PBS Portable Batch System

Selected User Portions of the

External Reference Specification

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

Numerical Aerospace Simulation Systems Division
NASA Ames Research Center

Release: 2.0
Printed: October 15, 1998

† MRJ Technology Solutions, NASA Contract NAS 2-14303, Moffett Field, CA

Portable Batch System, Rights to Use and Redistribute

This program is derived from prior work some of which was originally written 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.

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 1.1.7	June 6, 1996
Revision 1.1.8	August 19, 1996
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

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 is a set of execution “servers” or hosts on which a single batch server manages batch jobs. A cluster may be made up of a set of workstations, multiple cpu systems, or one or more parallel systems. Each allocatable unit is a node, see Node.

Complex or 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 complexes.

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.

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 computional unit, one or more of which can be allocated to a job in a shared or exclusive manner. A node is characterized by (a) a unique memory address space, (b) and operating system image, (c) an IP address, and (d) a PBS execution server (pbs_mom). If a set of nodes is allocated in an *exclusive* manner to a job, no other job may use that set of nodes during the execution of the job to which the nodes are assigned. This is often called *Space Sharing* If a set of nodes is allocated in a *shared* manner, multiple jobs may have processes executing on the set at the same time; often called *Time Sharing* .

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 are:

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.

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.

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.

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 complete.

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 debug-

gers 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.

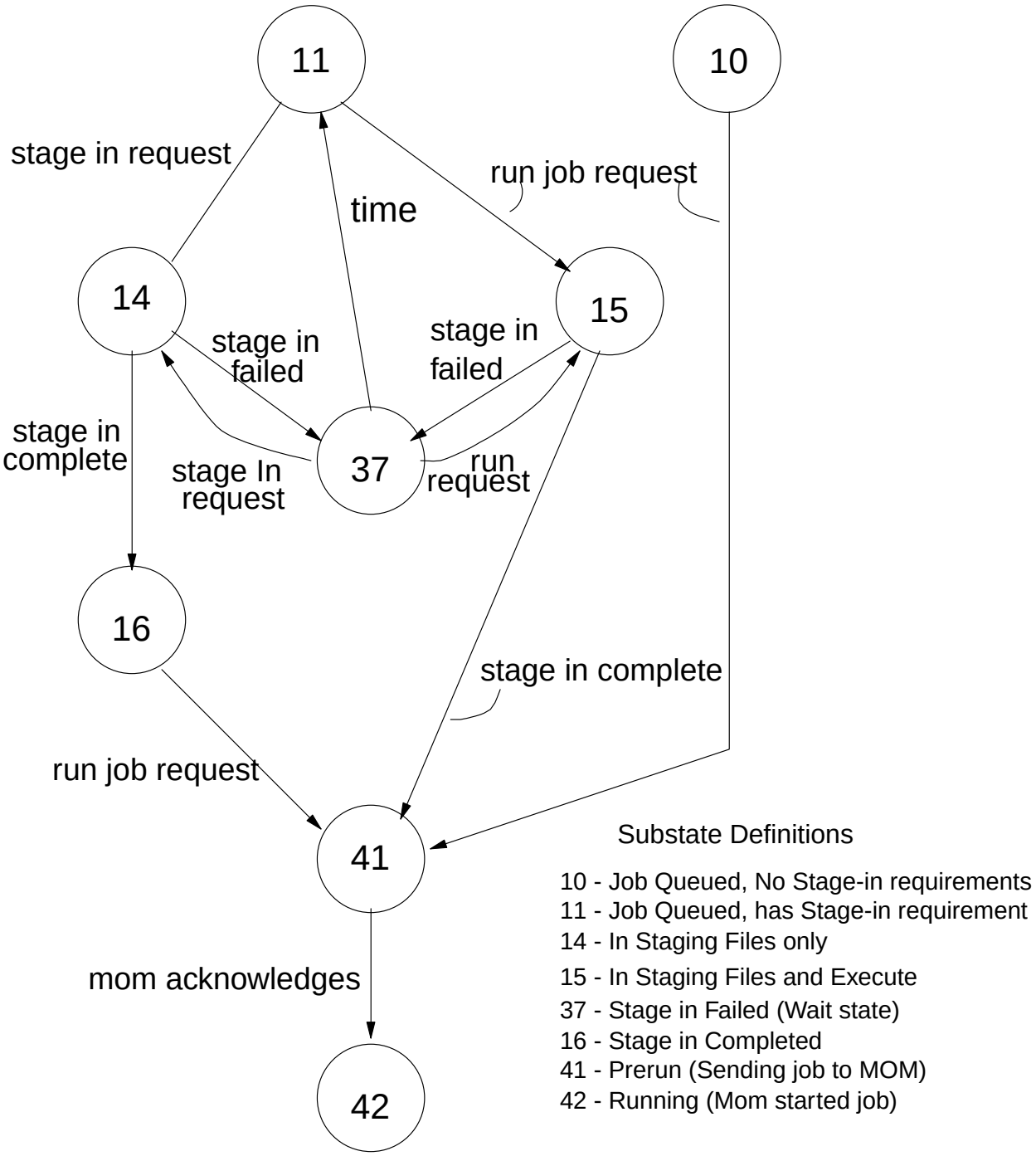


Figure 1−1: Job Substate Changes During File Stage In

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.

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. AttributeTypes

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]

depend
: The 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 9

server internally. The attribute is visible to privileged clients, such as the scheduler. 9
Format: interger. [internal type: long integer] 9
The values are defined in the header file job.h. See the ERS section on file staging for 9
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

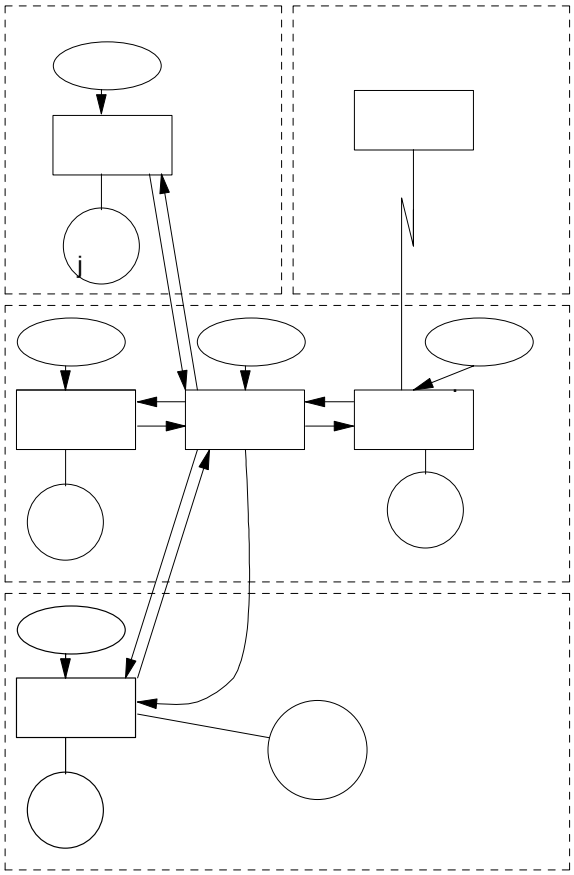
The following data items are internal to the server representation of a job. They are specifically described here because of their importance.

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.



2.7. 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.

.

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/yy 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/yy 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.

3.4.4.4. 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.

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

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

software Allows for 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

3.4.5. 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.

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 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)

5.2.15. 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.