users (see Users). A user must be a member of a group and may be a member of more than one. Within Unix and POSIX systems, membership in a group establishes one level of privilege. Group membership is also often used to control or limit access to system resources.

Hold — is an artificial restriction which prevents a job from being selected for processing. There are three types of holds, User which is applied by the job owner, Other (or operator) which is applied by the batch operator or administrator, and System which is applied by the system itself or the batch system administrator.

Job — or *batch job* is the basic execution object managed by the batch subsystem. A job is a collection of related processes which is managed as a whole. A job can often be thought of as a shell script. In POSIX terms, a job is a session group. A session is a processes group the member processes cannot leave.

Manager — or Batch System Manager is a person authorized to use all restricted capabilities of the batch system. The manager may act upon the the batch system, queues, or jobs. Also called the administrator.

Node — Within PBS, the term Node is used to mean a computational unit, one or more of which can be allocated to a job in a shared or exclusive manner. A node is characterized by (a) one or more virtual processors, (b)a unique memory address space, (c) and operating system image, (d) one or more IP addresses, and (e) a PBS execution server (pbs_mom). In prior versions of PBS, a node could only be allocated as a whole. Starting with version 2.2, pieces of a node, called virtual processors, VPs, may be defined and allocated individually. This is typically done where the node is a Shared Multiple Processor, SMP, system. References to allocating a *node* may still be found and should be taken to mean allocating a virtual processor. If a node only has one virtual processor defined, then of course, allocating that processor is synonymous with allocating the node itself.

If VPs on a set of nodes are allocated in an *exclusive* manner to a job, no other job may use that set of processors during the execution of the job to which the nodes are assigned. This is often called *Space Sharing* If a set of VPs on a set nodes are allocated in a *shared* manner, multiple jobs may have processes executing on the set at the same time.

Operator — or Batch Operator is a person authorized to use some but not all of the restricted capabilities of the batch system.

Owner — of a job is the user who submitted the job to the batch system.

PBS — is short for *Portable Batch System* .

POSIX — refers to the various standards being developed by the "Technical Committee on Operating Systems and Application Environments of the IEEE Computer Society" under standard P1003. There are a number of subcommittees under POSIX, those of interest to this project were:

- POSIX.1 System Application Program Interface (the system calls).
- POSIX.2 The command shell language.
- POSIX.3 Test Methods
- POSIX.10 Super Computing Profile
- POSIX.12 Protocol Independent Interfaces (one of the many network working groups)
- POSIX.14 Multiprocessor Working Group
- POSIX.15 Batch Queuing Extensions. This standard has been approved as 1003.2d and is the basis of the PBS external design.

Queue — is a collection of jobs (or job related tasks) within the batch queuing system. Each queue has a set of associated attributes which determine what actions are performed upon each job within the queue. Typical attributes include queue name, queue priority, resource limits, destination(s) and job count limits. Selection/scheduling of jobs is implementation defined. The use of the term "queue" does not imply the ordering is "first in, first out."

Rerunable — If a batch job can be terminated and its execution restarted from the beginning without harmful side effects, then the job is said to be rerunable.

Stage In is to move a file or files to the host prior to the batch job beginning execution.

Stage Out is to move a file or files off of the host after the batch job completes execution.

Timeshared In our context, to timeshare is to always allow multiple jobs to run concurrently on an execution host or node. A *timeshared node* is a node on which jobs are timeshared. Often the term *host* rather than node is used in conjunction with timeshared, as in *timeshared host*. If the term node is used without the timeshared prefix, the node is a cluster node which is allocated either exclusively or temporarily-shared.

User is a user of the compute system. Each user is identified by a unique character string, the user name; and by a unique number, the user id.

User ID is a numeric identifier uniquely assigned to each user. Privilege to access system resources and services is typically established by the user id.

Virtual Processor
An allocation unit that exists within a cluster (space-shared) node. Virtual processors, VPs, are allocated to jobs on an exclusive or temporarily shared basis. Virtual processors may or may not correspond in number to the number of real processors on the node. Fewer VPs than real processors may be defined to keep the batch load on the node down. More VPs may be defined to allow for over-subscription of the cpu resources.

1.3. Document Conventions

The following font conventions are used throughout this document.

New terms are introduced in italicized text.

Names of commands, library functions, and signals are shown in bold, serifed text.

[Error values] are shown in a sans-serif typeface and [inside brackets].

Option Argument and operands are shown as italic.

Attribute or data item names associated with jobs, queues, or the server are shown in a bold, sans-serif typeface.

Nonspecific values of attribute or data items are shown in a sans-serif typeface.

{Symbolic Constant} values, typically to be found in header files are shown in a sans-serif typeface and {inside braces}.

`Examples of formats and type-ins are in the fixed width typewriter font.`

Unresolved issues or areas which may require modification as the implementation progresses are called out by a note in a quoted paragraph, left and right indents, and are headed with the phrase "Author Note:". As these issues become resolved, this document will be updated and those sections removed.

1.4. Overview of The Portable Batch System

In the past, Unix systems were used in a totally 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 lead to the requirement of a networked batch capability.

The purpose of the **PBS** system is to provide additional controls over initiating or scheduling execution of batch jobs; and to allow routing of those jobs between different hosts. The batch system allows a site to define and implement policy as to what types of resources and how much of each resource can be used by different jobs. The batch system also provides a mechanism with which a user can insure a job will have access to the resources required to com-

plete.

The batch system is made up of a number of components, the server and clients such as user commands. A server component manages a number of different objects, such as queues or jobs. Each object consists of a number of data items or attributes. The attributes are characterized as *public* attributes, *read-only public* attributes, *private* attributes, and *internal* attributes.

Public attributes have values which are supplied by or can be changed by client requests. The behavior of the object changes when the value of an attribute is changed. The values of public attributes are available upon request to clients. Read-only attributes are public attributes whose values are available as status to clients, but the clients cannot change the values. Throughout this document, public and read-only attributes will be commonly referred to simply as "attributes". *Private* attributes are those data items which are permanent. They are passed as part of the object when the object changes ownership, for example when a job moves between servers. Private items are generally not made available to client programs. *Internal* data items are not visible, are not passed with the object between servers, and depend on the server implementation. Public and private attributes will be described in this ERS. Except in special cases, internal data items will not be described.

Typical interaction between the components is based upon the client - server model, with clients making (batch) requests to servers and the servers performing work on behalf of the clients. Clients do not create or modify objects directly, but depend upon the server which manages those objects.

A batch server is a persistent process or set of processes, such as a daemon. The batch server manages batch objects such as queues and jobs. It provides batch services like creating, routing, executing, modifying, or deleting jobs for batch clients. A batch server may at times request services of other servers. During that time, the server is acting in the role of a client.

User, operator, and administrator commands are batch clients. They allow users of the batch system to request batch services via the command line. While the commands may appear to accomplish certain services, they actually request and obtain the services from a batch server by means of a batch request.

The Interface Library provided as part of **PBS** supplies the interface to a server for the supplied commands and allows the development of other application clients.

1.5. PBS Features

This section describes some of the special features of PBS.

1.5.1. Interactive Batch Jobs

With a normal batch job, the input to the job is the script supplied via the qsub command and the output and error streams are spooled to disk files and delivered after the job completes. PBS provides support for scheduling and running jobs that require interactive user access to the input and output of the job during run time. This access is often required to run debuggers or other programs requiring feedback as part of a job that must have scheduled access to scarce hardware.

A PBS interactive job is submitted with the qsub command. If the `-I` option is specified on the qsub command line or in a #PBS directive in the script file [or if `-W interactive=true` is specified on the command line or in a directive] the job is a interactive job. A job may determine that it is an interactive batch job by the value of the environment variable **PBS_ENVIRONMENT**. Instead of the normal value of **PBS_BATCH**, it will have a value of **PBS_INTERACTIVE**. [Sessions not created by PBS do not have the PBS_ENVIRONMENT variable set.]

After submitting the job to the batch system, qsub remains active waiting for the job to connect over the network. When the job starts, data written to standard output and error is

sent to qsub which displays it on its terminal (qsub's standard output). The qsub command reads its standard input and passes the data to the job as the job's standard input. The job is connected via a pseudo tty, so job control signals and special characters are processed by the job, not the local qsub session. Since the job's stardard input is from the terminal, via qsub, the script is not executed as is a normal script. However, any #PBS directives in the script are processed by qsub and sent with the job. Therefore, job attributes and resources requirements may be specified in the script as with a normal batch job.

While qsub is waiting for the job to start, it will recognize the interrupt signal typically generated by entering CNTL-C on the keyboard. Qsub will ask if the user wishes to exit. If the user responds with yes (or any string starting with a 'y') qsub will send a request to the server asking that the batch job be aborted. Qsub will also periodically check on the batch job. If qsub finds that the job has been deleted, qsub will inform the user and exit.

When the job starts execution, qsub will inform the user and begin to relay the input, output and error streams. Control keys are passed to the job. Thus a CNTL-Z will suppend the job, not the qsub command, and CNTL-C will cause a interrupt signal to the job, not to qsub. During this time, qsub will three escape commands, if the line begins with:

~. Qsub will exit. This will end the job.

~^Z (CNTL-Z) Qsub will suspend itself, the job remains running. Neither input nor output is transferred. The user may issue commands to the local shell.

~^Y (CNTL-Y) Qsub will suspend the part of itself that sends input to the job. This allows the user to issue commands to the local shell while still receiving the output of the job.

Note, the last two escape lines do not work if the local shell does not support job control, e.g. is the Bourne shell sh.

When the job terminates, qsub will exit returning control to the local shell.

1.5.2. Job Prologue and Epilogue Scripts

PBS provides for the execution of two administrative supplied scripts with each job. The prologue script is run immediately before a job is executed, the epilogue script is run immediately after. Both scripts are run with root privilege and may used to place a "banner" on the job's output or establish part of the evironment for the job such as creating temporary directories or cleaning up after the job.

1.5.3. File Stage In and Stage Out

PBS provides for files to be staged, or moved, before and after a job is run. The user specifies a "remote" location and name of the file and the "local" name when submitting the job. The remote name includes a host name which is typically a remote host, but may be the local execution host. If the file is local to the execution host, /bin/cp is used to copy the file, if the file is remote, rcp is used.

Staging out of files occures as part of the post job processing. The job is shown to be in exiting, 'E', state during the staging out. Once the files are staged out to their destination, they are deleted from the execution host. If the user wishes to retain the files on the execution host, s/he should link the file to second file name using the *ln* (link) command.

Staging in files is a bit more complex. A decision must be made about when to begin to stage in files for a job. The files must be available before the job executes. The amount of time that will be required to copy the files is unknown to PBS, that being a function of file size and network speed. If file in-staging is not started until the job has been selected to run when the other required resources are available, either those resources are "wasted" while the stage in occurs, or another job is started which takes the resources away from the first job, and might prevent it from running. If the files are staged in well before the job is otherwise ready to run, the files may take up valuable disk space need by running jobs.

PBS provides two ways that file in-staging can be initiated for a job. If a run request is received for a job with a requirement for staging-in files, the staging in operation is begun and when completed, the job is run. Or, a specific stage-in request may be received for a job, see pbs_stagein(3B), in which case the files are staged in but the job is not run. When the job is run, it begins execution immediately because the files are already there.

In either case, if the files could not be staged-in for any reason, the job is placed into a wait state with a "execute at" time {PBS_STAGEFAIL_WAIT}, 30 minutes, in the future. A mail message is sent to the job owner requesting that s/he look into the problem. The reason the job is changed into wait state is to prevent the scheduler from constantly retrying the same job which likely would keep on failing.

Figure 1−1 shows the (sub)state changes for a job involving file in staging. The scheduler may note the substate of the job and chose to perform pre-staging via the pbs_stagein() call. The scheduler developer should carefully chose a stage in approach based on factors such as the likely source of the files, network speed, and disk capacity.

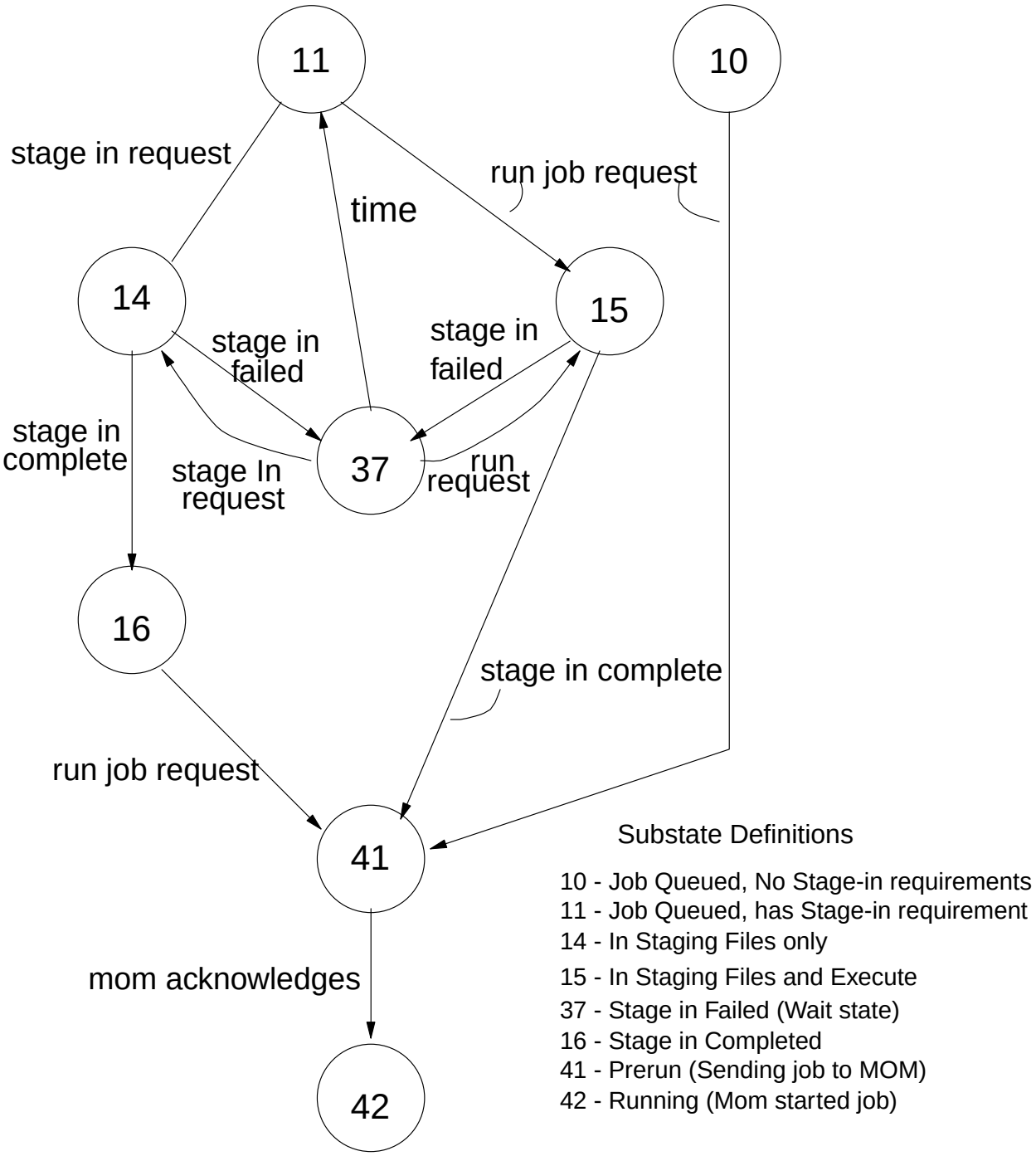


Figure 1−1: Job Substate Changes During File Stage In

1.6. Acknowledgements

Much assistance to the PBS project was given in the early days in terms of man power and ideas by both *Lawrence Livermore Nation Laboratory* and by the *National Energy Research Supercomputer Center*. Special thanks go to Bruce Kelly and Clark Streeter of NERSC, who directly assisted in the development of PBS. Additional help was provided by Kent Crispin and Terry Heidelberg of LLNL.

The supplied code for the mom_rcp utility was taken from the *bsd4.4-Lite* distribution. This code is copyrighted by *The Regents of the University of California*. The complete copyright notice and right to modify and redistribute the software is contained in the source code.

The Red Hat Linux port was done by the *Pittsburgh Supercomputing Center* under funding by the *National Institute of Standards and Technology*, NIST. Special thanks go to John Kochmar and Rob Pennington of PSC for doing the port.

The ports of PBS to Digitial Equipment Corporation Unix on the Alpha workstation and to HP-UX were provided by Dirk Grunwald at *University of Colorado, Boulder*.

No list of acknowledgements for PBS would possibly be complete without special recognition of the first two beta test sites and the brave individuals who were willing to try PBS. 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 tester and holds the honor of submitting more problem reports than anyone else outside of NASA. Without you two, the project would not be so successful.

[Page intentionally left bank.]

2. General Specifications

A server is a persistent process, a daemon, which manages several classes of objects and provides batch services. Each class of object has a set of attributes (variables) which contain information that is specific to that object. In the following sections, the objects and the services are described.

2.1. Attribute Types

Each attribute associated with an object has a defined data type. The following is a list of the general data types currently supported.

Boolean
: used for true/false yes/no variables. The true state may be input as `True`, `TRUE`, `y`, `Y`, or `1`. The false state may be input as `False`, `FALSE`, `n`, `N`, or `0`. Unset boolean attributes are generally treated as if set to false.

Integer
: used for numeric variables. The input data is a numeric string specifying a value which will fit into a long integer on the host system. Some attributes will place additional restrictions on the value range.

Size
: used for size of disk or memory related values. The input data is in the form of a numeric string with optional suffix. The suffix consist of an optional scale factor character `kKmMgGtT` and an optional byte or word indicator `bBwW`.

Character
: used for types containing a single alphanumeric character.

String
: used for types requiring a single null terminated character string. Additional format requirements may be placed on the string for a specific attribute.

Array of Strings
: used where the attribute value is a series of strings. The value is input as a set of comma separated stings. The value at the human interface level (command level) often requires quoting.

List
: used for those attributes requiring a linked list of data structures. The input data is typically of the same form as the "array of strings" above.

Resources
: is a special case of list for resource limit or resource usage items.

Access Control List
: is a special case of list for access control lists. More information on the access control lists can be found in the ERS chapter "Security", section "Authorization".

2.2. Batch Jobs

A batch job is a primary object managed by a batch server. From the user's view point, a job is a file which is submitted via the **qsub** command. Typically, this file is a shell script and is interpreted by a command shell. In fact the file may contain any data and the user can request any valid program to process the file as its standard input. In addition to the shell script, a batch job consists of many attributes which affect the processing of the job. These are covered in the next section.

The server maintains an internal representation of the job and a copy of that representation as a file on disk to insure the job information is not lost across server instantiations.

Jobs are created by the server upon receipt and successful processing of a *Queue Job* batch request. Jobs are maintained by the server until (1) the job executes and terminates, (2) the job is deleted by a *Delete Job* batch request, (3) the job is moved or routed to another server, or (4) the server determines it is impossible to process the job and the job is *aborted* .

When a job is created, it is named with a *job_identifier* by the server which created the job. The job_identifier is of the form

```
sequence_number.server_name
```

where the `sequence_number` is a unique number within the creating server and `server_name` is the name of that server.

2.2.1. Public Job Attributes

A batch job has the following public attributes shown in the following list. The attributes marked with the section symbol § are required by POSIX 1003.2d: If an attribute is unset, the indicated default value is assumed.

Account_Name §
: Used for accounting on some hosts. A server may not use the string, but allowances for it must be made. Format: string; default value: none, not used. [internal type: string]

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. [internal type: string]

dependThe type of inter-job dependencies specified by the job owner. Format: "type:jobid[,jobid...]"; default value: no dependencies. [internal type: special, dependency]

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). [internal type: list]

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. [internal type: integer]

group_list §
: A list of `group_names@hosts` which determines the group under which the job is run on a given host. [internal type: array of strings] 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]...]". [internal type: array of strings]

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. [internal type: string]

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. [internal type: string]

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. [internal type: string]

Keep_Files §
: If Keep_Files contains the values "o" {KEEP_OUTPUT} and/or "e" {KEEP_ERROR} 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. [internal type: string]

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 default value: 'a', send on job abort. [internal type: string]

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. [internal type: array of strings]

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). [internal type: string]

Priority §
: The job scheduling priority assigned by the user. Format: "[+|-]nnnnn"; default value: undefined. [internal type: integer]

Rerunable §
: The rerunable flag given by the user. Format: "y" or "n", see Join_Path; default value: y, job is rerunable. [internal type: boolean]

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. [internal type: 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. [internal type: array of strings]

stagein
: The list of files to be staged in prior to job execution. Format: local_path@remote_host:remote_path [internal type: array of strings]

stageout
: The list of files to be staged out after job execution. Format: local_path@remote_host:remote_path [internal type: array of strings]

User_List §
: The list of `user@hosts` which determines the user name under which the job is run on a given host. [internal type: array of strings] 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. [internal type: array of strings]

Variable_List §
: This is the list of environment variables passed with the *Queue Job* batch request. Format: "name=value[,name=value...]". [internal type: array of strings]

2.2.2. Privileged Job Attributes

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. [internal type: string]

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]

2.2.3. Read-Only Job 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).

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. [internal type: string]

ctime
: The time that the job was created. [internal type: integer, (seconds since epoch)]

etime
: The time that the job became eligible to run, i.e. in a queued state while residing in an execution queue. [internal type: integer, (seconds since epoch)]

exec_host
: If the job is running, this is set to the name of the host on which the job is executing. [internal type: string]

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 available only to the batch administrator.] [internal type: string]

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 available only to the batch administrator.] [internal type: string]

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. [This attribute is available only to the batch administrator.] [type: string]

interactive
True if the job is an interactive PBS job. Format: boolean, see Join_Paths; default value: false. [internal type: long] Internally, the value is the port number obtained by qsub when the job was submitted.

Job_Owner §
The login name on the submitting host of the user who submitted the batch job. [internal type: string]

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 (Job running on Unicos only), the job was executing and has been suspended. The job retains its assigned resources but does not use cpu cycle or wall-time.

T for Transiting, the job is in process of being routed or moved to a new destination.

W for Waiting, the job is not held but the Execution_Time attribute contains a time which has not yet been reached.

[internal type: character]

mtime The time that the job was last modified, changed state, or changed locations. Internally, maintained as number of seconds since epoch. [internal type: integer]

qtime The time that the job entered the current queue. Internally, maintained as number of seconds since epoch. [internal type: integer]

queue The name of the queue in which the job currently resides. [internal type: string]

queue_rank
An ordered, non-sequential number indicating the job's position with in the queue. This is provided as an aid to the scheduler. [This attribute is available to the batch manager only.] [internal type: integer] 7 7

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 available to the batch manager only.] Format: The letter "E" or the letter "r". [internal type: character]

resources_used §
The amount of resources used by the job. This is provided as part of job status information if the job is running. [internal type: resource]

server The name of the server which is currently managing the job. [internal type: string]

session_id
If the job is running, this is set to the session id of the first executing task. [internal

type: integer]

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. Format: interger. [internal type: long integer] The values are defined in the header file job.h. See the ERS section on file staging for why it is available to the scheduler.

2.2.4. Job Private Attributes

The following data items are private attributes of the job. These items are a permanent part of the job object and are passed with the job between servers or between the server and the execution server, but are not passed to user clients.

hopcount
The hop count is maintained by the server. It is set to zero when the job is created and incremented each time the job changes destination, queue or server. The hopcount attribute is used to prevent endless routing loops and to ensure correct ordering of updates of the job's current location for the **Job Locate** batch request. [type: integer]

security
Reserved for future implementation. [type: string]

2.2.5. Job Internal Data Items

The following data items are internal to the server representation of a job. They are specifically described here because of their importance.

destination
The destination_id supplied on the **qsub** or **qmove** commands.

job_substate
The secondary job state field, see "state" under Job Read-Only Attributes. As it is not visible to clients, the values are not defined in this document.

2.2.6. Interactive Batch Jobs

PBS supports "interactive batch jobs". An interactive batch job is a job submitted to PBS where the standard input, output, and error streams of the job are connected to the terminal session in which qsub is run. The qsub command acts as a conduit for the communication between the job and the terminal session.

2.3. Queues

A batch queue is an object managed by a batch server. A batch queue consists of a collection of zero or more batch jobs, a set of queue attributes, private attributes, and a set of internal data items. Jobs are said to reside in the queue or be members of the queue. In spite of the name, jobs residing in a queue need not be ordered first in, first out.

Access to a queue is limited to the server which owns the queue. All clients gain information about a queue or jobs within a queue through batch requests to the server.

Two main types of queues are defined: routing queues and execution queues. The type of queue is determined by which subset of queue attributes have been assigned to it.

When a job resides in a routing queue, it is a candidate for routing to a new destination. Each routing queue has a list of destinations to which jobs may be routed. The new destination may be a different queue within the same server or a queue under a different server.

Jobs are removed from a routing queue when:

- The job has been successfully routed to another queue.
- The job has been deleted.
- The job has been moved to another queue.
- The job has been aborted by the server.

When a job resides in an execution queue, it is a candidate for execution. A job in execution is still a member of the execution queue from which it was selected for execution. Jobs are removed from an execution queue when:

- The job has executed and terminated.
- The job has been deleted.
- The job has been moved to another queue.
- The job has been aborted.

2.3.1. Queue Public Attributes

Queue public attributes are alterable on request by a client. The client must be acting for a user with administrator (manager) or operator privilege. Certain attributes require the user to have full administrator privilege before they can be modified. The following attributes apply to both queue types:

acl_group_enable
: Attribute which 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. [internal type: boolean]

acl_groups
: List which allows or denies enqueuing of jobs 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 Specificaiton. Format: "[+|-]group_name[,...]"; default value: all groups allowed. [internal type: access control list]

acl_host_enable
: Attribute which when true directs the server to use the *acl_hosts* access list. Format: boolean (see acl_group_enable); default value: disabled. [internal type: boolean]

acl_hosts
: List of hosts which may enqueue jobs in the queue. See section 10.1, Authorization, in the PBS External Reference Specificaiton. Format: "[+|-]hostname[...]"; default value: all hosts allowed. [internal type: access control list]

acl_user_enable
: Attribute which when true directs the server to use the the *acl_users* access list for this queue. Format: boolean (see acl_group_enable); default value: disabled. [internal type: boolean]

acl_users
: List of users allowed or denied the ability to enqueue jobs in this queue. See section 10.1, Authorization, in the PBS External Reference Specificaiton. Format: "[+|-]user[@host][,...]"; default value: all users allowed. [internal type: access control list]

enabled
: Queue will or will not accept new jobs. When false the queue is “disabled” and will not accept jobs. Format: boolean (see acl_group_enable); default value: disabled. [internal type: boolean]

from_route_only
: When true, this queue will not accept jobs except when being routed by the server from a local routing queue. This is used to force user to submit jobs into a routing queue used to distribute jobs to other queues based on job resource limits. Format: boolean; default value: disabled. [internal type: boolean]

max_queuable
: The maximum number of jobs allowed to reside in the queue at any given time. Format: integer; default value: infinite. [internal type: integer]

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. For an execution queue, this attribute is advisory to the Scheduler, it is not enforced by the server. Format: integer. [internal type: integer]

Priority
: The priority of this queue against other queues of the same type on this server. May affect job selection for execution/routing. Advisory to the Scheduler, not used by the server. Format: integer. [internal type: integer]

queue_type
: The type of the queue: execution or route. Format: "execution", "e", "route", "r". This attribute must be explicitly set. [internal type: string]

resources_max
: The maximum amount of each resource which can be requested by a single job in this queue. The queue value superceeds any server wide maximum limit. Format: "resources_max.resource_name=value", see qmgr(1B); default value: infinite usage. [internal type: resource]

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. [internal type: resource]

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 unlimited resource usage. [internal type: resource]

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). [internal type: boolean]

The following attributes apply only to execution queues:

checkpoint_min §
: 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. [internal type: integer]

resources_available
: The list of resource 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 ex-

ceed the total amount listed here. Advisory to the Scheduler, not enforced by the server. Format: "resources_available.resource_name=value", see qmgr(1B). [internal type: resource]

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. [internal type: integer]

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. This attribute is advisory to the Scheduler, it is not enforced by the server. Format: integer; default value: none. [internal type: integer]

max_group_run
: The maximum number of jobs owned by any users in a single group that are allowed to be running from this queue at one time. This attribute is advisory to the Scheduler, it is not enforced by the server. Format: integer; default value: none. [internal type: integer]

The following attributes apply only to routing queues:

route_destinations
: The list of destinations to which jobs may be routed. [default value: none, should be set to at least one valid destination] [internal type: array of strings]

alt_router
: If true, an site supplied, alternative job router function is used to determine the destination for routing jobs from this queue. Otherwise, the default, round-robin router is used. Format: boolean (see acl_group_enable); default value: false. [internal type: boolean]

route_held_jobs
: If true, jobs with a hold type set may be routed from this queue. If false, held jobs are not to be routed. Format: boolean (see acl_group_enable); default value: false. [internal type: boolean]

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. [internal type: boolean]

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). [internal type: integer]

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. [internal type: integer]

2.3.2. Queue Read-Only Attributes

The following data items are read-only attributes of the queue. They are visible to but cannot be changed by clients.

Items which apply to all types of queues are:

total_jobs
: The number of jobs currently residing in the queue. [internal type: integer]

state_count
: The total number of jobs currently residing in the queue in each state. [internal type: special, array of integers]

These read-only attributes only apply to execution queues:

resources_assigned
: The total amount of certain types of resources allocated to jobs running from this queue. [internal type: resource]

2.4. Batch Server Attributes

The following attributes apply to the server.

2.4.1. Server Public Attributes

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. [internal type: boolean]

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 aways accepted even if the host is not included in the list. See section 10.1, Authorization, in the PBS External Reference Specificaiton. Requires full manager privilege to set or alter. Format: "[+|-]hostname.domain[,...]"; default value: all hosts. [internal type: access control list]

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 (see acl_group_enable); default value: disabled. [internal type: boolean]

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 Specificaiton. Requires full manager privilege to set or alter. Format: "[+|-]user[@host][,...]"; default value: all users allowed. [internal type: access control list]

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. Format: "[+|-]user[@host][,...]"; default value: no root jobs allowed. [internal type: access control list]

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. [internal type: string]

default_node
: A node specification to use if there is no other supplied specification. This attribute is only used by servers where a *nodes* file exist in the server_priv directory providing a list of nodes to the server. If the nodes file does not exist, this attribute is not set by default and is ignored if set. The default value allows for jobs

to share a single node. Format: a node specification string; default value: 1#shared. [internal type: string]

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. [internal type: string]

log_events
: A bit string which specifies the type of events which are logged, see the section on Event Logging in chapter 3 of the ERS. Format: integer; default value: 511, all events. [internal type: integer]

mail_uid
: The uid from which server generated mail is sent to users. Format: integer uid; default value: 0 for root. [internal type: integer]

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. [internal type: access control list]

max_running
: The maximum number of jobs allowed to be selected for execution at any given time. Advisory to the Scheduler, not enforced by the server. Format: integer. [internal type: integer]

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. This attribute is advisory to the Scheduler, it is not enforced by the server. Format: integer; default value: none. [internal type: integer]

max_group_run
: The maximum number of jobs owned by any users in a single group that are allowed to be running from this queue at one time. This attribute is advisory to the Scheduler, it is not enforced by the server. Format: integer; default value: none. [internal type: integer]

node_pack
: Controls how multiple processor nodes are allocated to jobs. If this attribute is set to true, jobs will be assigned to the multiple processor nodes with the fewest free processors. This packs jobs into the fewest possible nodes leaving multiple processor nodes free for jobs which need many processors on a node. If set to false, jobs will be scattered across nodes reducing conflicts over memory between jobs. If unset, the jobs are packed on nodes in the order that the nodes are declared to the server (in the nodes file). Default value: unset − assigned to nodes as nodes in order that were declared. [internal type: boolean]

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. [internal type: access control list]

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. [internal type: boolean]

resources_available
: The list of resource and amounts available to jobs run by this server. The sum of the resource of each type used by all jobs running by this server cannot exceed the total amount listed here. Advisory to the Scheduler, not enforced by the server. Format: "resources_available.resource_name=value[,...]". [internal type: resource]

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. Format: "resources_cost.resource_name=value[,...]"; default value: none, cost of resource is not computed. [internal type: list]

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. [internal type: resource]

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. [internal type: resource]

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. [internal type: integer, displays as name defined below]

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 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 recoved from the prior server run. If it has never been set, the value is "false". [internal type: boolean]

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." [default value: none, cost of resource is not computed] [internal type: list]

2.4.2. Read Only Server Attributes

The following attributes are read-only, they are maintained by the server and cannot be changed by a client.

resources_assigned
: The total amount of certain types of resources allocated to running jobs. [internal type: resource]

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`. [internal type: string]

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.

  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.

  [internal type: integer]

state_count
: The total number of jobs managed by the server currently in each state. [internal type: special, array of integers]

total_jobs
: The total number of jobs currently managed by the server. [internal type: integer]

PBS_version
: The release version number of the server. [internal type: string]

2.4.3. Server Sub-Objects

The following are lists of objects belonging to the server.

queues
: List of the queues managed by this server.

job List of the jobs managed by this server.

2.5. PBS Files

The PBS subsystem maintains its files under the directory {PBS_DIR} which is usually set to /usr/spool/PBS, see figure 2−1 for the layout directories used by PBS. There is a subdirectory, {PBS_DIR} /server_priv for private files accessible only by the server, see figure 2−2 used by the Server daemon for its private files. The Job Executor, MOM, has a similiar subdirectory mom_priv.

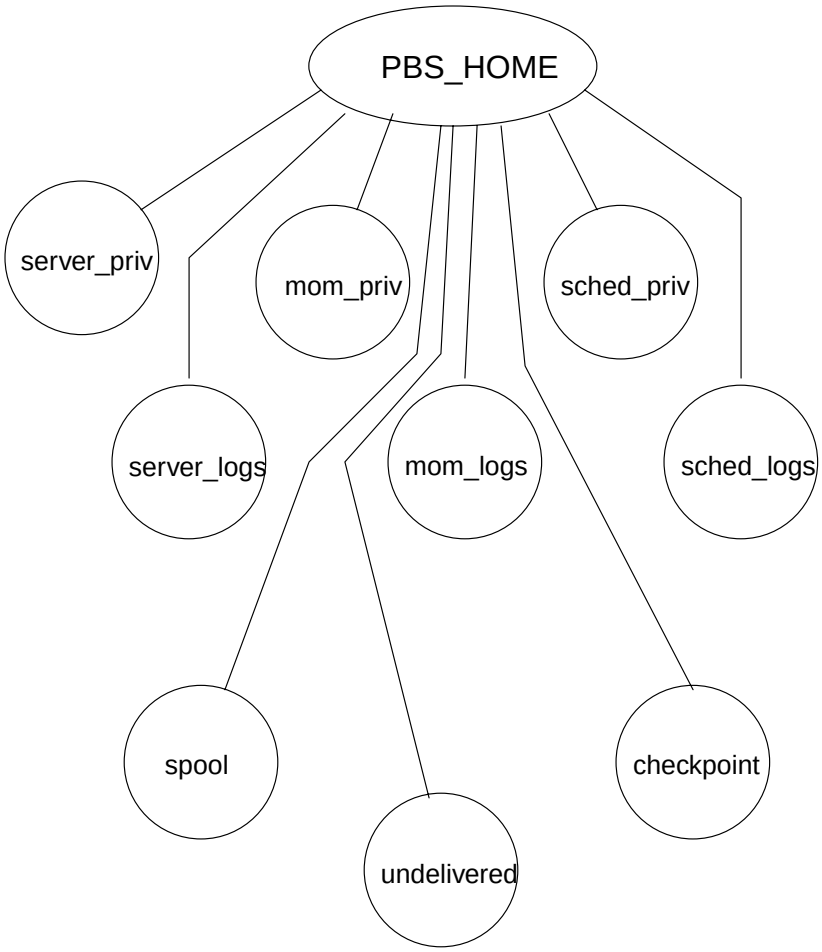


Figure 2−1: PBS Directory Structure

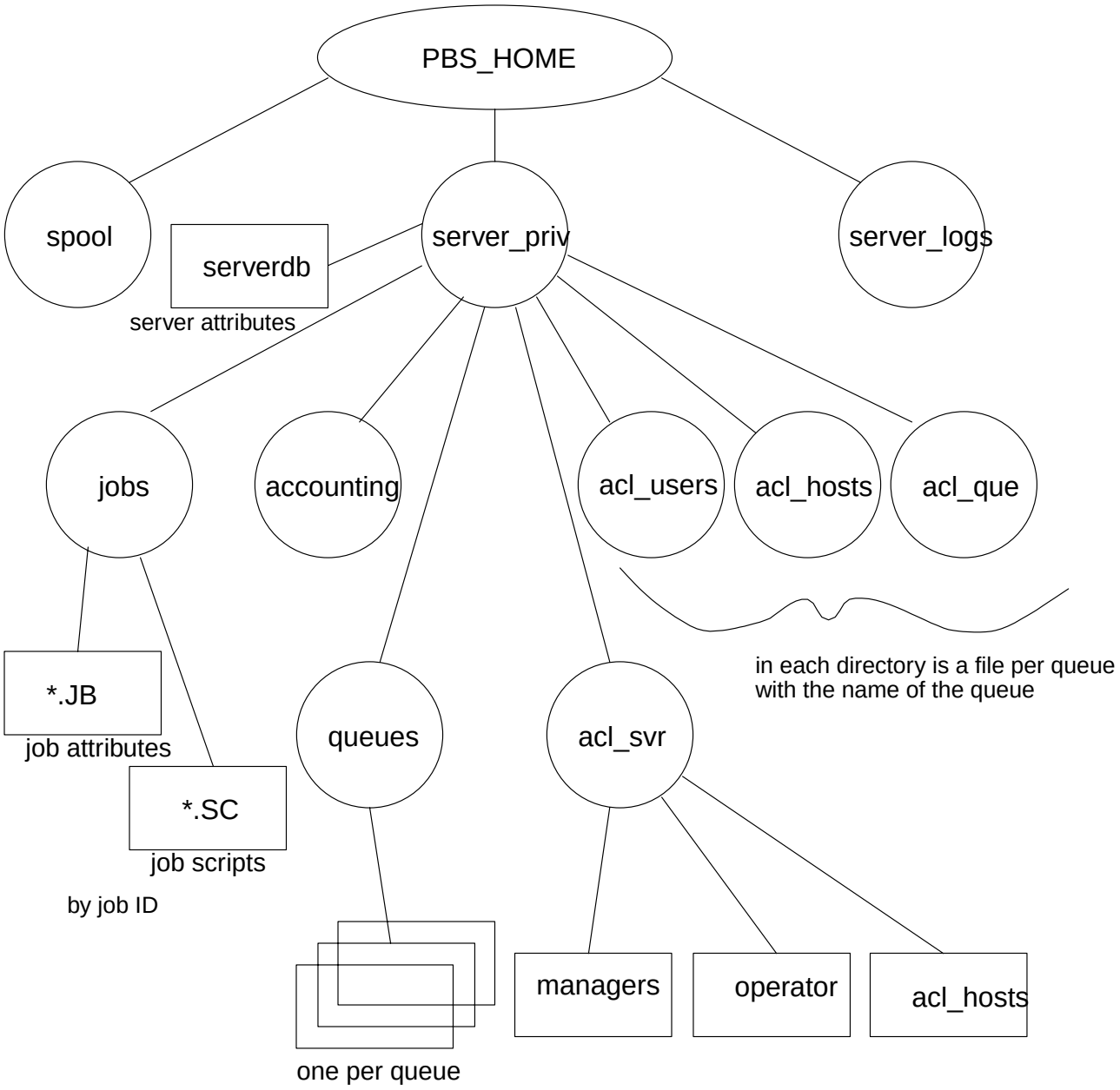


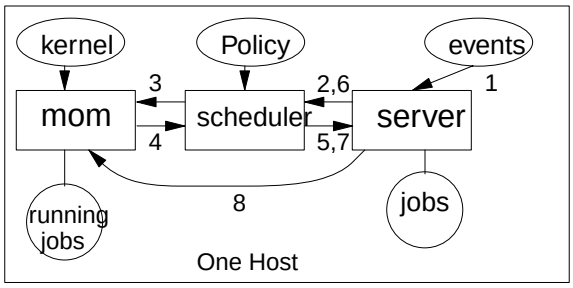
Figure 2−3: PBS Server Files

2.6. Job Selection / Scheduling

Job selection or job scheduling is the process of picking which eligible job is placed into execution or initiated. To be eligible, the job must reside in an execution queue and be in the **queued** state.

To maximize flexibility in implementing site policy, **PBS** provides a separate program as the selection process. This process operates on the principle of evaluating the eligible jobs according to a program written either in a yacc/lex based procedural language or in Tcl. This process, the **PBS Scheduler**, pbs_sched, communicates with the PBS Server via a socket based IPC, using the standard PBS API. This provides the capability of having the Scheduler and the Server reside on different hosts.

The Scheduler communicates with another process called the **Machine Oriented Miniserver**, pbs_mom, in its role as a resource manager to obtain information about the loading of the host system. Loading information may contain details of memory usage, cpu load average, and more. This information, taken as input to the scheduling program can be used to control job scheduling. The relationship between PBS, the Scheduler, and the Machine Oriented Miniserver can be seen in figures 2−4 and 2−5.



1. Event tells Server to initiate a scheduling cycle.
2. Server sends scheduling command to Scheduler.
3. Scheduler requests resource info from MOM.
4. MOM retuns requested info.
5. Scheduler requests job info from server.
6. Server sends job status info to scheduler.
Scheduler makes policy decision to run job.
7. Scheduler sends run request to server.
8. Server sends job to MOM to run.

Figure 2−4: Batch Scheduling on a Single Host

The concept of the Scheduler and Machine Oriented Miniserver can be extended to include multiple hosts. A single Scheduler would provide scheduling for one or more PBS servers. The Scheduler would talk with a Machine Oriented Miniserver on each host for which it had scheduling responsibility.

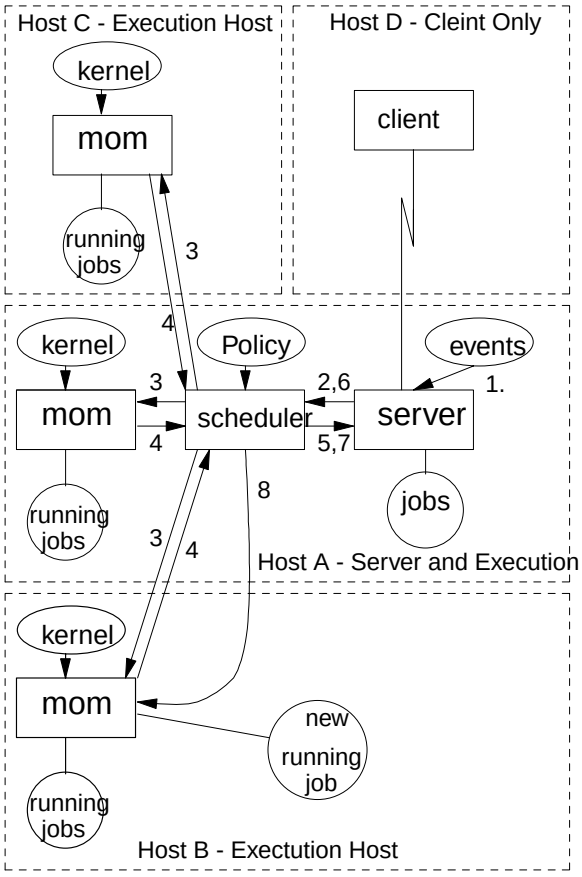


Figure 2−5: Batch Scheduling on Multiple Hosts

The procedural or Tcl script which drives the Scheduler is a plain text file. The administrator may update the script in the file or direct the scheduler to use a different file.

The script may be written to consider time and queue information, so different procedures could apply at different times and to different queues. The script may also state when the conditions are such that a job should be started, deleted, or suspended.

2.7. 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 sub-sections 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 enteried on the command line or in a PBS job directive.

2.7.1. 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.

2.7.2. 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.

2.7.3. 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, section 2.7.9. If `server_name` is omitted, the default server is assumed.

2.7.4. 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 by (b) the server name in the file specified by the $(PBS_DEFAULT_FILE) build parameter in the local.mk file.

2.7.5. 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 separater. The name must not contain the commerical 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.

2.7.6. 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. See the definition of default server in section 2.7.4 and the section 5.1.2, entitled "Directing Requests to Correct Server."

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.

2.7.7. 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.

2.7.8. Resource Name

A *Resource Name* identifies a job resource requirement and may also indentify a resource usage limit. The name must consist of alphanumeric characters plus the underscore, "_", character. It should start with an alphamumeric character. The length is not limited. Certain resource names are identified and reserved by POSIX 1003.2d and by PBS. They are listed in section 3.4.3, "Types of Resources".

2.7.9. 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 (see section 2.7.5). 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.

2.7.10. 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.

[Page intentionally left bank.]

3. Batch Server Functions

A batch server provides services in one of two ways:

- The server provides a service at the request of a client.
- 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.

3.1. 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.

The server is required to respond to all requests it receives. Those requests that cannot be successfully completed, are *rejected*. The reason for the rejection is returned in the reply to the client.

The following subsections describe the services provided by a batch server in response to a request from a client. The requests are grouped in the following sub-sections by the type of object affected by the request: server, queue, job, or resource.

3.1.1. Server Management

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.

3.1.1.1. 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.

3.1.1.2. Server Status Request

The status of the server may be requested with a *Server Status* request.

The batch server will reject the request if any of the following conditions are true:

- 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.

3.1.1.3. 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.

3.1.1.4. Shut Down

The batch server is "shut down" when it no longer reponds 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.

3.1.2. 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.

3.1.2.1. 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.

3.1.3. 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.

3.1.3.1. Abort Request

[The Abort Request has been aborted (deleted).]

3.1.3.2. 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.

3.1.3.3. 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 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 server will send a **SIGKILL** signal to the job process group.

The job is then placed into the **exiting** state.

3.1.3.4. 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 checkpoint/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 the *expedite modifier* to the hold type is specified, the sched_hint job attribute will be set to one (1) to indicate to the scheduler that this job should be placed into execution as soon as possible. If the *expedite modifier* is not specified, the job will be rescheduled for execution as if it had been re-submitted.

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.

3.1.3.5. Queue Job Request

The *Queue Job* sub-request is part of the Queue Job complex request. This sub-request identifies the destination for the job and, in effect, asks the receiving server to fork off a child to process the remaining sub-requests. It also passes the public and private job attributes to the receiving server.

3.1.3.6. 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 in section 10.2.

3.1.3.7. 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.

3.1.3.8. 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.

3.1.3.9. 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 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.

3.1.3.10. 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. This is detailed later in this subsection.
- 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.

The following table indicates which attributes are alterable in each state.

Attribute	q	h	w	r	t	e
Account_Name	x	x	x			
Checkpoint	x	x	x	1		
depend	x	x	x			
Error_Path	x	x	x	1		
Execution_Time	x	x	x	1		
group_list	x	x	x			
Hold_Types	x	x	x	1		
Job_Name	x	x	x	x		
Join_Path	x	x	x			
Keep_Files	x	x	x			
Mail_Points	x	x	x	x		
Mail_Users	x	x	x	x		
Output_Path	x	x	x			
Priority	x	x	x			
Rerunable	x	x	x	x		
Resource_List	x	x	x	2		
Shell_Path_List	x	x	x	1		
stagein	x	x	x			
stageout	x	x	x			
User_List	x	x	x			
Variable_List						

Notes:

1. May be altered with qalter, but changes will not take effect until job is requeued. Use of qhold produces special processing.
2. Only certain resources limits may be changed. Those resource limits may be lowered by job owner. Increasing those limits requires special privilege.

3.1.3.11. 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.

3.1.3.12. 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 target queue is an execution queue, the batch server does not support checkpoint, and the value of the Checkpoint attribute of the job is neither the single character "n" nor the single character "u".
- 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}. Then the job is placed in the **waiting** state.
- The job has a Hold_Types attribute with a value other than {NONE}. The job is placed in the **held** state.

When a job is placed in a routing 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, the Hold_Types attribute has value of {NONE}, and the route_waiting_jobs queue attribute is {FALSE}. Then the job is placed in the **waiting** state.
- The job has a Hold_Types attribute with a value other than {NONE} and the route_held_jobs queue attribute is {FALSE}. Then the job is placed in the **held** state.

A batch server that accepts a Queue Job Request for a *new* job will 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.

A batch server that accepts a Queue Job Request for a *new* job will 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.

A batch server that accepts a Queue Job Request for a *new* job will 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.

3.1.3.13. 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.

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. Otherwise the job is placed in the **queued** state.

3.1.3.14. 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. Unless 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 Rerunable 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, it performs the following actions:

- Send a **SIGKILL** signal to the session (process) group of the job.
- 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.

3.1.3.15. 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.

3.1.3.16. 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 any of the following conditions are true:

- 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.

3.1.3.17. 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.
- The server does not own the designated job.
- The requested signal is not supported by the host operating system. (The killpg system call returns [EINVAL].)

When the server accepts a request to signal a job, it will send the signal requested by the client to the session (process) group of the job.

3.1.3.18. 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.

3.2. 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.

3.2.1. 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.

3.2.2. Synchronize Job Starts

PBS provides for synchronizing the initiation of jobs across hosts. This is done to support distributing processing.

> Author note:
> There are several approaches that could be taken to solve this requirement, none of them simple and straightforward. The best approach for synchronization of jobs would be a single job scheduler for all hosts on which jobs could be concurrently started. However, this approach greatly complicates the already complicated scheduling problem. Whereas the number of concurrent starts will be small compared to the total number of jobs, the semaphore approach was selected.
>
> It is the intent of the developers that PBS will be expanded to encompass the concept of a single job whose execution is distributed among multiple hosts.

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 job 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.

The server calculates the *Resource Cost* of a job by summing the product of the amount of each resource multiplied by an assigned cost of the resource. A general system surcharge may also be assigned and added to the above sum. Resources with a "size" unit are converted to megabytes before the multiplication to keep the number from becoming too large. See the server attributes resources_cost and system_cost.

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.

3.2.3. 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.

When a server queues a job with a dependency type of syncwith, after, afterok, afternotok, or afterany 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. 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.

3.3. 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.

3.3.1. 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.

The Scheduler is then treated as a privileged client and may make any request of the Server, including Run Job, Delete Job, Hold Job, or Modify Job/Queue/Server.

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`.

3.3.2. 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:

```
-W stagein=local_file@host:remote_path[,local_file@host:remote_path,...]
-W stageout=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.

3.3.3. Job Initiation

Job initiation is to place a job into execution. The server creates 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 server will search its execution path, $PATH, for the shell. If the path name of the shell is omitted or is the null string, the server uses the login shell for the user under whose name the job is to be run.

The server 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 server will place the job into **running** state.

The server 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 - the value of which is the name of the execution queue.

The server 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 server 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 server will insure that the standard output and standard error streams of the job are appended to the prior streams, if any.

If the server and host system support accounting, the server will use the value of the Account_Name job attribute as required by the host system.

If the server and host system support checkpoint, the server 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 server 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 either (1) in a temporary file in the server's spool directory, or (2) a file in the user's home directory. The choice is determined by a server build time configuration parameter.

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

If the value of the job attribute Mail_Points contains the value {beginning}, the server will send mail to each mail address specified in the job attribute Mail_Users.

3.3.4. 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 number of tries in the queue attribute number_retries. 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.

3.3.5. Job Exit

When the session leader of a batch job exits, the server 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.

3.3.6. 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.

The job is removed from the queue.

3.3.7. 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 either by the routing queue attribute route_retry_time, or the general server attribute network_retry_time.

3.3.8. Event Logging

The various daemons including the **PBS** server will maintain a log file of events. This file is available to the batch administrator for analysis of past events.

The file will be maintained under the path name {PBS_SERVER_HOME}/server_log/date, where date is the date in the form yyyymmdd when the log file started (see the -L option in pbs_server(8B)).

The events recorded by the server in the file are specified by the server attribute log_events which is a bit string with each bit determining if a type of event is logged:

1 Internal PBS errors.

2 System (OS) errors such as malloc failed.

4 Administrator related events, such as chaning server or queue attributes.

8 Job related events events: submitted, ran, deleted, ...

16 (0x010)
Job resource usage, this duplicates the accounting information in the log.

32 (0x020)
Security releated events, such as attempts to connect from an unknown host.

64 (0x040)
When the scheduler was called and why.

128 (0x080)
First level, common, debug messages.

256 (0x100)
Second level, more rare, debug messages.

The log file is a text file with each entry terminated by a new line. The format of an entry is:

```
date time;event_code;server_name;object_type;object_name;message_text
```

The date time field is a date and time stamp in the format: mm/dd/yyyy hh:mm:ss. The event_code is the type of event which triggered the event logging. It correspondings to the bit position, 0 to n, in the log_events server attribute. The server_name is the name of the server which logged the message. This is recorded in case a site wishes to merge and sort the various logs in a single file. The object_type is the type of object which the message is about, Svr for server, Que for queue, Job for job, Req for request, or Fil for file. The object_name is the name of the specific object. message_text field is the text of the log message.

3.3.9. Accounting

The PBS server maintains an accounting file. The file will be maintained under the path name {PBS_SERVER_HOME}/server_priv/accounting/day. Where day is the date in the form yyyymmdd when the accounting file started (see the -A option in pbs_server(8B)).

The account file is a text file with each entry terminated by a new line. The format of an entry is:

```
date time;record_type;job_id;message_text
```

The date time field is a date and time stamp in the format: mm/dd/yyyy hh:mm:ss. The job_id is the job identifier. The messge_text is ascii text. The content depends on the record type. The message text format is blank separated keyword=value fields. The record_type is a single character indicating the type of record. The types are:

A Job was aborted by the server.

D Job was deleted by request. The message_text will contain requestor=user@host to identify who deleted the job.

E Job ended (terminated execution). The message_text field contains:
user=username - the user name under which the job executed.
group=groupname - the group name under which the job executed.
jobname=job_name - the name of the job.
queue=queue_name - the name of the queue from which the job is executed.
ctime=time - time in seconds when job was created (first submitted).
qtime=time - time in seconds when job was queued into current queue.
etime=time - time in seconds when job became eligable to run; no holds, etc.
start=time - time in seconds when job execution started.
exec_host=host - name of host on which the job is being executed.
Resource_List.resource=limit - list of the specified reource limits.
session=sesid - session number of job.
alt_id=id - Optional alternate job identifier. Will be included only for certain systems:

> Irix 6.x with Array Services − The alternate id is the Array Session Handle (ASH) assigned to the job.

end=time - time in secnds when job ended execution.
Exit_status=value - the exit status of the job. If the value is less than 10000 (decimal) it is the exit value of the top level process of the job, typically the shell. If the value is greater than 10000, the top process exited on a signal whose number is given by subtracting 10000 from the exit value.
Resources_used.resource=limit - list of the specified reource limits.

For Resource_List and Resources_used, there is one entry per resource.

C Job was checkpointed and held.

Q Job entered a queue. The message_text contains queue=name identifying the queue into which the job was placed. There will be a new Q record each time the job is routed or moved to a new (or the same) queue.

R Job was rerun.

S Job execution started. The message_text field contains:
`user=`username - the user name under which the job executed.
`group=`groupname - the group name under which the job executed.
`jobname=`job_name - the name of the job.
`queue=`queue_name - the name of the queue from which the job is executed.
`ctime=`time - time in seconds when job was created (first submitted).
`qtime=`time - time in seconds when job was queued into current queue.
`etime=`time - time in seconds when job became eligable to run; no holds, etc.
`start=`time - time in seconds when job execution started.
`exec_host=`host - name of host on which the job is being executed.
`Resource_List.`resource=limit - list of the specified reource limits.
`session=`sesid - session number of job.

T Job was restarted from a checkpoint file.

3.4. 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.

3.4.1. Non Reservable Resources

Most Unix systems do not provide for resource reservation, only for limits. Resources, like memory, disk space, and cpu time, are handed out by the kernel on a first come first served basis. When a request exceeds the users limits, the request is denied or the job is signaled. To add resource reservation to a system is generally a major undertaking. One example is the Session Reservable File System, SRFS, extension to Unicos® developed at NAS. This extension required several additions to the kernel.

For resources which are not reservable, **PBS** manages resource allocation based on the amount "allocated" to **PBS** by the administrator. This available amount of each resource is maintained in the server attribute resources_available. The share of the resource "distributed" to each queue are maintained in an attribute for each of those objects. This attribute limits the aggregate total of the resources used by the jobs running under each object. This allocation is made by the batch system administrator. Most host systems do not provide support for dynamically adjusting the allocation to the batch system. A site may build in a procedure for adjusting the values of certain resources based on system load.

An example of non reservable resources is memory. The host operating system kernel manages memory, dynamically assigning physical memory to running processes. A limit can be set which the process cannot exceed, but memory cannot be reserved for a particular process in advance.

The resources_max attribute for the server and queue declares the maximum amount of the resource that a single job may be allocated. The server's resources_max is examined if there is not a resources_max value for the type of resource defined at the queue level.

PBS insures that the resources requested by a job fall with in the two groups of limits before the job is initiated.

3.4.2. Reservable Resources

On some hosts, certain resources types may be requested and the amount guaranteed to the process. Session Reservable File System, SRFS, is such a resource. For these types of resources, **PBS** will attempt to reserve the requested amount before scheduling the job for execution.

When the request to reserve resources is denied by the system, the job selection function may assist or *expedite* the job. In this case resources allocated to the job are not "released". **PBS**

will attempt to acquire the remaining resources until it is successful and the job can be initiated, or until a time limit, specified by the queue attribute reserved_expedite, is reached. At that point all the resources are released. If a job is being expedited, other jobs whose resources do not conflict with the needs of the expedited job may be scheduled for execution.

When the reservable resources have been allocated to the job and the non reservable resources fit into what **PBS** has available, the job will be placed into execution.

3.4.3. 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 attrbute resources_max.
4. The server's attrbute 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.

3.4.4. Types of Resources

The following table lists the names recommend for various resources. Not all types are supported on a single server, some are not yet implemented on any system. Following sub-sections will list the resources supported by each system.

Keyword	Units	Definition
cput	time	job cpu time
pcput	time	process cput time
mem	size	job memory size
pmem	size	process memory size
pf	size	Amount of file systems block for the job
ppf	size	Amount of file systems block for any process in job
file	size	Amount of space for any single file
filsys	string:size	Amount of space on a file system
fileexist	string	file exists and is readable
srfs	size	Session Reservable File System space
walltime	time	wall clock time running
memt	size*time	Maximum job memory * time (byte_seconds)
ncpus	unitary	Number of cpus
typecpu	string	type of cpu
cpugroup	string	set of cpus
9trk	unitary	number of 9 track tape drives
3480	unitary	number of 18 track tape drives
3490	unitary	number of 36 track tape drives
8mm	unitary	number of 8mm tape drives

The attribute values take the following units:

time specifies a maximum time period the resource can be used. Time is expressed in seconds as an integer, or in the form:
`[[hours:]minutes:]seconds[.milliseconds]`
If specified, milliseconds are rounded to the nearest second.

size specifies the maximum amount in terms of bytes or words. It is expressed in the form `integer[suffix]` The suffix is a multiplier defined in the following table, "b" means bytes (the default) and "w" means words. The size of a word is calculated on the execution server as its word size.

Suffix		Multiplier
b	w	1
kb	kw	1024
mb	mw	1,048,576
gb	gw	1,073,741,824
tb	tw	1,099,511,627,776

string of characters which must be interpreted by the execution server. It is frequently a path name.

unitary The maximum amount of a resource which is expressed as a simple integer.

3.4.4.1. IBM AIX Version 4 Resources

cput Maximum amount of CPU time used by all processes in the job. Units: time.

file The largest size of any single file that may be created by the job. Units: size.

mem Maximum amount of physical memory (workingset) used by the job. Units: size.

vmem Maximum amount of virtual memory used by all concurrent processes in the job. Units: size.

nice The nice value under which the job is to be run. Units: unitary.

pcput Maximum amount of CPU time used by any single process in the job. Units: time.

pmem Maximum amount of physical memory (workingset) used by any single process of the job. Units: size.

pvmem Maximum amount of virtual memory used by any single process in the job. Units: size.

walltime Maximum amount of real time during which the job can be in the running state. Units: time.

arch Specifies the administrator defined system architecture requried. This defaults to whatever the PBS_MACH string is set to in "local.mk". Units: string.

host Name of host on which job should be run. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

nodes Number and/or type of nodes to be reserved for exclusive use by the job. The value is one or more *node_specs* joined with the '+' character, "node_spec[+node_spec...]. Each node_spec is an *number* of nodes required of the type declared in the node_spec and a *name* or one or more *properity* or properities desired for the nodes. The number, the name, and each properity in the node_spec are separated by a colon ':'. If no number is specified, one (1) is assumed. Units: string.

The name of a node is its hostname. The properities of nodes are:

. ppn=# specifying the number of processors per node requested. Defaults to 1.

. arbitrary string assigned by the system administrator, please check with your administrator as to the node names and properities available to you.

Examples:

. To ask for 12 nodes of any type: -l nodes=12

. To ask for 2 "server" nodes and 14 other nodes (a total of 16): -l nodes=2:server+14
The above consist of two node_specs "2:server" and "14".

. To ask for (a) 1 node that is a "server" and has a "hippi" interface, (b) 10 nodes that are not servers, and (c) 3 nodes that have a large amount of memory an have hippi:
-l nodes=server:hippi+10:noserver+3:bigmem:hippi

. To ask for three nodes by name:
-l nodes=b2005+b1803+b1813

. To ask for 2 processors on each of four nodes:
-l nodes=4:ppn=2

. To ask for 4 processors on one node:
-l nodes=1:ppn=4

. To ask for 2 processors on each of two blue nodes and three processors on one red node:
-l nodes=2:blue:ppn=2+red:ppn=3

host Allows a user to specify the desired execution location. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

other Allows a user to specify site specific information. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

software Allows a user to specify software required by the job. This is useful if certain software packages are only available on certain systems in the site. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

EXAMPLES

qsub -l nodes=15,walltime=2:00:00 script

or in a qsub script as a directive:

#PBS -l nodes=15,walltime=2:00:00

qsub -l cput=1:00:00,walltime=2:00:00,file=50gb,mem=15mb script
qalter -lcput=30:00,pmem=8mb 123.jobid

or in a qsub script as a directive:

#PBS -l cput=1:00:00,walltime=2:00:00,file=50gb,mem=15mb

3.4.4.2. Digital Unix Resources

cput Maximum amount of CPU time used by all processes in the job. Units: time.

file The largest size of any single file that may be created by the job. Units: size.

nice The nice value under which the job is to be run. Units: unitary.

pcput Maximum amount of CPU time used by any single process in the job. Units: time.

pvmem Maximum amount of virtual memory used by any single process in the job. Units: size.

vmem Maximum amount of virtual memory used by all concurrent processes in the job. Units: size.

walltime Maximum amount of real time during which the job can be in the running state. Units: time.

arch Specifies the administrator defined system architecture requried. This defaults to whatever the PBS_MACH string is set to in "local.mk". Units: string.

host Name of host on which job should be run. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

nodes Number and/or type of nodes to be reserved for exclusive use by the job. The value is one or more *node_specs* joined with the '+' character, "node_spec[+node_spec...]. Each node_spec is an *number* of nodes required of the type declared in the node_spec and a *name* or one or more *properity* or properities desired for the nodes. The number, the name, and each property in the node_spec are separated by a colon ':'. If no number is specified, one (1) is assumed. Units: string.

The name of a node is its hostname. The properities of nodes are:

. ppn=# specifying the number of processors per node requested. Defaults to 1.

. arbitrary string assigned by the system administrator, please check with your administrator as to the node names and properities available to you.

Examples:

. To ask for 12 nodes of any type: -l nodes=12

. To ask for 2 "server" nodes and 14 other nodes (a total of 16): -l nodes=2:server+14
The above consist of two node_specs "2:server" and "14".

. To ask for (a) 1 node that is a "server" and has a "hippi" interface, (b) 10 nodes that are not servers, and (c) 3 nodes that have a large amount of memory an have hippi:
-l nodes=server:hippi+10:noserver+3:bigmem:hippi

. To ask for three nodes by name:
-l nodes=b2005+b1803+b1813

. To ask for 2 processors on each of four nodes:
-l nodes=4:ppn=2

. To ask for 4 processors on one node:
-l nodes=1:ppn=4

. To ask for 2 processors on each of two blue nodes and three processors on one red node:
-l nodes=2:blue:ppn=2+red:ppn=3

host Allows a user to specify the desired execution location. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

other Allows a user to specify site specific information. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

software Allows a user to specify software required by the job. This is useful if certain software packages are only available on certain systems in the site. This resource is provided for use by the site's scheduling policy. The allowable values and effect on

job placement is site dependent. Units: string.

EXAMPLES

qsub -l nodes=15,walltime=2:00:00 script

or in a qsub script as a directive:

#PBS -l nodes=15,walltime=2:00:00

qsub -l cput=1:00:00,walltime=2:00:00,file=50gb,mem=15mb script
qalter -lcput=30:00,pmem=8mb 123.jobid

or in a qsub script as a directive:

#PBS -l cput=1:00:00,walltime=2:00:00,file=50gb,mem=15mb

3.4.4.3. SGI Irix 5 Resources

cput Maximum amount of CPU time used by all processes in the job. Units: time.

file The largest size of any single file that may be created by the job. Units: size.

nice The nice value under which the job is to be run. Units: unitary.

pcput Maximum amount of CPU time used by any single process in the job. Units: time.

pmem Maximum amount of physical memory (workingset) used by any single process of the job. Units: size.

pvmem Maximum amount of virtual memory used by any single process in the job. Units: size.

vmem Maximum amount of virtual memory used by all concurrent processes in the job. Units: size.

walltime Maximum amount of real time during which the job can be in the running state. Units: time.

arch Specifies the administrator defined system architecture requried. This defaults to whatever the PBS_MACH string is set to in "local.mk". Units: string.

host Name of host on which job should be run. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

nodes Number and/or type of nodes to be reserved for exclusive use by the job. The value is one or more *node_specs* joined with the '+' character, "node_spec[+node_spec...]. Each node_spec is an *number* of nodes required of the type declared in the node_spec and a *name* or one or more *properity* or properities desired for the nodes. The number, the name, and each properity in the node_spec are separated by a colon ':'. If no number is specified, one (1) is assumed. Units: string.

The name of a node is its hostname. The properities of nodes are:

. ppn=# specifying the number of processors per node requested. Defaults to 1.

. arbitrary string assigned by the system administrator, please check with your administrator as to the node names and properities available to you.

Examples:

. To ask for 12 nodes of any type: -l nodes=12

. To ask for 2 "server" nodes and 14 other nodes (a total of 16): -l nodes=2:server+14
The above consist of two node_specs "2:server" and "14".

. To ask for (a) 1 node that is a "server" and has a "hippi" interface, (b) 10 nodes that are not servers, and (c) 3 nodes that have a large amount of memory an have hippi:
-l nodes=server:hippi+10:noserver+3:bigmem:hippi

. To ask for three nodes by name:
-l nodes=b2005+b1803+b1813

. To ask for 2 processors on each of four nodes:
-l nodes=4:ppn=2

. To ask for 4 processors on one node:
-l nodes=1:ppn=4

. To ask for 2 processors on each of two blue nodes and three processors on one red node:
-l nodes=2:blue:ppn=2+red:ppn=3

host Allows a user to specify the desired execution location. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

other Allows a user to specify site specific information. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

software Allows a user to specify software required by the job. This is useful if certain software packages are only available on certain systems in the site. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

EXAMPLES

qsub -l nodes=15,walltime=2:00:00 script

or in a qsub script as a directive:

#PBS -l nodes=15,walltime=2:00:00

qsub -l cput=1:00:00,walltime=2:00:00,file=50gb,mem=15mb script
qalter -lcput=30:00,pmem=8mb 123.jobid

or in a qsub script as a directive:

#PBS -l cput=1:00:00,walltime=2:00:00,file=50gb,mem=15mb

3.4.4.4. SGI Irix 6 Resources

cput Maximum amount of CPU time used by all processes in the job. Units: time.

file The largest size of any single file that may be created by the job. Units: size.

ncpus The number of processors requested. Units: unitary.

cpupercent The maximum percentage of a cpu which the job used. A value of 100 means 1 cpu. This cannot be set, it is only reported. Units: percent.

nice The nice value under which the job is to be run. Units: unitary.

nodemask A bit mask specifying the nodes (a pair of processors) to be associated with this job. This resource is intended for use by PBS to optimize processor allocation and direct use of this field by the job owner is discourged. Units: bit mask.

pcput Maximum amount of CPU time used by any single process in the job. Units: time.

pmem Maximum amount of physical memory (workingset) used by any single process of the job. Units: size.

pvmem Maximum amount of virtual memory used by any single process in the job. Units: size.

vmem Maximum amount of virtual memory used by all concurrent processes in the job. Units: size.

walltime Maximum amount of real time during which the job can be in the running state. Units: time.

arch Specifies the administrator defined system architecture requried. This defaults to whatever the PBS_MACH string is set to in "local.mk". Units: string.

host Name of host on which job should be run. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

nodes Number and/or type of nodes to be reserved for exclusive use by the job. The value is one or more *node_specs* joined with the '+' character, "node_spec[+node_spec...]. Each node_spec is an *number* of nodes required of the type declared in the node_spec and a *name* or one or more *properity* or properities desired for the nodes. The number, the name, and each properity in the node_spec are separated by a colon ':'. If no number is specified, one (1) is assumed. Units: string.

The name of a node is its hostname. The properities of nodes are:

. ppn=# specifying the number of processors per node requested. Defaults to 1.

. arbitrary string assigned by the system administrator, please check with your administrator as to the node names and properities available to you.

Examples:

. To ask for 12 nodes of any type: -l nodes=12

. To ask for 2 "server" nodes and 14 other nodes (a total of 16): -l nodes=2:server+14
The above consist of two node_specs "2:server" and "14".

. To ask for (a) 1 node that is a "server" and has a "hippi" interface, (b) 10 nodes that are not servers, and (c) 3 nodes that have a large amount of memory an have hippi:
-l nodes=server:hippi+10:noserver+3:bigmem:hippi

. To ask for three nodes by name:
-l nodes=b2005+b1803+b1813

. To ask for 2 processors on each of four nodes:
-l nodes=4:ppn=2

. To ask for 4 processors on one node:
-l nodes=1:ppn=4

. To ask for 2 processors on each of two blue nodes and three processors on one red node:
-l nodes=2:blue:ppn=2+red:ppn=3

host Allows a user to specify the desired execution location. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

other Allows a user to specify site specific information. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

software Allows a user to specify software required by the job. This is useful if certain software packages are only available on certain systems in the site. This resource is provided for use by the site's scheduling policy. The allowable values and effect on

job placement is site dependent. Units: string.

EXAMPLES

qsub -l nodes=15,walltime=2:00:00 script

or in a qsub script as a directive:

#PBS -l nodes=15,walltime=2:00:00

qsub -l cput=1:00:00,walltime=2:00:00,file=50gb,vmem=15mb script
qalter -lcput=30:00,pmem=8mb 123.jobid

or in a qsub script as a directive:

#PBS -l cput=1:00:00,walltime=2:00:00,file=50gb,mem=15mb

3.4.4.5. Linux Resources

cput Maximum amount of CPU time used by all processes in the job. Units: time.

file The largest size of any single file that may be created by the job. Units: size.

nice The nice value under which the job is to be run. Units: unitary.

pcput Maximum amount of CPU time used by any single process in the job. Units: time.

pmem Maximum amount of physical memory (workingset) used by any single process of the job. Units: size.

pvmem Maximum amount of virtual memory used by any single process in the job. Units: size.

vmem Maximum amount of virtual memory used by all concurrent processes in the job. Units: size.

walltime Maximum amount of real time during which the job can be in the running state. Units: time.

arch Specifies the administrator defined system architecture requried. This defaults to whatever the PBS_MACH string is set to in "local.mk". Units: string.

host Name of host on which job should be run. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

nodes Number and/or type of nodes to be reserved for exclusive use by the job. The value is one or more *node_specs* joined with the '+' character, "node_spec[+node_spec...]. Each node_spec is an *number* of nodes required of the type declared in the node_spec and a *name* or one or more *properity* or properities desired for the nodes. The number, the name, and each properity in the node_spec are separated by a colon ':'. If no number is specified, one (1) is assumed. Units: string.

The name of a node is its hostname. The properities of nodes are:

. ppn=# specifying the number of processors per node requested. Defaults to 1.

. arbitrary string assigned by the system administrator, please check with your administrator as to the node names and properities available to you.

Examples:

. To ask for 12 nodes of any type: -l nodes=12

. To ask for 2 "server" nodes and 14 other nodes (a total of 16): -l nodes=2:server+14
The above consist of two node_specs "2:server" and "14".

. To ask for (a) 1 node that is a "server" and has a "hippi" interface, (b) 10 nodes that are not servers, and (c) 3 nodes that have a large amount of memory an have hippi:
-l nodes=server:hippi+10:noserver+3:bigmem:hippi

. To ask for three nodes by name:
-l nodes=b2005+b1803+b1813

. To ask for 2 processors on each of four nodes:
-l nodes=4:ppn=2

. To ask for 4 processors on one node:
-l nodes=1:ppn=4

. To ask for 2 processors on each of two blue nodes and three processors on one red node:
-l nodes=2:blue:ppn=2+red:ppn=3

host Allows a user to specify the desired execution location. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

other Allows a user to specify site specific information. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

software Allows a user to specify software required by the job. This is useful if certain software packages are only available on certain systems in the site. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

EXAMPLES

qsub -l nodes=15,walltime=2:00:00 script

or in a qsub script as a directive:

#PBS -l nodes=15,walltime=2:00:00

qsub -l cput=1:00:00,walltime=2:00:00,file=50gb,mem=15mb script
qalter -lcput=30:00,pmem=8mb 123.jobid

or in a qsub script as a directive:

#PBS -l cput=1:00:00,walltime=2:00:00,file=50gb,mem=15mb

3.4.4.6. IBM SP-2 Resources

nice The nice value under which the job is to be run. Units: unitary.

walltime
Maximum amount of real time during which the job can be in the running state. Units: time.

arch Specifies the administrator defined system architecture requried. This defaults to whatever the PBS_MACH string is set to in "local.mk". Units: string.

host Name of host on which job should be run. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

nodesNumber and/or type of nodes to be reserved for exclusive use by the job. The value is one or more *node_specs* joined with the '+' character, "node_spec[+node_spec...]. Each node_spec is an *number* of nodes required of the type declared in the node_spec and a *name* or one or more *properity* or properities desired for the nodes. The number, the name, and each properity in the node_spec are separated by a colon ':'. If no number is

specified, one (1) is assumed. Units: string.

The name of a node is its hostname. The properities of nodes are:

. ppn=# specifying the number of processors per node requested. Defaults to 1.

. arbitrary string assigned by the system administrator, please check with your administrator as to the node names and properities available to you.

Examples:

. To ask for 12 nodes of any type: -l nodes=12

. To ask for 2 "server" nodes and 14 other nodes (a total of 16): -l nodes=2:server+14
The above consist of two node_specs "2:server" and "14".

. To ask for (a) 1 node that is a "server" and has a "hippi" interface, (b) 10 nodes that are not servers, and (c) 3 nodes that have a large amount of memory an have hippi:
-l nodes=server:hippi+10:noserver+3:bigmem:hippi

. To ask for three nodes by name:
-l nodes=b2005+b1803+b1813

. To ask for 2 processors on each of four nodes:
-l nodes=4:ppn=2

. To ask for 4 processors on one node:
-l nodes=1:ppn=4

. To ask for 2 processors on each of two blue nodes and three processors on one red node:
-l nodes=2:blue:ppn=2+red:ppn=3

host Allows a user to specify the desired execution location. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

other Allows a user to specify site specific information. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

software
Allows a user to specify software required by the job. This is useful if certain software packages are only available on certain systems in the site. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

EXAMPLES

qsub -l nodes=15,walltime=2:00:00 script

or in a qsub script as a directive:

#PBS -l nodes=15,walltime=2:00:00

3.4.4.7. Sun SunOS Version 4 Resources

cput Maximum amount of CPU time used by all processes in the job. Units: time.

file The largest size of any single file that may be created by the job. Units: size.

nice The nice value under which the job is to be run. Units: unitary.

pcput Maximum amount of CPU time used by any single process in the job. Units: time.

pmem Maximum amount of physical memory (workingset) used by any single process of the job. Units: size.

pvmem Maximum amount of virtual memory used by any single process in the job. Units: size.

vmem Maximum amount of virtual memory used by all concurrent processes in the job. Units: size.

walltime Maximum amount of real time during which the job can be in the running state. Units: time.

arch Specifies the administrator defined system architecture requried. This defaults to whatever the PBS_MACH string is set to in "local.mk". Units: string.

host Name of host on which job should be run. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

nodes Number and/or type of nodes to be reserved for exclusive use by the job. The value is one or more *node_specs* joined with the '+' character, "node_spec[+node_spec...]. Each node_spec is an *number* of nodes required of the type declared in the node_spec and a *name* or one or more *properity* or properities desired for the nodes. The number, the name, and each properity in the node_spec are separated by a colon ':'. If no number is specified, one (1) is assumed. Units: string.

The name of a node is its hostname. The properities of nodes are:

. ppn=# specifying the number of processors per node requested. Defaults to 1.

. arbitrary string assigned by the system administrator, please check with your administrator as to the node names and properities available to you.

Examples:

. To ask for 12 nodes of any type: -l nodes=12

. To ask for 2 "server" nodes and 14 other nodes (a total of 16): -l nodes=2:server+14
The above consist of two node_specs "2:server" and "14".

. To ask for (a) 1 node that is a "server" and has a "hippi" interface, (b) 10 nodes that are not servers, and (c) 3 nodes that have a large amount of memory an have hippi:
-l nodes=server:hippi+10:noserver+3:bigmem:hippi

. To ask for three nodes by name:
-l nodes=b2005+b1803+b1813

. To ask for 2 processors on each of four nodes:
-l nodes=4:ppn=2

. To ask for 4 processors on one node:
-l nodes=1:ppn=4

. To ask for 2 processors on each of two blue nodes and three processors on one red node:
-l nodes=2:blue:ppn=2+red:ppn=3

host Allows a user to specify the desired execution location. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

other Allows a user to specify site specific information. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

software Allows a user to specify software required by the job. This is useful if certain software packages are only available on certain systems in the site. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

EXAMPLES

qsub -l nodes=15,walltime=2:00:00 script

or in a qsub script as a directive:

#PBS -l nodes=15,walltime=2:00:00

qsub -l cput=1:00:00,walltime=2:00:00,file=50gb,mem=15mb script
qalter -lcput=30:00,pmem=8mb 123.jobid

or in a qsub script as a directive:

#PBS -l cput=1:00:00,walltime=2:00:00,file=50gb,mem=15

3.4.4.8. Sun Solaris Resources

cput Maximum amount of CPU time used by all processes in the job. Units: time.

file The largest size of any single file that may be created by the job. Units: size.

nice The nice value under which the job is to be run. Units: unitary.

pcput Maximum amount of CPU time used by any single process in the job. Units: time.

pmem Maximum amount of physical memory (workingset) used by any single process of the job. Units: size.

pvmem Maximum amount of virtual memory used by any single process in the job. Units: size.

vmem Maximum amount of virtual memory used by all concurrent processes in the job. Units: size.

walltime Maximum amount of real time during which the job can be in the running state. Units: time.

arch Specifies the administrator defined system architecture requried. This defaults to whatever the PBS_MACH string is set to in "local.mk". Units: string.

host Name of host on which job should be run. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

nodes Number and/or type of nodes to be reserved for exclusive use by the job. The value is one or more *node_specs* joined with the '+' character, "node_spec[+node_spec...]. Each node_spec is an *number* of nodes required of the type declared in the node_spec and a *name* or one or more *properity* or properities desired for the nodes. The number, the name, and each properity in the node_spec are separated by a colon ':'. If no number is specified, one (1) is assumed. Units: string.

The name of a node is its hostname. The properities of nodes are:

. ppn=# specifying the number of processors per node requested. Defaults to 1.

. arbitrary string assigned by the system administrator, please check with your administrator as to the node names and properities available to you.

Examples:

. To ask for 12 nodes of any type: -l nodes=12

. To ask for 2 "server" nodes and 14 other nodes (a total of 16): -l nodes=2:server+14
The above consist of two node_specs "2:server" and "14".

. To ask for (a) 1 node that is a "server" and has a "hippi" interface, (b) 10 nodes that are not servers, and (c) 3 nodes that have a large amount of memory an have hippi:
-l nodes=server:hippi+10:noserver+3:bigmem:hippi

. To ask for three nodes by name:
-l nodes=b2005+b1803+b1813

. To ask for 2 processors on each of four nodes:
-l nodes=4:ppn=2

. To ask for 4 processors on one node:
-l nodes=1:ppn=4

. To ask for 2 processors on each of two blue nodes and three processors on one red node:
-l nodes=2:blue:ppn=2+red:ppn=3

host Allows a user to specify the desired execution location. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

other Allows a user to specify site specific information. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

software Allows a user to specify software required by the job. This is useful if certain software packages are only available on certain systems in the site. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

EXAMPLES

qsub -l nodes=15,walltime=2:00:00 script

or in a qsub script as a directive:

#PBS -l nodes=15,walltime=2:00:00

qsub -l cput=1:00:00,walltime=2:00:00,file=50gb,vmem=15mb script
qalter -lcput=30:00 123.jobid

or in a qsub script as a directive:

#PBS -l cput=1:00:00,walltime=2:00:00,file=50gb,vmem=15mb

3.4.4.9. CRI Computers running Unicos version 8, 9, or 10

ncpus Maximum number of multitask processes in the job. If this is set the environment variable NCPUS will be set to the given value. Units: unitary.

cput Maximum amount of CPU time used by all processes in the job. Root processes are not limited, see limit(2). Units: time.

mem Maximum amount of physical memory used by all concurrent processes in the job. Units: size.

mppe The number of processing elements used by a single process in the job. Units: unitary.

mppt Maximum amount of wall clock time used on the MPP in the job. Units: time.

mta, mtb, mtc, ..., mth
Maximum number of magnetic tape drives required in the corresponding device class of a, b, c, ..., h. Units: unitary.

nice The nice value under which the job is to be run. Units: unitary.

pncpus Maximum number of processors used by any single process in the job. Units: unitary.

pcput Maximum amount of CPU time used by any single process in the job. Units: time.

pf Maximum number of file system blocks that can be used by all process in the job. Units: size.

pmem Maximum amount of physical memory used by any single process in the job. Units: size.

pmppt Maximum amount of wall clock time used on the MPP by a single process in the job. Units: time.

ppf Maximum number of file system blocks that can be used by a single process in the job. Units: size.

procs Maximum number of processes in the job. Units: unitary.

psds Maximum number of data blocks on the SDS (secondary data storage) for any process in the job.

sds Maximum number of data blocks on the SDS (secondary data storage) for the job.

srfs_tmp Session Reservable File System (SRFS) space in TMPDIR. Note, SRFS is not supported by Cray. Units: size.

srfs_wrk SRFS space in WRKDIR. Units: size.

srfs_big SRFS space in BIGDIR. Units: size.

srfs_fast SRFS space in FASTDIR. Units: size.

walltime Maximum amount of real time during which the job can be in the running state. Units: time.

arch Specifies the administrator defined system architecture requried. This defaults to whatever the PBS_MACH string is set to in "local.mk". Units: string.

host Name of host on which job should be run. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

nodes Number and/or type of nodes to be reserved for exclusive use by the job. The value is one or more *node_specs* joined with the '+' character, "node_spec[+node_spec...]. Each node_spec is an *number* of nodes required of the type declared in the node_spec and a *name* or one or more *properity* or properities desired for the nodes. The number, the name, and each properity in the node_spec are separated by a colon ':'. If no number is specified, one (1) is assumed. Units: string.

The name of a node is its hostname. The properities of nodes are:

. ppn=# specifying the number of processors per node requested. Defaults to 1.

. arbitrary string assigned by the system administrator, please check with your administrator as to the node names and properities available to you.

Examples:

. To ask for 12 nodes of any type: -l nodes=12

. To ask for 2 "server" nodes and 14 other nodes (a total of 16): -l nodes=2:server+14
The above consist of two node_specs "2:server" and "14".

. To ask for (a) 1 node that is a "server" and has a "hippi" interface, (b) 10 nodes that are not servers, and (c) 3 nodes that have a large amount of memory an have hippi:
-l nodes=server:hippi+10:noserver+3:bigmem:hippi

. To ask for three nodes by name:
-l nodes=b2005+b1803+b1813

. To ask for 2 processors on each of four nodes:
-l nodes=4:ppn=2

. To ask for 4 processors on one node:
-l nodes=1:ppn=4

. To ask for 2 processors on each of two blue nodes and three processors on one red node:
-l nodes=2:blue:ppn=2+red:ppn=3

host Allows a user to specify the desired execution location. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

other Allows a user to specify site specific information. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

software Allows a user to specify software required by the job. This is useful if certain software packages are only available on certain systems in the site. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

EXAMPLES

qsub -l nodes=15,walltime=2:00:00 script

or in a qsub script as a directive:

#PBS -l nodes=15,walltime=2:00:00

qsub -l cput=1:00:00,pcput=20:00,file=50gb,mem=15mb,mta=2 script
qalter -lcput=30:00,pmem=8mb 123.jobid
qsub -lsrfs_tmp=50mb,srfs_big=2gw script

or in a qsub script as directives:

#PBS -l cput=1:00:00,pcput=20:00,file=50gb,mem=15mb,mta=2
#PBS -lsrfs_tmp=50mb,srfs_big=2gw

SPECIAL FEATURES

Cray Unicos supports checkpoint and restart. Any description of checkpoint features in PBS are applicable to Unicos.

3.4.4.10. CRI Computers running Unicos MK version 2.

cput Maximum amount of CPU time used by all processes in the job. Root processes are not limited, see limit(2). Units: time.

mem Maximum amount of physical memory used by all concurrent processes in the job. Units: size.

mppe The number of processing elements used by a single process in the job. Units: unitary.

mppt Maximum amount of wall clock time used on the MPP in the job. Units: time.

mta, mtb, mtc, ..., mth
Maximum number of magnetic tape drives required in the corresponding device class of a, b, c, ..., h. Units: unitary.

nice The nice value under which the job is to be run. Units: unitary.

pncpus
Maximum number of processors used by any single process in the job. Units: unitary.

pcput Maximum amount of CPU time used by any single process in the job. Units: time.

pf Maximum number of file system blocks that can be used by all process in the job. Units: size.

pmem Maximum amount of physical memory used by any single process in the job. Units: size.

pmppt Maximum amount of wall clock time used on the MPP by a single process in the job. Units: time.

ppf Maximum number of file system blocks that can be used by a single process in the job. Units: size.

procs Maximum number of processes in the job. Units: unitary.

psds Maximum number of data blocks on the SDS (secondary data storage) for any process in the job.

sds Maximum number of data blocks on the SDS (secondary data storage) for the job.

srfs_tmp
Session Reservable File System (SRFS) space in TMPDIR. Note, SRFS is not supported by Cray. Units: size.

srfs_wrk
SRFS space in WRKDIR. Units: size.

srfs_big
SRFS space in BIGDIR. Units: size.

srfs_fast
SRFS space in FASTDIR. Units: size.

walltime
Maximum amount of real time during which the job can be in the running state. Units: time.

arch Specifies the administrator defined system architecture requried. This defaults to whatever the PBS_MACH string is set to in "local.mk". Units: string.

host Name of host on which job should be run. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

nodes Number and/or type of nodes to be reserved for exclusive use by the job. The value is one or more *node_specs* joined with the '+' character, "node_spec[+node_spec...]. Each node_spec is an *number* of nodes required of the type declared in the node_spec and a *name* or one or more *properity* or properities desired for the nodes. The number, the name, and each properity in the node_spec are separated by a colon ':'. If no number is specified, one (1) is assumed. Units: string.

The name of a node is its hostname. The properities of nodes are:

. ppn=# specifying the number of processors per node requested. Defaults to 1.

. arbitrary string assigned by the system administrator, please check with your administrator as to the node names and properities available to you.

Examples:

. To ask for 12 nodes of any type: -l nodes=12

. To ask for 2 "server" nodes and 14 other nodes (a total of 16): -l nodes=2:server+14
The above consist of two node_specs "2:server" and "14".

. To ask for (a) 1 node that is a "server" and has a "hippi" interface, (b) 10 nodes that are not servers, and (c) 3 nodes that have a large amount of memory an have hippi:
-l nodes=server:hippi+10:noserver+3:bigmem:hippi

. To ask for three nodes by name:
-l nodes=b2005+b1803+b1813

. To ask for 2 processors on each of four nodes:
-l nodes=4:ppn=2

. To ask for 4 processors on one node:
-l nodes=1:ppn=4

. To ask for 2 processors on each of two blue nodes and three processors on one red node:
-l nodes=2:blue:ppn=2+red:ppn=3

host Allows a user to specify the desired execution location. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

other Allows a user to specify site specific information. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

software
Allows a user to specify software required by the job. This is useful if certain software packages are only available on certain systems in the site. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. Units: string.

EXAMPLES

qsub -l nodes=15,walltime=2:00:00 script

or in a qsub script as a directive:

#PBS -l nodes=15,walltime=2:00:00

qsub -l cput=1:00:00,pcput=20:00,file=50gb,mem=15mb,mta=2 script
qalter -lcput=30:00,pmem=8mb 123.jobid
qsub -lsrfs_tmp=50mb,srfs_big=2gw script

or in a qsub script as directives:

#PBS -l cput=1:00:00,pcput=20:00,file=50gb,mem=15mb,mta=2
#PBS -lsrfs_tmp=50mb,srfs_big=2gw

SPECIAL FEATURES

Cray Unicos MK supports checkpoint and restart. Any description of checkpoint features in PBS are applicable to Unicos MK.

3.4.5. Interactive Session Management

Author note:
This section is very incomplete. It is the beginnings of an idea based on the need to know, and perhaps control, the resource utilization by interactive sessions. In most implementations based on NQS, this is not done. A few implementations have been extended to periodically monitor the proc table in the kernel. The disadvantage of this method is the makeup of the proc table varies greatly for each kernel implementation.

To improve its ability to schedule jobs and manage resources, PBS must be aware of the load of the system produced by interactive jobs. It would be an advantage to have the capability to control the activities of interactive sessions.

One approach is to provide a communication capability between the login process and **PBS.** The number of login sessions and the amount of resources "assigned" to each session, based on the user limits, would be communicated to PBS. PBS would then be able to adjust the amount of resources available to batch jobs.

If a site wished to restrict interactive sessions based on the availability of resources under control of PBS, this capability would be extended such that PBS could direct the login process to disallow the user login attempt.

The sum of the resources (limits) used by all current interactive sessions would be treated as those assigned to a job.

[Page intentionally left bank.]

4. Interface Library

Part of the **PBS** package 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 have requirements to build jobs 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 which corresponds 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 *Interface Facility*, IFF, as shown in figure 4−1. The purpose of the IFF is to provide the user a credential which validates the user's identity. This credential is included in each batch request. The IFF provided credential prevents a user from spoofing another user's identity. See chapter 10, Security, for more detail.

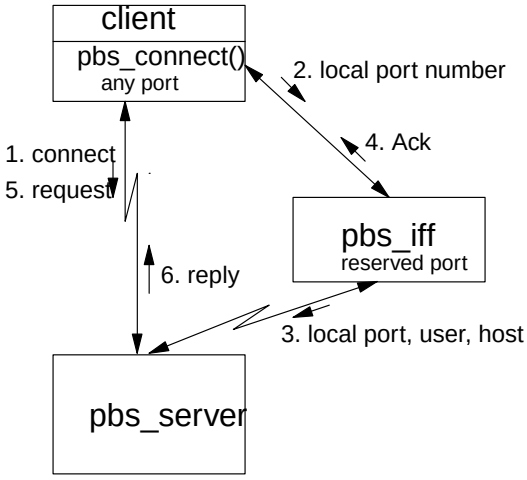


Figure 4−1: Interface Between Client, IFF, and Server

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 insure the parameters are in the 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. This request is then passed to the server communication routine.

4.2. Interface Library Routines

The following "man" pages describe the user callable functions in the IFL. The functions are found in files src/lib/Libifl/pbsD_*.c.

4.2.1. Alter Job

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. It is not currently used by this function.

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.

4.2.2. Connect with Server

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 in section 2.7.4.

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.

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.

4.2.3. Delete Job

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 two purposes. 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 a 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 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.

4.2.4. Disconnect from Server

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.

4.2.5. Get Error Message Text

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.

Because different server implementations may have different error conditions, the batch request - reply protocol allows for the server to include the text of an error message in the reply to a batch 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.

4.2.6. Hold Job

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. It is not currently used by this function.

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.

4.2.7. Locate Job

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.

4.2.8. Manager Server

NAME

pbs_manager - administrator 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
```

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 full range of batch_op values is `{ SET, UNSET, INCR, DECR, EQ, NE, GE, GT, LE, LT }` Only the SET, UNSET, INCR, and DECR are allowed in a manager call, others will be rejected by the server.

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

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

SEE ALSO

qmgr(8B) and pbs_connect(3B)

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.

4.2.9. Move Job

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. It is not currently used by this function.

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.

4.2.10. Message Job

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. It is not currently used by this function.

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.

4.2.11. Order Job

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 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. It is not currently used by this function.

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.

4.2.12. Rerun Job

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. It is not currently used by this function.

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.

4.2.13. Release Job

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. It is not currently used by this function.

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.

4.2.14. Run Job

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. If the server does not understand the location, it will reject the request and return an error.

The argument, *extend*, is reserved for implementation defined extensions. It is not currently used by these functions.

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.

4.2.15. Select Job

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 "Rriority"
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.

The full range of batch_op values is `{ SET, UNSET, INCR, DECR, EQ, NE, GE, GT, LE, LT }`. Only the relational operators are allowed in a select call, and others will be rejected by the server.

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 space for the array and the job identifier strings is malloc-ed by **pbs_selectjob**() as one allocation. The array of pointers starts at the address returned by malloc(), so free() will work.

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

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.

4.2.16. Status of Selected Jobs

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, 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. The full range of batch_op values is SET, UNSET, INCR, DECR, EQ, NE, GE, GT, LE, and LT, Only the relational operators are allowed in a selstat call, and others will be rejected by the server.

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

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

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.

4.2.17. Signal Job

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`

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.

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

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.

4.2.18. Stage In Files

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 the cluster managed by the server. If the server does not understand the location, it will reject the request and return an error. 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. It is not currently used by this function.

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.

4.2.19. Status Job

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 the string starts with a number, it is a job identifier. If it starts with a alphabetic character, it is 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 aways be a pointer to a null string.

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

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.

4.2.20. Status Queue

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 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 aways be a pointer to a null string.

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

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.

4.2.21. Status Server

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 aways be a pointer to a null string.

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

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 struc-

ture. Otherwise, a null pointer is returned and the error code is set in pbs_errno.

4.2.22. Submit Job

NAME

pbs_submit - submit a pbs batch job

SYNOPSIS

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

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. It is not currently used by this function.

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.

4.2.23. Terminate Server

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, runnings 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.

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

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

This call requires PBS Operator or Manager privilege.

SEE ALSO

qterm(8B) and pbs_connect(3B)

DIAGNOSTICS

When the batch request generated by **pbs_terminate**() 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.

4.2.24. Query Resource Availability

While the following functions may be used by any user, they are intended for use in a batch scheduler. They replace avail, totpool, and usepool calls made to pbs_mom in early versions of PBS.

NAME

pbs_rescquery, avail, totpool, usepool - query resource availability

SYNOPSIS

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

int pbs_rescquery(int connect, char **resourcelist, int *arraysize,
int *available, int *allocated, int *reserved, int *down )

char *avail(int connect, char *resc)

int totpool(int connect, int update)

int usepool(int connect, int update)
```

DESCRIPTION

pbs_rescquery

Issue a request to the batch server to query the availability of 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. *available*, *allocated*, *reserved*, and *down* are integer arrays of size arraysize. The amount of resource specified in the corresponding resourcelist string which is available, already allocated, reserved, and down/off-line is returned in the integer arrays.

At the present time the only resources which may be specified is "nodes". It may be specified as

```
nodes
nodes=
nodes=specification
```

where specification is what a user specifies in the -l option arguement list for nodes, see qsub(1B) and the various pbs_resource_* man pages.

Where the node resourcelist is a simple type, such as "nodes", "nodes=", or "nodes=*type*", the numbers returned reflect the actual number of nodes (of the specified type) which are *available*, *allocated*, *reserved*, or *down*.

For a more complex node resourcelist, such as "nodes=2" or "nodes=type1:type2", only the value returned in *available* has meaning. If the number in *available* is positive, it is the number of nodes requried to satisified the specification and that some set of nodes are available which will satisify it, see *avail*(). If the number in *available* is zero, some number of nodes requried to satisified the specification are currently unavailable, the request might be satisifed at a later time. If the number in *available* is negative, no combination of known nodes can satisified the specification.

avail

The *avail*() call is provided as conversion aid for scheduler written for early versions of PBS. The avail() routine uses pbs_rescquery() and returns a character string answer. *connect* is the connection returned by **pbs_connect**(). *resc* is a sin-

gle *node=specification* specification as discussed above. If the nodes to satisify the specification are currently available, the return value is the character string **yes**. If the nodes are currently unavailable, the return is the character string **no**. If the specification could never be satified, the return is the string **never**. An error in the specification returns the character string **?**.

totpool
The *totpool*() function returns the total number of nodes know to the PBS server. This is the sum of the number of nodes available, allocated, reserved, and down. The parameter *connection* is the connection returned by pbs_connect(). The parameter *update* if non-zero, causes totpool() to issue a pbs_rescquery() call to obtain fresh information. If zero, numbers from the prior pbs_rescquery() are used.

usepool
usepool() returns the number of nodes currently in use, the sum of allocated, reserved, and down. The parameter *connection* is the connection returned by pbs_connect(). The parameter *update* if non-zero, causes totpool() to issue a pbs_rescquery() call to obtain fresh information. If zero, numbers from the prior pbs_rescquery() are used.

SEE ALSO

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

DIAGNOSTICS

When the batch request generated by the **pbs_rescquery**() 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.

The functions usepool() and totpool() return -1 on error.

4.2.25. Make and Release Resource Reservations

The following functions require manager or operator privilege. They are intended for use in a batch scheduler.

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), pbs_rescquery(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.

4.2.26. Node Status

The following function provides a means to query the status of PBS nodes. A node has a name and two attributes. ATTR_NODE_state (state) may take the values of `free` or one or more of the following "or-ed" together:

`free` The node is not being used by PBS and isfree to be allocated to a job.

`down` The node is not responding to queries by the PBS job server.

`offline`
The node has been marked off-line by the administrator

`reserve`
The node has been reserved by the PBS job scheduler.

`job-exclusive`
The node has been assigned exclusively to a job, no other jobs can run on that node.

`job-sharing`
The node has been assigned to one or more jobs. Other jobs willing to share nodes may be allocated to it.

The attribute ATTR_NODE_properties (properties) is a list of alternative names which may be used to identify sets or groups of nodes by some arbitrary criteria.

NAME

pbs_statnode - obtain status of pbs nodes

SYNOPSIS

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

struct batch_status *pbs_statnode(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 PBS node or nodes.

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 node or the null string. If *id* specifies a node name, the status of that node will be returned. If the *id* is a null string (or null pointer), the status of all nodes 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 only supported attribute names relating to nodes are "state" and "properties". The `resource` member is not used and must be a pointer to a null string. The `value` member should aways be a pointer to a null string.

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

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_statnode**() 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.

[Page intentionally left bank.]

5. User Commands

This section describes the commands available to the general user. Unless otherwise noted, the command must conform to the POSIX 1003.2d specification of the command as to syntax and functionality.

5.1. General Specifications of User Commands

The following specifications apply to all user commands.

5.1.1. Error Checking

All user commands will validate their invocation for the correct syntax. Any syntax error or invalid option will terminate the command with an error message on standard error and an exit status greater than zero.

5.1.2. Directing Requests to Correct Server

A command performs its function by sending the corresponding request for service to a batch server. The choice of batch servers to which to send the request is governed by the following ordered set of rules:

1. For those commands which require or accept a job identifier operand, if the server is specified in the job identifier operand as `@server`, then the batch requests will be sent to the server named by `server`.
2. For those commands which require or accept a job identifier operand and the `@server` is not specified, then the command will attempt to determine the current location of the job by sending a *Locate Job* batch request to the server which created the job.
3. If a server component of a destination is supplied via the -q option, such as on **qsub** and **qselect**, but not **qalter**, then the server request is sent to that server.
4. The server request is sent to the server identified as the default server, see section 2.7.4.

5.1.3. Operands

Unless noted, when more than one operand is specified on the command line, the command processes each operand in turn. An error reply from a server on one operand will be noted in the standard error stream. The command continues processing the other operands. If an error reply was received for any operand, the final exit status for the command will be greater than zero.

Generally, the operands to commands will be job identifiers as described in section 2.7.6 or destination identifiers described in section 2.7.3.

5.2. General User Commands

5.2.1. Alter Job

NAME

qalter − alter pbs batch 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] [−r c] [-S path] [−u user_list] [−W additional_attributes] job_identifier...

DESCRIPTION

The **qalter** command modifies the attributes of the job or jobs specified by *job_identifier* on the command line. Only those attributes listed as options on the command will be modified. If any of the specified attributes cannot be modified for a job for any reason, none of that job's attributes will be modified.

The qalter command accomplishes the modifications by sending a *Modify Job* batch request to the batch server which owns each job.

OPTIONS

−a date_time

Replaces the attribute Execution_Time time at which the job becomes eligible for execution. The *date_time* argument syntax is: `[[[[CC]YY]MM]DD]hhmm[.SS]`.

If the month, `MM`, is not specified, it will default to the current month if the specified day `DD`, is in the future. Otherwise, the month will be set to next month. Likewise, 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. This *date_time* will be converted to the integer number of seconds since Epoch that is equivalent to the local time on the system where the command is being executed.

This attribute can be altered once the job has begun execution, but it will not take affect until the job is rerun.

−A account_string

Replaces the the Account_Name attribute, the account string associated with the job. The syntax of the *account_string* is defined in section 2.7.1. It is interpreted by the server which executes the job.

This attribute cannot be altered once the job has begun execution.

−c interval

Replaces the Checkpoint attribute, the interval at which the job will be checkpointed. If the job executes upon a host which does not support checkpoint, this option will be ignored.

The *interval* argument is specified as:

n No checkpointing is to be performed. The job's Checkpoint attribute is set to the string `"n"`.

s Checkpointing is to be performed only when the server executing the job is shutdown. The job's Checkpoint attribute is set to the string `"s"`.

c Checkpointing is to be performed at the default minimum cpu time for the queue from which the job is executing. The job's Checkpoint attribute is set to the string `"c"`.

c=minutes

Checkpointing is to be performed at an interval of *minutes*, which is the in-

teger number of minutes of CPU time used by the job. This value must be greater than zero. If the number is less than the default checkpoint time, the default time will be used. The Checkpoint attributes is set to the string specified by *"c=minutes"*.

This attribute can be altered once the job has begun execution, but the new value does not take affect until the job is rerun.

−e path Replaces the Error_Path attribute, the path to be used for the standard error stream of the batch job. The *path* argument is of the form:

```
[hostname:]path_name
```

where `hostname` is the name of a host to which the file will be returned and `path_name` is the path name on that host in the syntax recognized by POSIX 1003.1. The argument will be interpreted as follows:

`path_name`
Where path_name is not an absolute path name, then the qalter command will expand the path name relative to the current working directory of the command. The command will supply the name of the host upon which it is executing for the *hostname* component.

`hostname:path_name`
Where path_name is not an absolute path name, then the qalter command will not expand the path name. The execution server will expand it relative to the home directory of the user on the system specified by hostname.

`path_name`
Where path_name specifies an absolute path name, then qalter will supply the name of the host on which it is executing for the *hostname*.

`hostname:path_name`
Where path_name specifies an absolute path name, the path will be used as specified.

This attribute can be altered once the job has begun execution, but it will not take affect until the job is rerun.

−h hold_list

Updates the Hold_Types attribute, the types of holds on the job. The *hold_list* argument is a string of one or more of the following characters:

u Add the USER type hold.

s Add the SYSTEM type hold if the user has the appropriate level of privilege. [Typically reservered to the batch administrator.]

o Add the OTHER (or OPERATOR) type hold if the user has the appropriate level of privilege. [Typically reservered to the batch administrator and batch operator.]

n Set to none; that is clear the hold types which could be applied with the users level of privilege.

Repetition of characters is permitted, but "n" may not appear in the same option argument with the other three characters. This attribute can be altered once the job has begun execution, but the hold will not take affect until the job is rerun.

−j join Declares which standard streams of the job will be merged together. The *join* argument value may be the characters "oe" and "eo", or the single character "n".

A argument value of `oe` directs that the standard output and standard error streams of the job will be merged, intermixed, and returned as the standard

output. A argument value of eo directs that the standard output and standard error streams of the job will be merged, intermixed, and returned as the standard error. The Join_Path job attribute is set to the value.

A value of n directs that the two streams will be two separate files. The Join_Path attribute is set to "n". This attribute can be altered once the job has begun execution, but it will not take affect until the job is rerun.

−k keep Defines which if either of standard output or standard error of the job will be retained on the execution host. If set for a stream, this option overrides the path name for that stream.

The argument is either the single letter "e", "o", or "n", or one or more of the letters "e" and "o" combined in either order.

n No streams are to be retained. The Keep_Files attribute is set to KEEP_NONE, "n".

e The standard error stream is to retained on the execution host. The stream will be placed in the home directory of the user under whose user id the job executed. The file name will be the default file name given by: job_name.esequence where job_name is the name specified for the job, and sequence is the sequence number component of the job identifier. The attribute is set to include KEEP_ERROR, "e".

o The standard output stream is to be retained on the execution host. The stream will be placed in the home directory of the user under whose user id the job executed. The file name will be the default file name given by: job_name.osequence where job_name is the name specified for the job, and sequence is the sequence number component of the job identifier. The Output_Path attribute is set to include KEEP_OUTPUT, "o".

eo Both the standard output and standard error streams will be retained. The attribute is set to KEEP_OUTPUT | KEEP_ERROR.

oe Both the standard output and standard error streams will be retained. The attribute is set to KEEP_OUTPUT | KEEP_ERROR.

Repetition of characters is permitted, but "n" may not appear in the same option argument with the other two characters. This attribute cannot be altered once the job has begun execution.

−l resource_list

Modifies the Resource_List attribute, the list of resources that are required by the job. The *Resource_List* argument is in the following syntax:
resource_name[=[value]][,resource_name[=[value]],...]

For each resource listed, if a resource with the specified name already exist in the jobs resource attribute, the value for that resource will be updated. If the named resource does not exist in the job resource attribute, the resource name and value will be added. No white space is allowed in the value.

Because the list of supported resources vary from host to host, the command will perform no validation of the name or value.

If a requested modification to a resource would exceed the resource limits for jobs in the current queue, the server will reject the request.

If the job is running, only certain, resources can be altered. Which resources can be altered in the run state is system dependent. A user may only lower the limit for those resources. A PBS Manager or Operator may increase them.

−m mail_options

Replaces the set of conditions under which the execution server will send a mail message about the job. The *mail_options* argument is a string which con-

sists of one or more repetitions of the single character "n", or one or more repetitions of the characters "a", "b", and "e".

If the character "n" is specified, no mail will be sent. The Mail_Points attribute is set to NONE, "n".

For the letters "a", "b", and "e":

a mail is sent when the job is aborted by the batch system. The Mail_Points attribute is set to ABORT, "a".

b mail is sent when the job begins execution. The Mail_Points attribute is set to BEGINNING, "b".

e mail is sent when the job terminates. The Mail_Points attribute is set to EXIT, "e".

−M user_list

Replaces the list of users to whom mail is sent by the execution server when it sends mail about the job.

The *user_list* argument is of the form:

```
user[@host][,user[@host],...]
```

The Mail_Users attribute is set to the argument.

−N nameRenames the job. The name specified may be up to and including 15 characters in length. It must consist of printable, non white space characters with the first character alphabetic. [See the discussion of the -N option under qsub(1).] The Job_Name attribute is reset to the name value.

−o path Replaces the path to be used for the standard output stream of the batch job. The *path* argument is of the form:

```
[hostname:]path_name
```

where `hostname` is the name of a host to which the file will be returned and `path_name` is the path name on that host in the syntax recognized by POSIX. The argument will be interpreted as follows:

`path_name`

Where path_name is not an absolute path name, then the qalter command will expand the path name relative to the current working directory of the command. The command will supply the name of the host upon which it is executing for the *hostname* component.

`hostname:path_name`

Where path_name is not an absolute path name, then the qalter command will not expand the path name. The execution server will expand it relative to the home directory of the user on the system specified by hostname.

`path_name`

Where path_name specifies an absolute path name, then the qalter will supply the name of the host on which it is executing for the *hostname*.

`hostname:path_name`

Where path_name specifies an absolute path name, the path will be used as specified.

This attribute can be altered once the job has begun execution, but it will not take affect until the job is rerun.

−p priority

Replaces the priority of the job. The *priority* argument must be a integer between -1024 and +1023 inclusive. The Priority attribute is set to this signed integer value.

This attribute can be altered once the job has begun execution, but it will not take affect until the job is rerun.

−r c Declares whether the job is rerunable. See the **qrerun** command. The option argument *c* is a single character. PBS recognizes the following characters: y and n. Also see *rerunability* in the glossary.

If the argument is "y", the job is marked rerunable. The Rerunable attribute is set to 'y'. If the argument is "n", the job is marked as not rerunable. The Rerunable attribute is set to 'n'.

−S path Declares the shell that interprets the job script.

The option argument *path_list* is in the form:

```
path[@host][,path[@host],...]
```

Only one path may be specified for any host named. Only one path may be specified without the corresponding host name. The path selected will be the one with the host name that matched the name of the execution host. If no matching host is found, then if present the path specified without a host will be selected.

If the −*S* option is not specified, the option argument is the null string, or no entry from the *path_list* is selected, the execution will use the login shell of the user on the execution host. The Shell_Path_List attribute is set to the *path_list* argument if present, otherwise it is set to the null string.

This attribute can be altered once the job has begun execution, but it will not take affect until the job is rerun.

−u user_list

Replaces the user name under which the job is to run on the execution system.

The *user_list* argument is of the form:

```
user[@host][,user[@host],...]
```

Only one user name may be given for per specified host. Only one of the user specifications may be supplied without the corresponding host specification. That user name will be used for execution on any host not named in the argument list. The User_List attribute is set to the value of *User_List*.

This attribute cannot be altered once the job has begun execution.

-W additional_attributes

The -W option allows for the modification of additional job attributes. The general syntax of the -W is in the form:

```
-W attr_name=value[,attr_name=value...]
```

Note if white space occurs anywhere within the option argument string or the equal sign, "=", occurs within an *attribute_value* string, then the string must be enclosed with either single or double quote marks.

PBS currently supports the following attributes within the -W option.

depend=dependency_list

Redefines the depend attribute listing the dependencies between this and other jobs. The *dependency_list* is in the form: type[:argument[:argument...][,type:argument...].

The *argument* is either a numeric count or a PBS job id according to *type*. If argument is a count, it must be greater than 0. If it is a job id and is not fully specified in the form: seq_number.server.name, it will be expanded according to the default server rules. If *argument* is null (the preceeding colon need not be specified), the dependency of the cooresponding type is cleared (unset).

synccount:*count*

This job is the first in a set of jobs to be executed at the same time. *Count* is the number of additional jobs in the set.

`syncwith:`*jobid*
This job is an additional member of a set of jobs to be executed at the same time. *Jobid* is the job identifier of the first job in the set.

`after:`*jobid[:jobid...]*
This job may be scheduled for execution at any point after jobs *jobid* have started execution.

`afterok:`*jobid[:jobid...]*
This job may be scheduled for execution only after jobs *jobid* have terminated with no errors.

`afternotok:`*jobid[:jobid...]*
This job may be scheduled for execution only after jobs *jobid* have terminated with errors.

`afterany:`*jobid[:jobid...]*
This job may be scheduled for execution after jobs *jobid* have terminated, with or without errors.

`on:`*count*
This job may be scheduled for execution after *count* dependencies on other jobs have been satisfied. This form is used in conjunction with one of the `before` forms, see below.

`before:`*jobid[:jobid...]*
When this job has begun execution, then jobs *jobid...* may begin.

`beforeok:`*jobid[:jobid...]*
If this job terminates execution without errors, then jobs *jobid...* may begin.

`beforenotok:`*jobid[:jobid...]*
If this job terminates execution with errors, then jobs *jobid...* may begin.

`beforeany:`*jobid[:jobid...]*
When this job terminates execution, jobs *jobid...* may begin.

If any of the `before` forms are used, the job referenced by *jobid* must have been submitted with a dependency type of `on`.

The job specified in any of the `before` forms must have the same owner as the job being altered.. Otherwise, the dependency will not take effect.

Error processing of the existance, 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. These options are extensions to the POSIX 1003.2d standard.

`group_list=g_list`
Alters the group_list attribute, which lists the group name under which the job is to run on the execution system.

The *g_list* argument is of the form: `group[@host][,group[@host],...]` Only one group name may be given per specified host. Only one of the `group` specifications may be supplied without the corresponding `host` specification. That group name will used for execution on any host not named in the argument list. This option is an extension to the POSIX 1003.2d standard.

`stagein=file_list`
`stageout=file_list`
Alters the stageout attribute or the stagein attribute, which list which files are staged (copied) in before job start or staged out after the job completes execution. The *file_list* is in the form: `local_file@hostname:re-`

mote_file[,...]
The name local_file is the name on the system where the job executes. It may be an absolute path or a path relative to the home directory of the user. The name remote_file is the destination name on the host specified by hostname. The name may be absolute or relative to the user's home directory on the destination host. These options are extensions to the POSIX 1003.2d standard.

OPERANDS

The qalter command accepts one or more *job_identifier* operands of the form:

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

See the description under "Job Identifier" in section 2.7.6 in this ERS.

STANDARD ERROR

Any error condition, either in processing the options or the operands, or any error received in reply to the batch requests will result in a error message being written to standard error.

EXIT STATUS

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

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

SEE ALSO

qsub(1B), qstat(1B), pbs_alterjob(3B), pbs_statjob(3B), pbs_selectjob(3B), pbs_resources_*(7B), where * is system type, and the PBS ERS.

5.2.2. Delete Job

NAME

qdel - delete pbs batch job

SYNOPSIS

qdel [−W delay] job_identifier ...

DESCRIPTION

The **qdel** command deletes jobs in the order in which their job identifiers are presented to the command. A job is deleted by sending a *Delete Job* batch request to the batch server that owns the job. A job that has been deleted is no longer subject to management by batch services.

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

A batch job being deleted by a server will be sent a **SIGTERM** signal following by a **SIGKILL** signal. The time delay between the two signals is an attribute of the execution queue from which the job was run (settable by the administrator). This delay may be overridden by the −*W* option.

See the PBS ERS section 3.1.3.3, "Delete Job Request", for more information.

OPTIONS

−W delay Specify the delay between the sending of the SIGTERM and SIGKILL signals. The argument *delay* specifies a unsigned integer number of seconds. This option is an extension to POSIX 1003.2d.

OPERANDS

The qdel command accepts one or more *job_identifier* operands of the form:

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

See the description under "Job Identifier" in section 2.7.6 in this ERS.

STANDARD ERROR

The qdel 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 the qdel command, the exit status will be a value of zero.

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

SEE ALSO

qsub(1B), qsig(1B), and pbs_deljob(3B)

5.2.3. Hold Job

NAME

qhold - hold pbs batch jobs

SYNOPSIS

qhold [−h hold_list] job_identifier ...

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. There are three supported holds: USER, OTHER (also known as operator), and SYSTEM.

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.

If no *−h* option is given, the USER hold will be applied to the jobs described by the *job_identifier* operand list.

If the job identified by *job_identifier* 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 **held** state if it resides in an execution queue.

If the job is in **running** state, then the following additional action is taken to interrupt the execution of the job. This is an extension to POSIX.2d. 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 the requested hold attribute. This will have no effect unless the job is rerun with the **qrerun** command.

The qhold command sends a *Hold Job* batch request to the server as described in the general section.

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". The hold type associated with each letter is:

u − USER

o − OTHER

s − SYSTEM

n − None

Repetition of characters is permitted, but "n" may not appear in the same option argument with the other three characters.

OPERANDS

The qhold command accepts one or more *job_identifier* operands of the form:

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

See the description under "Job Identifier" in section 2.7.6 in this ERS.

STANDARD ERROR

The qhold command will write a diagnostic message to standard error for each error oc-

currence.

EXIT STATUS

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

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

SEE ALSO

qrls(1B), qalter(1B), qsub(1B), pbs_alterjob(3B), pbs_holdjob(3B), pbs_rlsjob(3B), pbs_job_attributes(7B), pbs_resources_unicos8(7B)

5.2.4. Move Job

NAME

qmove − move pbs batch job

SYNOPSIS

qmove destination job_identifier ...

DESCRIPTION

To move a job is to remove the job from the queue in which it resides and instantiate the job in another queue. The **qmove** command issues a *Move Job* batch request to the batch server that currently owns each job specified by *job_identifier*.

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

OPERANDS

The first operand is the new *destination*for

```
queue
@server
queue@server
```

See the PBS ERS section 2.7.3, "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]
```

See the description under "Job Identifier" in section 2.7.6 in this ERS.

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 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

qsub(1B), pbs_movejob(3B)

5.2.5. Message Job

NAME

qmsg − send message to pbs batch jobs

SYNOPSIS

qmsg [-E] [-O] message_string job_identifier ...

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.

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.

If neither the *-E* nor the *-O* 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]
```

See the description under "Job Identifier" in section 2.7.6 in this ERS.

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

qsub(1B), pbs_msgjob(3B)

5.2.6. Order Jobs

NAME

qorder − exchange order of two pbs batch jobs in a queue.

SYNOPSIS

qorder job_identifier job_identifier

DESCRIPTION

To order two jobs is to exchange the jobs positions in the queue or queues in which the jobs resides. The two jobs must be located at the same server. The **qorder** command issues an *Order Jobs* batch request to the batch server that currently owns the two jobs specified by the two *job_identifier* operands. No attribute of the job, such as priority is changed. The impact of interchanging the order with the queue(s) is dependent on local job scheduled policy, contact your systems administrator.

A job in the **running** state cannot be reordered.

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. See the description under "Job Identifier" in section 2.7.6 in this ERS.

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 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

qsub(1B), qmove(1B), pbs_orderjob(3B), pbs_movejob(3B)

5.2.7. Rerun Job

NAME

qrerun − rerun a pbs batch job

SYNOPSIS

qrerun job_identifier ...

DESCRIPTION

The **qrerun** command directs that the specified jobs are to be rerun if possible.

To rerun a job is is to terminate the session leader of the job and return the job to the queued state in the execution queue in which the job currently resides.

The qrerun command sends a *Rerun Job* batch request to the server which owns the job.

If a job is marked as not rerunable then the rerun request will fail for that job. See the Rerunable attribute and the *-r* option on the **qsub** and **qalter** commands.

OPERANDS

The qrerun command accepts one or more *job_identifier* operands of the form:

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

See the description under "Job Identifier" in section 2.7.6 in this ERS.

STANDARD ERROR

The qrerun 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 qrerun command, the exit status will be a value of zero.

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

SEE ALSO

qsub(1B), qalter(1B), pbs_alterjob(3B), pbs_rerunjob(3B)

5.2.8. Release Job

NAME

qrls − release hold on pbs batch jobs

SYNOPSIS

qrls [-h hold_list] job_identifier ...

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, and SYSTEM. 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.

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.

If the sched_hint attribute is set, when the job is returned to **queued** state, it may be given preference in selection for execution depending on site policy.

The qrls command sends a *Release Job* batch request to the server which owns the job.

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", an "s" in any combination, or one or more of the letters "n. The hold type associated with each letter is:

u − USER

o − OTHER

s − SYSTEM

n − None

Repetition of characters is permitted, but "n" may not appear in the same option argument with the other three characters.

OPERANDS

The qrls command accepts one or more *job_identifier* operands of the form:

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

See the description under "Job Identifier" in section 2.7.6 in this ERS.

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

qsub(1B), qalter(1B), qhold(1B), pbs_alterjob(3B), pbs_holdjob(3B), and pbs_rlsjob(3B).

5.2.9. Select Jobs

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]

DESCRIPTION

The **qselect** command provides a method to list the job identifier of those jobs which meet a list of selection criteria. The selection is accomplished by sending a *Select Jobs* batch request to the default server or the server specified by the −*q*option. Jobs are selected from those owned by a single server.

When qselect successfully completes, it will have written to standard output a list of zero or more jobs which meet the criteria specified by the options. Each option acts as a filter restricting the number of jobs which might be listed. With no options, the qselect command will list all jobs at the server which the user is authorized to list (query status of). The -u option may be used to limit the selection to jobs owned by this user or other specified users. Operators and system administrators have the privilege to select all jobs. Other uses access depends on the setting of the server attribute query_other_jobs.

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_listRestricts 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 one or more occurrences the single letter n, or one or more of the letters u, o, 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
s − system

-l resource_list

Restricts selection of jobs to those with specified resource amounts.

Only those jobs will be selected whose Resource_List attribute matches the specified relation with each resource and value listed in the *resource_list* argument. The *resource_list* is in the following format:

`resource_nameopvalue[,resource_nameopval,...]`
The relation operator *op* must be present.

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. See the ERS section 2.7.4 for a definition of 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: E, H, Q, R, T, and W. This set of state letters does not conform to the POSIX 1003.2d standard. It requires the sames letters, but in lower case. A repeated character will be accepted, but no additional meaning is assigned to it.

The characters in the *states* argument have the following interpretation:

E the Exiting state.

H the Held state.

Q the Queued state.

R the Running state.

T the Transiting state.

W the Waiting state.

Jobs will be selected which are in any of the specified states.

-u user_listRestricts 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 ability to select jobs owned by others is controllable by the server attribute query_other_jobs. Mapping between user names on different hosts and validation of privilege to access the specified user name is discussed under the server. This option may also be used by batch operators and batch administrators to select jobs belonging to other 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.

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

qsub(1B), qstat(1B), pbs_selectjob(3B), pbs_selstat(3B), pbs_statjob(3B)

5.2.10. Signal Job

NAME

qsig − signal pbs batch job

SYNOPSIS

qsig [-s signal] job_identifier ...

DESCRIPTION

The **qsig** command requests that a signal be sent to 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.

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 a 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. If it doesn't recognize the signal name, try issuing the signal number instead.

For Unicos on Cray systems only, two special signal names, "suspend" and "resume", 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 the server receives a *Signal Job* batch request with a signal that is unsupported on the server host, the server will reject the request.

OPERANDS

The qsig command accepts one or more *job_identifier* operands of the form:

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

See the description under "Job Identifier" in section 2.7.6 in this ERS.

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

qsub(1B), pbs_sigjob(3B), pbs_resources_*(7B) where * is system type, and the PBS ERS.

5.2.11. Status Jobs, Queues, or Server

NAME

qstat − show status of pbs batch jobs

SYNOPSIS

qstat [-f][-W site_specific] [job_identifier... | destination...]

qstat [-a|-i|-r] [-n] [-s] [-G|-M] [-R] [-u user_list] [job_identifier... | destination...]

qstat -Q [-f][-W site_specific] [destination...]

qstat -q [-G|-M] [destination...]

qstat -B [-f][-W site_specific] [server_name...]

DESCRIPTION

The **qstat** command is used to request the status of jobs, queues, or a batch server. The requested status is written to standard out. The status is obtained by sending a *Job Status*, a *Queue Status*, or a *Server Status* batch request to the appropriate server.

When requesting job status, synopsis format 1 or 2, **qstat** will output information about each *job_identifier* or all jobs at each *destination*. The capability to request status of all jobs at a destination is an extension to POSIX 1003.2d. Jobs for which the user does not have status privilege are not displayed.

When requesting queue or server status, synopsis format 3 through 5, qstat will output information about each *destination*. The syntax using the [-a|-i|-r] line or the -q line are extensions to POSIX.

OPTIONS

-f Specifies that a full status display be written to standard out.

-a "All" jobs are displayed in the alternative format, see the Standard Output section. If the operand is a destination id, all jobs at that destination are displayed. If the operand is a job id, information about that job is displayed.

-i Job status is displayed in the alternative format. For a destination id operand, status for jobs at that destination which are not running are displayed. This includeds jobs which are queued, held or waiting. If an operand is a job id, status for that job is displayed regardless of its state.

-r If an operand is a job id, status for that job is displayed. For a destination id operand, status for jobs at that destination which are running are displayed, this includeds jobs which are suspended. If an operand is a job id, status for that job is displayed.

-n In addition to the basic information, nodes allocated to a job are listed.

-s In addition to the basic information, any comment provided by the batch administrator or scheduler is shown.

-G Show size information in giga-bytes.

-M Show size information, disk or memory in mega-words. A word is considered to be 8 bytes.

-R In addition to other information, disk reservation information is shown. Not applicable to all systems.

-u Job status is displayed in the alternative format. If an operand is a job id, status for that job is displayed. For a destination id operand, status for jobs

at that destination which are owned by the user(s) listed in *user_list* are displayed. 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.

-Q Specifies that the request is for queue status and that the operands are destination identifiers.

-q Specifies that the request is for queue status which should be shown in the alternative format.

-B Specifies that the request is for batch server status and that the operands are the names of servers.

OPERANDS

If neither the *-Q* nor the *-B* option is given, the operands on the qstat command must be either job identifiers or destinations identifiers.

If the operand is a job identifier, it must be in the following form:

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

where `sequence_number.server_name` is the job identifier assigned at submittal time, see **qsub**. If the `.server_name` is omitted, the name of the default server will be used. If `@server` is supplied, the request will be for the job identifier currently at that Server. See ERS sections 2.7.6 and 2.7.3 for more details on job identifiers and batch destinations.

If the operand is a destination identifier, it is one of the following three forms:

```
queue
@server
queue@server
```

If `queue` is specified, the request is for status of all jobs in that queue at the default server. If the `@server` form is given, the request is for status of all jobs at that server. If a full destination identifier, `queue@server`, is given, the request is for status of all jobs in the named queue at the named server.

If the *-Q* option is given, the operands are destination identifiers as specified above. If `queue` is specified, the status of that queue at the default server will be given. If `queue@server` is specified, the status of the named queue at the named server will be given. If `@server` is specified, the status of all queues at the named server will be given. If no destination is specified, the status of all queues at the default server will be given.

If the *-B* option is given, the operand is the name of a server.

STANDARD OUTPUT

Displaying Job Status

If job status is being displayed in the default format and the *-f* option is not specified, the following items are displayed on a single line, in the specified order, separated by white space:

- the job identifier assigned by PBS.
- the job name given by the submitter.
- the job owner
- the CPU time used
- the job state:
  E - Job is exiting after having run.

H - Job is held.
Q - job is queued, eligable to run or routed.
R - job is running.
T - job is being moved to new location.
W - job is waiting for its execution time (-a option) to be reached.
S - (Unicos only) job is suspend.

This set of state letters does not conform to the POSIX 1003.2d standard. It requires the sames letters, but in lower case.

- the queue in which the job resides

If job status is being displayed and the *-f* option is specified, the output will depend on whether **qstat** was compiled to use a **Tcl** interpreter. See the configuration section for details. If **Tcl** is not being used, full display for each job consists of the header line:

```
Job Id:  job identifier
```

Followed by one line per job attribute of the form:

```
attribute_name = value
```

The attribute name is indented 4 spaces. There is a single space on each side of the equal sign. Long values wrap either at column 78 or following a comma beyond which the next comma separated segment will not fit before column 79. Continuation lines are indented by a tab (8 spaces). There is blank line following the last attribute.

If any of the options -a, -i, -r, -u, -n, -s, -G or -M are provided, the alternative display format for jobs is used. The following items are displayed on a single line, in the specified order, separated by white space:

- the job identifier assigned by PBS.
- the job owner.
- The queue in which the job currently resides.
- The job name given by the submitter.
- The session id (if the job is running).
- The number of nodes requested by the job.
- The number of cpus or tasks requested by the job.
- The amount of memory requested by the job.
- Either the cpu time, if specified, or wall time requested by the job, (hh:mm).
- The job's current state.
- The amount of cpu time or wall time used by the job (hh:mm).

If the -R option is provied, the line contains:

- the job identifier assigned by PBS.
- the job owner.
- The queue in which the job currently resides.
- The number of nodes requested by the job.
- The number of cpus or tasks requested by the job.
- The amount of memory requested by the job.
- Either the cpu time or wall time requested by the job.
- The job's current state.
- The amount of cpu time or wall time used by the job.
- The amount of SRFS space requested on the big file sysstem.
- The amount of SRFS space requested on the fast file sysstem.

- The amount of space requested on the parallel I/O file system.

The last three fields may not contain useful information at all sites or on all systems.

Displaying Queue Status

If queue status is being displayed and the *-f* option was not specified, the following items are displayed on a single line, in the specified order, separated by white space:

- the queue name
- the maximum number of jobs that may be run in the queue concurrently
- the total number of jobs in the queue
- the enable or disabled status of the queue
- the started or stopped status of the queue
- for each job state, the name of the state and the number of jobs in the queue in that state.
- the type of queue, execution or routing.

If queue status is being displayed and the *-f* option is specified, the output will depend on whether **qstat** was compiled to use a **Tcl** interpreter. See the configuration section for details. If **Tcl** is not being used, the full display for each queue consists of the header line:

```
Queue: queue_name
```

Followed by one line per queue attribute of the form:

```
attribute_name = value
```

The queue attributes are listed in the same format as job attributes.

If the -q option is specified, queue information is displayed in the alternative format: The following information is displayed on a single line:

- the queue name
- the maximum amount of memory a job in the queue may request
- the maximum amount of cpu time a job in the queue may request
- the maximum amount of wall time a job in the queue may request
- the maximum amount of nodes a job in the queue may request
- the number of jobs in the queue in the running state
- the number of jobs in the queue in the queued state
- the maximum number (limit) of jobs that may be run in the queue concurrently
- the state of the queue given by a pair of letters:
  - either the letter E if the queue is Enabled or D if Disabled, and
  - either the letter R if the queue is Running (started) or S if Stopped.

Displaying Server Status

If batch server status is being displayed and the *-f* option is not specified, the following items are displayed on a single line, in the specified order, separated by white space:

- the server name
- the maximum number of jobs that the server may run concurrently
- the total number of jobs currently managed by the server
- the status of the server

- for each job state, the name of the state and the number of jobs in the server in that state

If server status is being displayed and the *-f* option is specified, the output will depend on whether **qstat** was compiled to use a **Tcl** interpreter. See the configuration section for details. If **Tcl** is not being used, the full display for the server consist of the header line:

```
Server: server name
```

Followed by one line per server attribute of the form:

```
attribute_name = value
```

The server attributes are listed in the same format as job attributes.

STANDARD ERROR

The qstat command will write a diagnostic message to standard error for each error occurrence.

CONFIGURATION

If **qstat** is compiled with an option to include a **Tcl** interpreter, using the *-f* flag to get a full display causes a check to be made for a script file to use to output the requested information. The first location checked is $HOME/.qstatrc. If this does not exist, the next location checked is administrator configured. If one of these is found, a **Tcl** interpreter is started and the script file is passed to it along with three global variables. The command line arguments are split into two variable named **flags** and **operands**. The status information is passed in a variable named **objects**. All of these variables are **Tcl** lists. The **flags** list contains the name of the command (usually "qstat") as its first element. Any other elements are command line option flags with any options they use, presented in the order given on the command line. They are broken up individually so that if two flags are given together on the command line, they are separated in the list. For example, if the user typed

```
qstat -QfWbigdisplay
```

the **flags** list would contain

```
qstat -Q -f -W bigdisplay
```

The **operands** list contains all other command line arguments following the flags. There will always be at least one element in **operands** because if no operands are typed by the user, the default destination or server name is used. The **objects** list contains all the information retrieved from the server(s) so the Tcl interpreter can run once to format the entire output. This list has the same number of elements as the **operands** list. Each element is another list with two elements. The first element is a string giving the type of objects to be found in the second. The string can take the values "server", "queue", "job" or "error". The second element will be a list in which each element is a single batch status object of the type given by the string discussed above. In the case of "error", the list will be empty. Each object is again a list. The first element is the name of the object. The second is a list of attributes. The third element will be the object text. All three of these object elements corespond with fields in the structure `batch_status` which is described in detail for each type of object by the man pages for **pbs_statjob(3),**pbs_statque(3), Each attribute in the second element list whose elements correspond with the `attrl` structure. Each will be a list with two elements. The first will be the attribute name and the second will be the attribute value.

EXIT STATUS

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

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

SEE ALSO

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) where * is system type, and the PBS ERS.

5.2.12. Submit Job

NAME

qsub − submit pbs job

SYNOPSIS

qsub [-a date_time] [-A account_string] [-c interval] [-C directive_prefix] [-e path] [-h] [-I] [-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_list] [-u user_list] [-v variable_list] [-V] [-W additional_attributes] [-z] [script]

DESCRIPTION

To create a job is to submit an executable script to a batch server. The batch server will be the default server unless the *−q* option is specified. See discussion of PBS_DEFAULT under Environment Variables below. Typically, the script is a shell script which will be executed by a command shell such as sh or csh.

Options on the **qsub** command allow the specification of attributes which affect the behavior of the job. The job is created by sending a *Queue Job* batch request to the batch server.

The qsub command will pass certain environment variables in the Variable_List attribute of the job. These variables will be available to the job. The value for the following variables will be taken from the environment of the qsub command: **HOME**, **LANG**, **LOGNAME**, **PATH**, **MAIL**, **SHELL**, and **TZ**. These values will be assigned to a new name which is the current name prefixed with the string "PBS_O_". For example, the job will have access to an environment variable named **PBS_O_HOME** which have the value of the variable **HOME** in the qsub command environment.

In addition to the above, the following environment variables will be available to the batch job. (The values of the following environment variables are established by qsub.)

PBS_O_HOST
the name of the host upon which the qsub command is running.

PBS_O_QUEUE
the name of the original queue to which the job was submitted. (It is established by the server which creates the job, not qsub.)

PBS_O_WORKDIR
the absolute path of the current working directory of the qsub command.

The following are established by the server executing the job, not the qsub command.

PBS_ENVIRONMENT
set to PBS_BATCH to indicate the job is a batch job, or to PBS_INTERACTIVE to indicate the job is a PBS interactive job, see -I option.

PBS_JOBID
the job identifier assigned to the job by the batch system.

PBS_JOBNAME
the job name supplied by the user.

PBS_NODEFILE
the name of the file contain the list of nodes assigned to the job (for parallel and cluster systems).

PBS_QUEUE
the name of the queue from which the job is executed.

OPTIONS

-a date_time

Declares the time after which the job is eligible for execution.

The *date_time* argument is in the form: `[[[[CC]YY]MM]DD]hhmm[.SS]`

Where CC is the first two digits of the year (the century), YY is the second two digits of the year, 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.

If the month, `MM`, is not specified, it will default to the current month if the specified day `DD`, is in the future. Otherwise, the month will be set to next month. Likewise, 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 sumit a job at 11:15am with a time of `-a 1110`, the job will be eligable to run at 11:10am tomorrow. See the date_time operand for the touch(1) command defined by POSIX.2.

The Execution_Time job attribute will be set to the number of seconds since Epoch which is equivalent to the Universal time expressed by the local time in the *date_time* argument. If the *−a* option is not specified, the Execution_Time attribute is unset which represents a time zero or no delay.

-A account_string

Defines the account string associated with the job. The *account_string* is an undefined string of characters and is interpreted by the server which executes the job. See section 2.7.1 of the PBS ERS. The Account_Name attribute is set to the account string. If account_string is unset, it is not passed with the job to the job executor.

-c interval

Defines the interval at which the job will be checkpointed. If the job executes upon a host which does not support checkpoint, this option will be ignored.

The *interval* argument is specified as:

n No checkpointing is to be performed. The job's Checkpoint attribute is set to the string `"n"`.

s Checkpointing is to be performed only when the server executing the job is shutdown. The job's Checkpoint attribute is set to the string `"s"`.

c Checkpointing is to be performed at the default minimum time for the server executing the job. The job's Checkpoint attribute is set to the string `"c"`.

c=minutes

Checkpointing is to be performed at an interval of *minutes*, which is the integer number of minutes of CPU time used by the job. This value must be greater than zero. The Checkpoint attribute is set to the string specified by *"c=minutes"*.

If *-c* is not specified, the Checkpoint attribute is set to the value `"u"` meaning unspecified. Unless otherwise stated, "u" is treated the same as "s".

−C directive_prefix

Defines the prefix that declares a directive to the qsub command within the script file. See the paragraph on script directives in the Extended Description section.

If the *-C* option is presented with a *directive_prefix* argument that is the null string, qsub will not scan the script file for directives. The directive prefix is not a job attribute. It is used solely within the qsub command.

−e path Defines the path to be used for the standard error stream of the batch job. The *path* argument is of the form:

```
[hostname:]path_name
```

where hostname is the name of a host to which the file will be returned and path_name is the path name on that host in the syntax recognized by POSIX. The argument will be interpreted as follows:

path_name
Where path_name is not an absolute path name, then the qsub command will expand the path name relative to the current working directory of the command. The command will supply the name of the host upon which it is executing for the *hostname* component.

hostname:path_name
Where path_name is not an absolute path name, then the qsub command will not expand the path name relative to the current working directory of the command. On delivery of the standard error, the path name will be expanded relative to the user's home directory on the hostname system.

path_name
Where path_name specifies an absolute path name, then the qsub will supply the name of the host on which it is executing for the *hostname*.

hostname:path_name
Where path_name specifies an absolute path name, the path will be used as specified.

If the *-e* option is not specified, the default file name for the standard error stream will be used. The default name has the following form:

job_name.esequence_number

where job_name is the name of the job, see *-N* option, and sequence_number is the job number assigned when the job is submitted. This option sets the job attribute Error_Path.

-h Specifies that a user hold be applied to the job at submission time. The Hold_Types attribute will be set to USER, "u". If -h is not specified, then Hold_Types is set to NONE, "n".

−I Declares that the job is to be run "interactively". 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 through qsub to the terminal session in which qsub is running. See the "Extended Description" paragraph for addition information of interactive jobs. The −I option is a violation of the POSIX 1003.2d standard. Option key letters not defined by the standard, such as I, are reserved for future revisions of the standard. PBS can be built with the symbol PBS_NO_POSIX_VIOLATION defined, in which case the −I option is removed. The interactive attribute may still be specified via the −W option.

−j join Declares if the standard error stream of the job will be merged with the standard output stream of the job.

An option argument value of oe directs that the two streams will be merged, intermixed, as standard output. The Join_Path job attribute is set to "oe". An option argument value of eo directs that the two streams will be merged, intermixed, as standard error. The Join_Path job attribute is set to "eo".

If the *join* argument is n or the option is not specified, the two streams will be two separate files. The Join_Path job attribute is set to "n".

−k keep Defines which (if either) of standard output or standard error will be retained on the execution host. If set for a stream, this option overrides the path name for that stream. If not set, neither stream is retained on the execution host.

The argument is either the single letter "e" or "o", or the letters "e" and "o" combined in either order. Or the argument is the letter n. Repetition of characters is permitted, but "n" may not appear in the same option argument with

the other two characters. The attribute Keep_Files is set to the argument.

e The standard error stream is to retained on the execution host. The stream will be placed in the home directory of the user under whose user id the job executed. The file name will be the default file name given by: `job_name.esequence` where `job_name` is the name specified for the job, and `sequence` is the sequence number component of the job identifier. The attribute is set to KEEP_ERROR.

o The standard output stream is to retained on the execution host. The stream will be placed in the home directory of the user under whose user id the job executed. The file name will be the default file name given by: `job_name.osequence` where `job_name` is the name specified for the job, and `sequence` is the sequence number component of the job identifier. The attribute is set to KEEP_OUTPUT.

eo Both the standard output and standard error streams will be retained. The attribute is set to "KEEP_OUTPUT | KEEP_ERROR".

oe Both the standard output and standard error streams will be retained. The attribute is set to "KEEP_OUTPUT | KEEP_ERROR".

n Neither stream is retained.

-l resource_list

Defines the resources that are required by the job and establishes a limit to the amount of resource that can be consumed. If not set for a generally available resource, such as CPU time, the limit is infinite. The *resource_list* argument is of the form:

```
resource_name[=[value]][,resource_name[=[value]],...]
```

For each resource listed in the *resource_list*, one entry will be added to the Resource_List attribute of the job. The entry contains the resource name and its requested value. No white space is allowed in the value. Other than syntax, qsub performs no resource or value checking. The checking is performed by the execution server.

−m mail_options

Defines the set of conditions under which the execution server will send a mail message about the job. The *mail_options* argument is a string which consists of either the single character "n", or one or more of the characters "a", "b", and "e". Repeated letters are accepted, but n cannot be mixed with the other characters.

If the character "n" is specified, no mail will be sent. The Mail_Points attribute is set to NONE, "n".

For the letters "a", "b", and "e":

a mail is sent when the job is aborted by the batch system. The Mail_Points attribute is set to ABORT, "a".

b mail is sent when the job begins execution. The Mail_Points attribute is set to BEGINNING, "b".

e mail is sent when the job terminates. The Mail_Points attribute is set to EXIT, "e".

If the *−m* option is not specified, mail will be sent if the job is aborted. The Mail_Points attribute is set to ABORT, "a".

−M user_list

Declares the list of users to whom mail is sent by the execution server when it sends mail about the job.

The *user_list* argument is of the form:

```
user[@host][,user[@host],...]
```

If unset, the list defaults to the submitting user at the qsub host, i.e. the job owner.

The Mail_Users attribute is set to the argument.

−N name

Declares a name for the job. The name specified may be up to and including 15 characters in length. It must consist of printable, non white space characters with the first character alphabetic. [The POSIX 1003.2d Standard calls for only alphanumeric characters, but then calls for the use of the script file base name as the job name if a name is not specified. The file name may contain other than alphanumeric characters. Therefore I "interpret" the standard as allowing printable characters.] Names taken from the script name may have a non-alphabetic character first. If the script basename is greater than 15 characters, it will be truncated to 15.

If the *−N* option is not specified, the job name will be the base name of the job script file specified on the command line. If no script file name was specified and the script was read from the standard input, then the job name will be set to STDIN.

The Job_Name attribute is set to the name.

−o path Defines the path to be used for the standard output stream of the batch job. The *path* argument is of the form:

```
[hostname:]path_name
```

where hostname is the name of a host to which the file will be returned and path_name is the path name on that host in the syntax recognized by POSIX. The argument will be interpreted as follows:

path_name

Where path_name is not an absolute path name, then the qsub command will expand the path name relative to the current working directory of the command. The command will supply the name of the host upon which it is executing for the *hostname* component.

hostname:path_name

Where path_name is not an absolute path name, then the qsub command will not expand the path name relative to the current working directory of the command. On delivery of the standard output, the path name will be expanded relative to the user's home directory on the hostname system.

path_name

Where path_name specifies an absolute path name, then the qsub will supply the name of the host on which it is executing for the *hostname*.

hostname:path_name

Where path_name specifies an absolute path name, the path will be used as specified.

If the *-o* option is not specified, the default file name for the standard output stream will be used. The default name has the following form:

```
job_name.osequence_number
```

where job_name is the name of the job, see *-N* option, and sequence_number is the job number assigned when the job is submitted. This option sets the job attribute Output_Path.

−p priority

Defines the priority of the job. The *priority* argument must be a integer be-

tween -1024 and +1023 inclusive. The default is no priority which is equivalent to a priority of zero. The Priority job attribute is set to this signed integer value.

-q destination

Defines the destination of the job. The *destination* names a queue, a server, or a queue at a server.

The qsub command will submit the script to the server defined by the *destination* argument. The server named by the destination is the one to which qsub sends the *Queue Job* batch request. If the destination is a *routing queue,* the job may be routed by the server to a new destination.

If the *−q* option is not specified, the qsub command will submit the script to the default server. See PBS_DEFAULT under the Environment Variables section on this man page and the PBS ERS section 2.7.4, "Default Server".

If the *−q* option is specified, it is in one of the following three forms:

```
queue
@server
queue@server
```

If the *destination* argument names a queue and does not name a server, the job will be submitted to the named queue at the default server.

If the *destination* argument names a server and does not name a queue, the job will be submitted to the default queue at the named server.

If the *destination* argument names both a queue and a server, the job will be submitted to the named queue at the named server.

−r y|n Declares whether the job is rerunable. See the **qrerun** command. The option argument is a single character, either y or n. Also see *rerunable* in the glossary.

If the argument is "y", the job is rerunable. The Rerunable attribute is set to the character 'y'. If the argument is "n", the job is not rerunable. The default value is 'y', rerunable.

−S path_list

Declares the shell that interprets the job script.

The option argument *path_list* is in the form:

```
path[@host][,path[@host],...]
```

Only one path may be specified for any host named. Only one path may be specified without the corresponding host name. The path selected will be the one with the host name that matched the name of the execution host. If no matching host is found, then the path specified without a host will be selected, if present.

If the *−S* option is not specified, the option argument is the null string, or no entry from the *path_list* is selected, the execution will use the user's login shell on the execution host. The Shell_Path_List attribute is set to the *path_list* argument if present, otherwise it is set to the null string.

−u user_list

Defines the user name under which the job is to run on the execution system.

The *user_list* argument is of the form:

```
user[@host][,user[@host],...]
```

Only one user name may be given per specified host. Only one of the user specifications may be supplied without the corresponding host specification. That user name will used for execution on any host not named in the argument list. The User_List attribute is set to the value of *user_list*. If unset, the user

list defaults to the user who is running qsub.

-v variable_list

Expands the list of environment variables that are exported to the job.

In addition to the variables described in the "Description" section above, *variable_list* names environment variables from the qsub command environment which are made available to the job when it executes. The *variable_list* is a comma separated list of strings of the form `variable` or `variable=value`. These variables and their values are passed to the job. The Variable_List attribute is appended with the variables in *user_list* and their values.

-V Declares that all environment variables in the qsub command's environment are to be exported to the batch job. The Variable_List attribute is appended with the variables in the qsub command's environment and their values.

-W additional_attributes

The -W option allows for the specification of additional job attributes. POSIX.2 reserves all undefined option letters for future versions of the standard. The single letter 'W' is allowed for extensions. PBS makes use of the -W to specify attributes which are extensions to POSIX 1003.2d. The general syntax of the -W is in the form:

```
-W attr_name=attr_value[,attr_name=attr_value...]
```

Note if white space occurs anywhere within the option argument string or the equal sign, "=", occurs within an *attribute_value* string, then the string must be enclosed with either single or double quote marks.

PBS currently supports the following attributes within the -W option.

`depend=dependency_list`

Defines the dependency between this and other jobs. The *dependency_list* is in the form:

`type[:argument[:argument...][,type:argument...]`.

The *argument* is either a numeric count or a PBS job id according to *type*. If argument is a count, it must be greater than 0. If it is a job id and not fully specified in the form `seq_number.server.name`, it will be expanded according to the default server rules which apply to job ids on most commands. If *argument* is null (the preceeding colon need not be specified), the dependency of the cooresponding type is cleared (unset).

`synccount:`*count*

This job is the first in a set of jobs to be executed at the same time. *Count* is the number of additional jobs in the set.

`syncwith:`*jobid*

This job is an additional member of a set of jobs to be executed at the same time. In the above and following dependency types, *jobid* is the job identifier of the first job in the set.

`after:`*jobid[:jobid...]*

This job may be scheduled for execution at any point after jobs *jobid* have started execution.

`afterok:`*jobid[:jobid...]*

This job may be scheduled for execution only after jobs *jobid* have terminated with no errors. See the csh warning under "Extended Description".

`afternotok:`*jobid[:jobid...]*

This job may be scheduled for execution only after jobs *jobid* have terminated with errors. See the csh warning under "Extended Description".

`afterany:`*jobid[:jobid...]*

This job may be scheduled for execution after jobs *jobid* have terminated, with or without errors.

on:*count*
This job may be scheduled for execution after *count* dependencies on other jobs have been satisfied. This form is used in conjunction with one of the before forms, see below.

before:*jobid[:jobid...]*
When this job has begun execution, then jobs *jobid...* may begin.

beforeok:*jobid[:jobid...]*
If this job terminates execution without errors, then jobs *jobid...* may begin. See the csh warning under "Extended Description".

beforenotok:*jobid[:jobid...]*
If this job terminates execution with errors, then jobs *jobid...* may begin. See the csh warning under "Extended Description".

beforeany:*jobid[:jobid...]*
When this job terminates execution, jobs *jobid...* may begin.

If any of the before forms are used, the jobs referenced by *jobid* must have been submitted with a dependency type of on.

The depend attribute is set to the value of the *dependency* option argument.

If any of the before forms are used, the jobs referenced by *jobid* must have the same owner as the job being submitted. Otherwise, the dependency is ignored.

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.big.iron.com /tmp/script
qsub -W depend=before:234.hunk1.com:235.hunk1.com /tmp/script
```

group_list=g_list
Defines the group name under which the job is to run on the execution system. The *g_list* argument is of the form:

```
group[@host][,group[@host],...]
```

Only one group name may be given per specified host. Only one of the group specifications may be supplied without the corresponding host specification. That group name will used for execution on any host not named in the argument list. The group_list attribute is set to the value of *g_list*. If not set, the group_list defaults to the primary group of the user under which the job will be run.

interactive=true
If the interactive attribute is specified, the job is an interactive job. The −I option is a alternative method of specifying this attribute.

stagein=file_list
stageout=file_list
Specifies the stagein or stageout attribute, listing which files are staged (copied) in before job start or staged out after the job completes execution. On completion of the job, all staged-in and staged-out files are removed from the execution system. The *file_list* is in the form

```
local_file@hostname:remote_file[,...]
```

regardless of the direction of the copy. The name local_file is the name of the file on the system where the job executed. It may be an absolute path or relative to the home directory of the user. The name remote_file is the destination name on the host specified by hostname. The name may be absolute

or relative to the user's home directory on the destination host. The use of wildcards in the file name is not recommended. Since rcp (or scp) is run via rsh, it will pick up matching names from the remote system. However, pbs_mom will does not expand the wildcards and will fail to delete the staged files on job termination. The file names map to a remote copy program (rcp) call on the execution system in the follow manner:
For stagein: rcp hostname:remote_file local_file
For stageout: rcp local_file hostname:remote_file

-z Directs that the qsub command is not to write the job identifier assigned to the job to the command's standard output.

OPERANDS

The qsub command accepts a *script* operand that is the path to the script of the job. If the path is relative, it will be expanded relative to the working directory of the qsub command.

If the *script* operand is not provided or the operand is the single character "−", the qsub command reads the script from standard input.

STANDARD INPUT

The qsub command reads the script for the job from standard input if the *script* operand is missing or is the single character "−".

INPUT FILES

The *script* file is read by the qsub command. Qsub acts upon any directives found in the script.

When the job is created, a copy of the script file is made and that copy cannot be modified.

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 values of some or all of the variables in the qsub command's environment are exported with the job, see the -v and -V options.

The environment variable **PBS_DEFAULT** defines the name of the default server. Typically, it corresponds to the system name of the host on which the server is running. If PBS_DEFAULT is not set, the default is defined by an administrator established file.

The environment variable **PBS_DPREFIX** determines the prefix string which identifies directives in the script.

EXTENDED DESCRIPTION

Script Processing:

A job script may consist of PBS directives, comments and executable statements. A PBS directive provides a way of specifying job attributes in addition to the command line options. For example:

```
:
#PBS -N Job_name
```

```
#PBS -l walltime=10:30,mem=320kb
#PBS -m be
#
step1 arg1 arg2
step2 arg3 arg4
```

The qsub command scans the lines of the script file for directives. An initial line in the script that begins with the characters "#!" or the character ":" will be ignored and scanning will start with the next line. Scanning will continue until the first executable line, that is a line that is not blank, not a directive line, nor a line whose first non white space character is "#". If directives occur on subsequent lines, they will be ignored.

A line in the script file will be processed as a directive to qsub if and only if the string of characters starting with the first non white space character on the line and of the same length as the directive prefix matches the directive prefix.

The remainder of the directive line consists of the options to qsub in the same syntax as they appear on the command line. The option character is to be preceded with the "−" character.

If an option is present in both a directive and on the command line, that option and its argument, if any, will be ignored in the directive. The command line takes precedence.

If an option is present in a directive and not on the command line, that option and its argument, if any, will be processed as if it had occurred on the command line.

The directive prefix string will be determined in order of preference from:

The value of the *−C* option argument if the option is specified on the command line.

The value of the environment variable **PBS_DPREFIX** if it is defined.

The four character string #PBS.

If the *−C* option is found in a directive in the script file, it will be ignored.

User Authorization:

When the user submits a job from a system other than the one on which the PBS Server is running, the name under which the job is to be executed is selected according to the rules listed under the -u option. The user submitting the job must be authorized to run the job under the execution user name. This authorization is provided if

(1) The host on which qsub is run is trusted by the execution host (see /etc/hosts.equiv),

(2) The execution user has an .rhosts file naming the submitting user on the submitting host.

C-Shell .logout File:

The following warning applies for users of the c-shell, csh. If the job is executed under the 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. To preserve the job exit status, either remove the .logout file or place the following line as the first line in the .logout file

```
set EXITVAL = $status
```

and the following line as the last executable line in .logout

```
exit $EXITVAL
```

Interactive Jobs:

If the *−I* option is specified on the command line or in a script directive, or if the "interactive" job attribute declared true via the −W option, `-W interactive=true`, either on the command line or in a script directive, the job is an interactive job. The script will be processed for directives, but will not be included with the job. When the job begins execution, all input to the job is from the terminal session in which qsub is running.

When an interactive job is submitted, the qsub command will not terminate when the job is submitted. Qsub will remain running until the job terminates, is aborted, or the user interrupts qsub with an SIGINT (the control-C key). If qsub is interrupted prior to job start, it will query if the user wishes to exit. If the user response "yes", qsub exits and the job is aborted.

One the interactive job has started execution, input to and output from the job pass through qsub. Keyboard generated interrupts are passed to the job. Lines entered that begin with the tilde ('~') character and contain special sequences are escaped by qsub. The recognized escape sequences are:

~. Qsub terminates execution. The batch job is also terminated.

~susp Suspend the qsub program if running under the C shell. "susp" is the suspend character, usually CNTL-Z.

~asusp Suspend the input half of qsub (terminal to job), but allow output to continue to be displayed. Only works under the C shell. "asusp" is the auxiliary suspend character, usually CNTL-Y.

EXIT STATUS

Upon successful processing, the qsub exit status will be a value of zero.

If the qsub command fails, the command exits with a value greater than zero.

SEE ALSO

qalter(1B), qdel(1B), qhold(1B), qmove(1B), qmsg(1B), qrerun(1B), qrls(1B), qselect(1B), qsig(1B), qstat(1B), pbs_connect(3B), pbs_job_attributes(7B), pbs_queue_attributes(7B), pbs_resources_irix5(7B), pbs_resources_sp2(7B), pbs_resources_sunos4(7B), pbs_resources_unicos8(7B), pbs_server_attributes(7B), and pbs_server(8B)

5.2.13. Convert NQS Scripts

NAME

nqs2pbs − convert NQS job scripts to PBS

SYNOPSIS

nqs2pbs nqs_script [pbs_script]

DESCRIPTION

This utility converts a 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 encounterred, an error message is displayed. The new PBS script will be deleted if any errors occur.

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)

5.2.14. BASL Compiler

NAME

basl2c − converts a BASL (BAtch Scheduling Language) code into a C scheduler code.

SYNOPSIS

basl2c [-d] [-l lexerDebugFile] [-p parserDebugFile] [-y symtabDebugFile] [-s semanticDebugFile] [-g codegenDebugFile] [-c cFile] baslFile

DESCRIPTION

basl2c is the BASL to C compiler that produces an intermediate code that can be fed into a regular C compiler, and linked with the PBS libraries to produce the scheduler executable. Basl2c takes as input a *baslFile*, which is a program written in the BAtch Scheduling Language, containing the main scheduling code. Basl2c then converts the BASL constructs in the file into C statements, and it also attaches additional code to produce the PBS scheduler source code. By default, the resulting C code is written into the file *pbs_sched.c*.

The full pathname to the resulting C file is what needs to be specified in the **SCHD_CODE** variable in local.mk before compiling the BASL scheduler to produce the *pbs_sched* executable.

OPTIONS

-d Prints additional debugging messages to the lexer (see -l option), parser (see -p option), symbol table (see -y option), semantic analyzer (see -s option), and code generator (see -g option).

-l lexerDebugFile
lexerDebugFile is the name of a file to write into the debugging messages generated while scanning for tokens.

-p parserDebugFile
parserDebugFile is the name of a file to write into the debugging messages generated while putting together tokens in a usable way.

-y symtabDebugFile
symtabDebugFile is the name of a file to write into the debugging messages related to the symbol table.

-s semanticDebugFile
semanticDebugFile is the name of a file to write into the debugging messages generated while checking to make sure variables and operators are used in a consistent way.

-g codegenDebugFile
codegenDebugFile is the name of a file to write into the debugging messages generated while converting BASL statements to C statements.

-c cFile
cFile is the name of a file where the generated C code is written into.

MAIN STRUCTURE

The basic structure of a scheduler code written in BASL is as follows:

zero or more FUNCTIONS definitions
zero or more global VARIABLE DECLARATIONS
zero or more assignment statements (to initialize global variables)
sched_main()
{
one or more VARIABLE DECLARATIONS

zero or more STATEMENTS
}

For example,

```
% cat sched.basl
Int sum(Int a, Int b)
{
        Int s;
        s = a + b;
        return(s);
}
Int glob;
sched_main()
{
        Int c;

        a = 3;
        b = 4;
        c = sum(a, b);
        print(c);
glob = 5;
print(glob);
}
```

`sched_main()` is the function that gets called at every scheduling iteration.

FUNCTIONS

To define a function that can be called in subsequent functions, the syntax is:

ReturnType function-name (DATATYPE1 IDENTIFIER1,
DATATYPE2 IDENTIFIER2, ...)
{
one or more VARIABLE DECLARATIONS

zero or more STATEMENTS
}

For example,

```
Void printStuff(Dayofweek dow, DateTime t, String str,
                                        Size sz, CNode cn)
{
    print(dow);
    print(t);
    print(str);
    print(sz);
    print(cn);
}
```

Valid function **ReturnType** are: Void, Int, Float, Dayofweek, DateTime, String, Size, Server, Que, Job, CNode, Set Server, Set Que, Set Job, Set CNode.

Valid data types (**DATATYPE1, DATATYPE2, ...**) for the parameter identifiers are: Int, Float, Dayofweek, DateTime, String, Size, Server, Que, Job, CNode, Set Server, Set Que, Set Job, Set CNode, Range Int, Range Float, Range Dayofweek, Range DateTime, Range Size, Fun Int, Fun Float, Fun Void, Fun Dayofweek, Fun DateTime, Fun String, Fun Size, Fun Server, Fun Que, Fun Job, Fun CNode, Fun Set Server, Fun Set Que, Fun Set Job, Fun Set CNode. These data types will be discussed in the next topic.

Functions are invoked by their name and their arguments as in:

```
printStuff( MON, (5|1|1997@14:32:00), "sched begins",
                                          30gb, node );
```

basl2c will actually add a "basl_" prefix to the function name given by the scheduler writer to minimize chance of name collision, which can result when the resulting C code is linked with the PBS, BASL libraries. For example, if you look at the generated C code for *printStuff*, you would see,

```
basl_printStuff( MON, (5|1|1997@14:32:00),
                "sched begins", 30gb, node );
```

As in C, all function calls must have been previously defined. The BASL compiler will check to make sure that arguments in the function call match up exactly (in terms of types) with the parameters in the function definition.

Two kinds of functions exist in BASL: user-defined functions and predefined functions. User-defined functions are those that the scheduler writer provided a definition for, while predefined functions are those that can immediately be called without a need for defining it. For a list of predefined functions, see section on **PREDEFINED FUNCTIONS**.

VARIABLE DECLARATIONS

Like in C, all variables in a BASL code must be explicitly declared before use. Those variables declared outside of any function are referred to as global variables, while variables that are declared within a function body are called local variables. Global variables are usable anywhere within the BASL code, while local variables are readable only within the function from which they were declared.

The syntax of a variable declaration is:

DATATYPE IDENTIFIER ;

where **DATATYPE** can be: Int, Float, Dayofweek, DateTime, String, Size, Server, Que, Job, CNode, Set Server, Set Que, Set Job, Set CNode, Range Int, Range Float, Range Dayofweek, Range DateTime, Range Size.

DATA TYPE

Void used for functions that don't return a value.

Int signed, whole numbers given in base 10.

Sample constants:
5, +1, -3, SUCCESS (=1), FAIL (=0), TRUE (=1), FALSE (=0)

Float real numbers which are represented as doubles in the translated C code.
Sample constants: 4.3, +1.2, -2.6

Dayofweek
constant values: SUN, MON, TUE, WED, THU, FRI, SAT, internally represented as integer valued constants with SUN=0, MON=1, and so on.

DateTime
specify in one of 3 formats:

[1] (m|d|y) where 1 <= m <= 12, 1 <= d <= 31, 0 <= y, ex. (4|4|1997);

[2] (hh:mm:ss) where 0 <= hh <= 23, 0 <= mm <= 59, 0 <= ss <= 61, ex. (12:01:00);

[3] (m|d|y@hh:mm:ss), ex. (4|4|1997@12:01:00)
During dates/times comparison, "now" time is substituted if the time portion is not given (format [1]); the "now" date is substituted if the date portion is not given (format [2]). Also, the full year portion must be given (i.e. 1997 instead of 97) in dates to avoid ambiguity.

String A string is enclosed in quotes (") and it can contain anything except another quote, a newline, and left and right parentheses.
Sample constants: "a sample string", NULLSTR

Size format: <integer><suffix> where suffix is a multiplier of the form: <multiplier><unit>:

multiplier	unit (bytes or words)
k,m,g,t,p,K,M,G,T,P	b,B,w,W

where k=K=1024, m=M=1,048,576, g=G=1,073,741,824, t=T=1,099,511,627,776, p=P=1,125,899,906,842,624, b=B=1, and word size w=W is locally defined (i.e. 4 bytes in a 32-bit machine).

When operating on 2 size operands that are of different suffixes, the suffix of the "lower" of the two will be the resultant suffix. For example,
10mb + 10gb = 10250mb
Sample constants: -1b, 2w, 1kb, 2mw, +3gb, 4tw, 6Pb

Range Int
format: (low Int value, high Int value)
where low Int value <= high Int value. Sample constant: (1,3)

Range Float
format: (low Float value, high Float value)
where low value <= high value. Sample constant: (2.3, 4.6)

Range Dayofweek
format: (earlier day, later day)
where earlier day <= later day. Sample constant: (WED, FRI)

Range DateTime
format: (earlier date/time, later date/time)
where earlier date/time <= later date/time.
NOTE: if range contains only time portions, and earlier time "appears" to be > later time as in "((18:0:0), (6:0:0))", then during date/time comparisons, the "later" time will be adjusted by one day so that it will look like: "((<now date>@18:0:0), (<tomorrow date>@6:0:0)

)"

Sample constants:
((4|4|1997), (4|10|1997)), ((12:01:00), (12:30:00)), ((4|4|1997@12:01:00), (4|10|1997@12:30:00))

Range Size

format: (low size, high size)
where low size <= high size. Sample constants: (23gb, 50gb)

Server Maps directly to the PBS server object. A **Server** manages one or more **Que** objects.
Sample constant: NOSERVER

CNode for computational node consisting of a single OS image, a shared memory, and a set of cpus. CNode runs 1 PBS MOM.
Sample constant: NOCNODE

Que Maps directly to the PBS queue object. A **Que** object spools one or more **Job** objects.
Sample constant: NOQUE

Job Maps directly to the PBS job object. A **Job** object carries some attributes and resource requirements.
Sample constant: NOJOB

Set Server
list of Server objects.
Sample constant: EMPTYSETSERVER

Set CNode
list of CNode objects.
Sample constant: EMPTYSETCNODE

Set Que list of Que objects.
Sample constant: EMPTYSETQUE

Set Job list of Job objects.
Sample constant: EMPTYSETJOB

BASL-DEFINED CONSTANTS

These are constants that cannot be used for naming an identifier (see next topic). These are always in uppercase.

DATA TYPE	BASL-DEFINED CONSTANT
Dayofweek	SUN, MON, TUE, WED, THU, FRI, SAT
Int	SUCCESS, FAIL, FALSE, TRUE, SYNCRUN, ASYNCRUN, DELETE, RERUN, HOLD, RELEASE, SIGNAL, MODIFYATTR, MODIFYRES, SERVER_ACTIVE, SERVER_IDLE, SERVER_SCHED, SERVER_TERM, SERVER_TERMDELAY, QTYPE_E, QTYPE_R, SCHED_DISABLED, SCHED_ENABLED, TRANSIT, QUEUED, HELD, WAITING, RUNNING, EXITING, CNODE_OFFLINE, CNODE_DOWN, CNODE_FREE, CNODE_RESERVE, CNODE_INUSE_EXCLUSIVE, CNODE_INUSE_SHARED, CNODE_TIMESHARED, CNODE_CLUSTER, CNODE_UNKNOWN, OP_EQ, OP_NEQ, OP_LE, OP_LT, OP_GE, OP_GT, OP_MAX, OP_MIN,

```
                          ASC, DESC

Server                    NOSERVER
Set Server                EMPTYSETSERVER

CNode                     NOCNODE
Set CNode                 EMPTYSETCNODE

Que                       NOQUE
Set Que                   EMPTYSETQUE

Job                       NOJOB
Set Job                   EMPTYSETJOB

String                    NULLSTR
```

IDENTIFIER

Identifiers (used for variable names and function names) are in alphanumeric format, with the special underscore (_) character allowed. Currently, BASL can only handle identifiers with length of up to 80 chars. Also, you cannot use the BASL-defined constant names for naming an identifier.

STATEMENTS

In BASL(2), you can have a single statement terminated by a semi-colon, or a group of statements (called compound statement or block) delimited by '{' and '}'. The different kinds of statements that can appear in a BASL code are:

1. expression statement

Expression statements are anything of the form:

expr ;

where **expr** can be:

a) Arithmetic expressions

lexpr + rexpr (add)
lexpr - rexpr (subtract)
lexpr * rexpr (multiply)
lexpr / rexpr (divide)
lexpr % rexpr (modulus or remainder)

NOTE: Adding, subtracting, multiplying, dividing, and remaindering will only be allowed for proper types and if the left and right expressions are of consistent types. The table below illustrates what types are consistent among the various operators:

For +:

```
lexpr          rexpr
============   ============
Int or Float   Int or Float
Size           Size
String         String
```

For -, *, /:

lexpr	rexpr
Int or Float	Int or Float
Size	Size

For %:

lexpr	rexpr
Int or Float	Int or Float

Here are some sample arithmetic expressions statements:

```
Int     i1;
Int     i2;
Float   f1;
Float   f2;
Size    sz1;
Size    sz2;
String  str1;
String  str2;

i1 + i2;
f1 - i2;
sz1 * sz2 * 2b;
sz1 / 1024b;

str1 = "basl";
str2 = " cool";

// the following is a string concatenation
// operation resulting in the string:
//      "basl cool"
str1 + str2;

i1 % 10;
```

b) Unary expressions

```
+expr   // positive - multiplies by 1 an
        //      expression that is
        //      of Int, Float, or
        //      Size type

-expr   // negative - multiplies by -1 an
        //      expression that is
        //      of Int, Float, or
        //      Size type

!expr   // not - converts a non-zero expr
        //      value into 0, and a
        //      zero expr value into 1
        //      where expr type must be
        //      of type Int or Float
```

Some sample unary expressions:

```
Int i;

+3;
-(i + 4);
!i;
```

c) Logical expressions

lexpr EQ rexpr
lexpr NEQ rexpr
lexpr LT rexpr
lexpr LE rexpr
lexpr GT rexpr
lexpr GE rexpr
lexpr AND rexpr
lexpr OR rexpr

lexpr and **rexpr** must have types that are mutually consistent as shown in the following table:

lterminal-expr	rterminal-expr
Int or Float	Int or Float
Dayofweek	Dayofweek
DateTime	DateTime
String	String
Size	Size
Server	Server
Que	Que
Job	Job
CNode	CNode
Set Server	Set Server
Set Que	Set Que
Set Job	Set Job
Set CNode	Set CNode

For **AND, OR** operators, the **lexpr, rexpr** consistent types are Int or Float.

Some sample logical expressions:

```
i1 EQ  i2;
i1 NEQ f2;
dow1 LE dow2;
d1 LT d2;
str1 GT str2;
sz1 GE sz2;
```

d) Post-operator expressions
These are expressions that are merely shortcut to assignment statements.

IDENTIFIER++; // identifier=identifier+1
IDENTIFIER--; // identifier=identifier-1

IDENTIFIER must be of Int or Float type.

Example:

```
Int i;
Float f;

i++;
f--;
```

e) Function call

function-name (arg1 ,arg2 ... , argN)

where **arg1, ..., argN** can be any constant or variable. You can't have another function call as an argument.
Example:

```
Void pr(Int a) {
        print(a);
}

pr(5);
```

There are certain predefined functions that a scheduler writer can automatically call in his/her BASL code without a need to define it. These functions are referred to as assist functions (or helper functions) and they are discussed under **PREDEFINED FUNCTIONS** topic.

f) Constants
Some valid constant expressions are given in the following:

5;
+1.2;
SUN;
MON;
TUE;
WED;
THU;
FRI;
SAT;
(4|4|1997);
(12:01:00);
(4|4|1997@12:01:00);
"wonderful";
-1b;
SYNCRUN;
ASYNCRUN;
DELETE;
RERUN;
HOLD;
RELEASE;
SIGNAL;
MODIFYATTR;
MODIFYRES;
(1, 3);
(2.3, 4.6);
(WED, FRI);
((4|4|1997), (4|10|1997));

((12:01:00), (12:30:00));
((4|4|1997@12:01:00), (4|10|1997@12:30:00));
(23gb, 50gb);
NOSERVER;
NOCNODE;
NOQUE;
NOJOB;
EMPTYSETSERVER;
EMPTYSETCNODE;
EMPTYSETQUE;
EMPTYSETJOB;
NULLSTR;
SUCCESS;
FAIL;
SERVER_ACTIVE;
SERVER_IDLE;
SERVER_SCHED;
SERVER_TERM;
SERVER_TERMDELAY;
QTYPE_E;
QTYPE_R;
SCHED_DISABLED;
SCHED_ENABLED;
FALSE;
TRUE;
TRANSIT;
QUEUED;
HELD;
WAITING;
RUNNING;
EXITING;
CNODE_OFFLINE;
CNODE_DOWN;
CNODE_FREE;
CNODE_RESERVE;
CNODE_INUSE_EXCLUSIVE;
CNODE_INUSE_SHARED;
CNODE_TIMESHARED;
CNODE_CLUSTER;
CNODE_UNKNOWN;
OP_EQ;
OP_NEQ;
OP_LE;
OP_LT;
OP_GE;
OP_GT;
OP_MAX;
OP_MIN;

g) Identifier

Example:

Int i;

i;

2. Assignment statement

IDENTIFIER = expr ;

IDENTIFIER and **expr** must have types that are mutually consistent as illustrated in the following table:

identifier	expr
Int	Int, Float
Float	Int, Float
Dayofweek	Dayofweek
DateTime	DateTime
String	String
Size	Size
Que	Que
Job	Job
CNode	CNode
Server	Server
Dayofweek	Dayofweek
DateTime	DateTime
Set Server	Set Server
Set Que	Set Que
Set Job	Set Job
Set CNode	Set CNode
Range Int	Range Int
Range Float	Range Float
Range Dayofweek	Range Dayofweek
Range DateTime	Range DateTime
Range Size	Range Size

3. if...else statement

The format of an if statement is similar to that in C with the delimiting "{" and "}" always present:

```
if( expr ) {
     zero or more (true) STATEMENTS
}

if( expr ) {
     zero or more (true) STATEMENTS
} else {
     zero or more (false) STATEMENTS
}
```

The **expr**'s type must be either Int or Float, and after evaluation if its value is non-zero, then the true statements are executed. On the second form, if the **expr** evaluates to zero, then the false statements are executed.

Some sample **if** statements are given below:

```
if (2 * x )
{
        y = y + 3;
        print(y);
```

```
}

if (2 * x ) {
        y = y + 3;
} else {
        if( 3 * x ) {
                y = 4;
        } else {
                y = 5;
        }
}
```

4. for loop statement

The format of a for statement is as follows:

for(start; test; action) {
zero or more STATEMENTS
}

Just like in C, **for** first executes **start**, then evaluates the **test** condition to see if it returns a non-zero value. If it does, the **for** statements are executed. After the **for** statements are executed, then **action** is evaluated, and then it checks the **test** condition again in the same manner as before. **start** and **action** can be a simple assignment expression or a post-operator expression. **test** is a logical/relational expression. Some sample for statements are given in the following:

```
for (i = 0; i LT 3 ; i = i + 1)
{
        print(i);
}


for (i = 0; i LT 2 * x; i++)
{
        if (x GT 3)
       {
                y = 99;
        } else
        {
                x = 73;
        }
}
```

5. foreach loop statement

This statement is primarily used for successively retrieving each element of a Set data type: Set Server, Set CNode, Set Job, Set Que. The syntax is:

foreach (IDENTIFIER1 in IDENTIFIER2) {
zero or more STATEMENTS
}

where the following pairing of types for the identifiers are allowed:

```
IDENTIFIER1   IDENTIFIER2
```

```
===========   ===========
Server        Set Server
Que           Set Que
Job           Set Job
CNode         Set CNode
```

Example:

```
Server  s;
Que     q;
Job     j;
CNode   c;

Set Server ss;
Set Que    sq;
Set Job    sj;
Set CNode  sc;


foreach(s in ss){
        print(s);
}
foreach(q in sq){
        print(q);
}
foreach(j in sj){
        print(j);
}
foreach(c in sc){
        print(c);
}
```

6. while loop statement

The syntax of a while loop is:

while (expr) {
zero or more STATEMENTS
}

where **expr** must be of Int or Float type. If **expr** is non-zero, then the zero or more STATEMENTS are executed and **expr** is re-evaluated.

Example:

```
Int i;
i = 3;
while(i) {
        if( i EQ 0 ) {
                print("break on i = 1");
                break;
        }
        i--;
}
```

7. switch statement

The switch statement is a mult-way decision that tests whether an identifier's value matches one of a number of values, and branches to a group of statements accordingly.

The syntax for a switch statement is:

```
switch( IDENTIFIER ) {
    case constant-expr :
        {
            zero or more STATEMENTS
        }
    case constant-expr :
        {
            zero or more STATEMENTS
        }
     ...
    case in constant-rangeOrSet-expr :
        {
            zero or more STATEMENTS
        }
    case in IDENTIFIER-rangeOrSettype :
        {
            zero or more STATEMENTS
        }
    default :
        {
            zero or more STATEMENTS
        }
}
```

where **constant-expr** is an **expr** of type Int, Float, Dayofweek, DateTime, Size, String, Server, Que, Job, or CNode. **constant-rangeOrSet-expr** and **IDENTIFIER-rangeOrSettype** can be of type Set Server, Set CNode, Set Que, Set Job, Range Int, Range Float, Range Dayofweek, Range DateTime, or Range Size.

IDENTIFIER cannot be of type Void. **IDENTIFIER**'s type must be consistent with **constant-expr**'s, **constant-rangeOrSet-expr**'s, and **IDENTIFIER-rangeOrSettype**'s type as illustrated in the following table:

```
IDENTIFIER  constant-range-expr, IDENTIFIER-rangetype
========== ========================================
Server      Set Server
Que         Set Que
Job         Set Job
CNode       Set CNode
Int         Range Int
Float       Range Float
Dayofweek   Range Dayofweek
DateTime    Range DateTime
Size        Range Size
```

If a case expression matches the **IDENTIFIER**'s value, then the corresponding block of statements are executed. Unlike in C, execution does NOT fall through to the next case statement. The reason for this is that *basl2c* will translate this **switch** statement into if-elseif-else construct. The case labeled default is executed if none of the other cases are satisfied. The **default** is optional; if it isn't there, and if none of the cases match, no action takes place.

Example:

```
        Dayofweek dow;

        switch(dow)
        {
                case MON:
                {
                        print("case MON");
                }
                case TUE:
                {
                        print("case TUE");
                }
                case WED:
                {
                        print("case WED");
                }
                case THU:
                {
                        print("case THU");
                }
                case FRI:
                {
                        print("case FRI");
                }
                case SAT:
                {
                        print("case SAT");
                }
                case SUN:
                {
                        print("case SUN");
                }
                default:
                {
```

```
                print("case defaulted");
        }
}

Int a;
Range Int ri;
ri = (10, 12);
switch(a)
{
        case in (1,5):
        {
                print("case 1,5");
        }
        case in (6,9):
        {
                print("case 6,9");
        }
        case in ri:
        {
                print("case ri");
        }
}
```

8. print statement

Print statement is capable of printing to stdout the value of any **identifier** or **constant** of type Int, Float, Dayofweek, DateTime, String, Size, Que, Job, CNode, Server, Range Int, Range Float, Range Dayofweek, Range DateTime, Range Size.
The syntax is as follows:

print (IDENTIFIER);
print (constant);

Example:

```
DateTime dt;
CNode cn;

dt = (4|4|1997@12:13:36);
cn = AllNodesLocalHostGet();

print(dt);
print(cn);
```

For Set types, use **foreach** to go through each element and print as in:

```
Server s;
Set Server ss;

ss = AllServersGet();

foreach(s in ss) {
        print(s);
}
```

9. continue statement

continue ;

The **continue** statement must have been invoked within a **for, foreach,** and **while** loop. It causes the next iteration of the enclosing loop to begin.

10. break statement

break ;

The **break** statement must have been invoked within a **for, foreach,** and **while** loop. It provides an early exit from the enclosing loop.

11. return statement

return(IDENTIFIER) ;
return(constant) ;
return() ;

The return statement provides the value (if any) to be returned by a function. The type returned by **IDENTIFIER** and **constant** must match the calling function's return type. **constant** types allowed are anything except Set and Range types. The last format, **return()** is usually called within a function that doesn't return any value (like **sched_main()**).

12. exit statement

exit(constant);

where **constant** is of type Int. Calling this will terminate the scheduler.

13. Comment statement

These are statements prefixed by "//" and they are ignored by the BASL compiler.

```
// this line is ignored
Int i;  // string following the slashes is ignored
```

OPERATOR PRECEDENCE AND ASSOCIATIVITY

The following table shows the various operator precedence levels and associativity defined in the BASL language. The operators are listed in the order of decreasing precedence. The higher the precedence of an operator, the earlier it gets executed. The order in which the operators on the same level are executed depends on the associativity: left means the operators are seen from left to right, while right means they are seen from right to left.

Operator	Associativity
! ++ -- + (unary plus) - (unary minus)	right
* / %	left
+ -	left
LT LE GT GE	left

```
EQ NEQ                                  left
AND                                     left
OR                                      left
=                                       right
```

PREDEFINED FUNCTIONS

In BASL(2), a **Server** data type maps directly to a batch server object. Similarly, **CNode** is to mom/resmom, **Job** is to batch job, and **Que** is to batch queue. However, not all attributes to the PBS objects can be accessed from BASL. Only a subset of attributes, those that seemed to make sense in the context of a scheduler, are made available, and values to these attributes can be accessed by calling the following predefined functions, known also as assist/helper functions.

(1) Server-related functions

Set Server AllServersGet(void)

Returns the list of servers specified in the configuration file for which the scheduler writer wants the system to periodically check for status, queues and jobs info. See **pbs__sched__basl(8B)** for a discussion on the format of the configuration file.

CAUTION: This function must be called from inside **sched_main()** so that at every scheduling iteration, the most up to date **Set Server** structure is returned.

Server AllServersLocalHostGet(void)

Returns the Server object that represents the local host. unset value: NOSERVER. This is a simple function to call for non-cluster environments where only one server host exists.

CAUTION: This function must be called from inside **sched_main()** (or from within function called by sched_main) so that at every scheduling iteration, the most up to date **Server** structure is returned.

String ServerInetAddrGet(Server s)

Returns name for Server s. unset value: NULLSTR

String ServerDefQueGet(Server s)

Returns the default_queue attribute of Server s. unset value: NULLSTR

Int ServerStateGet(Server s)

Returns server_state attribute of Server s.

Return value:

SERVER_ACTIVE, SERVER_IDLE, SERVER_SCHED, SERVER_TERM, SERVER_TERMDELAY, -1 (unset value)

Int ServerMaxRunJobsGet(Server s)

Returns max_running attribute of Server s. unset value: 0

Int ServerMaxRunJobsPerUserGet(Server s)

Returns max_user_run attribute of Server s. unset value: 0

Int ServerMaxRunJobsPerGroupGet(Server s)

Returns max_group_run attribute of Server s. unset value: 0

Set Que ServerQueuesGet(Server s)

Returns list of queues managed by Server s.

Set Job ServerJobsGet(Server s)

Returns list of jobs managed by Server s. For obtaining a subset of this list, see **QueJobsGet()**.

Int ServerIntResAvailGet(Server s, String name)

Returns the value to resource specified in **name** that is available to jobs run by this server (Server resources_available.name attribute). Call this

function for resources with values that are of Int type. Sample resource names are: cput, pcput, walltime, mppt, pmppt, nice, procs, mppe, ncpus, pncpus, nodect, srfs_assist, mta,..., mth. For a description of these resource names, see pbs_resources_irix5(7B), pbs_resources_sp2(7B), pbs_resources_sunos4(7B), pbs_resources_unicos8(7B), pbs_server_attributes(7B), pbs_resources_irix6(7B), pbs_resources_linux(7B).

Example:

```
Int cpuAvail;
// return the # of cpus currently available in
// the server
cpuAvail = ServerIntResAvailGet(server, "ncpus");
```

Size ServerSizeResAvailGet(Server s, String name)

Returns the value to resource specified in **name** that is available to jobs run by this server (Server resources_available.name attribute). Call this function for resources with values that are of Size type. Sample resource names are: file, mem, pmem, workingset, pf, ppf, srfs_tmp, srfs_wrk, srfs_big, srfs_fast, sds, psds. For a description of these resource names, see pbs_resources_irix5(7B), pbs_resources_sp2(7B), pbs_resources_sunos4(7B), pbs_resources_unicos8(7B), pbs_server_attributes(7B), pbs_resources_irix6(7B), pbs_resources_linux(7B).

Example:

```
Size memAvail;
// return the amount of available memory in
// the server
memAvail = ServerSizeResAvailGet(server, "mem");
```

String ServerStringResAvailGet(Server s, String name)

Returns the value to resource specified in **name** that is available to jobs run by this server (Server resources_available.name attribute). Call this function for resources with values that are of String type. Sample resource names are: nodes, arch, neednodes. For a description of these resource names, see pbs_resources_irix5(7B), pbs_resources_sp2(7B), pbs_resources_sunos4(7B), pbs_resources_unicos8(7B), pbs_server_attributes(7B), pbs_resources_irix6(7B), pbs_resources_linux(7B).

Example:

```
String type;
// return the architecture (or os type) of
// the server
type = ServerStringResAvailGet(server, "arch");
```

Int ServerIntResAssignGet(Server s, String name)

Returns the value to resource specified in **name** that is allocated to running jobs (Server resources_assigned.name attribute). Call this function for resources with values that are of Int type. Sample resource names are: cput, pcput, walltime, mppt, pmppt, nice, procs, mppe, ncpus, pncpus, nodect, srfs_assist, mta,..., mth. For a description of these resource names, see pbs_resources_irix5(7B), pbs_resources_sp2(7B), pbs_resources_sunos4(7B), pbs_resources_unicos8(7B), pbs_server_attributes(7B), pbs_resources_irix6(7B), pbs_resources_linux(7B).

Example:

```
Int cpuAssn;
// return the # of cpus currently assigned in
```

```
// the server
cpuAssn = ServerIntResAssignGet(server, "ncpus");
```

Size ServerSizeResAssignGet(Server s, String name)

Returns the value to resource specified in **name** that is allocated to running jobs (Server resources_assigned.name attribute). Call this function for resources with values that are of Size type. Sample resource names are: file, mem, pmem, workingset, pf, ppf, srfs_tmp, srfs_wrk, srfs_big, srfs_fast, sds, psds. For a description of these resource names, see pbs_resources_irix5(7B), pbs_resources_sp2(7B), pbs_resources_sunos4(7B), pbs_resources_unicos8(7B), pbs_server_attributes(7B), pbs_resources_irix6(7B), pbs_resources_linux(7B).

Example:

```
Size sdsAssn;
// return the amount of sds space currently assigned
// in the server
sdsAssn = ServerSizeResAssignGet(server, "sds");
```

String ServerStringResAssignGet(Server s, String name)

Returns the value to resource specified in **name** that is allocated to running jobs (Server resources_assigned.name attribute). Call this function for resources with values that are of String type. Sample resource names are: nodes, arch, neednodes. For a description of these resource names, see pbs_resources_irix5(7B), pbs_resources_sp2(7B), pbs_resources_sunos4(7B), pbs_resources_unicos8(7B), pbs_server_attributes(7B), pbs_resources_irix6(7B), pbs_resources_linux(7B).

Set CNode ServerNodesGet(Server s)

Returns the set of nodes managed by server s. unset value: EMPTYSETCNODE.

NOTE: You can usually call the following functions for the nodes returned by this call: CNodeStateGet(), CNodePropertiesGet(), and CNodeTypeGet().

Int ServerNodesQuery(Server s, String spec)

Issues a request to the specified server to query the availability of resources specified in **spec.** At the present time, the only resource specification allowed is one that involves "nodes" and it can be of the format "nodes", "nodes=", or "nodes=<type>". The query results can be accessed by calling the following functions: ServerNodesNumAvailGet(), ServerNodesNumAllocGet(), ServerNodesNumRsvdGet(), ServerNodesNumDownGet().

NOTE: This is a wrapper to the pbs_rescquery(3B) server function.

Return value:

SUCCESS, FAIL

Int ServerNodesNumAvailGet(Server s)

Returns the number of nodes available for those managed by the specified server, or as reflected by the most recent query specified by ServerNodesQuery(). If the return value is zero, then this means that some number of nodes currently needed to satisfy the specification of ServerNodesQuery() are currently unavailable. The request maybe satisfied at some later time. If the result is negative, no combination of known nodes can satisfy the specification.

Int ServerNodesNumAllocGet(Server s)

Returns the number of nodes allocated for those managed by the specified

server, or as reflected by the most recent query specified by ServerNodes-Query().

Int ServerNodesNumRsvdGet(Server s)
Returns the number of nodes reserved for those managed by the specified server, or as reflected by the most recent query specified by ServerNodes-Query().

Int ServerNodesNumDownGet(Server s)
Returns the number of nodes down for those managed by the specified server, or as reflected by the most recent query specified by ServerNodes-Query().

Int ServerNodesReserve(Server s,String spec,Int resId)
Issues a request to the specified server to reserve the resources specified in **spec.** A value of 0 for **resId** means that this is for doing a new reservation. Otherwise, the number will represent an existing (partial) reservation. Resources currently reserved for this **resId** will be released and the full reservation will be attempted again. 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).
NOTE: This is a wrapper to the pbs_rescreserve(3B) server function.

Return value:
a reference number to a successful or partially-successful reservation, or FAIL

Int ServerNodesRelease(Server s, Int resId)
This releases or frees resources reserved with the reference number specified in **resId.**
NOTE: This is a wrapper to the pbs_rescrelease(3B) server function.

Return value:
SUCCESS, or FAIL

(2) Que-related functions:

String QueNameGet(Que que)
Returns name of Que que. unset value: NULLSTR

Int QueTypeGet(Que que)
Returns queue_type attribute of Que que.
Return value: QTYPE_E (Execution), QTYPE_R (Routing), -1 (unset value)

Int QueNumJobsGet(Que que)
Returns number of jobs residing in Que que. unset value: 0

Int QueMaxRunJobsGet(Que que)
Returns max_running attribute of Que que. unset value: 0

Int QueMaxRunJobsPerUserGet(Que que)
Returns max_user_run attribute of Que que. unset value: 0

Int QueMaxRunJobsPerGroupGet(Que que)
Returns max_group_run attribute of Que que. unset value: 0

Int QuePriorityGet(Que que)
Returns Priority attribute of Que que. unset value: 0

Int QueStateGet(Que que)
Returns started attribute of Que que - the job execution selection state of the que: SCHED_DISABLED, SCHED_ENABLED. unset value: SCHED_DISABLED

Set Job QueJobsGet(Que que)
Returns the list of jobs currently residing in que.

Int QueIntResAvailGet(Que q, String name)
Returns the value to resource specified in **name** that is available to jobs running from this q (Que resources_available.name attribute). Call this function for resources with values that are of Int type. Sample resource names are: cput, pcput, walltime, mppt, pmppt, nice, procs, mppe, ncpus, pncpus, nodect, srfs_assist, mta,..., mth. For a description of these resource names, see pbs_resources_irix5(7B), pbs_resources_sp2(7B), pbs_resources_sunos4(7B), pbs_resources_unicos8(7B), pbs_server_attributes(7B), pbs_resources_irix6(7B), pbs_resources_linux(7B).

Size QueSizeResAvailGet(Que q, String name)
Returns the value to resource specified in **name** that is available to jobs running from this q (Que resources_available.name attribute). Call this function for resources with values that are of Size type. Sample resource names are: file, mem, pmem, workingset, pf, ppf, srfs_tmp, srfs_wrk, srfs_big, srfs_fast, sds, psds. For a description of these resource names, see pbs_resources_irix5(7B), pbs_resources_sp2(7B), pbs_resources_sunos4(7B), pbs_resources_unicos8(7B), pbs_server_attributes(7B), pbs_resources_irix6(7B), pbs_resources_linux(7B).

String QueStringResAvailGet(Que q, String name)
Returns the value to resource specified in **name** that is available to jobs running from this q (Que resources_available.name attribute). Call this function for resources with values that are of String type. Sample resource names are: nodes, arch, neednodes. For a description of these resource names, see pbs_resources_irix5(7B), pbs_resources_sp2(7B), pbs_resources_sunos4(7B), pbs_resources_unicos8(7B), pbs_server_attributes(7B), pbs_resources_irix6(7B), pbs_resources_linux(7B).

Int QueIntResAssignGet(Que q, String name)
Returns the value to resource specified in **name** that is allocated to jobs running from this queue (Que resources_assigned.name attribute). Call this function for resources with values that are of Int type. Sample resource names are: cput, pcput, walltime, mppt, pmppt, nice, procs, mppe, ncpus, pncpus, nodect, srfs_assist, mta,..., mth. For a description of these resource names, see pbs_resources_irix5(7B), pbs_resources_sp2(7B), pbs_resources_sunos4(7B), pbs_resources_unicos8(7B), pbs_server_attributes(7B), pbs_resources_irix6(7B), pbs_resources_linux(7B).

Size QueSizeResAssignGet(Que q, String name)
Returns the value to resource specified in **name** that is allocated to jobs running from this q (Que resources_assigned.name attribute). Call this function for resources with values that are of Size type. Sample resource names are: file, mem, pmem, workingset, pf, ppf, srfs_tmp, srfs_wrk, srfs_big, srfs_fast, sds, psds. For a description of these resource names, see pbs_resources_irix5(7B), pbs_resources_sp2(7B), pbs_resources_sunos4(7B), pbs_resources_unicos8(7B), pbs_server_attributes(7B), pbs_resources_irix6(7B), pbs_resources_linux(7B).

String QueStringResAssignGet(Que q, String name)
Returns the value to resource specified in **name** that is allocated to jobs running from this q (Que resources_assigned.name attribute). Call this function for resources with values that are of String type. Sample resource names are: nodes, arch, neednodes. For a description of these

resource names, see pbs_resources_irix5(7B), pbs_resources_sp2(7B), pbs_resources_sunos4(7B), pbs_resources_unicos8(7B), pbs_server_attributes(7B), pbs_resources_irix6(7B), pbs_resources_linux(7B).

(3) Job-related functions

String JobIdGet(Job job)
Returns job identifier of Job job. unset value: NULLSTR

String JobNameGet(Job job)
Returns Job_Name attribute of Job job. unset value: NULLSTR

String JobOwnerNameGet(Job job)
Returns Job_Owner attribute of Job job. unset value: NULLSTR

String JobEffectiveUserNameGet(Job job)
Returns euser attribute of Job job.

String JobEffectiveGroupNameGet(Job job)
Returns egroup attribute of Job job. unset value: NULLSTR

Int JobStateGet (Job job)
Returns job_state attribute of Job job.

Return value:
TRANSIT, QUEUED, HELD, WAITING, RUNNING, EXITING, -1 (unset value)

Int JobPriorityGet(Job job)
Returns Priority attribute of Job job. unset value: 0

Int JobRerunFlagGet(Job job)
Returns Rerunable attribute of Job job.
Return value: FALSE, TRUE, -1 (unset value)

Int JobInteractiveFlagGet(Job job)
Returns interactive attribute of Job job.
Return value: FALSE, TRUE. unset value: FALSE

DateTime JobDateTimeCreatedGet(Job job)
Returns the ctime attribute of Job job. unset value: (0|0|0@-1:-1:-1)

String JobEmailAddrGet(Job job)
Returns the Mail_Users attribute of Job job. unset value: NULLSTR

String JobStageinFilesGet(Job job)
Returns the stagein attribute of Job job. unset value: NULLSTR

String JobStageoutFilesGet(Job job)
Returns stageout attribute of Job job. unset value: NULLSTR

Int JobIntResReqGet(Job job, String name)
Returns the value to resource specified in **name** as required by the job (Job Resource_List.name attribute). Call this function for resources with values that are of Int type. Sample resource names are: cput, pcput, walltime, mppt, pmppt, nice, procs, mppe, ncpus, pncpus, nodect, srfs_assist, mta,..., mth. For a description of these resource names, see pbs_resources_irix5(7B), pbs_resources_sp2(7B), pbs_resources_sunos4(7B), pbs_resources_unicos8(7B), pbs_server_attributes(7B), pbs_resources_irix6(7B), pbs_resources_linux(7B).

Example:

```
Int cputReq;
// returns the cput requirement of the job
cputReq = JobIntResReqGet(job, "cput");
```

Size JobSizeResReqGet(Job job, String name)

Returns the value to resource specified in **name** as required by the job (Job Resource_List.name attribute). Call this function for resources with values that are of Size type. Sample resource names are: file, mem, pmem, workingset, pf, ppf, srfs_tmp, srfs_wrk, srfs_big, srfs_fast, sds, psds. For a description of these resource names, see pbs_resources_irix5(7B), pbs_resources_sp2(7B), pbs_resources_sunos4(7B), pbs_resources_unicos8(7B), pbs_server_attributes(7B), pbs_resources_irix6(7B), pbs_resources_linux(7B).

Example:

```
Size memReq;
// returns the memory requirement of the job
memReq = JobSizeResReqGet(job, "mem");
```

String JobStringResReqGet(Job job, String name)

Returns the value to resource specified in **name** as required by the job (Job Resource_List.name attribute). Call this function for resources with values that are of String type. Sample resource names are: nodes, arch, neednodes. For a description of these resource names, see pbs_resources_irix5(7B), pbs_resources_sp2(7B), pbs_resources_sunos4(7B), pbs_resources_unicos8(7B), pbs_server_attributes(7B), pbs_resources_irix6(7B), pbs_resources_linux(7B).

Example:

```
String nodes;
// returns the nodes requirement property of
// the job
nodes = JobStringResReqGet(job, "nodes");
```

Int JobIntResUseGet(Job job, String name)

Returns the value to resource specified in **name** used by the job (Job resources_used.name attribute). Call this function for resources with values that are of Int type. Sample resource names are: cput, pcput, walltime, mppt, pmppt, nice, procs, mppe, ncpus, pncpus, nodect, srfs_assist, mta,..., mth. For a description of these resource names, see pbs_resources_irix5(7B), pbs_resources_sp2(7B), pbs_resources_sunos4(7B), pbs_resources_unicos8(7B), pbs_server_attributes(7B), pbs_resources_irix6(7B), pbs_resources_linux(7B).

Example:

```
Int walltUse;
// returns the amount of walltime used by
// the job
walltUse = JobIntResUseGet(job, "walltime");
```

Size JobSizeResUseGet(Job job, String name)

Returns the value to resource specified in **name** used by the job (Job resources_used.name attribute). Call this function for resources with values that are of Size type. Sample resource names are: file, mem, pmem, workingset, pf, ppf, srfs_tmp, srfs_wrk, srfs_big, srfs_fast, sds, psds. For a description of these resource names, see pbs_resources_irix5(7B), pbs_resources_sp2(7B), pbs_resources_sunos4(7B), pbs_resources_unicos8(7B), pbs_server_attributes(7B), pbs_resources_irix6(7B), pbs_resources_linux(7B).

Example:

```
Size srfsUse;
// returns the amount of srfs_fast used by
// the job
srfsUse = JobSizeResUseGet(job, "srfs_fast");
```

String JobStringResUseGet(Job job, String name)

Returns the value to resource specified in **name** used by the job (Job resources_used.name attribute). Call this function for resources with values that are of String type. Sample resource names are: nodes, arch, neednodes. For a description of these resource names, see pbs_resources_irix5(7B), pbs_resources_sp2(7B), pbs_resources_sunos4(7B), pbs_resources_unicos8(7B), pbs_server_attributes(7B), pbs_resources_irix6(7B), pbs_resources_linux(7B).

(4) CNode-related functions

Set CNode AllNodesGet(void)

Returns list of nodes that are managed by the server running on the local host. This could also include those nodes that were specified in the scheduler configuration file for which the scheduler writer wants the system to periodically check for information like state, property, and so on. See **pbs_sched_basl(8B)** for a discussion of configuration file format.

CAUTION: This function must be called from inside **sched_main()** so that at every scheduling iteration, the most up to date **Set CNode** structure is returned. Do not call this from an assignment statement intended to initialize a global variable, as the statement will only be called once.

CNode AllNodesLocalHostGet(void)

Returns the CNode object that represents the local host. This is a simple function to call for non-clustered systems where only 1 CNode exists. unset value: NOCNODE

CAUTION: This function must be called from inside **sched_main()** (or from within functions called by sched_main) so that at every scheduling iteration, the most up to date **CNode** structure is returned. Do not call this from an assignment statement intended to initialize a global variable, as the statement will only be called once.

String CNodeNameGet(CNode node)

Returns the unique (official) name of the node (i.e. ResMom hostname in a 1 mom/node model). This returns the same string that was specified in the configuration file. unset value: NULLSTR

String CNodeOsGet(CNode node)

Returns the os architecture of the node (i.e. "irix5", "sp2"). unset value: NULLSTR

Int CNodeStateGet(CNode node)

Returns the node's state.

Return value:

CNODE_OFFLINE, CNODE_DOWN, CNODE_FREE, CNODE_RESERVE, CNODE_INUSE_EXCLUSIVE, CNODE_INUSE_SHARED, CNODE_UNKNOWN

Int CNodeTypeGet(CNode node)

Returns the node's type.

Return value:

CNODE_TIMESHARED, CNODE_CLUSTER, CNODE_UN-

KNOWN

String CNodePropertiesGet(CNode node)

Returns the comma-separated list of other names the node is known by (properties, other network name). For example, "babbage.nas.nasa.gov" maybe the node name, but it could also be known via "babbage1, babbage2". unset value: NULLSTR

String CNodeVendorGet(CNode node)

Returns the name of the vendor for the hardware of the machine (i.e. "sgi", "ibm"). unset value: NULLSTR

Int CNodeNumCpusGet(CNode node)

Returns the number of processors attached to the node. unset value: -1

Size CNodeMemTotalGet(CNode node, String type)

Returns total memory of **type** for the node. **type** is an arbitrary string that the scheduler writer defines in the scheduler configuration file. unset value: -1b

Example:

```
// get total physical memory
CNodeMemTotalGet(node, "real")
// get total virtual memory
CNodeMemTotalGet(node, "virtual")
```

Size CNodeMemAvailGet(CNode node, String type)

Returns available memory of **type** for the node. **type** is an arbitrary string that the scheduler writer defines in the scheduler configuration file. unset value: -1b

So sample calls will be:

```
// get available physical memory
CNodeMemAvailGet(node, "real")
// get available virtual memory
CNodeMemAvailGet(node, "virtual")
```

Int CNodeIdletimeGet(CNode node)

Returns number of seconds in which no keystroke or mouse movement has taken place on any terminal connected to the node. unset value: -1

Float CNodeLoadAveGet(CNode node)

Returns node's load average for all cpus. unset value: -1.0

Int CNodeCpuPercentIdleGet(CNode node)

Returns the percent of idle time that all the processors of the node have experienced.

Int CNodeCpuPercentSysGet(CNode node)

Returns the percent of time that all the processors of the node have spent running kernel code.

Int CNodeCpuPercentUserGet(CNode node)

Returns the percent of time that all the processors of the node have spent running user code.

Int CNodeCpuPercentGuestGet(CNode node)

Returns the percent of time that all the processors of the node have spent running a guest operating system.

Int CNodeNetworkBwGet(CNode node, String type)

Returns the bandwidth of the node's network of **type** in bytes/second. **type** is defined by the scheduler writer in the scheduler configuration file. unset value: -1

Some sample calls are:

```
CNodeNetworkBwGet( node, "hippi" );
CNodeNetworkBwGet( node, "fddi" );
```

Size CNodeDiskSpaceTotalGet(CNode node, String name)

Returns the node's total space on disk identified by **name** where **name** is the device name arbitrarily defined by the scheduler writer in the scheduler configuration file. unset value: -1b

Example:

```
CNodeDiskSpaceTotalGet( node, "/scratch2" );
```

Size CNodeDiskSpaceAvailGet(CNode node, String name)

Returns the node's available space on disk identified by **name** where **name** is arbitrarily defined by the scheduler writer in the scheduler configuration file. unset value: -1b

Example:

```
CNodeDiskSpaceAvailGet( node, "/scratch1" );
```

Size CNodeDiskSpaceReservedGet(CNode node, String name)

Returns the node's reserved space on disk (user quota?) identified by **name** where **name** is arbitrarily defined by the scheduler writer in the scheduler configuration file. unset value: -1b

Example:

```
CNodeDiskSpaceReservedGet( node, "/scratch1" );
```

Int CNodeDiskInBwGet(CNode node, String name)

Returns the write bandwith (bytes/sec) of the node's disk identified by **name**. unset value: -1

Example:

```
CNodeDiskInBwGet( node, "/fast" );
```

Int CNodeDiskOutBwGet(CNode node, String name)

Returns read bandwith (bytes/sec) of the node's disk identified by **name**. unset value: -1

Example:

```
CNodeDiskOutBwGet( node, "/big" );
```

Size CNodeSwapSpaceTotalGet(CNode node, String name)

Returns the node's total space on swap identified by **name** where **name** is arbitrarily defined by the scheduler writer in the scheduler configuration file. unset value: -1b

Example:

```
CNodeSwapSpaceTotalGet( node, "primary" );
```

Size CNodeSwapSpaceAvailGet(CNode node, String name)

Returns node's available space on swap identified by **name** where **name** is the device name arbitrarily defined by the scheduler writer in the scheduler configuration file. unset value: -1b

Example:

```
CNodeSwapSpaceAvailGet( node, "secondary" );
```

Int CNodeSwapInBwGet(CNode node, String name)

Returns swapin rate of the node's swap device identified by **name.**

Example:

```
CNodeSwapInBwGet(node, "secondary");
```

Int CNodeSwapOutBwGet(CNode node, String name)

Returns the swapout rate of the node's swap device identified by **name.** unset value: -1

Example:

```
CNodeSwapOutBwGet(node, "primary");
```

Size CNodeTapeSpaceTotalGet(CNode node, String name)
Returns the node's total space on tape identified by **name** where **name** is arbitrarily defined by the scheduler writer in the scheduler configuration file. unset value: -1b
Example:
```
CNodeTapeSpaceTotalGet(node, "4mm");
```

Size CNodeTapeSpaceAvailGet(CNode node, String name)
Returns the node's available space on tape identified by **name** where **name** is arbitrarily defined by the scheduler writer in the scheduler configuration file. unset value: -1b
Example:
CNodeTapeSpaceAvailGet(node, "8mm");

Int CNodeTapeInBwGet(CNode node, String name)
Returns the write bandwith (bytes/sec) of the node's tape identified by **name**. unset value: -1
Example:
```
CNodeTapeInBwGet( node, "4mm" );
```

Int CNodeTapeOutBwGet(CNode node, String name)
Returns the read bandwith (bytes/sec) of the node's tape identified by **name**. unset value: -1
Example:
```
CNodeTapeOutBwGet( node, "8mm" );
```

Size CNodeSrfsSpaceTotalGet(CNode node, String name)
Returns the node's total space on srfs device identified by **name** where **name** is arbitrarily defined by the scheduler writer in the scheduler configuration file. unset value: -1b
Example:
```
CNodeSrfsSpaceTotalGet(node, "/fast");
```

Size CNodeSrfsSpaceAvailGet(CNode node, String name)
Returns the node's available space on srfs device identified by **name** where **name** is arbitrarily defined by the scheduler writer in some configuration file. unset value: -1b
Example:
```
CNodeSrfsSpaceAvailGet( node, "/big" );
```

Size CNodeSrfsSpaceReservedGet(CNode node, String name)
Returns the node's total amount of reserved space on srfs device identified by **name** where **name** is arbitrarily defined by the scheduler writer in the scheduler configuration file. unset value: -1b
Example:
```
CNodeSrfsSpaceReservedGet( node, "/fast" );
```

Int CNodeSrfsInBwGet(CNode node, String name)
Returns the write bandwith (bytes/sec) of the node's srfs device identified by **name**. unset value: -1
Example:
CNodeSrfsInBwGet(node, "/fast");

Int CNodeSrfsOutBwGet(CNode node, String name)
Returns the read bandwith (bytes/sec) of the node's srfs device identified by **name**. unset value: -1
Example:
CNodeSrfsOutBwGet(node, "/big");

(5) Miscellaneous Functions

DateTime datetimeGet()
gets the current date/time.

Int datetimeToSecs(DateTime dt)
returns the # of seconds since epoch (beginning of UNIX time - 00:00:00, January 1, 1970) for the given date/time **dt.**

Int JobAction(Job job, Int action, String param)
Performs **action** on **job** with a **param** specified depending on the action. **action** can be: SYNCRUN, ASYNCRUN, DELETE, RERUN, HOLD, RELEASE, SIGNAL, MODIFYATTR, MODIFYRES where:

Action	Description
SYNCRUN	runs the job synchronously, meaning the call to JobAction() will only return when the job has started running or when an error has been encountered. Param value: name of host(s) to run job under.
ASYNCRUN	runs the job asynchronously, meaning the call to JobAction() will return immediately as soon as the run request is validated by the PBS server, and not necessarily when the job has started execution. Param value: name of host(s) to run job under.
DELETE	deletes the job. Param value: "deldelay=<# of secs>" - delay # of seconds between the sending of SIGTERM and SIGKILL to the job before getting deleted.
RERUN	reruns the running job, which involves terminating the session leader of the job and returning the job to the queued state.
HOLD	places one or more holds on the job. Param value: "u", "o", "s", "uo", "os",

```
                        "uos"
                        - type of holds to place
                          on job: u(ser), o(ther),
                          s(ystem).

  RELEASE               removes or releases
                        holds placed on jobs.
                        Param value:
                         "u", "o", "s", "uo", "os",
                         "uos"
                         - type of holds to remove
                           from job: u(ser), o(ther),
                           s(ystem).

  SIGNAL                sends a signal to the
                        executing job.
                        Param value:
                          "HUP", "SIGHUP",...

  MODIFYATTR            modifies the specified
                        attribute of the job to
                        the given value, when
                        the attrib_name is
                        != "Resource_List" or
                        "resources_used".
                        Param value:
                          "attrib_name=value"

  MODIFYRES             modifies the job's
                        Resource_List
                        attribute given the
                        res_name and the
                        res_value:
                        Resource_List.res_name=
                                      res_value
                        Param value:
                         "res_name=res_val"
```

param value depends on the action. Specify NULLSTR if no value for this parameter is desired.
Return value: SUCCESS or FAIL.
NOTE: Any unrecognized **action** is ignored.
Example:

```
  // run Job j synchronously
  JobAction(j, SYNCRUN, NULLSTR);

  // run Job j asynchronously on host "db"
  JobAction(j, ASYNCRUN, "db");

  // delete Job j
  JobAction(j, DELETE, NULLSTR);

  // delete Job j with a delay of 5 secs
  // between the sending of SIGTERM and
  // SIGKILL
```

```
JobAction(j, DELETE, "deldelay=5");

// rerun Job j
JobAction(j, RERUN, NULLSTR);

// place a u(ser) hold on Job j
JobAction(j, HOLD, "u");

// place an o(ther) hold on Job j
JobAction(j, HOLD, "o");

// place a s(ystem) hold on Job j
JobAction(j, HOLD, "s");

// place a default hold (u) on Job j
JobAction(j, HOLD, NULLSTR);

// release u(ser) hold from Job j
JobAction(j, RELEASE, "u");

// release o(ther) hold from Job j
JobAction(j, RELEASE, "o");

// release s(ystem) hold from Job j
JobAction(j, RELEASE, "s");

// release default hold (u) from Job j
JobAction(j, RELEASE, NULLSTR);

// send SIGHUP signal to Job j
JobAction(j, SIGNAL, "SIGHUP");

// update the comment attribute of Job
// j to "a message".
// The param format is: attribute_name=new_value
// Consult PBS documentation for a list of job
// attribute names that can be specified.
JobAction(j, MODIFYATTR, "comment=a message");
// update the Resource_List.cput attribute of Job
// j to 3600 seconds.
// The param format is: resource_name=new_value
// See pbs_resources* man page for a list of
// resource_names that can be specified.
JobAction(j, MODIFYRES, "cput=3600");
```

QueJobFind(Que que,Fun Int func,Int cpr,Int value);

QueJobFind(Que que,Fun String func,Int cpr,String value);

QueJobFind(Que que,Fun DateTime func,Int cpr,DateTime value);

QueJobFind(Que que,Fun Size func,Int cpr,Size value);

where **cpr** is one of: OP_EQ, OP_NEQ, OP_LE, OP_LT, OP_GE, OP_GT. **func** is a function whose ONLY argument is of Job type. **Job** is the return type.

Description: Applies **func** to every job in **que**, and return the first job that satisfies the logical comparison: **func(job) cpr value**

Example:

```
Size JobVirtualMemAvailGet(Job job)
{
    Size sz;

    sz = JobSizeResReqGet(job, "mem");
    return(sz);
}
Int JobWallTimeReqGet(Job job)
{
    Int wallt;

    wallt = JobIntResReqGet(job, "walltime");
    return(wallt);
}

Int JobCpuTimeUsedGet(Job job)
{
    Int cput;

    cput = JobIntResUseGet(job, "cput");
    return(cput);
}

Que findQueByName(Set Que queues, String qname)
{
        Que q;

        foreach(q in queues) {
                if( QueNameGet(q) EQ qname ) {
                        return(q);
                }
        }
        return(NOQUE);
}
sched_main()
{
   Server s;
   Que    que;
   Set Que sq;

   // get local server
   s = AllServersLocalHostGet();

   // get the queues of the Server s
   sq = ServerQueuesGet(s);

   // get the queue named "fast" from the
   // local server
   que = findQueByName( sq, "fast" );
```

```
    // Find the 1st job whose walltime requirement
    // is == 300s:
    QueJobFind(que, JobWallTimeReqGet, OP_EQ, 300);

    // Find the 1st job whose email address to
    // notify about job activity != "bayucan":
    QueJobFind(que, JobEmailAddrGet, OP_NEQ,
                                        "bayucan");

    // Find the 1st job that was created after
    // or on 3/3/1997:
    QueJobFind(que, JobDateTimeCreatedGet, OP_GE,
                                          (3|3|1997));

    // Find the 1st job that was created after
    // 3:3:44:
    QueJobFind(que, JobDateTimeCreatedGet, OP_GT,
                                           (3:3:44));

    // Find the 1st job that was created after
    // 3:3:44 on 3/3/1997:
    QueJobFind(que, JobDateTimeCreatedGet, OP_GT,
                                   (3|3|1997@3:3:44));

    // Find the 1st job whose cpu time used < 1600s:
    QueJobFind(que, JobCpuTimeUsedGet, OP_LT, 1600);

    // Find the 1st job whose virtual memory
    // requirement <= 300mb:
    QueJobFind(que, JobVirtualMemAvailGet, OP_LE,
                                             300mb);
}
```

Job QueJobFind(Que que, Fun Int func, Int cpr)

Job QueJobFind(Que que, Fun String func, Int cpr)

Job QueJobFind(Que que, Fun DateTime func, Int cpr)

Job QueJobFind(Que que, Fun Size func, Int cpr)

where **cpr** can be one of the following: OP_MAX, OP_MIN, **func** is a function whose only argument is of Job type.

Description: Returns the Job with the max or min value found for **func(job)** as it is applied to every job in **que**.

Example:

```
Int JobCpuTimeReqGet(Job job)
{
  Int cput;

  cput = JobIntResReqGet(job, "cput");
  return(cput);
}
sched_main()
{
```

```
  Que que;
  Job job;

  // Find the Job with the highest cpu time
  // requirement:
  job = QueJobFind(que, JobCpuTimeReqGet, OP_MAX);

  // Find the Job with the minimum cpu time
  // requirement:
  job = QueJobFind(que, JobCpuTimeReqGet, OP_MIN);
}
```

Que QueFilter(Que que,Fun Int func,Int cpr,Int value)

Que QueFilter(Que que,Fun String func,Int cpr,String value)

Que QueFilter(Que que,Fun DateTime func,Int cpr,Date value)

Que QueFilter(Que que,Fun Size func,Int cpr,Size value)

where **cpr** can be one of the following: OP_EQ, OP_NEQ, OP_LE, OP_LT, OP_GE, OP_GT, **func** is a function whose only argument is of Job type.

Description: Applies **func** to every job in **que**, and returns a new que containing all jobs that satisfies the comparison condition: **func(job) cpr value**

Example:

```
Int JobWallTimeReqGet(Job job)
{
  Int wallt;

  wallt = JobIntResReqGet(job, "walltime");
  return(wallt);
}
sched_main()
{
  Que que;
  Que newq;

  // Returns a new que containing all jobs in "que"
  // with a walltime requirement == 300s:
  newq = QueFilter(que, JobWallTimeReqGet, OP_EQ, 300);

  // Returns a new que containing all jobs in "que"
  // with an email address != "bayucan":
  newq = QueFilter(que, JobEmailAddrGet, OP_NEQ, "bayucan");

  // Returns a new que containing all jobs in "que"
  // created after or on 3/3/1997:
  newq = QueFilter(que, JobDateTimeCreatedGet, OP_GE,
                                        (3|3|1997));

  // Returns a new que containing all jobs in "que"
  // created after 3:3:44:
  newq = QueFilter(que, JobDateTimeCreatedGet, OP_GT,
```

```
                                                    (3:3:44));

        // Returns a new que containing all jobs in "que"
        // created after 3:3:44 on 3/3/1997:
        newq = QueFilter(que, JobDateTimeCreatedGet, OP_GT,
                                        (3|3|1997@3:3:44));

        // NOTE: The original "que" is not modified
        // whatsoever.
      }
```

Int Sort(Set Job s, Fun Int key, Int order)

Int Sort(Set Job s, Fun String key, Int order)

Int Sort(Set Job s, Fun Float key, Int order)

Int Sort(Set Job s, Fun DateTime key, Int order)

Int Sort(Set Job s, Fun Size key, Int order)

where **s** the set of jobs to sort. **key** is the sorting key which is a function whose only argument is of Job type, **order** is the sorting order: ASC, DESC.

Description: sorts the elements of **s**, in either ASCending or DESCending order of values that were returned by the **key** function, as applied to every member of the set of jobs. The **s** object is modified with this call. This returns SUCCESS or FAIL depending on outcome of the sort.

Examples:

```
Size JobMemReqGet(Job job)
{
      Size mem;

      mem = JobSizeResReqGet(job, "mem");
      return(mem);
}

sched_main()
{
      Server  master;

      Set Job jobs;

      Int     order;

      // get local server
      master = AllServersLocalHostGet();

      jobs = ServerJobsGet(master);
      Sort(jobs, JobPriorityGet, ASC);
      Sort(jobs, JobIdGet, DESC);
      order = ASC;
      Sort(jobs, JobDateTimeCreatedGet, order);
      order = DESC;
      Sort(jobs, JobMemReqGet, order);
}
```

Int Sort(Set Que s, Fun Int key, Int order)

Int Sort(Set Que s, Fun String key, Int order)

Int Sort(Set Que s, Fun Float key, Int order)

Int Sort(Set Que s, Fun DateTime key, Int order)

Int Sort(Set Que s, Fun Size key, Int order)

where **s** the set of queues to sort. **key** is the sorting key which is a function whose only argument is of Que type, **order** is the sorting order: ASC, DESC.

Description: sorts the elements of **s**, in either ASCending or DESCending order of values that were returned by the **key** function, as applied to every member of the set of queues. The **s** object is modified with this call. This returns SUCCESS or FAIL depending on outcome of the sort.

Examples:

```
Size QueMemAvailGet(Que que)
{
      Size mem;

      mem = QueSizeResAvailGet(que, "mem");
      return(mem);
}

sched_main()
{
      Server  master;

      Set Que ques;
      Int     order;

      // get local server
      master = AllServersLocalHostGet();

      ques = ServerQueuesGet(master);
      Sort(ques, QuePriorityGet, ASC);
      Sort(ques, QueNameGet, ASC);
      order = DESC;
      Sort(ques, QueMemAvailGet, order);
}
```

Int Sort(Set Server s, Fun Int key, Int order)

Int Sort(Set Server s, Fun String key, Int order)

Int Sort(Set Server s, Fun Float key, Int order)

Int Sort(Set Server s, Fun DateTime key, Int order)

Int Sort(Set Server s, Fun Size key, Int order)

where **s** the set of servers to sort. **key** is the sorting key which is a function whose only argument is of Server type, **order** is the sorting order: ASC, DESC.

Description: sorts the elements of **s**, in either ASCending or DESCending order of values that were returned by the **key** function, as applied

to every member of the set of servers. The **s** object is modified with this call. This returns SUCCESS or FAIL depending on outcome of the sort.

Examples:

```
Size ServerMemAvailGet(Server serv)
{
      Size mem;

      mem = ServerSizeResAvailGet(serv, "mem");
      return(mem);
}

sched_main()
{
      Set Server sserver;

      Int     order;

      Int     ret;

      sserver = AllServersGet();

      ret = Sort(sserver, ServerMaxRunJobsGet, ASC);
      Sort(sserver, ServerInetAddrGet, ASC);

      order = DESC;
      Sort(sserver, ServerMemAvailGet, order);
}
```

Int Sort(Set CNode s, Fun Int key, Int order)

Int Sort(Set CNode s, Fun String key, Int order)

Int Sort(Set CNode s, Fun Float key, Int order)

Int Sort(Set CNode s, Fun DateTime key, Int order)

Int Sort(Set CNode s, Fun Size key, Int order)

where **s** the set of nodes to sort. **key** is the sorting key which is a function whose only argument is of CNode type, **order** is the sorting order: ASC, DESC.

Description: sorts the elements of **s**, in either ASCending or DESCending order of values that were returned by the **key** function, as applied to every member of the set of nodes. The **s** object is modified with this call. This returns SUCCESS or FAIL depending on outcome of the sort.

Examples:

```
Size CNodeMyMemAvailGet(CNode cn)
{
      Size mem;

      mem = CNodeMemAvailGet(cn, "virtual");
      return(mem);
}

sched_main()
```

```
{
      Set CNode scnode;

      Int     order;

      scnode = AllNodesGet();

      Sort(scnode, CNodeIdletimeGet, ASC);
      Sort(scnode, CNodeNameGet, ASC);
      order = DESC;
      Sort(scnode, CNodeMyMemAvailGet, order);
}
```

CNode..Get() FUNCTIONS

The return values of the CNode..Get() functions discussed in the previous section are obtained by sending resource queries to the CNode's MOM at every scheduling iteration. For example, `CNodeLoadAveGet(node)` will return the value obtained from some <host resource> query (this could be the string "loadave") as sent to the node's MOM. The "<host resource> -> CNode..Get()" mappings are established internally, but they can be modified or more mappings can be added via the scheduler configuration file. The config file is discussed in **pbs_sched_basl(8B).** Mappings already established are given in the following:

For all architectures:

CNode..Get() actual call	host resource
CNodeOsGet(node)	arch
CNodeLoadAveGet(node)	loadave
CNodeIdletimeGet(node)	idletime

SEE ALSO

pbs_sched_basl(8B) pbs_job_attributes(7B), pbs_queue_attributes(7B), pbs_resources_irix5(7B), pbs_resources_sp2(7B), pbs_resources_sunos4(7B), pbs_resources_unicos8(7B), pbs_server_attributes(7B), and pbs_server(8B), pbs_resources_irix6(7B), pbs_resources_linux(7B).

Figure 5−2 shows how the BASL data types Server, Set Server, Que, Set Que, Job, Set Job relate to one another. The items in italics are the predefined functions that must be called in a BASL program in order to instantiate the data types. The figure also illustrates the PBS framework: in a scheduling system, one or more server hosts exist, and each server manages one or more queues, and one or more jobs are spooled inside a queue. Jobs are run on one or more computational nodes (machines). A machine could act both as a server and a computation node.

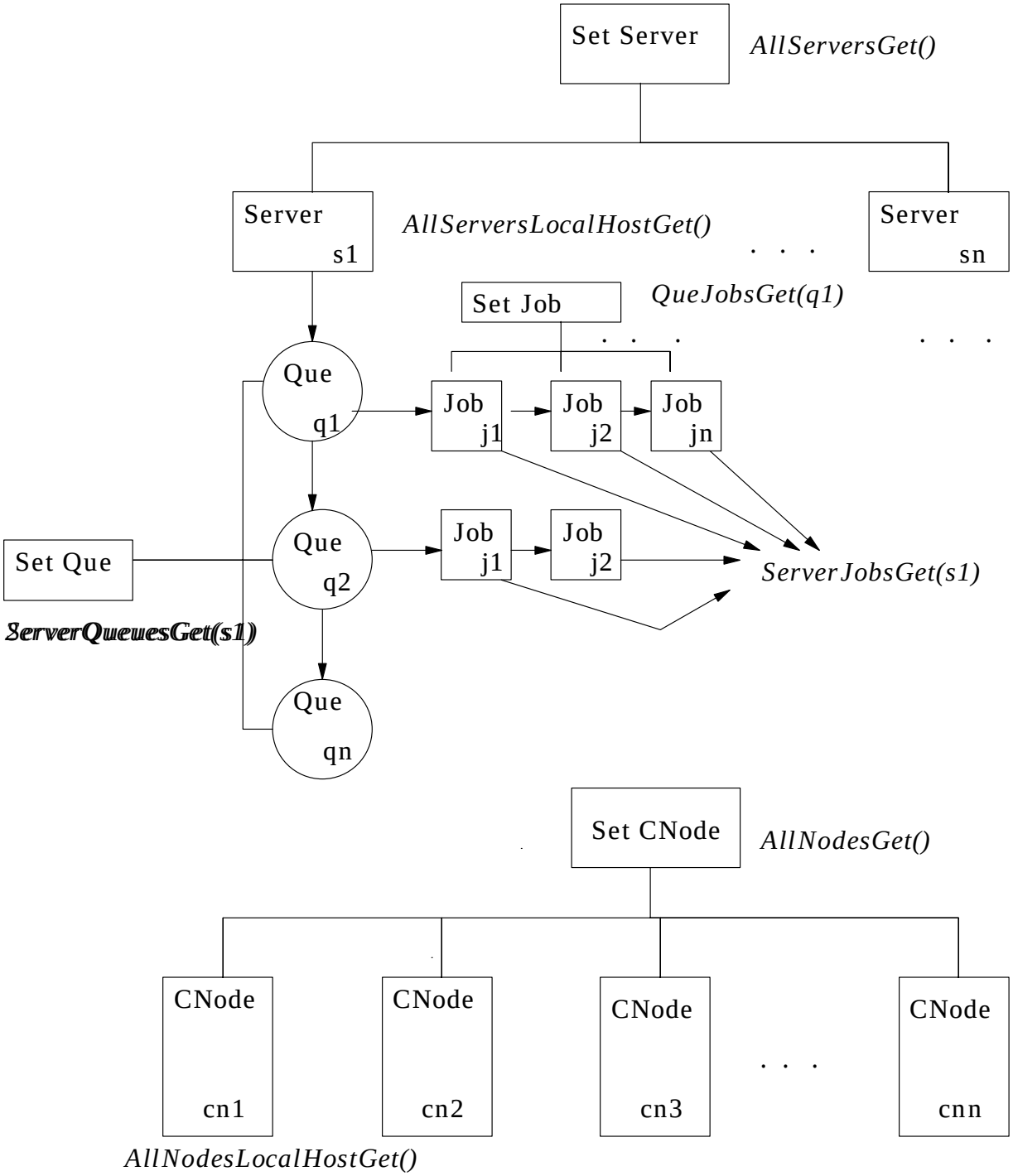


Figure 5−2: BASL Complex Data Types

5.2.15. Graphical User Interface: xpbs

xpbs is the graphical user interface to PBS user and operator commands. The current implementation will actually call the PBS commands like qsub, qstat, qselect, and so on. In addition, 2 new commands were created specifically for xpbs: xpbs_scriptload, and xpbs_datadump. These 2 binaries are compiled using the same libraries used by the standard PBS commands.

xpbs is written in Tcl/Tk. The way it works is that a main driver script/program called "xpbs" is what a user invokes, and this script calls other programs found in XPBS_LIB directory (as set in Makefile).

NAME

xpbs − GUI front end to PBS commands

SYNOPSIS

xpbs [-admin]

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.

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 property 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 listbox.
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**.

*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 checkbutton.

*selectHosts
list of hosts (space separated) to automatically select/highlight in the HOSTS listbox.

*selectQueues
list of queues (space separated) to automatically select/highlight 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	**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

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).

5.2.16. Graphical User Interface: xpbsmon

xpbsmon is the graphical user interface for displaying, monitoring the nodes/execution hosts under PBS.

xpbsmon is also written in Tcl/Tk. The way it works is that a main driver script/program called "xpbsmon" is what a user invokes, and this script calls other programs found in XPBSMON_LIB directory (as set in Makefile).

NAME

xpbsmon − GUI for displaying, monitoring the nodes/execution hosts under PBS

SYNOPSIS

xpbsmon

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 and properties as these 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.

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 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 Buttonshows 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 buttonshows the full view of Site Information where nodes are represented in bigger boxes, still colored depending on the status, and some information 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 sub-divided 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 is colored according to the status of the node that it represents, and if the view type is FULL, it will 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, *nodeColorDOWN, *nodeColorRSVD, *nodeColorOFFL.

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,

properties 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.

*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 scrollbars.

*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 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

pbs_sched_tcl(8B), pbs_mom(8B), the PBS External Reference Specification, and the PBS Internal Design Specification.

6. Operator Commands

A batch operator is a user of the system who is granted privilege to perform actions beyond those available to the normal user. Typical of those actions are starting and stopping of individual queues and the server itself, and modification or deleting the jobs of other users.

The operator will be able to use the following general user commands with increased privilege: qalter, qdel, qhold, qmove, qmsg, qrerun, qrls, qsig, and qstat. Additionally, the operator will have access to the new commands described in this section: qdisable, qenable, qrun, qstart, qstop, qterm, pbs_mom, pbs_sched, pbs_server, and pbsnodes.

6.1. Initiate Server Operation

NAME

pbs_server − start a pbs batch server

SYNOPSIS

pbs_server [-a active] [-d config_path] [-p port] [-A acctfile] [-L logfile] [-M mom_port] [-R momRPP_port] [-S scheduler_port] [-t type]

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.

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.

-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_SERVER_HOME which is typically `/usr/spool/PBS`.

-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.

-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.

-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).

-R mom_RPPport — Specifies the port number on which the the server should query the up/down status of Mom. See the -R option for pbs_mom(8).

-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.

-t type
: Specifies the impact on jobs which were in execution, running, when the server shut down. If the running job is not rerunnable or restartable from a checkpoint image, the job is aborted. If the job is rerunnable or restartable, then the actions described below are taken. When the *type* argument is:

  hot
  : All jobs are requeued except non-rerunnable jobs that were executing. Any rerunnable job which was executing when the server went down will be run immediately. 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.

  warm
  : All rerunnable jobs which were running when the server went down are requeued. All other jobs are maintained. New selections are made for which jobs are placed into execution. Warm is the default if -t is not specified.

  cold
  : All jobs are deleted. Positive confirmation is required before this direction is accepted.

  create
  : The server will discard any existing configuration files, queues and jobs, and initialize configuration files to the default values. The server is idled.

FILES

$PBS_SERVER_HOME/server_priv
: default directory for configuration files, typically /usr/spool/pbs/server_priv

$PBS_SERVER_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.

SIGINT
: Causes an orderly shutdown of pbs_server, identical to "qterm".

SIGTERM
: Causes an orderly shutdown of pbs_server, identical to "qterm".

SIGSHUTDN
: On systems (Unicos) where SIGSHUTDN is defined, it also causes an orderly 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

qsub (1B), pbs_connect(3B), pbs_mom(8B), pbs_sched_basl(8B), pbs_sched_tcl(8B), pbsnodes(8B), qdisable(8B), qenable(8B), qmgr(8B), qrun(8B), qstart(8B), qstop(8B), qterm(8B), and the PBS External Reference Specification.

6.2. Initiate Execution Mini-server (MOM) Operation

NAME

pbs_mom − start a pbs batch execution mini-server

SYNOPSIS

pbs_mom [-C chkdirectory] [-c config] [-d directory] [-L logfile] [-M MOMport] [-R RPPport] [-p | -r] [-x]

DESCRIPTION

The **pbs_mom** command starts the operation of a batch **M**achine **O**riented **M**ini−server, MOM, 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 command is not runnable by the general user community, the server will only execute if its real and effective uid is zero.

One function of pbs_mom is to place jobs into execution as directed by the server, establish resource usage limits, monitor the job's usage, and notify the server when the job completes. If they exist, pbs_mom will execute a prologue script before executing a job and an epilogue script after executing the job. The next function of pbs_mom is to respond to resource monitor requests. This was done by a separate process in previous versions of PBS but has now been combined into one process. The resource monitor function is provided mainly for the PBS scheduler. It provides information about the status of running jobs, memory available etc. The next function of pbs_mom is to respond to task manager requests. This involves communicating with running tasks over a tcp socket as well as communicating with other MOMs within a job (aka a "sisterhood").

Pbs_mom will record a diagnostic message in a log file for any error occurrence. The log files are maintained in the *mom_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

-C chkdirectory — Specifieds the path of the directory used to hold checkpoint files. [Currently this is only valid on Cray systems.] The default directory is PBS_HOME/spool/checkpoint, see the -d option. The 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.

-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_priv, see the -d option. If the specified file cannot be opened, pbs_mom will abort. If the -c option is not supplied, pbs_mom will 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 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. 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 will open a file named for the current date in the PBS_HOME/mom_logs directory, see the -d option.

-M port — Specifies the port number on which the mini-server (MOM) will listen for batch requests.

-R port
: Specifies the port number on which the mini-server (MOM) will listen for resource monitor requests, task manager requests and inter-MOM messages. Both a UDP and a TCP port of this number will be used.

-p
: Specifies the impact on jobs which were in execution when the mini-server shut down. On any restart of MOM, the new mini-server will not be the parent of any running jobs, MOM has lost control of her offspring (not a new situation for a mother). With the -p option, Mom will allow the jobs to continue to run and monitor them indirectly via polling. The -p option is mutually exclusive with the -r option.

-r
: Specifies the impact on jobs which were in execution when the mini-server shut down. With the -r option, MOM will kill any processes belonging to jobs, mark the jobs as terminated, and notify the batch server which owns the job. The -r option is mutual exclusive with the -p option.

  Normally the mini-server is started from the system boot file without the -p or 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.

  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.

-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.

-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: 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 %xxx %yyy
```

If a query for "escape" is sent with no qualifiers, the command executed would be "echo %xxx %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, "localhost" and the name returned to pbs_mom by the system call gethostname(). These names need not be specified in the configuration file. The hosts listed as "clienthosts" comprise a "sisterhood" of machines. Any one of the sisterhood will accept connections from a server from within the sisterhood. They will also accept Resource Monitor (RM) requests and Internal MOM (IM) messages from within the sisterhood. For a sisterhood to be able to communicate IM messages to each other, they must all share the same RM port.

restricted

which causes a host name to be added to the list of hosts which will be allowed to connect to MOM without needing to use a privilaged 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. 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.

cputmult

which sets a factor used to adjust cpu time used by a job. This is provided to allow adjustment of time charged and limits enforced where the job might run on systems with different cpu performance. If Mom's system is faster

than the reference system, set cputmult to a decimal value greater than 1.0. If Mom's system is slower, set cputmult to a value between 1.0 and 0.0. For example:

```
$cputmult 1.5
$cputmult 0.75
```

wallmult
which sets a factor used to adjust wall time usage by to job to a common reference system. The factor is used for walltime calculations and limits the same as cputmult is used for cpu time.

The configuration file must be "secure". It must be owned by a user id and group id less than 10 and not be world writtable.

FILES

$PBS_SERVER_HOME/mom_priv
the default directory for configuration files, typical (/usr/spool/pbs)/mom_priv.

$PBS_SERVER_HOME/mom_logs
directory for log files recorded by the server.

$PBS_SERVER_HOME/mom_priv/prologue
the administrative script to be run before job execution.

$PBS_SERVER_HOME/mom_priv/eiplogue
the administrative script to be run after job execution.

Signal Handling

Pbs_mom handles the following signals:

SIGHUP
causes pbs_mom 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 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

pbs_server(8B), pbs_scheduler_basl(8B), pbs_scheduler_tcl(8B), the PBS External Reference Specification, and the PBS Administrator's Guide.

6.3. Disable Queue

NAME

qdisable − disable input to a pbs destination

SYNOPSIS

qdisable destination ...

DESCRIPTION

The **qdisable** command directs that a destination should no longer accept batch jobs.

The qdisable command sends a *Manage* request to the batch server specified by *destination*. The enabled queue attribute is set to FALSE. If the batch request is accepted, the server 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.

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

pbs_server(8B), qmgr(8B), and qenable(8B)

6.4. Enable Queue

NAME

qenable − enable input to a pbs destination

SYNOPSIS

qenable destination ...

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*. The enabled queue attribute is set to TRUE. If the request is accepted, that server will accept *Queue Job* requests which specified the queue.

In order to execute qenable, the user must have PBS Operation or Manager privilege.

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

pbs_server(8B), qdisable(8B), and qmgr(8B)

6.5. Run Job

NAME

qrun − run a pbs batch job

SYNOPSIS

qrun [-H host] job_identifier ...

DESCRIPTION

The **qrun** command is used to force a batch server to initiate the execution of a batch job. The job is run regardless of scheduling position, resource requirements, or state.

In order to execute qrun, the user must have PBS Operation or Manager privilege.

OPTIONS

-H host — Specifies the host within the cluster on which the job(s) are to be run. The *host* argument is the name of a host that is a member of the cluster of hosts managed by the server. If the option is not specified, the server will select the "worst possible" host on which to execute the job.

OPERANDS

The qrun command accepts one or more *job_identifier* operands of the form:

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

See the description under "Job Identifier" in section 2.7.6 in the ERS.

STANDARD ERROR

The qrun 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 qrun command, the exit status will be a value of zero.

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

SEE ALSO

pbs_server(8B), pbs_sched(8B), qmgr(8B)

6.6. Start Queue

NAME

qstart − start pbs batch job processing at a destination

SYNOPSIS

qstart destination ...

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.

The qstart command sends a *Manage* request to the batch server specified by *destination*. The started queue attribute is set to TRUE.

In order to execute qstart, the user must have PBS Operation or Manager privilege.

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

pbs_server(8B), qstop(8B), and qmgr(8B)

6.7. Stop Queue

NAME

qstop − stop pbs batch job processing at a destination

SYNOPSIS

qstop destination ...

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.

The qstop command sends a *Manage* request to the batch server specified by *destination*. The started queue attribute is set to FALSE.

In order to execute qstop, the user must have PBS Operation or Manager privilege.

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

pbs_server(8B), qstart(8B), and qmgr(8B)

6.8. Terminate Server Operation

NAME

qterm − terminate processing by a pbs batch server

SYNOPSIS

qterm [-t type] [server...]

DESCRIPTION

The **qterm** command terminates a batch server. The command sends a *Server Shutdown* batch request to the server. When a server receives a Shutdown request, The server will go into *Terminating* state. No new jobs will be allowed to be started into execution nor enqueued into the server. The impact on jobs currently being run by the server depends on the type of shut down requested as described below. The qterm command will not exit until the server has completed it shutdown procedure.

In order to execute qterm, the user must have PBS Operation or Manager privilege.

OPTIONS

-t type Specifies the type of shut down. The types are:

immediate

This is the default action if the *-t* option is not specified. All running jobs are to immediately stop execution. The value passed in the *Shutdown* request is {SHUT_IMMEDIATE}.

If checkpoint is supported, running jobs that can be checkpointed are checkpointed, terminated, and requeued. If checkpoint is not supported or the job cannot be checkpointed, running jobs are requeued if the rerunable attribute is true. Otherwise, jobs are killed.

Normally the server will not shutdown until there are no jobs in the running state. If the server is unable to contact the MOM of running job, the job is still listed as running. The server may be forced down by a second "qterm -t immediate" command.

delay If checkpoint is supported, running jobs that can be checkpointed are checkpointed, terminated, and requeued. If a job cannot be checkpointed, but can be rerun, the job is terminated and requeued. Otherwise, running jobs are allowed to continue to run. Note, the operator or administrator may use the qrerun and qdel commands to remove running jobs. The value passed in the *Shutdown* request is {SHUT_DELAY}.

quick This option is used when you wish that running jobs be left running when the server shuts down. The server will cleanly shutdown and can be restarted when desired. Upon restart of the server, jobs that continue to run are shown as running; jobs that terminated during the server's absence will be placed into the exiting state. The value passed in the *Shutdown* request is

OPERANDS

The *server* operand specifies which servers are to shutdown. If no servers are given, then the default server will be terminated.

STANDARD ERROR

The qterm 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 qterm command, the exit status will be a value of zero.

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

SEE ALSO

pbs_server(8B), qmgr(8B), pbs_resources_aix4(7B), pbs_resources_irix5(7B), pbs_resources_sp2(7B), pbs_resources_sunos4(7B), and pbs_resources_unicos8(7B)

7. Administrator Commands

A batch administrator is a user of the system who is granted the highest level of privilege in the batch system. The batch administrator is able to perform all operator functions and additionally modify queue and server configurations.

In addition to the general user commands available to an operator with increased privilege and the special operator commands, the administrator has access to the **qmgr** command. Information on the Resource Monitor and Schedulers are also provided for the administrator.

There are three different schedulers provided with PBS. The yacc/lex based scheduler is driven by a procedure written in a "C - like language." This BAtch Scheduler Language (BASL) scheduler is sometimes referred to (incorrectly) as the "rules" based scheduler.

Another scheduler based on Tcl[1] another is a C language framework. is provided for sites which wish a different approach. The third scheduler is a framework (main program) written in the C language. The site must supply the primary function (in C) which implements the scheduling policy.

[1] Tool Control Langague - see "Tcl and the Tk Toolkit" by John K. Ousterhout.

7.1. PBS Job Schedulers

7.1.1. C Based Job Scheduler

NAME

pbs_sched_cc − pbs C scheduler

SYNOPSIS

pbs_sched [-a alarm] [-d home] [-L logfile] [-p file] [-S port] [-R port] [-c file]

DESCRIPTION

The **pbs_sched** program runs in conjunction with the PBS server. It queries the server about the state of PBS and communicates with **pbs_resmon** to get information about the status of running jobs, memory available etc. It then makes decisions as to what jobs to run.

pbs_sched must be executed with root permission.

OPTIONS

-a alarm — This specifies the time in seconds to wait for a schedule run to finish. If a script 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 180 seconds.

-d home — This specifies the PBS home directory, PBS_HOME. The current working directory of the scheduler is PBS_HOME/sched_priv. If this option is not given, PBS_HOME defaults to $PBS_SERVER_HOME as defined during the PBS build procedure.

-L logfile — Specifies an absolute path name of the file to use as the log file. If not specified, the scheduler will open a file named for the current date in the PBS_HOME/sched_logs directory (see the -d option).

-p file — This specifies the "print" file. Any output from the C code which is written to standard out or standard error will be written to this file. If this option is not given, the file used will be *PBS_HOME/sched_priv/sched_out*. See the -d option.

-S port — This specifies the port to use. If this option is not given, the default port for the PBS scheduler is used.

-R port — This specifies the resource monitor port to use. If this option is not given, the default port for the PBS mom is used. NOTE: this option only makes the mom port available to the scheduler writer. It doesn't force them to use it.

-c file — Specify a configuration file, see description below. If this is a relative file name it will be relative to PBS_HOME/sched_priv, see the -d option. If the -c option is not supplied, pbs_sched will not attempt to open a configuration file.

The options that specify file names may be absolute or relative. If they are relative, their root directory will be PBS_HOME/sched_priv.

USAGE

This version of the scheduler requires knowledge of the C language and the PBS API. Source code is provided for a main program for the scheduler. The site must supply the heart of the program. When invoked, the main program performs general initialization

and housekeeping chores. Then a locally supplied function, *schedinit()* is called to perform site specific initialization.

In the main loop, a locally supplied function, *schedule()* is called to make the scheduling decisions and perform any required actions. Information about jobs and queues is obtained from the Server through the standard PBS API as found in libifl.a. Information about the execution host(s) is obtained from the Resource Monitor. Routines to communicate with the Resource Monitor are found in libnet.a.

If the processing takes more than the allotted time, the scheduler will restart itself. The default amount of time is three minutes. This can be changed with the -a option.

On receipt of a SIGHUP signal, the scheduler will close and reopen its log file and reread its configuration file (if any).

CONFIGURATION FILE

A configuration file may be specified with the -c option. This file may be used to specify the hosts (servers) which are allowed to connect to pbs_sched. The hosts are specified in the configuration file in a manor identical to that used in pbs_mom. There is one line per host with the syntax:

```
$clienthost    hostname
```

where clienthost and hostname are separated by white space.

Two host names are always allowed to connection to pbs_sched, "localhost" and the name returned to pbs_sched by the system call gethostname(). These names need not be specified in the configuration file.

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/sched_priv
: the default directory for configuration files, typically (/usr/spool/pbs)/sched_priv.

Signal Handling

A C based scheduler will handle the following signals:

SIGHUP
: The server will close and reopen its log file and reread the config file if one exists.

SIGALRM
: If the site supplied scheduling module 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

Upon normal termination, an exit status of zero is returned.

SEE ALSO

pbs_sched_rule(8B), pbs_sched_tcl(8B), pbs_server(8B), pbs_mom(8B), the PBS External Reference Specification, and the PBS Internal Design Specification.
PBS Internal Design Specification

7.1.2. Yacc/lex Based Job Scheduler

NAME

pbs_sched_basl − pbs BASL scheduler

SYNOPSIS

pbs_sched [-d home] [-L logfile] [-p print_file] [-a alarm] [-S port] [-c configfile]

DESCRIPTION

The **pbs_sched** command starts the operation of a batch scheduler on the local host. It runs in conjunction with the PBS server. It queries the server about the state of PBS and communicates with **pbs_mom** to get information about the status of running jobs, memory available etc. It then makes decisions as to what jobs to run.

Typically, this command will be in a local boot file such as */etc/rc.local*.

pbs_sched must be executed with root permission.

OPTIONS

-d home

Specifies the name of the PBS home directory, PBS_HOME. If not specified, the value of $PBS_SERVER_HOME as defined at compile time is used. Also see the -L option.

-L logfile

Specifies an absolute path name of the file to use as the log file. If not specified, the scheduler will open a file named for the current date in the PBS_HOME/sched_logs directory. See the -d option.

-p print_file

This specifies the "print" file. Any output from the scheduler code which is written to standard out or standard error will be written to this file. If this option is not given, the file used will be $PBS_HOME/sched_priv/sched_out. See the -d option.

-a alarm

This specifies the time in seconds to wait for a schedule run to finish. If a scheduling iteration 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 180 seconds.

-S port

Specifies a port on which to talk to the server. This option is not required. It merely overides the default PBS scheduler port.

-c configfile

Specify a configuration file, see description below. If this is a relative file name it will be relative to PBS_HOME/sched_priv, see the -d option. If the -c option is not supplied, pbs_sched will not attempt to open a configuration file. In BASL, this config file is almost always needed because it is where the list of servers, nodes, and host resource queries are specified by the administrator.

USAGE

This version of the scheduler requires knowledge of the BASL language. The site must first write a function called *sched_main()* (and all functions supporting it) using BASL constructs, and then translate the functions into C using the BASL compiler *basl2c*, which would also attach a main program to the resulting code. This main program performs general initialization and housekeeping chores such as setting up local socket to communicate with the server running on the same machine, cd-ing to the priv directory,

opening log files, opening configuration file (if any), setting up locks, forking the child to become a daemon, initializing a scheduling cycle (i.e. get node attributes that are static in nature), setting up the signal handlers, executing global initialization assignment statements specified by the scheduler writer, and finally sitting on a loop waiting for a scheduling command from the server. When the server sends the scheduler an appropriate scheduling command {SCH_SCHEDULE_NEW, SCH_SCHEDULE_TERM, SCH_SCHEDULE_TIME, SCH_SCHEDULE_RECYC, SCH_SCHEDULE_CMD, SCH_SCHEDULE_FIRST}, information about server(s), jobs, queues, and execution host(s) are obtained, and then *sched_main()* is called.

SCHEDULING LANGUAGE

The BAtch Scheduling Language (BASL) is a C-like procedural language. It provides a number of constructs and predefined functions that facilitate dealing with scheduling issues. Information about a PBS server, the queues that it owns, jobs residing on each queue, and the computational nodes where jobs can be run, are accessed via the BASL data types Server, Que, Job, CNode, Set Server, Set Que, Set Job, and Set CNode.

The following simple sched_main() will cause the server to run all queued jobs on the local server:

```
sched_main()
{
        Server  s;
        Que     q;
        Job     j;
        Set Que queues;
        Set Job jobs;

        s = AllServersLocalHostGet(); // get local server
        queues = ServerQueuesGet(s);

        foreach( q in queues ) {
                jobs = QueJobsGet(q);
                foreach( j in jobs ) {
                        JobAction(j, SYNCRUN, NULLSTR);
                }
        }

}
```

For a more complete discussion of the Batch Scheduler Language, see **basl2c(1B)**.

CONFIGURATION FILE

A configuration file may be specified with the -c option. This file is used to specify the (1) hosts which are allowed to connect to pbs_sched, (2) the list of server hosts for which the scheduler writer wishes the system to periodically check for status, queues, and jobs info, (3) list of execution hosts for which the scheduler writer wants the system to periodically check for information like state, property, and so on, and (4) various queries to send to each execution host.

(1) specifying client hosts:

The hosts allowed to connect to pbs_sched are specified in the configuration file in a manner identical to that used in pbs_mom. There is one line per host using the syntax:

```
$clienthost   hostname
```

where `clienthost` and `hostname` are separated by white space. Two host names are always allowed to connection to pbs_sched: "localhost" and the name returned to pbs_sched by the system call gethostname(). These names need not be specified in the configuration file.

(2) specifying list of servers:

The list of servers is specified in a one host per line manner, using the syntax:

```
$serverhost hostname port_number
```

or where `$server_host, hostname,` and `port_number` are separated by white space.

If `port_number` is 0, then the default PBS server port will be used.

Regardless of what has been specified in the file, the list of servers will always include the local server - one running on the same host where the scheduler is running.

Within the BASL code, access to data of the list of servers is done by calling *AllServersGet()*, or *AllServersLocalHostGet()* which returns the local server on the list.

(3) specifying the list of execution hosts:

The list of execution hosts (nodes), whose MOMs are to be queried from the scheduler, is specified in a one host per line manner, using the syntax:

```
$momhost hostname port_number
```

where `$momhost, hostname,` and `port_number` are separated by white space.

If `port_number` is 0, then the default PBS MOM port will be used.

The BASL function *AllNodesGet()* , or *ServerNodesGet(AllServersLocalHostGet())* is available for getting the list of nodes known to the local system.

(4) specifying the list of host resources:

For specifying the list of host resource queries to send to each execution host's MOM, the following syntax is used:

```
$node node_name CNode..Get host_resource
```

`node_name` should be the same hostname string that was specified in a `$momhost` line. A `node_name` value of "*" (wildcard) means to match any node.

Please consult section 9 of the PBS ERS (Resource Monitor/Resources) for a list of possible values to `host_resource` parameter.

`CNode..Get` refers to the actual function name that is called from the scheduler code to obtain the return values to host resource queries. The list of `CNode..Get` function names that can appear in the configuration file are:

STATIC:
==========
CNodePropertiesGet
CNodeVendorGet
CNodeNumCpusGet
CNodeOsGet
CNodeMemTotalGet[type]
CNodeNetworkBwGet[type]
CNodeSwapSpaceTotalGet[name]
CNodeDiskSpaceTotalGet[name]
CNodeDiskInBwGet[name]
CNodeDiskOutBwGet[name]
CNodeTapeSpaceTotalGet[name]
CNodeTapeInBwGet[name]
CNodeTapeOutBwGet[name]
CNodeSrfsSpaceTotalGet[name]
CNodeSrfsInBwGet[name]
CNodeSrfsOutBwGet[name]

DYNAMIC:
==========
CNodeIdletimeGet
CNodeLoadAveGet
CNodeMemAvailGet[type]
CNodeSwapSpaceAvailGet[name]
CNodeSwapInBwGet[name]
CNodeSwapOutBwGet[name]
CNodeDiskSpaceReservedGet[name]
CNodeDiskSpaceAvailGet[name]
CNodeTapeSpaceAvailGet[name]
CNodeSrfsSpaceReservedGet[name]
CNodeSrfsSpaceAvailGet[name]
CNodeCpuPercentIdleGet
CNodeCpuPercentSysGet
CNodeCpuPercentUserGet
CNodeCpuPercentGuestGet

STATIC function names return values that are obtained only during the first scheduling cycle, or when the scheduler is instructed to reconfig; whereas, DYNAMIC function names return attribute values that are taken at every subsequent scheduling cycle.

name and **type** are arbitrarily defined. For example, you can choose to have **name** defined as "$FASTDIR" for the CNodeSrfs* calls, and a sample configuration file entry would look like:

```
$node unicos8 CNodeSrfsSpaceAvailGet[$FASTDIR]
                    quota[type=ares_avail,dir=$FASTDIR]
```

So in a BASL code, if you call CNodeSrfsSpaceAvailGet(node, "$FASTDIR"), then it will return the value to the query "quota[type=ares_avail,dir=$FASTDIR]" (3rd parameter) as sent to the node's MOM.

By default, the scheduler has already internally defined the following mappings,

which can be overriden in the configuration file:

```
keyword  node_name   CNode..Get         host_resource
=======  =========   ================   =============
$node    *           CNodeOsGet         arch
$node    *           CNodeLoadAveGet    loadave
$node    *           CNodeIdletimeGet   idletime
```

The above means that for all declared nodes (via $momhost), the host queries `arch`, `loadave`, and `idletime` will be sent to each node's MOM. The value to `arch` is obtained internally by the system during the first scheduling cycle because it falls under STATIC category, while values to `loadave` and `idletime` are taken at every scheduling iteration because they fall under the DYNAMIC category. Access to the return values is done by calling *CNodeOsGet(node)*, *CNodeLoadAveGet(node)*, and *CNodeIdletimeGet(node)*, respectively. The following are some sample $node arguments that you may put in the configuration file.

```
node_name            CNode..Get                          host res
==================   =========================           ==========
<sunos4_nodename>    CNodeIdletimeGet                    idletime
<sunos4_nodename>    CNodeLoadAveGet                     loadave
<sunos4_nodename>    CNodeMemTotalGet[real]              physmem
<sunos4_nodename>    CNodeMemTotalGet[virtual]           totmem
<sunos4_nodename>    CNodeMemAvailGet[virtual]           availmem

<irix5_nodename>     CNodeNumCpusGet                     ncpus
<irix5_nodename>     CNodeMemTotalGet[real]              physmem
<irix5_nodename>     CNodeMemTotalGet[virtual]           totmem
<irix5_nodename>     CNodeIdletimeGet                    idletime
<irix5_nodename>     CNodeLoadAveGet                     loadave
<irix5_nodename>     CNodeMemAvailGet[virtual]           availmem

<linux_nodename>     CNodeNumCpusGet                     ncpus
<linux_nodename>     CNodeMemTotalGet[real]              physmem
<linux_nodename>     CNodeMemTotalGet[virtual]           totmem
<linux_nodename>     CNodeIdletimeGet                    idletime
<linux_nodename>     CNodeLoadAveGet                     loadave
<linux_nodename>     CNodeMemAvailGet[virtual]           availmem

<solaris5_nodename> CNodeIdletimeGet                     idletime
<solaris5_nodename> CNodeLoadAveGet                      loadave
<solaris5_nodename> CNodeNumCpusGet                      ncpus
<solaris5_nodename> CNodeMemTotalGet[real]               physmem

<aix4_nodename>      CNodeIdletimeGet                    idletime
<aix4_nodename>      CNodeLoadAveGet                     loadave
<aix4_nodename>      CNodeMemTotalGet[virtual]           totmem
<aix4_nodename>      CNodeMemAvailGet[virtual]           availmem

<unicos8_nodename>   CNodeIdletimeGet                    idletime
<unicos8_nodename>   CNodeLoadAveGet                     loadave
<unicos8_nodename>   CNodeNumCpusGet                     ncpus
<unicos8_nodename>   CNodeMemTotalGet[real]              physme
<unicos8_nodename>   CNodeMemAvailGet[virtual]           availmem
<unicos8_nodename>   CNodeSwapSpaceTotalGet[primary]     swaptotal
<unicos8_nodename>   CNodeSwapSpaceAvailGet[primary]     swapavail
<unicos8_nodename>   CNodeSwapInBwGet[primary]           swapinrate
<unicos8_nodename>   CNodeSwapOutBwGet[primary]          swapoutrate
<unicos8_nodename>   CNodePercentIdleGet                 cpuidle
<unicos8_nodename>   CNodePercentSysGet                  cpuunix
<unicos8_nodename>   CNodePercentGuestGet                cpuguest
<unicos8_nodename>   CNodePercentUsrGet                  cpuuser
<unicos8_nodename>   CNodeSrfsSpaceAvailGet[$FASTDIR]    quota[type
                                                        =ares_avail,
                                                        dir=$FASTDIR]

<unicos8_nodename>   CNodeSrfsSpaceAvailGet[$BIGDIR]     quota[type
                                                        =ares_avail,
                                                        dir=$BIGDIR]

<unicos8_nodename>   CNodeSrfsSpaceAvailGet[$WRKDIR]     quota[type
```

```
                                                    =ares_avail,
                                                    dir=$WRKDIR]

    <sp2_nodename>      CNodeLoadAveGet                 loadave
```

Suppose you have an execution host that is of irix5 os type, then the <irix5_node_name> entries will be consulted by the scheduler. The initial scheduling cycle would involve sending the STATIC queries `ncpus`, `physmem`, `totmem` to the execution host's MOM, and access to return values of the queries is done via `CNodeNumCpusGet(node)`, `CNodeMemTotalGet(node, "real")`, `CNodeMemTotalGet(node, "virtual")` respectively, where node is the CNode representation of the execution host. The subsequent scheduling cycles will only send DYNAMIC queries `idletime`, `loadave`, and `availmem`, and access to the return values of the queries is done via `CNodeIdleTimeGet(node)`, `CNodeLoadAveGet(node)`, `CNodeMemAvailGet(node, "virtual")`. respectively.

"Later" entries in the config file take precedence.

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.

On receipt of a SIGHUP signal, the scheduler will close and reopen its log file and reread its configuration file (if any).

FILES

$PBS_SERVER_HOME/sched_priv
: the default directory for configuration files, typically (/usr/spool/pbs)/sched_priv.

Signal Handling

A C based scheduler will handle the following signals:

SIGHUP
: The server will close and reopen its log file and reread the config file if one exists.

SIGALRM
: If the site supplied scheduling module 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

Upon normal termination, an exit status of zero is returned.

SEE ALSO

basl2c(1B), pbs_sched_tcl(8B), pbs_server(8B), pbs_mom(8B), the PBS External Reference Specification, and the PBS Internal Design Specification.
PBS Internal Design Specification

7.1.3. Tcl Based Job Scheduler

NAME

pbs_sched_tcl − pbs Tcl scheduler

SYNOPSIS

pbs_sched [-a alarm] [-b file] [-d home] [-i file] [-L logfile] [-p file] [-S port] [-t file] [-v] [-c file]

DESCRIPTION

The **pbs_sched** program runs in conjunction with the PBS server. It queries the server about the state of PBS and communicates with **pbs_mom** to get information about the status of running jobs, memory available etc. It then makes decisions as to what jobs to run.

pbs_sched must be executed with root permission.

OPTIONS

-a alarm — This specifies the time in seconds to wait for a schedule run to finish. If a script 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 180 seconds.

-b file — This specifies the "body" file. The file given is read into memory once at program start or after the program receives a SIGHUP and executed each time the scheduler is awakened by the server. If this option is not given, the file "sched_tcl" in the directory PBS_HOME/sched_priv is read for the body code.

-d home — This specifies the PBS home directory, PBS_HOME. The current working directory of the scheduler is PBS_HOME/sched_priv. If this option is not given, PBS_HOME defaults to $PBS_SERVER_HOME as defined during the PBS build procedure.

-i file — This specifies the "initialize" file. The file given is executed once before the main processing loop is entered. If this option is not given, no initialization code is executed.

-L logfile — Specifies an absolute path name of the file to use as the log file. If not specified, the scheduler will open a file named for the current date in the PBS_HOME/sched_logs directory (see the -d option).

-p file — This specifies the "print" file. Any output from the Tcl code which is written to standard out or standard error will be written to this file. If this option is not given, the file used will be *PBS_HOME/sched_priv/sched_out*. See the -d option.

-S port — This specifies the port to use. If this option is not given, the default port for the PBS scheduler is used.

-t file — This specifies the "terminator" file. If a QUIT command is sent from the server, this code is executed before the scheduler exits. If this option is not given, no special termination handling is done.

-v — This puts the scheduler into "verbose" mode. Any errors will be shown no matter what this may be set to, but some "uninteresting" events may be logged by using this flag. An example is a message each time the server contacts the scheduler.

-c file Specify a configuration file, see description below. If this is a relative file name it will be relative to PBS_HOME/sched_priv, see the -d option. If the -c option is not supplied, pbs_sched will not attempt to open a configuration file.

The options that specify file names may be absolute or relative. If they are relative, their root directory will be PBS_HOME/sched_priv.

USAGE

This version of the scheduler requires knowledge of the Tcl language. 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.

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 nodes'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
: 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.

The Tcl interpreter is started at program initialization and after a reset (the receipt of a SIGHUP signal). It is not deleted between scheduling runs so variables which are set in one can be accessed later.

The "initialize" and "terminator" files are run with no supplied connection to the server. This means that none of the above functions which talk to the server will work unless **pbsconnect** is called first. The "body" file is run with a connection to the server already established.

CONFIGURATION FILE

A configuration file may be specified with the -c option. This file may be used to specify

the hosts (servers) which are allowed to connect to pbs_sched. The hosts are specified in the configuration file in a manor identical to that used in pbs_mom. There is one line per host with the syntax:

```
$clienthost    hostname
```

where clienthost and hostname are separated by white space.

Two host names are always allowed to connection to pbs_sched, "localhost" and the name returned to pbs_sched by the system call gethostname(). These names need not be specified in the configuration file.

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/sched_priv
: the default directory for configuration files, typically (/usr/spool/pbs)/sched_priv.

Signal Handling

A C based scheduler will handle the following signals:

SIGHUP
: The server will close and reopen its log file and reread the config file if one exists.

SIGALRM
: If the site supplied scheduling module 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

Upon normal termination, an exit status of zero is returned.

SEE ALSO

pbs_scheduler_cc(8B), pbs_scheduler_rule(8B), pbs_server(8B), pbs_mom(8B), the PBS External Reference Specification, and the PBS Internal Design Specification.
PBS Internal Design Specification

7.2. Node Management

NAME

pbsnodes − pbs node manipulation

SYNOPSIS

pbsnodes [-{c|o|r}] [-s server] [nodename ...]
pbsnodes -{a|l} [-s server]

DESCRIPTION

The **pbsnodes** command is used to mark nodes down, free or off line. It can also be used to list nodes and their state. Node information is obtained by sending a request to the PBS job server.

If **pbsnodes** is run without options (other than **−s**), the nodes specified as operands are marked DOWN and unavailable to run jobs. It is importaint that all the nodes known to be down are given as arguments on the command line. This is because nodes which are not listed are assumed to be UP and will be indicated as such if they were previously marked DOWN. None of the other options will change the marking for nodes which are not given on the command line.

In order to execute pbsnodes with other than the **−a** or **−l** options, the user must have PBS Manager or Operator privilege.

OPTIONS

-a	All nodes and their attributes are listed. The attributes include "state" and "properities".
-c	Clear OFFLINE or DOWN from listed nodes. The listed nodes are "free" to be allocated to jobs.
-l	List all nodes marked in any way.
-o	Mark listed nodes as OFFLINE even if currently in use. This is different from being marked DOWN. An automated script that checks nodes being up or down and calls **pbsnodes** with a list of nodes down will not change the status of nodes marked OFFLINE. This gives the administrator a tool to hold a node out of service without changing the automatic script.
-r	clear OFFLINE from listed nodes.
-r	specify the PBS server to which to connect.

SEE ALSO

pbs_server(8B) and the PBS External Reference Specification

7.3. Manage Server

NAME

qmgr − pbs batch system manager

SYNOPSIS

qmgr [-a] [-c command] [-e] [-n] [-z] [server...]

DESCRIPTION

The **qmgr** command provides an administrator interface to the batch system.

The command reads directives from standard input. The syntax of each directive is checked and the appropriate request is sent to the batch server or servers.

The list or print subcommands of qmgr can be executed by general users. Creating or deleting a queue requries PBS Manager privilege. Setting or unsetting server or queue attributes requires PBS Operator or Manager privilege.

OPTIONS

-a	Abort **qmgr** on any syntax errors or any requests rejected by a server.
-c command	Execute a single *command* and exit **qmgr**.
-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.

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.

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 back-slash "\".

Comments begin with the # character and continue to end of the line. Comments and blank lines are ignored by qmgr.

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,...]]
```

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 — is to print all the queue and server attributes in a format that will be usable as input to the qmgr command.

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 effected.

attr specifies the name of an attribute of the object which is to be set or modified. The attributes of objects are described in section 2 of the ERS. 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 a 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.
- -= — decrease the current value of the attribute by the amount in the new value.

value the value to assign to an attribute. If the value includes white space, commas or other special characters, such as the # character, the value string must be inclosed in quote marks (").

The following are examples of qmgr directives:

```
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

pbs_server(8B), pbs_queue_attributes(7B), pbs_server_attributes(7B), qstart(8B), qstop(8B), qenable(8B), qdisable(8), and the PBS External Reference Specification

8. The PBS Job Scheduler

The PBS Job Scheduler is a daemon that performs job scheduling for a particular PBS server.

8.1. Scheduler Overview

The PBS Scheduler performs job scheduling upon request from a PBS Server. Except for the initial message, which is transmitted through a normal socket connection, the Server and Scheduler communicate by using functions defined in the pbs_ifl (PBS interface library).

The entity which is given the role of scheduler for PBS may be a special purpose program or even a shell script. The fundamental parts have been provided to offer maximum flexability. Two example schedulers have been included. One scheduler is based on **yacc** and **lex** and creates its own language BASL, for BAtch Scheduler Language. The other scheduler is based on One is based on **tcl** and augments an already existing language with some new commands that give access to PBS functions. The other is a C language framework.

8.2. The Scheduling Cycle

If the scheduler has been asked to initiate a scheduling cycle, it first reverses the sense of the connection between the scheduler and server, and requests information concerning the current job pool from the server, and possibly, information about the queues that the server has defined. The scheduler then evaluates the job set according to the procedure expressed in the language file, and determines if any action should be taken concerning any jobs in the pool. At the end of the cycle, the scheduler once more resumes listening on the designated port.

The scheduler may also request information from one or more Resource Monitors. Resource Monitors are daemons that provide information about the state of the resources of particular machines, or collections of processors, such as clusters or MPP partitions. Communication between the scheduler and resource Monitors is conducted over a well-known port. The protocol used between the scheduler and a resource Monitor is described in the PBS Scheduler IDS.

8.3. The Tcl Scheduler

Tcl is a flexable language which can be extended fairly easily. The library to communicate with the Resource Monitor and the regular PBS Interface Library have been wrapped into Tcl commands to create a special purpose tcl interpreter which waits to be awakened by the PBS server before swinging into action. The indispensable book for dealing with Tcl is *Tcl and the Tk Toolkit* by John K. Ousterhout.

The Tcl scheduler man page gives all the particulars about the functions which are available to access the PBS Resource Monitor and Server. The section of the ERS which describes the Resource Monitor discusses the information available to the script writter for a particular type of system.

8.4. The C Scheduler

This is a skeleton program designed as a starting point for someone to use to create a complete C language scheduler program. It consists of a main program that processes options and sets up the network connection to the server. It was patterned after the Tcl scheduler main program in the hope it would not need to be chaneged yet the scheduler algorithm could still modified easily. A directory of examples is included that show the required entry points for the scheduler writter.

[Page intentionally left bank.]

9. Resource Monitor

This section describes the PBS resource monitor functions of the Machine Oriented Miniserver (MOM). MOM is the part of PBS which is machine dependent rather than pure POSIX. The resource monitor communicates with the world (mainly the PBS scheduler) over the network. All the resource requests discussed below should be sent in as large a group as practical. This helps reduce the impact on the machine answering the request because pbs_mom tries to gather information only once for each request. Each resource reported within each request will then also be consistent with all others.

Each resource request to pbs_mom is sent with a command number telling what the rest of the request will contain. For the command "tell me about these resources", the following strings are each a separate resource request. These consist of a resource name with parameters inclosed in square brackets. Each parameter has a parameter name followed by an equal sign and then a value. For example `size[fs=/tmp]`.

9.1. Resources

The machines and operating systems supported by *pbs_mom* are

- Cray systems using *Unicos 8, 9 or 10, or Unicos MK 2*
- IBM 590 Workstations using *AIX 4*
- IBM SP using *AIX 4* with *PSSP 2.1* or *PSSP 3.1*.
- Silicon Graphics systems using *IRIX 5.x or 6.x*,
- Sun Sparc using *SunOS 4.1.x*or *Solaris 2.5 (5.5)*,
- Fujitsu VPP300 systems using *UXP/V 4.1 ES*,
- AMD/Intel/Cyrix systems using *Linux*or *FreeBSD*

All of these machines will support a standard list of resources.

arch This returns a string which specifies the machine architecture of the host being queried. By default, it will be one of: "aix4", "irix5", "fujitsu", "irix6", "irix6array", "linux", "freebsd", "solaris5", "sunos4", "unicos8" or "unicosmk2". The string returned may be changed by adding a *static resource* or *shell command* to Mom's configuration file. The value of the configuration file entry will override that of the standard return for arch.

Note, an configure file entry with the same name as any standard resource will override the standard value when reported to a resource request from a privileged port on a *client host*. A resource request from an unprivileged port on a *restricted host* will receive the standard value; the configure file entries are ignored.

cput This reports cpu time in seconds. Two different parameters are accepted. One is *proc* which can be used to specify a process, the other is *session* which can be used to specify a session. For example: `cput[proc=8765]` and `cput[session=4567]`.

idletime
This is the time in seconds in which no keystroke or mouse movement has taken place on any terminal connected to the system.

loadave
This reports the smoothed system loadave (number of processes in the kernel run queue).

mem This reports memory usage in bytes. The *proc* and *session* parameters are accepted. For example: `mem[proc=8765]` and `mem[session=4567]`.

ncpus Returns the number of processors that are available.

nsessions
Returns the number of sessions which exist in the system. Sessions owned by root are excluded. No parameters are allowed.

nusers
Reports the number of users who have processes running in the system, excluding root. No parameters are allowed.

pids Reports a list of processes for a session. A parameter *session* must be given. For example `pids[session=34615]`.

physmem
Returns the physical (main) memory size in kilobytes.

sessions
This will return a list of the sessions which exist in the system. This is different from PBS "jobs" and the sessions listed may not be part of any PBS job. Sessions owned by root are excluded. No parameters are allowed. (Was "jobs" in prior versions)

size Reports the size of file system objects in kilobytes. Two different parameters may be given. One is *file* which specifies a filename whose size is returned. The other is *fs* which specifies a directory in a file system which is examined to find the amount of free space available.

uname
This returns a string with the POSIX specified information returned from the **uname()** function spearated by spaces. The strings are in order: *sysname*, *nodename*, *release*, *version* and *machine*.

validuser
This function returns a string of either `yes` or `no` if the user name is valid, i.e. has a password entry. A parameter of `user=`*name* must be given. For example: `validuser[user=joe]`.

walltime
Reports the time in seconds which a *proc* or *session* has existed in the system.

The following resources formerly were available under prior versions of PBS built with {NEEDNODES} from pbs_mom: **avail**, **reserve**, **totpool**, and **usepool**.

They are now available from the job server, pbs_server, via the standard PBS API. See the man page for *pbs_rescquery*() for more information.

The following are general examples of resource queries to pbs_mom:

```
pids[session=456]
mem[proc=123]
size[file=/etc/passwd]
idletime
```

In the given order, these resource requests would return a list of pid's for the session "456", the memory usage for process "123", the size of the file "/etc/passwd", and the idletime of the system.

9.1.1. SunOS Resources

On a Sun system running SunOS, PBS supports several resources in additon to the above list.

availmem
Returns the virtual memory available to be used.

quota Returns information about disk quotas. Several parameters are used. The first must be *type* which can have a value of *harddata*, *softdata*, *currdata*, *hardfile*, *softfile*, *cur-*

rfile, *timedata* or *timefile*. The second must be *dir* which is used to specify the directory of the file system for which quota information is desired. The last parameter must be *user* which is used to specify the user name or id number whose quota information is to be retrieved. The type "harddata" returns the hard limit for data storage in kilobytes. The type "softdata" returns the warning limit for data storage in kilobytes. The type "currdata" returns the current usage of data storage in kilobytes. The type "hardfile" returns the hard limit for the number of files. The type "softfile" returns the warning limit for the number of files. The type "currfile" returns the current number of files. The type "timedata" returns the number of seconds that a user has left in the grace period for excessive disk use, or zero if the grace period is not active. The type "timefile" returns the number of seconds that a user has left in the grace period for having an excessive number of files, or zero if the grace period is not active.

resi Returns the resident memory size in kilobytes for a *proc* or *session*.

totmem
Returns the total virtual memory which exists in the system in kilobytes.

9.1.2. Digital Unix

The resources for systems running Digital Unix include the standard set plus:

platform
Returns the cpu type as a string.

9.1.3. FreeBSD Resources

The FreeBSD resources include the standard set plus all those supported by the Sun under SunOS except totmem and availmem.

9.1.4. SGI Resources

The Silicon Graphics resources for either Irix 5 or 6 include the standard set plus all those supported by the Sun under SunOS. If the MOM is built for "irix6array", an additional resource is available:

availmask
Return a MAXCNODES-bit string with a '1' in each position where there are two CPUs available for a node.

9.1.5. Solaris Resources

The Sun Solaris resources include the standard set with the addition of the following.

platform
Returns the cpu type as a string, for example `SUNW,Ultra-1`.

9.1.6. Fujitsu Resources

The Fujitsu UXP/V resources include the standard set with the addition of the following.

totmem
Returns the total virtual memory which exists in the system in kilobytes.

availmem
Returns the virtual memory available to be used.

9.1.7. Cray Unicos Resources

Cray Unicos has some fundamental differences from both the Sun and SGI. It does not use virtual memory and it has a much more sophisticated quota system. Some of the same resource names are used with slightly different meanings. All of the standard resources are included as well as all the resources of the SGI except "walltime". Unicos uses a periodic data gathering routine to calculate swap rate and cpu idle values. The time period that is used for

each calculation is SAMPLE_DELTA which is set to 10 seconds by default.

availmem
: Returns the memory which is available for use by programs.

cpuidle
: This is the percent of idle time that all the processors have experienced within the previous SAMPLE_DELTA seconds. This is a time filtered value.

cpuguest
: This is the percent of time that all the processors have spent running a guest operating system within the previous SAMPLE_DELTA seconds. This is a time filtered value.

cpusysw
: This is the percent of time that all the processors have spent in system wait within the previous SAMPLE_DELTA seconds. This is a time filtered value.

cpuunix
: This is the percent of time that all the processors have spent running kernel code within the previous SAMPLE_DELTA seconds. This is a time filtered value.

cpuuser
: This is the percent of time that all the processors have spent running user code within the previous SAMPLE_DELTA seconds. This is a time filtered value.

totmem
: Returns the total memory in kilobytes minus what is taken by the kernel.

swapavail
: The number of characters of free space in the swap device(s).

swapinrate
: The swap in activity within the previous SAMPLE_DELTA seconds in characters per second. This is also a time filtered value.

swapoutrate
: The swap out activity within the previous SAMPLE_DELTA seconds in characters per second. This is also a time filtered value.

swaprate
: The swap activity within the previous SAMPLE_DELTA seconds in characters per second. This includes both swap in and swap out transfers and is time filtered such that previous values of *swaprate* are used to smooth the current calculated value.

swaptotal
: The total number of characters in the swap device(s).

swapused
: The number of characters of active data in the swap device(s).

quota
: The same quota types as the Sun and SGI are supported by the C90. The cray supports several others as well. The additional types only operate if the resource monitor is compiled with the symbol `SRFS`. They are "snap_avail", "ares_avail", "res_total", "soft_res", "delta" and "reserve". Additionally, if `SRFS` is defined, the "dir" attribute can specify one of several special directories by starting with a dollar sign ($) character. The allowed special names are `$TMPDIR`, `$BIGDIR`, `$FASTDIR` or `$WRKDIR`. These names are read from the file `/etc/tmpdir.conf` so in theory they could be changed but in practice they have been hardwired into MOM and the Server as resources. The file just gives the administrator the ability to change the actual directory where space will be allocated for a user making a SRFS request.

  If the "type" attribute is one of the SRFS values, there can be no other parameters. If the "type" attribute is one of the standard values, there must be one more parameter. It can have a name of "user", "group" or "account" and a value of a name or id number.

The standard types have the same meaning as the other machines. The meanings of the SRFS types are very UNICOS specific. They are *snap_avail*, *ares_avail*, *res_total*, *soft_res*, *delta*, and *reserve*. The type "snap_avail" returns the amount of total space, in characters. The type "ares_avail" returns *snap_avail* less unused job reserved space. The type "res_total" returns the total amount of space reserved for jobs. The type "soft_res" returns "true" if soft reservation is allowed and "false" otherwise. The type "delta" returns the setting of over or under commitment. The type "reserve" returns the number of characters committed to "srfs_assist" mode jobs. For more information, see the UNICOS SRFS documentation.

srfs_reserve
: This gives the ability to set the number of characters committed to "srfs_assist" mode jobs. The number is not a delta value so if there is more than one job in "srfs_assist" mode, their requests must be added together before making this call.

9.1.8. Cray Unicos MK 2 Resources

Cray Unicos MK 2 has some of the same resources as Unicos. Quite a few are missing because Unicos MK 2 typically comes in a "binary" release that does not include header files that allow programs such as PBS's MOM to retrieve information about processes in a general way. The resources available are "cput", "totmem", "availmem", "ncpus" (the comment for this says "Number of started processors" but it does not show the number of PEs), "physmem", "size", "idletime" and "quota".

In addition to these, PBS under Unicos/MK2 support the following resources:

mppe_app
: The number of application PEs currently configured.

mppe_free
: The total number of application PEs currently unallocated.

mppe_avail
: The size of largest contiguous block of application PEs.

mppe_info
: Returns the above three resources at the same time (intended to provide a scheduler interface with lower overhead).

9.1.9. IBM SP

Basically, the SP is a special case of AIX. However, because the SP Parallel Operating Environment works outside of PBS PBS does not have access to memory usage or cpu time of tasks other than serial processes run from the job script.

9.2. Libraries

A number of libraries exist to communicate with MOM. One is specialized as a Resource Monitor interface and another as a Task Manager interface. There is also a general network interface that uses UDP. These libraries are described below.

9.2.1. Resource Monitor Library

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

9.2.2. Task Management Library

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 − task management API

```
#include <tm.h>
int tm_init(info, roots)
        void *info;
        struct tm_roots *roots;
int tm_nodeinfo(list, nnodes)
        tm_node_id [2]*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 [3]event;
int tm_obit(tid, obitval, event)
        tm_task_id tid;
        int *obitval;
        tm_event_t [4]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 [5]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 [6]event;
int tm_publish(name, info, len, event)
        char *name;
        void *info;
        int len;
```

tm_event_t [7]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 [8]event;

int tm_finalize()

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 PSCHED Task Management API description used to create this library was commited to paper on Novermber 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 implementaion 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;  /* event handle, > 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.`

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* paramenter 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** be TM_SUCCESS if the library initialization was successful, or an error return 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 followed by a colon (:). The order of the strings is **sysname, nodename, release, version, machine.**

A comma spearated set of strings giving the components of the "Resource_List" attribute of the job. 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_finalize( ) may be called to free any memory in use by the library and close the connection to MOM.

pbs_mom, PSCHED:An http://parallel.nas.nasa.gov/Psched/psched-api-report.ps

9.2.3. Reliable Packet Protocol Library

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

```
#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;

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).

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 resceived 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 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 attepmt 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.

tcp(4P), udp(4P)

[Page intentionally left bank.]

10. Security

Security in **PBS** is composed of two facets, authorization and authentication. Any user making a request of a batch server must be authorized to request that service. To provide the basis for authorization, each request contains the identity of the user making the request. To insure the user identity is correct, the identity must be authenticated in some manner.

10.1. Authorization

There is an authorization method, usually an *access control list*, or ACL, to control access to each object managed by a batch server, jobs, queues, and the server itself.

While the exact format of the entries in access control lists varies depending upon the list, they have several properties in common. Each list is maintained by the server as an editable text file. Each list entry consists of a single text line. Entries in the list may be prefixed with a single "+" or "-" character to indicate allowing or denying access. If neither character is present, allowing access is assumed. For example, a list containing:

```
-Kenneth
+Lora
-Julie
-Bob
```

Would explicitly grant access to Lora and explicitly deny access to Kenneth, Julie, and Bob.

Note that while each entry in the list is maintained in the file as a separate line, when entering the entries through **qmgr**, the entries are comma separated on a single line. For example:

```
set server some_list="-Kenneth,Lora,-Julie,-Bob"
```

The quote marks are required to correctly parse the string as a single value for the keyword "some_list" as the string contains commas. Also be aware that when entered via qmgr, entries for ACLs which contain a host−domain part are sorted such that the more restrictive (fully specified, least wild carded) entries occur first. This is important to the comparison algorithm used when checking privilege. If the file is edited by hand, the order should be maintained.

10.1.1. Server Access

Server access is controlled by a host access control list, and by two lists which grant special privilege to certain users.

10.1.1.1. Server Host ACL

Each server has an access control list that specifies which hosts are allowed, or not allowed, to send requests to the server. The server depends on the network to identify the host which originated the connection (request). The entries in this list are fully specified host-domain names:

```
+host_one.domain.name.one
-host_two.domain.name.one
host_three.another.domain
*.my.domain
*.edu
-*
```

As shown in the last three entries, fields on the left, host, sub-domain, or domain names, may be "wild carded" by the use of a single asterisk. Thus any host in the domain `my.domain` or

any host in any subdomain under the top level edu domain is granted access, while the last line denies access to any other host.

10.1.1.2. Managers and Operators

Each server has a list of "users at hosts" who have the privilege to request services restricted to operators and managers (administrators). Requests for those services from other users will be rejected. Requests from the named users on hosts other than the named host will also be rejected. The format of these lists is:

```
user_name@host.domain.name
```

As above, the host, sub-domain, or domain name may be wild carded.

10.1.2. Queue Access

Queue access is controlled by three levels: host, group, and user. Unlike other ACLs, the user and group list may have an extra first entry which determines the default for unnamed members. This entry in the list consists solely of a single "+" or "-" character with no additional characters. Any name, user or group, not specified will implicitly be granted or denied access based on this line. If multiple default lines, single + or -, appear in a list, it is undefined as to which takes precedence.

10.1.2.1. Queue Host Access

As with the server, queue access can also be controlled by originating host. Each queue has an Host ACL identifying hosts from which jobs may or may not be entered into the particular queue. The format of the list entries is the same as the server's host list.

10.1.2.2. Queue Group Access

An access control list can be established which limits access to jobs run under certain group names. This is the group name under which the job would be executed. It may be different than the group name under which the user submitted the job, see the job attribute group_list. The format of this list is

```
[+|-]
[+|-]group_name
```

The optional first line of a single '+' or '-' establishes the default privilege for unlisted groups. If the first line is '-', any unlisted group is denied access. If the first line is '+' or contains additional characters, any unlisted group is allowed access.

10.1.2.3. Queue User Access

The user identification in each request may also control queue access. Each queue has an access control list identifying the users who are permitted or denied the ability to enter jobs into the queue. The format of this list is:

```
[+|-]
[+|-]user[@host.domain]
```

The optional first line is identical to the group ACL. The "host.domain" may be wild carded as described under the Server Host ACL. If a user name is given without a "@host.domain", then it is equivalent to that user name at any host, "user@*".

10.1.3. Job Access

The owner of a job is allowed full access to his or her own job. Operators and administrators also have access to all jobs.

Users without special privilege, manager or operator, are not allowed to modify or delete a job that they do not own. The ability to query the status of a job of another user is determined by the server attribute query_other_jobs. If true, users may query the status of any user's jobs.

10.1.4. Root Jobs

Generally, the super user, root, is not allowed to submit batch jobs. This prevents someone who has cracked the system from doing additional damage. However, there is a build time option to allow root on the server's local system to submit jobs. This option is controlled by the symbol PBS_ROOT_JOBS. Please see the server_limits.h section of the PBS Administrator Guide for more information.

10.2. Authentication

There must be a guarantee that the identity of the user making a batch request is correct. A basic method described below is provided with **PBS**. It is called the PBS Interface Facility, or **pbs_iff** (also known as Identify, Friend or Foe). The method of authenticating the user is somewhat modular. It can be removed and replaced by other authentication methods such as Kerberos with a bit of work in the server.

In the basic method provided with **PBS**, an Authentication Batch Request is sent to the server from a privileged port. This request contains the user's true identity, the name of the host on which the client is running, and the port name to which the client is bound and will be communicating with the server. This information is obtained by the *pbs_iff* program. This program is a "setuid root" program allowing it to access privileged ports (those requiring root privilege) and thus giving some amount of credence to its claims.

The PBS user application programmable interface (API) library, **libpbs.a**, is the client side of the credential process. Whenever a user requests an open connection with a batch server through the API, the **pbs_connect**(3B) routine will open a connection to the PBS server on a non-privileged port. It will then *fork*() and *exec*() **pbs_iff** passing it the server's name and the port to which pbs_connect() is bound. The authentication type is returned by **pbs_iff** to the pbs_connect routine over a pipe.

When the server receives the authentication request, it associates the user and host name with the network connection established by the client and identified by in the request. If the request is accepted by the server, the connection is considered authenticated for the remainder of its life.

11. Network Protocol

The batch 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.

As new batch processing features will become part of the base, and as vendors will extend the product, the data exchange format must be extensible.

As the batch system will run on many systems within a complex, the data exchange format must provide for interoperability among systems with:

Different architectures, including word size and byte order.
Non uniform extensions to the batch systems.

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.

PBS originally used the ISO standard ASN.1 data encoding between all pieces of PBS except for the Scheduler to Resource Monitor communication. A public domain package, ISODE, provided for quick implementation and it was believed that ASN.1 would be required by POSIX when the batch protocol was standardized. In release 1.1.9, the user and administrative command utilities still used the ASN.1 encoding and the Job Server, pbs_server, still accepted ASN.1 encoded requests from clients (user commands or other servers), but ASN.1 has being phased out in favor of the *Data Is Strings*, or DIS, encoding and is no longer used or supported.

11.1. 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. Human readability was a secondary goal and considered only when it would not cost.

Data are represented as people thing 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 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.

The crux of the the encoding is the integer. It's BNF is:

integer =: <count><sign><decimal string> | <count><integer>
count =: <decimal string>
sign =: '+' | '-'
decimal string =: <digit> | <digit><decimal string>
digit =: '0' | '1' | '2' | '3' |'4' | '5' | '6' | '7' | '8' | '9'

Single digit numbers do not require a count, just a sign followed by the one digit. Otherwise, the left most count is one digit long. Each count gives the number of digits in the decimal string to its right. The last decimal string is the actual integer value. The following table shows some examples of values and their encoding.

Value	Encoded String
1	+1
2	+2
-3	-3
12	2+12
-13	2-13
1234567890	210+1234567890

The last value may take some explaining, the first digit, '2', is the first count and there for the length of the following integer. That integer, '10' is the second count. Since it is followed by a sign, its value follows and is therefore '10' digits in length.

Counted character strings (or simply strings) are defined as:

counted string =: <integer><characters>
characters =: <character> | <character><characters>

The number of characters in the "characters" value is given by "integer". Some examples:

Value	Encoded String
"abc"	+3abc
"This is a long string."	2+22This is a long string.

Real numbers are defined as:

real number =: <integer><integer>

The first integer is a real number is the coefficient, with no leading or trailing zeros and with the decimal point to its right. The second integer of a real number is the exponent to the base 10.

PBS provides a collection of the functions to perform the encoding/decoding of the basic data types. The collection is in the source tree under lib/Libdis. The objects are placed into the API library, libpbs.a, for ease of linking. This library may be used outside of PBS if desired. The routines are described (briefly) in the PBS Internal Design Specification.

Layered on top of the above routines, PBS makes use of a set of batch request and reply routines which use the DIS routines to encode/decode the data specific to the request or reply. This routines are called by the public PBS API routines defined in chapter 4 of this document.

11.2. DIS Encoded Batch Requests

The following describes the format of the PBS batch requests and replies using the DIS encoding. These requests are the basis of all communication between the client commands (qalter, qsub, ...) and the server, between two servers, and between the server and MOM. The Job Scheduler uses a different message format when talking with the Resource Manager side of MOM.

11.2.1. Batch Requests

Each request is encoded as a *Message Header*, a *Message Body*, and an *Message Extension*. The format of the Message Header and the Message Extension is the same for all batch requests. The Message Body varies for each request.

11.2.1.1. Message Header

```
unsigned integer:    Batch Protocol Type Identifier     (=2)
unsigned integer:    Batch Protocol Version Identifier  (=1)
unsigned integer:    Batch Request Identifier
string:              User Name
```

11.2.1.2. Message Body

Copy Files Request

```
string:              Job Id
string:              Job Owner (may be null string)
string:              Execution User Name
string:              Execution Group Name (may be null string)
unsigned integer:    direction
unsigned integer:    number of file pairs that follow
array of file pairs:
    unsigned integer:    file type flag
    string:              local (to mom) path name
    string:              remote path name
```

Job Credential Request

```
unsigned integer:    credential type
string:              opque data
```

Job Files Move Request

```
unsigned integer:    file block number
unsigned integer:    file type (stdout, stderr, checkpoint, ...)
unsigned integer:    amount of data in this block
string:              Job Id
string:              file data block
```

Job Obituary (Notice) Request

```
string:              Job Id
unsigned integer:    exit status
array of Attributes
```

Manager Request

This request is used for a number of operations including Alter Job, Delete Job, Hold Job, and Release Job, as well as the request associated with the qmgr command.

```
unsigned integer:    command (create, delete, set, unset, increment, decr)
unsigned integer:    object type
string:              object name
array of Attributes
unsigned integer:    object type
string:              object name
array of Attributes
```

Message Job Request

```
string:              Job Id
unsigned integer:    which file (stderr, stdout, default)
string:              message text
```

Move Job Request

```
string:              Job Id
string:              Destination
```

Queue Job Request

```
string:              Job Id
string:                  Destination (queue name, null string implies default)
array of Attributes
```

Register Dependency Request

```
string:                  Job Owner
string:                  Parent Job Id
string:          Child Job Id
unsigned integer:    dependency type
unsigned integer:    operation
signed long:         dependency cost
```

Run Job Request

```
string:              Job Id
string:              Destination (where to run it)
```

Shut Down Server Request

```
unsigned integer:    manner of shutdown
```

Signal Job Request

```
string:              Job Id
string:              Signal name
```

Status Job, Queue, or Server Request

```
string:              Object Id (or null string)
```

```
array of Attributes
```

Track Job (Notice) Request

```
string:              Job Id
unsigned integer:    hop count
string:              location
unsigned character:  current job state
```

11.2.1.3. Complex Data Encoding

Certain data items are complex, such as the array of attributes, and require special encoding as a set.

Array of Attributes − An array or list of attributes, regardless for job, queues, or a server, is encoded as follows:

```
unsigned integer:    number of attributes in array (list)
```

followed by the attributes (if the above integer is non-zero):

```
unsigned integer:    sum of length of the name, resource, and value strings,
                     including the terminating null characters
string:              attribute name
unsigned integer:    flag if resource name is present (1=yes,  0=no)
optional string:     resource name (if above flag is 1)
string:              value
unsigned integer:    operation
```

Status Object (see Batch Reply)

Each status object in an array there of is encoded as:

```
unsigned integer:    object type (server, queue, job)
string:              object name
array of Attributes
```

11.2.1.4. Message Extension

```
unsigned integer:    flag indicates presence of extension string (1=yes, 0=no)
optional string:     extension string (present only if above flag is 1)
```

11.2.2. Batch Reply Format

The PBS batch reply is in two parts, a fixed header and a body which depends on the type of reply indicated in the header.

11.2.2.1. Reply Header

The reply header is encoded as:

```
unsigned integer:    Batch Protocol type (2)
unsigned integer:    Batch Protocol version (1)
unsigned integer:    return code
unsigned integer:    auxiliary code
unsigned integer:    body type indicator
```

11.2.2.2. Reply Body

The reply bodies are encoded depending on type as:

Null body − no additional data

Job Id Reply − (used for Queue, Ready to Commit, and Commit requests

```
string:              Job Id
```

Select Reply

```
unsigned integer:    count of job id strings to follow
string:              job id, repeated as required
```

Status Reply

```
unsigned integer:    count of status objects:
array of status objects
```

Text Reply

```
string:              text
```

Locate reply

```
string:              location of job
```

11.3. Description of Data Exchange Format

A batch client makes a *service request* of a batch server. The request consists of one or more *subrequests* from the client to the server. For each subrequest, the server returns a *reply* to the client. As a minimum, the reply indicates success or failure of the subrequest. For some subrequests, such as a status request, additional information is included in the reply.

11.4. Request Format

A general request consists of two required parts, a *fixed header* and a *variable body*, and an optional third part, a *request extension* byte string. The header contains three (3) fields:

request code which is a number identifying the type of request.

request user a string which is the effective user name of the user making the request,

request credential an "any" structure which contains client authentication credentials.

The format and content of the request body depends on the type of request. The body is decoded separately from the header and after the header identifies the type of body.

The request extension byte string is intended to allow portable extensions to the batch command parameters. PBS makes use of this capability for the **qhold** and **qdel** commands. When a server receives a request extension that it does not recognize, it is free to abort the request.

11.5. Reply Format

The reply returned by a server to a client consists of a *fixed header* and an optional *variable body*. The body is only required in replies to certain types of requests.

The header contains the following integer numbers:

reply code which contains the primary success or failure code.

auxiliary code which contains extended status or failure information.

The optional body format depends on the type of request to which the reply is being returned. The format of the reply will be discussed under those requests which require the additional information contained in a body.

11.6. Sequence of Subrequests

The following sequence of requests is followed in the client - server communication.

11.6.1. Open Connection

The client opens a reliable stream connection with the server. The PBS mechanism for opening the connection is the **connect**(2) call to connect to an address associated with the requested server.

11.6.2. Service Requests

The client sends one or more service requests to the server.

The server returns a reply following processing of each request. The client does not send an additional request until it has received a reply to the prior request.

11.6.3. Connection Closure

The client closes the connection to the server to indicate that there are no more requests. The server may close the connection. If this occurs before the reply is sent, the client treats this as error condition.

11.7. Description of Service Request

The following sections describe the data items in each request.

11.7.1. Queue Job Request

Queue Job is a complex request, made up of four or more sub-requests:

- Queue Job (subrequest)
- Job Credentials
- Job Script, zero or more sub-requests.
- Ready to Commit.
- Commit ownership.

The server must respond to each of the subrequests with a reply. The request may be aborted without the job being queued by the server at any point prior to receiving the Commit request.

11.7.1.1. Queue Job

The **Queue Job** subrequest informs the server that the client wishes the server to accept and enqueue a batch job.

The *DestinationID* field identifies the destination of the job. It is a string in the form: `[queue][@server]` If queue is not specified, the default queue is assumed. If @server is specified, it is ignored.

If the optional field *JobIdentifier* is present, the job is an existing job being moved from one server to another. The *JobIdentifier* field contains the job identifier assigned to the job by the creating server. If *JobIdentifier* is not present, the job is being newly submitted by a client qsub. The receiving server will assign a job identifier to the job. The server, in either case, replies with the job identifier.

Also specified in the Queue Job subrequest are the batch job's private and public attributes, including required resources and the inherited environment variables, the *Variable_List*.

11.7.1.2. Job Credential

The Job Credential is an optional subrequest. The request is provided as a means of passing service credentials or tickets for use by the batch job. These services are system or network services unrelated to the batch system. An example is a ticket authenticating the batch job to AFS (Andrew File System).

If present, the server will just decode the data from the network presentation. Sites or vendors must add the additional functions to the server to present the credentials to the system when the job is placed into execution.

11.7.1.3. Job Script

The body of the **Job Script** request consists of sections of the job script file supplied to the qsub command. Each section contains up to 8192 bytes of the file. The last section may be shorter than 8192 bytes. The sections are send in order and each must be acknowledged before the next is sent.

Each Job Script sub-request also contains a sequence number. This integer starts with one (1) and is incremented by one for each additional Job Script sub-request.

11.7.1.4. Ready to Commit

Upon receipt of the Ready to Commit, the receiving server **must** record the job to permanent storage **before** acknowledging.

The reply to the Ready to Commit, contains the job identifier. This is provided to assist in the recovery of a server or network failure.

If the client is another batch server which had the batch job queued, upon receipt of a acknowledgment to the Ready to Commit, the client removes the job from its queue and permanent storage, deleting the job.

11.7.1.5. Commit

When the sending client has received a positive reply to the Ready to Commit request and had deleted the job locally, then and only then should the client send a *Commit* sub-request to the server. This ensures the job cannot be duplicated.

When the server receives the commit sub-request, the server assumes ownership of the job and perform the steps required to place the job into a queue. The server **does not reply** to the client following the commit sub-request.

11.7.1.6. Error Recovery

The following procedures are used to ensure the job is not lost or duplicated if the event of a communication failure between the sending client and the receiving server:

Client Side Recovery - Failure of Server

The follow procedures should be followed by the client attempting to send a job to a server when communications are lost with the server.

1. If the communication link is lost before the client sends the *Ready to Commit*, it is to assume the job is lost and restart from the beginning. The server will replace

the job with the new copy when the client re-established the connection.

2. If the communication link is lost after the client sends the *Ready to Commit* but before sending the *Commit*, the client can assume that prior job information is at the server. Upon re-establishment of a connection with the server, the client re-sends the *Ready to Commit* followed by the *Commit*.
3. If the communication is lost after sending the *Ready to Commit* and at a later time the client receives a positive acknowledgement to the *Ready to Commit*, even if one has already been received, the client sends or resends the *Commit*. Note, the job identifier is required in the acknowledgement for this purpose. Also note that at this point the client should not have and does not need a copy of the job.
4. If the client is a temporary process, for example qsub, rather than restarting the Queue Job request, it may abort and issue an error message to the user. If the problem occurs after the *Ready to Commit* is sent but before an acknowledgement is received, it is unclear if the server has the complete job. The user should be warned to check for the job before attempting to resubmit it. Once the acknowledgement for the *Ready to Commit* has been received by the temporary client, the client may safely assume the job has been queued by the server.

Client Side Recovery - Failure of Client

Should the client side crash, upon recovery a permanent process (a sending server) performs the actions described in 1 through 3 above.

Server Side Recovery - Failure of Client

The following recovery procedures should be followed by a server receiving a job upon loss of communications with the client.

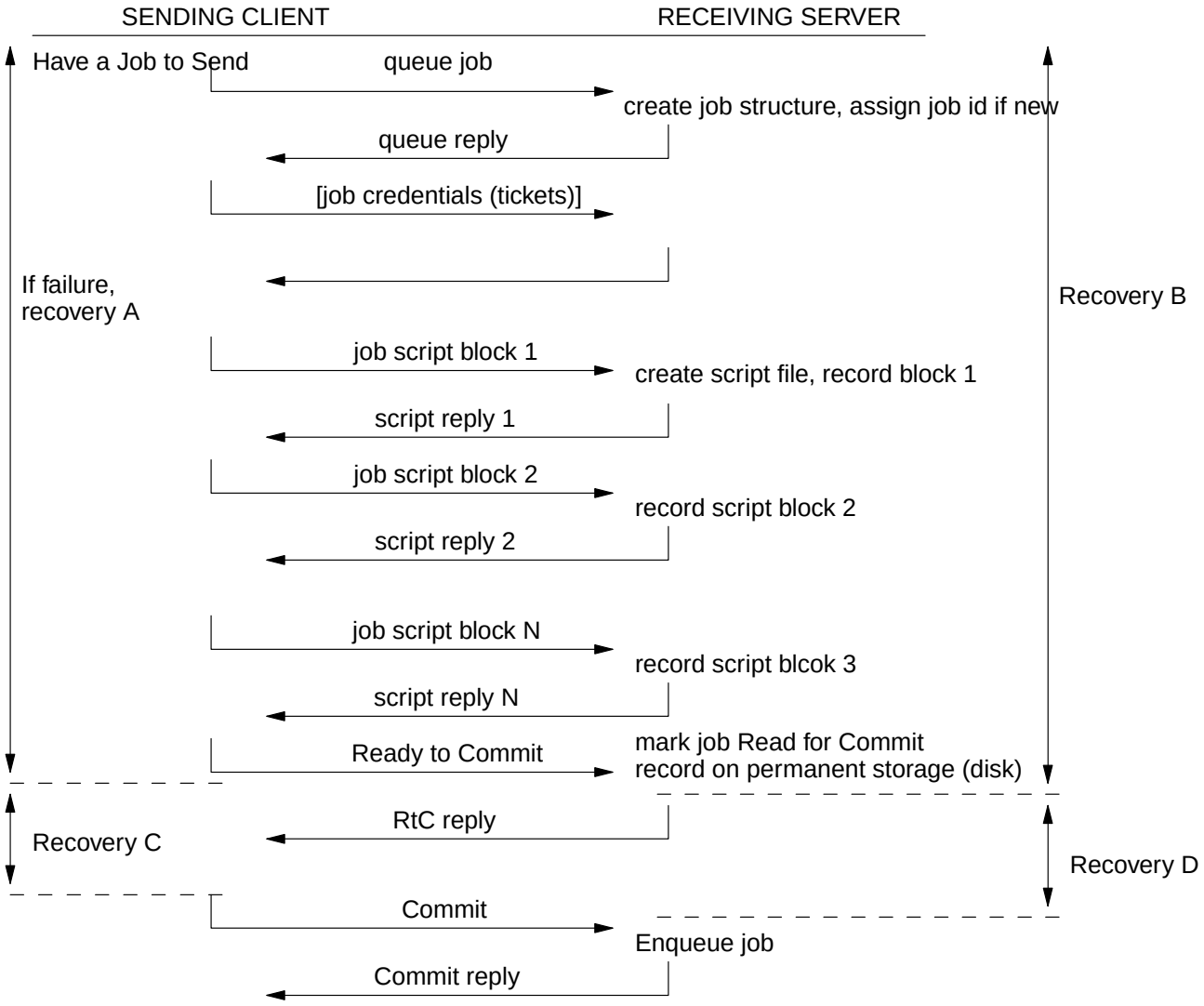
1. If the failure occurs before the *Ready to Commit* is received, the server discards the job. The client must restart from the beginning.
2. If the failure occurs after the *Ready to Commit* is received, the server should have recorded the job in permanent storage. The server keeps the job until (a) a request is received to delete it, or (b) the client resends the Read to Commit, Commit sequence. If after a "site defined" period of time, the server has not received any directions, it may notify the batch administrator and request instruction.

Server Side Recovery - Failure of Server

Should the receiving server crash, it should perform the following actions upon being restarted.

1. If the server detects a job for which a *Ready to Commit* had not been received, the server purges the job.
2. If the server detects a job for which the *Ready to Commit* has been received, and the sending client was a permanent process (another server), the server sends an acknowledgement of the Ready to Commit to the client. The acknowledge will include the job identifier. This will trigger the client to resend the the commit sequence. In transaction terminology, this is called a *restart* message.

The send — receive — recovery steps are show in figure 11−1.



Recovery Proceedures
A. If sender is a server, restart from top, else abort.
B. Discard job.
C. Resend "Ready to Commit".
D. Commit (enqueue) job.

Figure 11−2: Two Phase Commit and Error Recovery

11.7.2. Delete Job Request

The **Delete Job** request is used by a client, such as the **qdel** command, to request that a job be deleted. The request is a *manage* request with the object type set to "job", the name set to the job id, and the operation is "delete".

11.7.3. Hold Job Request

The **Hold Job** request is used by a client, such as the **qhold** command, to request that a hold be placed on a job.

The request is a *manage* form with the object type set to "job", the name set to the job identifier, the operation set to "set", and the attribute list contains the Hold_Types attribute with the

value being the holds to apply to the job.

PBS will use the request extension to pass the re-execution preference information, see pbs_holdjob(3B).

11.7.4. Modify Job Request

A **Modify Job** request is sent by a client such as the **qalter** command to a server to request that certain attributes of the job be altered or modified.

The request is a *manage* form with the object type set to "job", the name set to the job identifier, the operation set to "set", and the attribute list contains each of the attributes to be modified.

11.7.5. Move Job Request

The **Move Job** request is used by a client, such as the **qmove** command, to request that a server route the specified job to another destination.

The *jobid* field specifies the job to be routed. The *destin* field specifies the destination to which the job is to be routed.

11.7.6. Message Job Request

The **Message Job** request is used by a client, such as the **qmsg** command, to ask a server to write a message into an output file of a running job.

The *jobid* field specifies the id of the running job for which the message is destined. The destination file for the message is defined by the *fileopt*; if either the {oflg} bit and / or the {eflg} bit is set, the message will be written in the standard output and / or standard error file of the job. If neither bit is set, the message will be written to an file determined by the implementation of the server.

The message string will be in the *message* field.

11.7.7. Rerun Job Request

The **Rerun Job** request is used by a client such as **qrerun**, to request that a server terminate and re-queue a currently running job.

The request specifies the job identifier of the running job which is to be rerun.

11.7.8. Select Jobs Request

The **Select Jobs** request, issued by the command qselect, requests that the server return job identifiers for all jobs which meet the specified criteria.

The *attribute list* specifies the attributes which are to be used in filtering jobs. Resources, the job state (even though not a true attribute) and destination may be included in the attribute set.

The server replies to the client with the special reply **select-reply**. The count of jobs selected in response to the request is returned in *reply-auxcode*. The list of job identifiers selected is returned in *SelectReply*.

11.7.9. Signal Job Request

The **Signal Job** request, used by a client such as **qsig**, requests that the server send a signal to a running batch job. The *jobid* field indicates which job is to be sent the signal. The *signal* field contains the name or value of the signal to be sent.

11.7.10. Status Job Request

A **Status Job** request directs a server to return status information about either a job whose job identifier is in *id* or the set of jobs in a destination given by *id* .

If the server owns the job or destination specified and the client is entitled to see the status information, the server replies with a **stat** reply. The stat reply is a list of zero or more **objstat** structures, one per job. Each objstat structure includes:

- The type of object, i.e "job".
- The job name
- All public attributes of the job, see next paragraph.
- Optionally, a general text string for messages, implementation defined.

If the status request includes a set of specified attributes, only those attributes are included in the reply.

11.7.11. Status Queue Request

A **Status Queue** request directs the server to return status information about the queue identified in the destination specified in the request.

If a queue with the specified name exists at the server, the server returns a **stat** reply containing the attribute information about that queue.

If a queue name is not specified, is null, then the server will return status about all queues.

11.7.12. Status Server Request

A **Status Server** request directs the server to return status information about itself. The server responds with a **stat** reply.

11.7.13. Server Shutdown Request

The **Server Shutdown** request directs a batch server to terminate batch service and exit. The integer value is used to indicate the manner of shutdown: {immediate} or {delay}.

11.7.14. Locate Job Request

The **Locate Job** request is used by a client to search for the current location of a job. The job identifier is specified in the request.

A server who receives a Locate Job Request for a job which the server owns or knows the name of the server which owns the job, replies with a **locate reply**. If the server does not know the location, it replies with a failure code.

The locate reply is a expanded form of reply which contains the *locate-reply* field. This field contains the name of the batch server currently owning the job.

11.7.15. Track Job Request

Job tracking is an optional implementation feature in POSIX 1003.2d. Its purpose is to provide a method for recording the current location and status of jobs in a central location to facilitate job status reporting.

Job tracking is supported by **PBS.** When the server accepts a job for queuing, it sends a **Track Job** request to the server which originated the job and optionally to one or more "location servers" as configured by the batch system administrator.

When a server receives the Track Job request, that server records the current location, specified in the *location* field, of the job identified in the *jobid* field. The *hopcount* is a count of the current number of routes or hops made by the job. This is used to insure proper sequencing of the information.

The *jobstate* field is provided to allow for an abbreviated status following job completion.

11.7.16. Run Job

The **Run Job** request is used to direct a server to schedule a batch job immediately for execution regardless of its state, required resources, or the servers scheduling parameters.

11.7.17. Manage Request

The **Manage** request is used by the **qmgr** command (and others) to create, delete, or alter managed objects.

The *command* field specifies the operation: create, delete, set, or unset. The *objtype* field identifies the type of object and the *objname* field names the specific object. Attributes and associated values of the object are listed in the field *attrib*.

11.7.18. Register Dependent Job

This request is used to communicate job dependency requirements between servers.

The *parentid* field specifies the parent or master job identifier. The *childid* field specifies the id of the child job the server is registering with the parent.

The *dependtype* specifies the type of dependency.

The *op* field is mostly meaningful on synchronization request. It indicates if the child job has reserved its resources and is ready to run.

[Page intentionally left bank.]

12. Tools

This section describes the PBS tools. These are programs or scripts designed to make life easier for administrators and users of PBS. The existence of the PBS interface libraries provides the basis for a rich functionality. The tools give a higher level access to this functionality.

Right now only two tools exist. Both are based on Tcl so if your system doesn't have Tcl, it doesn't have these tools either. As PBS becomes more accepted and widely used, it is hoped that programs that various sites come up with will find there way into this section of the distribution (hint hint).

12.1. Pbs_tclsh

This is a Tcl shell program which includes the same extra commands as the Tcl scheduler. These can be found in the man page for **pbs_sched_tcl**.

12.2. Pbs_wish

This is a Tk window shell program with the same additional commands as *pbs_tclsh*. This program should make it possible to write an X11 interface program to PBS.

[Page intentionally left bank.]

13. Task Manger

This section describes the PBS task manager functions of the Machine Oriented Miniserver (MOM). This is a new feature of PBS and is an attempt to follow the emerging "PSCHED" API. For more information on PSCHED, please see the web page:

http://parallel.nas.nasa.gov/Psched/index.html

A paper (which can be found on the web site above) shows the current state of the effort: "PSCHED: An API for Parallel Job/Resource Management".

In the past, a running job under the control of a MOM was a single POSIX session. Now, multiple sessions may be spawned, each one called a "task", and tracked separately within the job by MOM. When the job is run, one task is started with the user selected shell running the job script as the program. This is the equivalent of the single session of the past. New variables are included in the job's environment which give it information needed to call back to MOM for task management service.

MOM will use another tcp port to listen for task management requests from its children. This communication will be handled by a library which sends and receives messages over a tcp socket to MOM using the DIS protocol.

[Page intentionally left bank.]