PBS Professional® 14.2

Programmer's Guide

Altair | PBS Works™

PBS Works is a division of Altair

You are reading the Altair PBS Professional 14.2

Programmer's Guide (PG)

Updated 1/23/17

Please send any questions or suggestions for improvements to agu@altair.com.

Please send any questions or suggestions for improvements to agu@altair.com.

Contact Us

For the most recent information, go to the PBS Works website, www.pbsworks.com, select "My PBS", and log in with your site ID and password.

Altair

Altair Engineering, Inc., 1820 E. Big Beaver Road, Troy, MI 48083-2031 USA www.pbsworks.com

Sales

pbssales@altair.com 248.614.2400

Technical Support

Need technical support? We are available from 8am to 5pm local times:

Location	Telephone	e-mail
Australia	+1 800 174 396	anz-pbssupport@india.altair.com
China	+86 (0)21 6117 1666	es@altair.com.cn
France	+33 (0)1 4133 0992	pbssupport@europe.altair.com
Germany	+49 (0)7031 6208 22	pbssupport@europe.altair.com
India	+91 80 66 29 4500 +1 800 425 0234 (Toll Free)	pbs-support@india.altair.com
Italy	+39 800 905595	pbssupport@europe.altair.com
Japan	+81 3 5396 2881	pbs@altairjp.co.jp
Korea	+82 70 4050 9200	support@altair.co.kr
Malaysia	+91 80 66 29 4500 +1 800 425 0234 (Toll Free)	pbs-support@india.altair.com
North America	+1 248 614 2425	pbssupport@altair.com
Russia	+49 7031 6208 22	pbssupport@europe.altair.com
Scandinavia	+46 (0)46 460 2828	pbssupport@europe.altair.com
Singapore	+91 80 66 29 4500 +1 800 425 0234 (Toll Free)	pbs-support@india.altair.com
South America	+55 11 3884 0414	br_support@altair.com
UK	+44 (0)1926 468 600	pbssupport@europe.altair.com

Contents

Manual Pages

About PBS Documentation

The PBS Professional Documentation

PBS Professional Installation & Upgrade Guide:

How to install and upgrade PBS Professional. For the administrator.

PBS Professional Administrator s Guide:

How to configure and manage PBS Professional. For the PBS administrator.

PBS Professional Hooks Guide:

How to write and use hooks for PBS Professional. For the administrator.

PBS Professional User s Guide:

How to submit, monitor, track, delete, and manipulate jobs. For the job submitter.

PBS Professional Reference Guide:

Covers PBS reference material.

PBS Professional Programmer s Guide:

Discusses the PBS application programming interface (API). For integrators.

PBS Manual Pages:

PBS commands, resources, attributes, APIs.

PBS Professional Quick Start Guide:

Quick overview of PBS Professional installation and license file generation.

Where to Keep the Documentation

To make cross-references work, put all of the PBS guides in the same directory.

Ordering Software and Licenses

To purchase software packages or additional software licenses, contact your Altair sales representative at pbssales@altair.com.

Document Conventions

`abbr`eviation

The shortest acceptable abbreviation of a command or subcommand is underlined.

`command`

Commands such as `qmgr` and `scp`

input

Command-line instructions

`manpage(x)`

File and path names. Manual page references include the section number in parentheses appended to the manual page name.

format

Syntax, template, synopsis

Attributes

Attributes, parameters, objects, variable names, resources, types

Values

Keywords, instances, states, values, labels

Definitions

Terms being defined

`Output`

Output, example code, or file contents

Examples

Examples

`Filename`

Name of file

`Utility`

Name of utility, such as a program

1 Introduction

This book, the **Programmer's Guide** for PBS Professional, is provided to document the external application programming interfaces to the PBS Professional software.

1.1 Location of API Libraries

All of the libraries containing the PBS API are installed by default in *$PBS_EXEC/lib/*.

1.2 Location of Header Files

Header files used by customer-written code are found in `$PBS_EXEC/include`.

1.3 Example Compilation Line

An example of a compile command might look like the following:

```
cc mycode.c -I/usr/pbs/include -L/usr/pbs/lib -lpbs
```

1.4 Deprecations

The following are deprecated:

`pbs_tclapi`

`pbs_rescquery`

2 PBS Architecture

PBS is a distributed workload management system which manages and monitors the computational workload on a set of one or more computers.

2.1 PBS Components

PBS consists of two major component types: system processes and user-level commands. You can install PBS on one or more machines.

2.1.1 Single Execution System

If PBS is to manage a single system, all components are installed on that same system. During installation (as discussed in the next chapter) be sure to select option 1 (all components) from the PBS Installation tool. The following illustration shows how communication works when PBS is on a single host in TPP mode. For more on TPP mode, see Chapter 4, "Communication", on page 49.

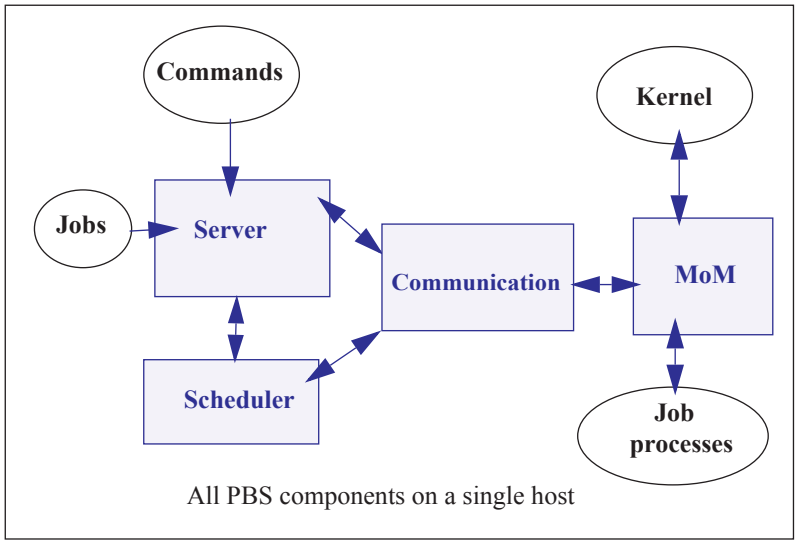

Figure 2-1:PBS daemons on a single execution host

2.1.2 Single Execution System with Front End

The PBS server and Scheduler (`pbs_server` and `pbs_sched`) can run on one system and jobs can execute on another. The following illustration shows how communication works when the PBS server and scheduler are on a front-end system and MoM is on a separate host, in TPP mode. For more on TPP mode, see Chapter 4, "Communication", on page 49.

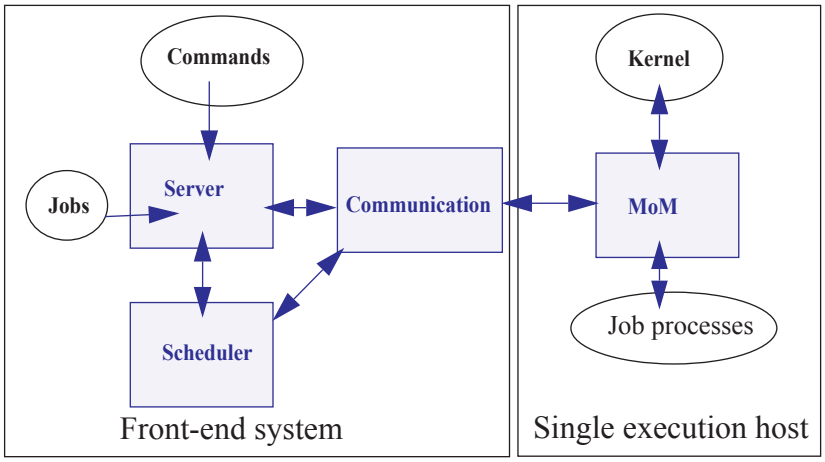

Figure 2-2:PBS daemons on single execution system with front end

2.1.3 Multiple Execution Systems

When you run PBS on several systems, normally the server (`pbs_server`), the scheduler (`pbs_sched`), and the communication daemon (`pbs_comm`) are installed on a "front end" system (option *1* from the PBS Installation tool), and a MoM (`pbs_mom`) is installed (option *2* from the Installation tool) and run on each execution host. The following diagram illustrates this for an eight-host complex in TPP mode.

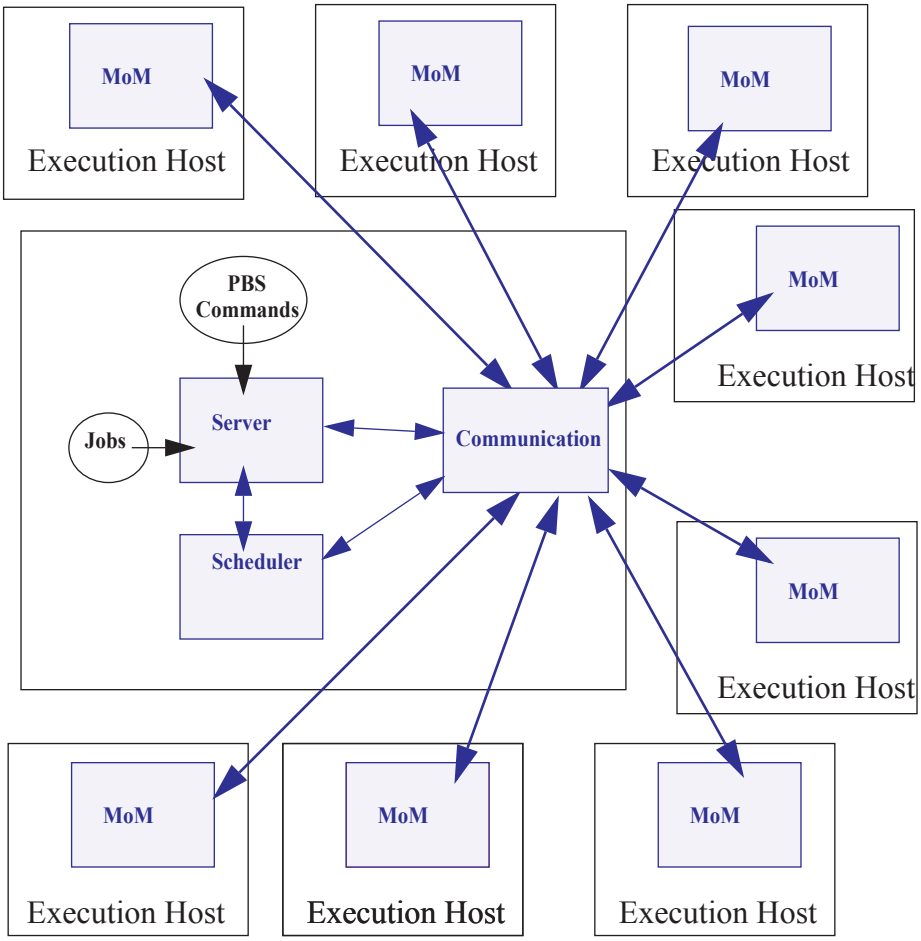

Figure 2-3:Typical PBS daemon locations for multiple execution hosts

2.1.4 Server

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

The server contains a licensing client which communicates with the licensing server for licensing PBS jobs.

2.1.5 Job Executor (MoM)

The *Job Executor* is the component that actually places the job into execution. This process, *pbs_mom,* is informally called *MoM* as it is the mother of all executing jobs. MoM places a job into execution when it receives a copy of the job from a server. MoM creates a new session that is as identical to a user login session as is possible. For example, if the user's login shell is `csh`, then MoM creates a session in which `.login` is run as well as `.cshrc`. MoM also has the responsibility for returning the job's output to the user when directed to do so by the server. One MoM runs on each computer which will execute PBS jobs.

2.1.6 Scheduler

The *Scheduler*, *pbs_sched*, implements the site's policy controlling when each job is run and on which resources. The Scheduler communicates with the various MoMs to query the state of system resources and with the server to learn about the availability of jobs to execute. See "The Scheduler" on page 73 in the PBS Professional Administrator's Guide.

2.1.7 Communication Daemon

The *communication daemon*, `pbs_comm`, handles communication between the other PBS daemons. For a complete description, see section 4.5, "Inter-daemon Communication Using TPP", on page 52.

2.1.8 The pbs_rshd Windows Service

The Windows version of PBS contains a service called `pbs_rshd` for supporting remote file copy requests for delivering job output and error files to destination hosts.

The `pbs_rshd` service supports rcp, but does not allow normal `rsh` activities. This service is used only by PBS; it is not intended to be used by administrators or users.

`pbs_rshd` reads either the `%WINDIR%\system32\drivers\etc\hosts.equiv` file or the user's `.rhosts` file to determine the list of accounts that are allowed access to the local host during remote file copying.

PBS uses this same mechanism for determining whether a remote user is allowed to submit jobs to the local server.

`pbs_rshd` is started automatically during installation; you can also start it manually.

To start `pbs_rshd` as a service:

```
net start pbs_rshd
```

To start pbs_rshd in debug mode, so that it prints logging output on the command line:

```
pbs_rshd -d
```

If user1 on *hostA* causes a file to be copied to *hostB* (running `pbs_rshd`) so that the source is `file1` and the destination is `hostB:file2, pbs_rshd` authenticates user1 as follows:

- If *user1* is a non-administrator account (e.g. not belonging to the Administrators group), then the copy will succeed in one of 2 ways:
  - *user1@hostA* is authenticated via *hostB's* `hosts.equiv` file; or
  - *user1@hostA* is authenticated via user's `[PROFILE_PATH]/.rhosts` on *hostB*.

  See also section 2.3.2, "Windows User HOMEDIR", on page 18.

  One of the following must be configured:
  - The system-wide `hosts.equiv` file on *hostB* includes as one of its entries:
    ```
    hostA
    ```
  - or, `[PROFILE_PATH]\.rhosts` on *userA's* account on *hostB* includes:
    ```
    hostA userA
    ```
- If *userA* is an administrator account, or the source is `file1` and the target is `userB@hostB:file2`

  then use of the account's `[PROFILE_PATH]\.rhosts` file is the only way to authenticate, and it needs to have the entry:

  ```
  hostA userA
  ```

2.1.9 Commands

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

There are three classifications of commands: user commands (which any authorized user can use), operator commands, and manager (or administrator) commands. Operator and Manager commands require specific access privileges.

PBS commands are described in "PBS Commands" on page 17 of the PBS Professional Reference Guide.

3
Server Functions

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

3.1 General Identifiers

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

3.1.1 Account String

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

3.1.2 Attribute Name

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

3.1.3 Destination Identifiers

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

queue@server_name

queue

@server_name

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

3.1.4 Default Server

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

3.1.5 Host Name

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

3.1.6 Job Identifiers

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

sequence_number.server[@server]

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

3.1.7 Job Name

A Job Name is a string assigned by the user to provide a meaningful label to identify the job. The job name is up to and including 236 characters in length and may contain any printable characters other than white space. It must start with an alphanumeric character. If the user does not assign a name, PBS will assign a default name as described under the -N option of the qsub(1) command.

3.1.8 Resource Name

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

3.1.9 Server Name.

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

3.1.10 User Name

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

3.2 Batch Server Functions

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

3.2.1 Client Service Requests

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

3.2.2 Deferred Services

The server may, depending on conditions being monitored, defer a client service request until a later time. (Deferred services include file staging, job scheduling, etc.) Detailed discussion of the deferred services provided by the server is given in section 3.7, "Deferred Services", on page 19 below.

3.3 Server Management

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

Table 3-1: Batch Request Codes

Numeric Value	Name	Numeric Value	Name
0	PBS_BATCH_Connect	24	PBS_BATCH_Rescq
1	PBS_BATCH_QueueJob	25	PBS_BATCH_ReserveResc
2	UNUSED	26	PBS_BATCH_ReleaseResc
3	PBS_BATCH_jobscript	27	PBS_BATCH_FailOver
4	PBS_BATCH_RdytoCommit	48	PBS_BATCH_StageIn
5	PBS_BATCH_Commit	49	PBS_BATCH_AuthenUser
6	PBS_BATCH_DeleteJob	50	PBS_BATCH_OrderJob

Table 3-1: Batch Request Codes

Numeric Value	Name	Numeric Value	Name
7	PBS_BATCH_HoldJob	51	PBS_BATCH_SelStat
8	PBS_BATCH_LocateJob	52	PBS_BATCH_RegistDep
9	PBS_BATCH_Manager	54	PBS_BATCH_CopyFiles
10	PBS_BATCH_MessJob	55	PBS_BATCH_DelFiles
11	PBS_BATCH_ModifyJob	56	PBS_BATCH_JobObit
12	PBS_BATCH_MoveJob	57	PBS_BATCH_MvJobFile
13	PBS_BATCH_ReleaseJob	58	PBS_BATCH_StatusNode
14	PBS_BATCH_Rerun	59	PBS_BATCH_Disconnect
15	PBS_BATCH_RunJob	60-61	UNUSED
16	PBS_BATCH_SelectJobs	62	PBS_BATCH_JobCred
17	PBS_BATCH_Shutdown	63	PBS_BATCH_CopyFiles_Cred
18	PBS_BATCH_SignalJob	64	PBS_BATCH_DelFiles_Cred
19	PBS_BATCH_StatusJob	65	PBS_BATCH_GSS_Context
20	PBS_BATCH_StatusQue	66-69	UNUSED
21	PBS_BATCH_StatusSvr	70	PBS_BATCH_SubmitResv
22	PBS_BATCH_TrackJob	71	PBS_BATCH_StatusResv
23	PBS_BATCH_AsyrunJob	72	PBS_BATCH_DeleteResv

3.3.1 Manage Request

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

3.3.2 Server Status Request

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

3.3.3 Start Up

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

3.3.4 Shut Down

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

3.4 Queue Management

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

3.4.1 Queue Status Request

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

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

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

3.5 Job Management

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

3.5.1 Queue Job Request

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

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

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

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

When a job is placed in a routing queue, its state may change based on the conditions described in section 3.7.4, “Job Routing”, on page 21.

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

3.5.2 Job Credential Request

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

3.5.3 Job Script Request

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

3.5.4 Commit Request

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

3.5.5 Message Job Request

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

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

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

3.5.6 Locate Job Request

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

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

3.5.7 Delete Job Request

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

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

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

3.5.8 Modify Job Request

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

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

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

3.5.9 Run Job

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

3.5.9.1 Rerun Job Request

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

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

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

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

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

3.5.10 Hold Job Request

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

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

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

3.5.11 Release Job Request

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

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

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

3.5.12 Move Job Request

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

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

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

3.5.13 Select Jobs Request

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

3.5.14 Signal Job Request

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

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

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

3.5.15 Status Job Request

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

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

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

3.6 Server to Server Requests

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

3.6.1 Track Job Request

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

3.6.2 Synchronize Job Starts

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

```
-W synccount=count
```

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

```
-W syncwith=job_identifier
```

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

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

3.6.3 Job Dependency

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

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

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

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

3.7 Deferred Services

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

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

3.7.1 Job Scheduling

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

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

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

3.7.2 File Staging

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

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

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

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

3.7.3 Job Initiation

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

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

1. Select the user identifier from the `User_List` job attribute which has a host name that matches the execution host.
2. Select the user identifier from the `User_List` job attribute which has no associated host name.
3. Use the user name from the `job_owner` attribute of the job.

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

- The stream will be located in a temporary file in the MoM's spool directory.
- If the job attribute Join_Path has the value eo or the value oe, the MoM connects the standard error stream of the job to the same file as the standard output stream.

3.7.4 Job Routing

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

3.7.5 Job Exit

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

- Place the job in the exiting state.
- 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_Users`.
- If out staging of files is supported, the files listed in the outfile resource will be copied to the specified destination.
- "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.
- The job will be removed from the execution queue.

3.7.6 Job Aborts

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

3.7.7 Timed Events

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

3.7.8 Event Logging

The PBS server maintains an event logfile, the format and contents of which are documented in "Event Logging" on page 468 in the PBS Professional Administrator's Guide.

3.7.9 Accounting

The PBS server maintains an accounting file, the format and contents of which are documented in "Accounting" on page 489 in the PBS Professional Administrator's Guide.

3.8 Resource Management

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

3.8.1 Resource Limits

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

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

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

3.8.2 Resource Names

For additional information, see "Resources" on page 241 of the PBS Professional Reference Guide where all resource names are documented.

3.9 Network Protocol

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

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

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

3.9.1 General DIS Data Encoding

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

4
Batch Interface Library (IFL)

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

4.1 Interface Library Overview

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

The credential that is sent to the server consists of: a) user's name from the password file based on running pbs_iff's "real uid" value, and b) unprivileged, client-side port value associated with the original pbs_connect request message to the server. The server looks at the entries in its connection table to try and find the entry having these two pieces of information, and which is not yet marked authenticated. To be believed, this information must be gotten from a connection having a privileged, remote-end, port value.

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

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

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

Any user-written programs using the IFL API must link with the pthread library.

4.2 Interface Library Routines

The following manual pages describe the user-callable functions in the IFL.

pbs_alterjob

alter PBS batch job

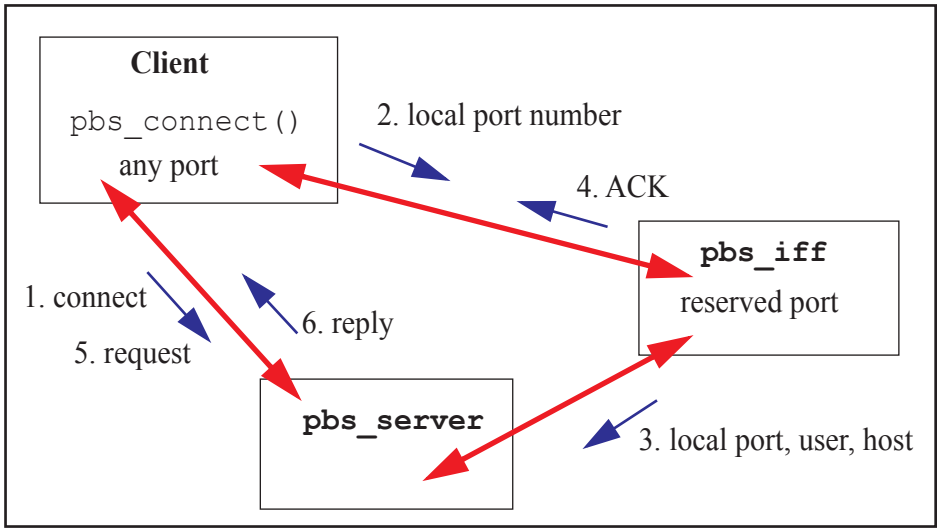

Figure 4-1: Interface Between Client, IFF, and Server

SYNOPSIS

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

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

DESCRIPTION

Issue a batch request to alter a batch job.

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

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

sequence_number.server

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

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

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

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

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

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

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

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

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

SEE ALSO

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

DIAGNOSTICS

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

pbs_connect

connect to a PBS batch server

SYNOPSIS

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

int pbs_connect(char *server)
extern char *pbs_server;
```

DESCRIPTION

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

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

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

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

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

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

pbs_connect() determines whether or not the complex has a failover server configured. It also determines which server is the primary and which is the secondary. pbs_connect() is called by client commands, and directs traffic to the correct server.

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

SEE ALSO

qsub(1B), pbs_alterjob(3B), pbs_deljob(3B), pbs_disconnect(3B), pbs_geterrmsg(3B), pbs_holdjob(3B), pbs_locjob(3B), pbs_manager(3B), pbs_movejob(3B), pbs_msgjob(3B), pbs_rerunjob(3B), pbs_rlsjob(3B), pbs_runjob(3B), pbs_selectjob(3B), pbs_selstat(3B), pbs_sigjob(3B), pbs_statjob(3B), pbs_statque(3B), pbs_statserver(3B), pbs_submit(3B), pbs_terminate(3B), pbs_server(8B)

DIAGNOSTICS

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

pbs_default

return the name of the default PBS server

SYNOPSIS

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

char *pbs_default()
```

DESCRIPTION

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

DIAGNOSTICS

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

SEE ALSO

qsub(1B), pbs_connect(3B), pbs_disconnect(3B)

pbs_deljob

delete a PBS batch job

SYNOPSIS

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

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

DESCRIPTION

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

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

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

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

SEE ALSO

qdel(1B) and pbs_connect(3B)

DIAGNOSTICS

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

pbs_delresv

delete a reservation

SYNOPSIS

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

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

DESCRIPTION

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

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

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

The argument, extend is currently unused.

SEE ALSO

pbs_rdel(1B) and pbs_connect(3B)

DIAGNOSTICS

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

pbs_disconnect

disconnect from a PBS batch server

SYNOPSIS

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

int pbs_disconnect(int connect)
```

DESCRIPTION

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

SEE ALSO

pbs_connect(3B)

DIAGNOSTICS

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

pbs_geterrmsg

get error message for last PBS batch operation

SYNOPSIS

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

char *pbs_geterrmsg(int connect)
```

DESCRIPTION

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

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

SEE ALSO

pbs_connect(3B)

DIAGNOSTICS

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

pbs_holdjob

place a hold on a PBS batch job

SYNOPSIS

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

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

DESCRIPTION

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

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

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

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

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

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

SEE ALSO

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

DIAGNOSTICS

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

pbs_locjob

locate current location of a PBS batch job

SYNOPSIS

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

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

DESCRIPTION

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

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

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

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

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

SEE ALSO

qsub(1B) and pbs_connect(3B)

DIAGNOSTICS

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

pbs_manager

modifies a PBS batch object

SYNOPSIS

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

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

DESCRIPTION

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

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

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

MGR_CMD_CREATE creates the object
MGR_CMD_DELETE deletes the object
MGR_CMD_SET sets the value
MGR_CMD_UNSET unsets the value
MGR_CMD_IMPORT imports the hook
MGR_CMD_EXPORT exports the hook

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

MGR_OBJ_SERVER Server object
MGR_OBJ_QUEUE Queue object
MGR_OBJ_NODE Node object
MGR_OBJ_HOOK Hook object
MGR_OBJ_PBS_HOOK Built-in hook object

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

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

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

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

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

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

The value member points to a string which is the new value of the attribute. For parameterized limit attributes, this string contains all parameters for the attribute.

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

For MGR_CMD_IMPORT, specify attropl "name" as "content-type", "content-encoding", and "input-file" along with the corresponding "value" and an "op" of SET.

For MGR_CMD_EXPORT, specify attropl "name" as "content-type", "content-encoding", and "output-file" along with the corresponding "value" and an "op" of SET.

The parameter extend is reserved for implementation-defined extensions.

Privilege required for functions depends on whether those functions are used with hooks. When not used with hooks:

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

When used with hooks:

All commands require root privilege on the server host.

Functions MGR_CMD_IMPORT, MGR_CMD_EXPORT, and MGR_OBJ_HOOK are used only with hooks, and therefore require root privilege on the server host.

When obj_name is MGR_OBJ_PBS_HOOK, the only allowed options for command are MGR_CMD_SET, MGR_CMD_UNSET, MGR_CMD_IMPORT, and MGR_CMD_EXPORT.

If MGR_CMD_IMPORT or MGR_CMD_EXPORT is specified when obj_name is MGR_OBJ_PBS_HOOK, the attropl content-type must be "application/x-config".

DIAGNOSTICS

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

The pbs_geterrmsg() function can be called to determine the last error message received from the pbs_manager() call.

SEE ALSO
qmgr(1B), pbs_connect(3B)

pbs_movejob

move a PBS batch job to a new destination

SYNOPSIS

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

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

DESCRIPTION

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

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

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

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

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

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

SEE ALSO

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

DIAGNOSTICS

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

pbs_msgjob

record a message for a running PBS batch job

SYNOPSIS

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

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

DESCRIPTION

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

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

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

The parameter, file , indicates the file or files to which the message string is to be written. See pbs_ifl.h for acceptable values.

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

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

SEE ALSO

qmsg(1B) and pbs_connect(3B)

DIAGNOSTICS

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

pbs_orderjob

reorder PBS batch jobs in a queue

SYNOPSIS

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

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

DESCRIPTION

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

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

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

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

SEE ALSO

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

DIAGNOSTICS

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

pbs_rerunjob

rerun a PBS batch job

SYNOPSIS

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

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

DESCRIPTION

Issue a batch request to rerun a batch job.

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

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

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

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

SEE ALSO

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

DIAGNOSTICS

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

pbs_rlsjob

release a hold on a PBS batch job

SYNOPSIS

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

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

DESCRIPTION

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

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

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

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

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

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

SEE ALSO

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

DIAGNOSTICS

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

pbs_runjob, pbs_asyrunjob

run a PBS batch job, asynchronous batch job

SYNOPSIS

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

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

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

DESCRIPTION

Issue a batch request to run a batch job.

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

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

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

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

sequence_number.server

The argument, location, if not the null pointer or null string, specifies the location where the job should be run, and optionally the resources to use. The location is the same as the -H option to the qrun command. See the description of qrun -H, both with and without resources specified, in the qrun.8B man page.

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

SEE ALSO

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

DIAGNOSTICS

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

pbs_selectjob

select PBS batch jobs

SYNOPSIS

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

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

DESCRIPTION

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

The attropl struct contains the list of selection criteria.

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

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

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

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

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

The name member points to a string which is the name of the attribute. Not all of the job attributes may be used as a selection criteria. The resource member points to a string which is the name of a resource. This member is only used when name is set to ATTR_l. Otherwise, resource should be a pointer to a null string. The value member points to a string which is the value of the attribute or resource. The attribute names are listed in pbs_job_attributes.7B.

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

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

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

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

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

Finished and Moved Jobs
In order to get information on finished and moved jobs, you must add an 'x' character to the extend parameter. The extend parameter is a character string; set one character to be the 'x' character. For example:

pbs_selectjob (..., ..., extend) ...

To get information on finished and moved jobs only, specify the Finished ('F') and moved ('M') job states. You must also use the extend character string containing the 'x' character.

Subjobs are not considered finished until the parent array job is finished.

SEE ALSO

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

DIAGNOSTICS

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

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

pbs_selstat

obtain status of selected PBS batch jobs

SYNOPSIS

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

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

void pbs_statfree(struct batch_status *psj)
```

DESCRIPTION

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

The sel_list struct holds the selection criteria. The rattrib struct holds the list of attributes whose values are to be returned.

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

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

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

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

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

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

The name member points to a string which is the name of the attribute. Not all of the job attributes may be used as a selection criteria. The resource member points to a string which is the name of a resource. This member is only used when name is set to ATTR_l, otherwise it should be a pointer to a null string. The value member points to a string which is the value of the attribute or resource. The attribute names are listed in pbs_job_attributes.7B.

The op member defines the operator in the logical expression:

value operator current_value

The logical expression must evaluate as true for the job to be

selected. The permissible values of op are defined in pbs_ifl.h as:
"enum batch_op { ..., EQ, NE, GE, GT, LE, LT, ... };" .
The attributes marked with (E) in the description above may only be selected with the equal, EQ, or not equal, NE, operators.

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

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

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

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

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

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

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

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

Finished and Moved Jobs
In order to get information on finished and moved jobs, you must add an 'x' character to the extend parameter. The extend parameter is a character string; set one character to be the 'x' character. For example:

```
pbs_selstat ( ..., ..., ..., extend) ...
```

To get information on finished and moved jobs only, specify the Finished ('F') and moved ('M') job states. You must also use the extend character string containing the 'x' character. For example:

```
sel_list->next = sel_list;
sel_list->name = ATTR_state;
```

```
sel_list->value = "MF";
sel_list->op = EQ;
pbs_selstat ( ..., sel_list, ..., extend) ...
```

Subjobs are not considered finished until the parent array job is finished.

SEE ALSO

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

DIAGNOSTICS

When the batch request generated by pbs_selstat() function has been completed successfully by a batch server, the routine will return a pointer to the list of batch_status structures. If no jobs met the criteria or an error occurred, the return will be the null pointer. If an error occurred, the global integer pbs_errno will be set to a non-zero value.

pbs_sigjob

send a signal to a PBS batch job

SYNOPSIS

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

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

DESCRIPTION

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

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

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

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

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

SEE ALSO

qsig(1B) and pbs_connect(3B)

DIAGNOSTICS

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

pbs_stagein

request that files for a PBS batch job be staged in.

SYNOPSIS

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

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

DESCRIPTION

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

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

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

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

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

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

SEE ALSO

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

DIAGNOSTICS

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

pbs_statfree

NAME

pbs_statfree - free a PBS status object

SYNOPSIS

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

void pbs_statfree(struct batch_status *psj)
```

DESCRIPTION

Frees the specified PBS status object returned by pbs_statque, pbs_statserver, pbs_stathook, etc.

The argument is a pointer to a batch_status structure. The batch_status structure is defined in pbs_ifl.h as

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

No error information is returned.

pbs_statjob

obtain status of PBS batch jobs

SYNOPSIS

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

struct batch_status *pbs_statjob(int connect, char *id, struct attrl *attrib, char *extend)
void pbs_statfree(struct batch_status *psj)
```

DESCRIPTION

Issue a batch request to query and return the status of a specified batch job, a list of batch jobs, or a set of batch jobs at a queue or server.

A Status Job batch request is generated and sent to the server over the connection specified by connect.

PARAMETERS

The connect parameter is the return value of pbs_connect().

The id parameter can be a job identifier, a list of job identifiers, or a queue or server identifier.

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

```
sequence_number.server
or, for an array job,
sequence_number[].server
```

where the first character of the identifier must be a digit. If the identifier is for an array job but the 't' character is not included in the extend parameter, the status of the array job is returned, but not the status of its subjobs. If the 't' character is included, status for each subjob is returned.

If id is a list of job identifiers, it must be a comma-separated list of job identifiers in a single string. The first character of the string must be a digit. White space outside of a job identifier is ignored. There is no limit to the length of the string except as imposed by available memory.

If id is a destination (server or queue) identifier, the status of all jobs at the destination which the user is authorized to see is returned.

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

The attrib parameter is a pointer to a list of attributes. If attrib is specified, then only the attributes in the list are returned by the server. Otherwise, all the attributes of a job are returned. The attrib parameter is a pointer to an attrl structure which is defined in pbs_ifl.h as:

```
struct attrl {
```

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

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

The name member is a pointer to an attribute name. Attribute names are listed in pbs_ifl.h.

The resource member specifies the name of the resource in the job's Resource_List attribute. When attrl describes the job's Resource_List job attribute, the name member is ATTR_l. If attrl is not ATTR_l, resource should be a pointer to a null string.

The value member should always be a pointer to a null string.

The extend parameter is used for extensions. This parameter can consist of characters in any order.

When the extend parameter includes 't', if any array job identifiers are in the set of IDs being queried, the status of each array job is followed by status of each subjob in the array job.

When the extend parameter includes 'x', finished and moved jobs and subjobs can be queried, and their status is included. Subjobs are not considered finished until the parent array job is finished.

RETURN VALUES and ERRORS

For a single job, if the job can be queried, the return value is a pointer to a batch_status structure containing the status of the specified job. If the job cannot be queried, a NULL pointer is returned, and pbs_errno is set to an error number.

For a list of jobs, if any of the specified jobs can be queried, the return value is a pointer to a batch_status structure containing the status of all the queryable jobs. If none of the jobs can be queried, a NULL pointer is returned, and pbs_errno is set to the error number that indicates the reason that the last job in the list could not be queried.

For a queue, if the queue exists, the return value is a pointer to a batch_status structure containing the status of all the queryable jobs in the queue. If the queue does not exist, a NULL pointer is returned, and pbs_errno is set to PBSE_UNKQUE (15018). If the queue exists but contains no queryable jobs, a NULL pointer ia returned and pbs_errno is set to PBSE_NONE (0).

When querying a server, the connection to the server is already established by pbs_connect(). If there are jobs at the server, the return value of the query is a pointer to a batch_status structure containing the status of all the queryable jobs at the server. If the server does not contain any queryable jobs, a NULL pointer ia returned and pbs_errno is set to PBSE_NONE (0).

The batch_status structure is defined in pbs_ifl.h as

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

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

SEE ALSO

qstat(1B) and pbs_connect(3B)

pbs_statnode, pbs_statvnode, pbs_stathost

obtain status of PBS vnodes or hosts

SYNOPSIS

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

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

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

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

void pbs_statfree(struct batch_status *psj)
```

DESCRIPTION

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

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

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

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

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

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

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

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

The attrib list is terminated by the first entry where next is a null pointer. If attrib is given, then only the attributes in the list are returned by the server. Otherwise, all the attributes of a node are returned. When an attrib list is specified, the name member is a

pointer to a attribute name. The resource member is not used and must be a pointer to a null string. The value member should always be a pointer to a null string.

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

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

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

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

DIAGNOSTICS

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

SEE ALSO

qstat(1B), pbs_connect(3B)

pbs_statque

obtain status of PBS batch queues

SYNOPSIS

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

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

void pbs_statfree(struct batch_status *psj)
```

DESCRIPTION

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

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

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

queue_name

or the null string. If

queue_name

is specified, the status of the queue named

queue_name

at the server is returned. If the id is a null string or null pointer, the status of all queues at the server is returned.

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

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

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

When pbs_statque is used to get the attributes of an object, a single attrl data structure is returned for each parameterized attribute.

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

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

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

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

SEE ALSO

qstat(1B) and pbs_connect(3B)

DIAGNOSTICS

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

pbs_statresv

obtain status information about reservations

SYNOPSIS

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

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

void pbs_statfree(struct batch_status *psj)
```

DESCRIPTION

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

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

The parameter, id , is a reservation identifier. A reservation identifier is of the form:

"R<sequence_number>.<server>"

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

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

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

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

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

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

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

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

SEE ALSO

pbs_rstat(1B) and pbs_connect(3B)

DIAGNOSTICS

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

pbs_statsched

obtain status of PBS scheduler

SYNOPSIS

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

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

void pbs_statfree(struct batch_status *psj)
```

DESCRIPTION

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

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

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

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

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

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

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

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

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

SEE ALSO

qstat(1B) and pbs_connect(3B)

DIAGNOSTICS

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

pbs_statserver

obtain status of a PBS batch server

SYNOPSIS

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

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

void pbs_statfree(struct batch_status *psj)
```

DESCRIPTION

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

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

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

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

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

When pbs_statserver is used to get the attributes of an object, a single attrl data structure is returned for each parameterized attribute.

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

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

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

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

SEE ALSO

qstat(1B) and pbs_connect(3B)

DIAGNOSTICS

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

pbs_submit

submit a PBS batch job

SYNOPSIS

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

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

DESCRIPTION

Issue a batch request to submit a new batch job.

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

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

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

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

The name member points to a string which is the name of the attribute. The value member points to a string which is the value of the attribute. The attribute names are defined in pbs_job_attributes(7B).

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

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

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

The parameter, script , is the path name to the job script. If the path name is relative, it will be expanded to the processes current working directory. If script is a null pointer or the path name

pointed to is specified as the null string, no script is passed with the job.

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

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

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

SEE ALSO

qsub(1B) and pbs_connect(3B)

DIAGNOSTICS

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

pbs_submit_resv

submit a PBS reservation

SYNOPSIS

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

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

DESCRIPTION

Issue a batch request to submit a new reservation.

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

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

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

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

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

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

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

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

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

The return value is a character string which is the reservation_identifier assigned to the job by the server. The space for the reserva-

tion_identifier string is allocated by pbs_submit_resv() and should be released via a call to free() by the user when no longer needed.

SEE ALSO

pbs_rsub(1B) and pbs_connect(3B)

DIAGNOSTICS

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

pbs_terminate

terminate a PBS batch server

SYNOPSIS

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

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

DESCRIPTION

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

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

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

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

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

This call requires PBS Operator or Manager privilege.

SEE ALSO

qterm(8B) and pbs_connect(3B)

DIAGNOSTICS

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

5
RPP Library

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

5.1 RPP Library Routines

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

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

reliable packet protocol

SYNOPSIS

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

int rpp_open(addr)
    struct sockadd_in *addr;

int rpp_bind(port)
    int port;

int rpp_poll()

int rpp_io()

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

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

int rpp_close(stream)
    u_int stream;

struct sockadd_in *rpp_getaddr(stream)
    u_int stream;

int rpp_flush(stream)
    u_int stream;

int rpp_terminate()

int rpp_shutdown()

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

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

int rpp_eom(stream)
```

```
    u_int stream;

int rpp_getc(stream)
    u_int stream;

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

DESCRIPTION

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

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

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

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

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

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

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

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

rpp_write() adds information to the current message on a stream. The data in buf numbering len characters is transferred to the stream. The

number of characters added to the stream are returned or a -1 on error. In this case, errno will be set. A -2 is returned if the peer has closed its connection.

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

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

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

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

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

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

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

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

SEE ALSO

tcp(4P), udp(4P)

6
TM Library

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

6.1 TM Library Routines

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

tm_init, tm_nodeinfo, tm_poll, tm_notify, tm_spawn, tm_kill, tm_obit, tm_taskinfo, tm_atnode, tm_rescinfo, tm_publish, tm_subscribe, tm_finalize, tm_attach

task management API

SYNOPSIS

```
#include <tm.h>

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

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

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

int tm_notify(tm_signal)
    int tm_signal;

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

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

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

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

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

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

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

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

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

int tm_finalize()
```

DESCRIPTION

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

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

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

```
typedef  int          tm_node_id;   /* job-relative node id */
#define  TM_ERROR_NODE  ((tm_node_id)-1)
typedef  int          tm_event_t;   /* > 0 for real events */
#define  TM_NULL_EVENT  ((tm_event_t)0)
#define  TM_ERROR_EVENT ((tm_event_t)-1)
typedef  unsigned long  tm_task_id;
#define  TM_NULL_TASK   (tm_task_id)0
```

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

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

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

tm_me The task id of this calling task.

tm_parent The task id of the task which spawned this task or TM_NULL_TASK if the calling task is the initial task started by PBS.

tm_nnodes The number of nodes allocated to the job.

tm_ntasks This will always be 0 for PBS.

tm_taskpoolid PBS does not support task pools so this will always be -1.

tm_tasklist This will be NULL for PBS.

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

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

tm_poll() is the function which will retrieve information about the task management system to locations specified when other routines request an action take place. The bookkeeping for this is done by generating an event for each action. When the task manager (MoM) sends a message that an action is complete, the event is reported by tm_poll and information is

placed where the caller requested it. The argument poll_event is meant to be used to request a specific event. This implementation does not use it and it must be set to TM_NULL_EVENT or an error is returned. Upon return, the argument result_event will contain a valid event number or TM_ERROR_EVENT on error. If wait is zero and there are no events to report, result_event is set to TM_NULL_EVENT. If wait is non-zero and there are no events to report, the function will block waiting for an event. If no local error takes place, TM_SUCCESS is returned. If an error is reported by MoM for an event, then the argument tm_errno will be set to an error code.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

7
RM Library

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

7.1 RM Library Routines

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

openrm, closerm, downrm, configrm, addreq, allreq, getreq, flushreq, activereq, fullresp

resource monitor API

SYNOPSIS

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

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

int closerm (stream)
int stream;

int downrm (stream)
int stream;

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

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

int allreq (line)
char *line;

char *getreq(stream)
int stream;

int flushreq()

int activereq()

void fullresp(flag)
int flag;
```

DESCRIPTION

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

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

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

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

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

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

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

fullresp() turns on, if flag is true, "full response" mode where getreq() returns a pointer to the beginning of a line of response. This is the default. If flag is false, the line returned by getreq() is just the answer following the equal sign.

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

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

SEE ALSO

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

8
TCL/tk Interface

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

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

8.1 TCL/tk API Functions

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

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

pbs_tclapi

PBS TCL Application Programming Interface

DESCRIPTION

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

USAGE

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

openrm host ?port?

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

closerm connection

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

downrm connection

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

configrm connection filename

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

addreq connection request

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

getreq connection

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

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

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

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

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

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

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

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

pbsstatnode
The server is sent a status request for information about all nodes defined within the server. If the request succeeds, a list is returned, otherwise an empty string is returned. The list contains an entry for each node. Each element is a list with

three elements. This first is the node's name. The second is a list of attributes similar to pbsstatjob. The third is the "text" associated with the node (usually blank).

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

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

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

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

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

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

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

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

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

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

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

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

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

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

pbsrescrelease resource_id
Deprecated. The reservation specified by resource_id is released.

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

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

pbsdisconnect
Disconnect from the currently connected server.

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

datetime ?day? ?time?

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

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

YYMMDDhhmmss

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

The second option with one argument is a relative time. The format for this is

HH:MM:SS

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

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

strftime format time

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

logmsg tag message

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

SEE ALSO

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

pbs_sched(8B)

9 Hooks

This chapter describes the PBS hook APIs. For more information on hooks, see the PBS Professional Administrator's Guide.

9.1 Introduction

A hook is a block of Python code that is triggered in response to queueing a job, modifying a job, moving a job, running a job, submitting a PBS reservation, MoM receiving a job, MoM starting a job, MoM killing a job, a job finishing, and MoM cleaning up a job. Each hook can *accept* (allow) or *reject* (prevent) the action that triggers it. The hook can modify the input parameters given for the action. The hook can also make calls to functions external to PBS. PBS provides an interface for use by hooks. This interface allows hooks to read and/or modify things such as job and server attributes, the server, queues, and the event that triggered the hook.

The Administrator creates any desired hooks.

This chapter contains the following man pages:

- `pbs_module(7B)`
- `pbs_stathook(3B)`

See the following additional man pages:

- `qmgr(1B)`
- `qsub(1B)`
- `qmove(1B)`
- `qalter(1B)`
- `pbs_rsub(1B)`
- `pbs_manager(3B)`

9.2 How Hooks Work

9.2.1 Hook Contents and Permissions

A hook contains a Python script. The script is evaluated by a Python 2.5 or later interpreter, embedded in PBS.

Hooks have a default Linux umask of 022. File permissions are inherited from the current working directory of the hook script.

9.2.2 Accepting and Rejecting Actions

The hook script always accepts the current event request action unless an unhandled exception occurs in the script, a hook alarm timeout is triggered or there's an explicit call to "pbs.event().reject()".

9.2.3 Exceptions

A hook script can catch an exception and evaluate whether or not to accept or reject the event action. In this example, while referencing the non-existent attribute pbs.event().job.interactive, an exception is triggered, but the event action is still accepted:

```
…
try:
    e = pbs.event()
    if e.job.interactive:
         e.reject("Interactive jobs not allowed")
except SystemExit:
    pass
except:
    e.accept()
```

9.2.4 Unsupported Interfaces and Uses

Site hooks which read, write, close, or alter stdin, stdout, or stderr, are not supported. Hooks which use any interfaces other than those described are unsupported.

9.3 Interface to Hooks

Two PBS APIs are used with hooks. These are pbs_manager() and pbs_stathook(). The pbs module provides a Python interface to PBS.

9.3.1 The pbs Module

Hooks have access to a special module called "pbs", which contains functions that perform PBS-related actions. This module must be explicitly loaded by the hook writer via the call "import pbs" .

The *pbs module* provides an interface to PBS and the hook environment. The interface is made up of Python objects, which have attributes and methods. You can operate on these objects using Python code.

9.3.1.1 Description of pbs Module

pbs_module

The interface is made up of Python objects, which have attributes and methods. You can operate on these objects using Python code. For a description of each object, see the PBS Professional Administrator's Guide.

9.3.1.2 pbs Module Objects

See "The pbs Module" on page 76 in the PBS Professional Hooks Guide.

9.3.1.3 pbs Module Global Attribute Creation Methods

See "Global Methods" on page 136 in the PBS Professional Hooks Guide.

9.3.1.4 Attributes and Resources

See "Using Attributes and Resources in Hooks" on page 42 in the PBS Professional Hooks Guide.

9.3.1.5 Exceptions

See "Table of Exceptions" on page 40 in the PBS Professional Hooks Guide and "Hook Alarm Calls and Unhandled Exceptions" on page 41 in the PBS Professional Hooks Guide.

9.3.1.6 See Also

The PBS Professional Administrator's Guide, pbs_hook_attributes(7B), pbs_resources(7B), qmgr(1B)

9.3.2 The pbs_manager() API

The pbs_manager() API is described in "pbs_manager" on page 37.

The pbs_manager() API contains an obj_name called "hook" defined as MGR_OBJ_HOOK, for use with non-built-in hooks.

The pbs_manager() API contains an obj_name called "pbshook" defined as MGR_OBJ_PBS_HOOK, for use with built-in hooks.

To run, hooks require root privilege on Linux, and local Administrators privilege on Windows. Hooks run only on the server host.

The pbs_manager() API contains the following hook commands, which operate only on hook objects:

MGR_CMD_IMPORT
This command is used for loading the hook script contents into a hook.

MGR_CMD_EXPORT
This command is used for dumping to a file the contents of a hook script.

The parameters to MGR_CMD_IMPORT and MGR_CMD_EXPORT are specified via the attrib parameter of `pbs_manager()`.

For MGR_CMD_IMPORT, specify attropl "name" as "content-type", "content-encoding", and "input-file" along with the corresponding "value" and an "op" of SET.

For MGR_CMD_EXPORT, specify attropl "name" as "content-type", "content-encoding", and "output-file" along with the corresponding "value" and an "op" of SET.

Functions MGR_CMD_IMPORT, MGR_CMD_EXPORT, and MGR_OBJ_HOOK are used only with hooks, and therefore require root privilege on the server host.

When obj_name is MGR_OBJ_PBS_HOOK, the only allowed options for command are MGR_CMD_SET, MGR_CMD_UNSET, MGR_CMD_IMPORT, and MGR_CMD_EXPORT.

If MGR_CMD_IMPORT or MGR_CMD_EXPORT is specified when obj_name is MGR_OBJ_PBS_HOOK, the attropl content-type must be “application/x-config”.

9.3.2.1 Troubleshooting

You can use pbs_geterrmsg() to determine the last error message received from the pbs_manager() call. For instance, with a MGR_OBJ_PBS_HOOK where command is either MGR_CMD_IMPORT or MGR_CMD_EXPORT, but attropl 'content-type' is not “application/x-config”, pbs_geterrmsg() returns:

```
“<content-type> must be application/x-config”
```

If an unrecognized hook configuration file suffix is given, whether for MGR_OBJ_HOOK or MGR_OBJ_PBS_HOOK, pbs_geterrmsg() returns:

```
“<input-file> contains an invalid suffix, should be one of: .json .py .txt .xml .ini”
```

If the hook configuration file failed to be precompiled by PBS, pbs_geterrmsg() show:s

```
“Failed to validate config file, hook '<hook_name>' config file not overwritten"
```

9.3.2.2 Examples of Using pbs_manager()

Example 9-1: The following:

```
# qmgr -c 'import hook hook1 application/x-python base64 hello.py.b64'
```

is programmatically equivalent to:

```
static struct attropl imp_attribs[] = {
    {   "content-type",
        (char *)0,
        "application/x-python",
        SET,
        (struct attropl *)&imp_attribs[1]
    },
    {   "content-encoding",
        (char *)0,
        "base64",
        SET,
        (struct attropl *)&imp_attribs[2]},
    {   "input-file",
        (char *)0,
        "hello.py.b64",
        SET,
        (struct attropl *)0
    }
};

pbs_manager(con, MGR_CMD_IMPORT, MGR_OBJ_HOOK, "hook1", &imp_attribs[0], NULL);
```

Example 9-2: The following:

```
# qmgr -c 'export hook hook1 application/x-python default hello.py'
```

is programmatically equivalent to:

```
static struct attropl exp_attribs[] = {
    {   "content-type",
        (char *)0,
        "application/x-python",
        SET,
        (struct attropl *)&exp_attribs[1]},
    {   "content-encoding",
        (char *)0,
        "default",
        SET,
        (struct attropl *)&exp_attribs[2]},
    {   "output-file",
        (char *)0,
        "hello.py",
        SET,
        (struct attropl *)0
```

```
    }
};

pbs_manager(con, MGR_CMD_EXPORT, MGR_OBJ_HOOK, "hook1", &exp_attribs[0], NULL);
```

Example 9-3: The following:

```
# qmgr -c 'import pbshook hook1 application/x-config default hello.json'
```

is programmatically equivalent to:

```
static struct attropl imp_attribs[] = {
    {   "content-type",
        (char *)0,
        "application/x-config",
        SET,
        (struct attropl *)&imp_attribs[1]},
    {   "content-encoding",
        (char *)0,
        "default",
        SET,
        (struct attropl *)&imp_attribs[2]},
    {   "input-file",
        (char *)0,
        "hello.json",
    <   SET,
        (struct attropl *)0
    }
};

pbs_manager(con, MGR_CMD_IMPORT, MGR_OBJ_PBS_HOOK, "hook1", &imp_attribs[0], NULL);
```

Example 9-4: The following:

```
# qmgr -c 'export pbshook hook1 application/x-config default hello.json'
```

is programmatically equivalent to:

```
static struct attropl exp_attribs[] = {
    {   "content-type",
        (char *)0,
        "application/x-config",
        SET,
        (struct attropl *)&exp_attribs[1]},
    {   "content-encoding",
        (char *)0,
        "default",
        SET,
        (struct attropl *)&exp_attribs[2]},
    {   "output-file",
        (char *)0,
        "hello.json",
        SET,
```

```
        (struct attropl *)0
    }
};

pbs_manager(con, MGR_CMD_EXPORT,
MGR_OBJ_PBS_HOOK, "hook1", &exp_attribs[0], NULL);
```

9.3.3 The pbs_stathook() API

The PBS API called "pbs_stathook()" is used to get attributes and values for site hooks and built-in hooks.

The prototype for pbs_stathook() is as follows:

```
struct batch_status *pbs_stathook(int connect, char *hook_name, struct attrl *attrib, char
    *extend)
```

To query status for site hooks:

The call to pbs_stathook() causes a PBS_BATCH_StatusHook request to be sent to the server. In reply, the PBS server returns a batch reply status of object type MGR_OBJ_HOOK listing the attributes and values that were requested relating to a particular hook or all hooks of type HOOK_SITE. Leave the extend value blank.

To query status for built-in hooks:

Pass PBS_HOOK as the extend value. The server returns a batch reply status of object type MGR_OBJ_PBS_HOOK.

9.3.3.1 Example of Using pbs_stathook()

To list all site hooks using `qmgr`:

```
qmgr -c "list hook"
```

To list all site hooks using the `pbs_stathook()` API:

```
pbs_stathook()
```

The result is the same. For example, if there are two site hooks, c3 and c36:

```
Hook c3
        type = site
        enabled = true
        event = queuejob, modifyjob
        user = pbsadmin
        alarm = 30
        order = 1
Hook c36
        type = site
        enabled = true
        event = resvsub
        user = pbsadmin
        alarm = 30
        order = 1
```

9.3.3.2 Description of pbs_stathook() API

pbs_stathook(3B)

SYNOPSIS

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

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

void pbs_statfree(struct batch_status *psj)
```

DESCRIPTION

Issue a batch request to obtain the status of a specified site hook or a set of site hooks at the current server.

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

This API can be executed only by root on the local server host.

The parameter, id, may be either a hook name or the null string. If id specifies a name, the attribute-value list for that hook is returned. If id is a null string or a null pointer, the status of a all hooks at the current server is returned.

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

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

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

If an attrib list is given, then only the attributes in the list are returned by the server. Otherwise, all the attributes of a hook are returned.

The resource member is only used if the name member is ATTR_l, otherwise it should be a pointer to a null string.

The value member should always be a pointer to a null string.

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

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

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

```
    char            *text;
}
```

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

SEE ALSO

pbs_hook_attributes(7B), pbs_connect(3B)

DIAGNOSTICS

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

9.3.4 Error Messages

The following error message is returned by the pbs_geterrmsg() API after calling pbs_manager() operating on a hook object, with the MGR_CMD_IMPORT command, with “content-type” of “application/x-config”:

```
“Failed to validate config file, hook 'submit' config file not overwritten”
```

If the input config file given is of unrecognized suffix, then the following message is returned by the pbs_geterrmsg() API after calling pbs_manager() operating on a hook object, MGR_CMD_IMPORT command with “content-type” of “application/x-config”:

```
“<input-file> contains an invalid suffix, should be one of: .json .py .txt .xml .ini”
```

If you specify an unknown hook event, pbs_geterrmsg() returns the following after calling pbs_manager():

```
invalid argument (<bad_event>) to event. Should be one or more of:
    queuejob,modifyjob,resvsub,movejob,runjob,provision,execjob_begin,execjob_prologue,execjob_e
    pilogue,execjob_preterm,execjob_end,exechost_periodic,execjob_launch,exechost_startup or ""
    for no event
```

If you specify an invalid value for a hook’s debug attribute, the following error message appears in qmgr's STDERR and is returned by pbs_geterrmsg() after calling pbs_manager():

```
“unexpected value '<bad_val>' must be (not case sensitive) true|t|y|1|false|f|n|0”
```

A runjob hook cannot set the value of a Resource_LIst member other than those listed in "Table: Reading & Setting Job Resources in Hooks" on page 56 in the PBS Professional Hooks Guide. Setting any of the wrong resources results in the following:

- The hook request is rejected
- The following message is the output from calling pbs_geterrmsg() after the failed pbs_runjob():

```
“ request rejected by filter hook: '<hook name>' hook failed to set job's
    Resource_List.<resc_name> = <resc_value> (not allowed)”
```

Index