Altair PBS Professional 2022.1

Programmer's Guide

You are reading the Altair PBS Professional 2022.1

Programmer's Guide (PG)

Updated 7/19/22

Copyright © 2003-2022 Altair Engineering, Inc. All rights reserved.

ALTAIR ENGINEERING INC. Proprietary and Confidential. Contains Trade Secret Information. Not for use or disclosure outside of Licensee's organization. The software and information contained herein may only be used internally and are provided on a non-exclusive, non-transferable basis. Licensee may not sublicense, sell, lend, assign, rent, distribute, publicly display or publicly perform the software or other information provided herein, nor is Licensee permitted to decompile, reverse engineer, or disassemble the software. Usage of the software and other information provided by Altair (or its resellers) is only as explicitly stated in the applicable end user license agreement between Altair and Licensee. In the absence of such agreement, the Altair standard end user license agreement terms shall govern.

Use of Altair's trademarks, including but not limited to "PBS™", "PBS Professional®", and "PBS Pro™", "PBS Works™", "PBS Control™", "PBS Access™", "PBS Analytics™", "PBScloud.io™", and Altair's logos is subject to Altair's trademark licensing policies. For additional information, please contact Legal@altair.com and use the wording "PBS Trademarks" in the subject line.

For a copy of the end user license agreement(s), log in to https://secure.altair.com/UserArea/agreement.html or contact the Altair Legal Department. For information on the terms and conditions governing third party codes included in the Altair Software, please see the Release Notes.

This document is proprietary information of Altair Engineering, Inc.

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

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

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	pbs@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 +1 800 208 9234 (Toll Free)	pbs-support@india.altair.com
Italy	+39 800 905595	pbssupport@europe.altair.com
Japan	+81 3 6225 5821	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 Africa	+27 21 831 1500	pbssupport@europe.altair.com
South America	+55 11 3884 0414	br_support@altair.com
UK	+44 (0)1926 468 600	pbssupport@europe.altair.com

Contents

List of APIs vii

About PBS Documentation ix

1 PBS Architecture 1

1.1 PBS Components . . . 1

2 Server Functions 5

2.1 Roles and Required Privilege . . . 5
2.2 Batch Server Functions . . . 5
2.3 Server Management . . . 5
2.4 Queue Management . . . 6
2.5 Vnode Management . . . 6
2.6 Job Management . . . 6
2.7 Server to Server Requests . . . 11
2.8 Deferred Services . . . 12
2.9 Resource Management . . . 17
2.10 Network Protocol . . . 17

3 Developer Headers and Libraries 19

3.1 Location of API Libraries . . . 19
3.2 Location of Header Files . . . 19
3.3 Developer Package . . . 19
3.4 Batch Interface Library . . . 20
3.5 Example Compilation Line . . . 20

4 Batch Interface Library (IFL) 21

4.1 Interface Library Overview . . . 21
4.2 Batch Library Routines . . . 22

5 TM Library 95

5.1 TM Library Routines . . . 95

6 RM Library 101

6.1 RM Library Routines . . . 101

7 TCL/tk Interface 105

7.1 TCL/tk API Functions . . . 105

8 Hooks 111

8.1 Introduction . . . 111
8.2 How Hooks Work . . . 111
8.3 Interface to Hooks . . . 112

9 Custom Authentication and Encryption Library APIs 123

Index 135

List of APIs

4.3 pbs_alterjob . . . 24
4.4 pbs_asyrunjob . . . 26
4.5 pbs_confirmresv . . . 28
4.6 pbs_connect . . . 30
4.7 pbs_default . . . 32
4.8 pbs_deljob . . . 33
4.9 pbs_delresv . . . 35
4.10 pbs_disconnect . . . 36
4.11 pbs_geterrmsg . . . 37
4.12 pbs_holdjob . . . 38
4.13 pbs_locjob . . . 39
4.14 pbs_manager . . . 41
4.15 pbs_modify_resv . . . 45
4.16 pbs_movejob . . . 47
4.17 pbs_msgjob . . . 49
4.18 pbs_orderjob . . . 51
4.19 pbs_preempt_jobs . . . 52
4.20 pbs_relnodesjob . . . 54
4.21 pbs_rerunjob . . . 56
4.22 pbs_rlsjob . . . 57
4.23 pbs_runjob . . . 58
4.24 pbs_selectjob . . . 60
4.25 pbs_selstat . . . 63
4.26 pbs_sigjob . . . 67
4.27 pbs_statfree . . . 69
4.28 pbs_stathost . . . 70
4.29 pbs_statjob . . . 72
4.30 pbs_statnode . . . 75
4.31 pbs_statque . . . 77
4.32 pbs_statresv . . . 79
4.33 pbs_statrsc . . . 81
4.34 pbs_statsched . . . 83
4.35 pbs_statserver . . . 85
4.36 pbs_statvnode . . . 87
4.37 pbs_submit . . . 89
4.38 pbs_submit_resv . . . 91
4.39 pbs_terminate . . . 93
5.2 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 . . . 96
6.2 openrm, closerm, downrm, configrm, addreq, allreq, getreq, flushreq, activereq,

	fullresp	102
7.2	pbs_tclapi	106
8.4	pbs_module	113
8.5	pbs_stathook(3B)	119
9.1	pbs_auth_set_config	124
9.2	pbs_auth_create_ctx	125
9.3	pbs_auth_destroy_ctx	127
9.4	pbs_auth_get_userinfo	128
9.5	pbs_auth_process_handshake_data	130
9.6	pbs_auth_encrypt_data	132
9.7	pbs_auth_decrypt_data	133

About PBS Documentation

The PBS Professional guides and release notes apply to the *commercial* releases of PBS Professional.

Document Conventions

`Abbr`eviation

The shortest acceptable abbreviation of a command or subcommand is underlined

Attribute

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

`Command`

Commands such as `qmgr` and `scp`

Definition

Terms being defined

`File name`

File and path names

`Input`

Command-line instructions

Method

Method or member of a class

`Output`

Output, example code, or file contents

Syntax

Syntax, template, synopsis

`Utility`

Name of utility, such as a program

Value

Keywords, instances, states, values, labels

Notation

Optional Arguments

Optional arguments are enclosed in square brackets. For example, in the `qstat` man page, the `-E` option is shown this way:

qstat [-E]

To use this option, you would type:

```
qstat -E
```

Variable Arguments

Variable arguments (where you fill in the variable with the actual value) such as a job ID or vnode name are enclosed in angle brackets. Here's an example from the `pbsnodes` man page:

pbsnodes -v <vnode>

To use this command on a vnode named "my_vnode", you'd type:

```
pbsnodes -v my_vnode
```

Optional Variables

Optional variables are enclosed in angle brackets inside square brackets. In this example from the `qstat` man page, the job ID is optional:

qstat [<job ID>]

To query the job named "1234@my_server", you would type this:

```
qstat 1234@my_server
```

Literal Terms

Literal terms appear exactly as they should be used. For example, to get the version for a command, you type the command, then "--version". Here's the syntax:

qstat --version

And here's how you would use it:

```
qstat --version
```

Multiple Alternative Choices

When there are multiple options and you should choose one, the options are enclosed in curly braces. For example, if you can use either "-n" or "--name":

```
{-n | --name}
```

List of PBS Professional Documentation

The PBS Professional guides and release notes apply to the *commercial* releases of PBS Professional.

PBS Professional Release Notes

Supported platforms, what's new and/or unexpected in this release, deprecations and interface changes, open and closed bugs, late-breaking information. For administrators and job submitters.

PBS Professional Big Book

All your favorite PBS guides in one place: *Installation & Upgrade, Administrator's, Hooks, Reference, User's*, *Programmer's, Cloud, Budget,* and *Simulate* guides in a single book.

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 PBS administrator.

PBS Professional Reference Guide

Covers PBS reference material: the PBS commands, resource, attributes, configuration files, etc.

PBS Professional User's Guide

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

PBS Professional Programmer's Guide

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

PBS Professional Manual Pages

PBS commands, resources, attributes, APIs.

PBS Professional Licensing Guide

How to configure licensing for PBS Professional. For the PBS administrator.

PBS Professional Cloud Guide

How to configure and use the PBS Professional Cloud feature in order to burst jobs to the cloud.

PBS Professional Budgets Guide

How to configure Budgets and use it to track and manage resource usage by PBS jobs.

PBS Professional Simulate Guide

How to configure and use the PBS Professional Simulate feature.

Where to Keep the Documentation

If you're not using the *Big Book*, make cross-references work by putting 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.

1 PBS Architecture

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

1.1 PBS Components

You can manage one or more machines using PBS. PBS consists of commands, a data service, and the following daemons:

- Server daemon for central management; this daemon runs on Linux only.
- One or more scheduler daemons to schedule jobs; schedulers run on Linux only.
- Communication daemon to manage communication; this daemon also runs only on Linux.
- Job management daemon called MoM to manage each execution host; this daemon can run on Linux or Windows.

The data service runs on Linux only. Commands can run on Linux or Windows.

1.1.1 Single Execution System

If PBS is to manage a single system, all components are installed on that same system. For installation instructions, see the *PBS Professional Installation & Upgrade Guide*.

The following illustration shows how communication works when PBS is on a single host in TPP mode. For more on TPP mode, see "Communication" on page 45 in the PBS Professional Installation & Upgrade Guide.

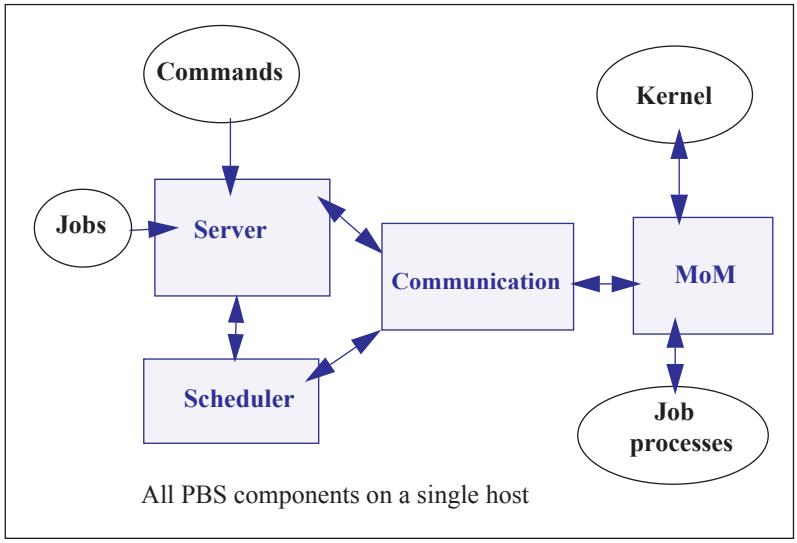

Figure 1-1:PBS daemons on a single execution host

1.1.2 Single Execution System with Front End

The PBS server and default scheduler (`pbs_server` and `pbs_sched`) can run on one system and jobs can execute on another. Job execution is managed by the MoM daemon. 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 "Communication" on page 45 in the PBS Professional Installation & Upgrade Guide.

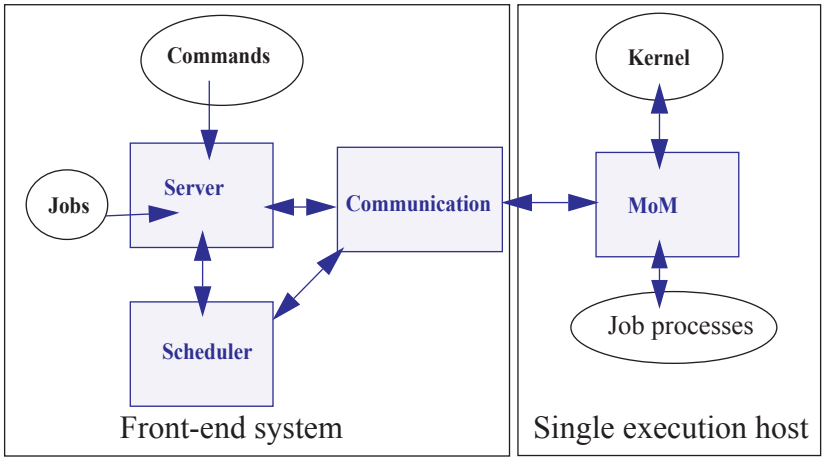

Figure 1-2:PBS daemons on single execution system with front end

1.1.3 Multiple Execution Systems

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

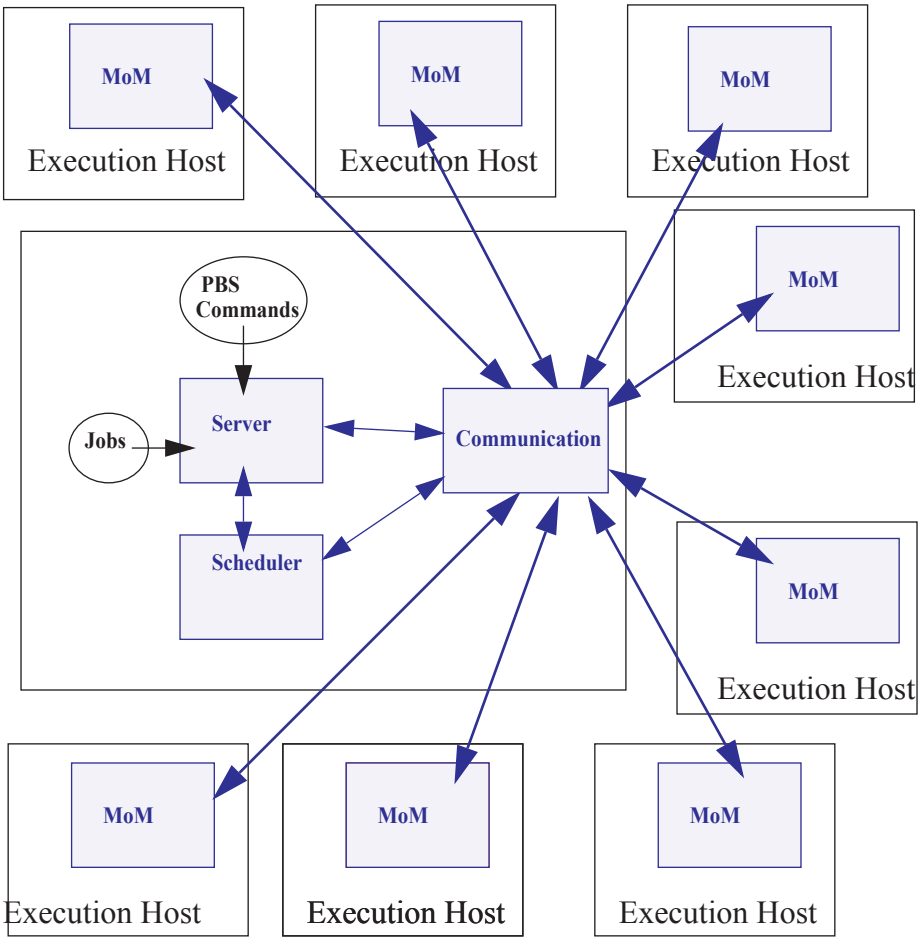

Figure 1-3:Typical PBS daemon locations for multiple execution hosts

1.1.4 Server

The *server* process is the central focus for PBS. In our documentation, it is generally referred to as *the server, the PBS server,* or by the execution name *pbs_server.* All commands and communication with the server are via an *Internet Protocol* (IP) network. The server provides core batch services such as receiving batch job requests, creating batch jobs, modifying jobs, protecting jobs against system crashes, and sending jobs to MoM for execution. One server manages each PBS complex.

1.1.5 Job Executor (MoM)

The *Job Executor* is the component that actually places the job into execution. This daemon, *pbs_mom,* is informally called *MoM* as it is the mother of all executing jobs on that host. MoM places a job into execution when it receives a copy of the job from the 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 returns the job's output to the user. One MoM runs on each execution host (a host where PBS jobs execute).

1.1.6 Schedulers

PBS has a default scheduler; if you want to schedule individual partitions separately, you can add any number of additional schedulers, called *multischeds*. Each PBS scheduler follows its own scheduling policy.

Each scheduler daemon implements a policy that you define that controls when each job is run and on which resources. See "About Schedulers" on page 91 in the PBS Professional Administrator's Guide.

Each scheduler makes a persistent connection to the server via `pbs_connect()`. If the scheduler does not have a connection to the server, it continues trying every 2 seconds until it gets a connection.

1.1.7 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 49.

1.1.8 Privilege

PBS recognizes separate roles and levels of privilege: Manager role is required for sensitive operations, Operator role can perform various less-sensitive functions, and User role allows access to only the user's own jobs. Root privilege is required for some of the Manager operations; the combination of root privilege and Manager role is called PBS Administrator. See "Security" on page 489 in the PBS Professional Administrator's Guide.

1.1.9 Commands

PBS provides a set of commands for submitting and managing jobs, and for managing PBS. PBS commands are described in "PBS Commands" on page 21 of the PBS Professional Reference Guide.

2 Server Functions

2.1 Roles and Required Privilege

PBS recognizes specific roles and levels of privilege, and these are required for some operations on PBS. For details, see "User Roles and Required Privilege" on page 489 in the PBS Professional Administrator's Guide.

2.2 Batch Server Functions

A batch server provides services in the following ways:

- The server provides a service at the request of a client. 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, the server itself, etc. Any requests that cannot be successfully completed are rejected. The reason for the rejection is returned in the reply to the client.
- The server provides a deferred service when it detects a change in conditions that it monitors. The server may, depending on conditions being monitored, defer a client service request until a later time. Deferred services include file staging, sending jobs for execution, etc. See section 2.8, "Deferred Services", on page 12.

The server also performs a number of internal bookkeeping functions.

2.3 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, reservation, vnode, hook, 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. For a list of batch request codes, see "Request Codes" on page 393 of the PBS Professional Reference Guide.

2.3.1 Manage Request

The *Manage* request supports `qmgr` and other commands. For more information, see "qmgr" on page 152 of the PBS Professional Reference Guide.

2.3.2 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 "qstat" on page 200 of the PBS Professional Reference Guide details of which server attributes are returned to the client.

2.3.3 Starting the PBS Server

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. For how to start the server, see "Server: Starting, Stopping, Restarting" on page 145 in the PBS Professional Installation & Upgrade Guide.

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 "pbs_server" on page 107 of the PBS Professional Reference Guide.

2.3.4 Stopping the PBS Server

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 in "qterm" on page 236 of the PBS Professional Reference Guide. 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.

2.4 Queue Management

The following client requests operate on the queues managed by the server.

2.4.1 Queue Status Request

The status of a queue at the server may be requested with a *Queue Status* request. The batch server requires that the following conditions are true:

- The user of the client is authorized to query the status of the designated queue
- The specified queue exists 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"qstat" on page 200 of the PBS Professional Reference Guide for details of which queue attributes are returned to the client.

2.5 Vnode Management

2.5.1 Modify Vnode Request

The *ModifyVnode* request tells the server to modify a vnode. Modifications include state changes.

2.6 Job Management

The following client requests operate on 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.

2.6.1 Queue Job Request

A *Queue Job* request consists 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. After the successful request, the job resides in a queue managed by the server. When a queue is not specified in the request, the job is placed in the default queue. The administrator can specify the default queue. We call the queue where the job ends up the *target queue*. The batch server requires that the following conditions are true:

- The client is authorized to create a job in the target queue
- The target queue exists at the server.
- The target queue is enabled.
- If the target queue is an execution queue, no resource requirement of the job exceeds the limits set for the queue
- If the target queue is an execution queue, all resources requested by the job are recognized
- The job requires access to a user identifier that the client is authorized to access

When a job is placed in an execution queue, it is put 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 *no hold*, in which case the job is placed in the waiting state
- The job has a Hold_Types attribute with a value other than *no hold*, in which case 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 2.8.4, "Job Routing", on page 16.

A server that accepts a *Queue Job* request for a new job will do the following:

- 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
- 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
- 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.

2.6.2 Job Credential Request

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

2.6.3 Job Script Request

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

2.6.4 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.

2.6.5 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 accept a *Message Job* request if all of the following conditions are true:

- The specified job is in the running state
- The user of the client is authorized to post a message to the designated job
- The specified job is owned by the server

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

2.6.6 Locate Job Request

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

- The server owns (manages) the job
- The server created the job
- The server is maintaining a record of the current location of the job

2.6.7 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 accept a *Delete Job* request if all of the following conditions are true:

- The user of the client is authorized to delete the designated job.
- The designated job is owned by the server.
- The designated job is 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 responsible for the job. 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 in seconds between the SIGTERM and SIGKILL signals, as well as to force the deletion of the job even if the node it is running on is not responding.

2.6.8 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 accept a *Modify Job* request if all of the following conditions are true:

- The user of the client is authorized to make the requested modification to the job.
- The designated job is owned by the server.
- The requested modification is consistent with the state of the job.
- A requested resource change would not exceed the limits of the queue or server.
- An recognized 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.

2.6.9 Run Job Request

The *Run Job* request directs the server to place the specified job into immediate execution. The request is issued by a `qrun` command or by the PBS job scheduler.

2.6.10 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 accept the *Rerun Job* request if all of the following conditions are true:

- The user of the client is authorized to rerun the designated job
- The Rerunable attribute of the job is set to *True*
- The job is in the running state
- The server owns 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:

1. Send a SIGKILL signal to the session (process) group of the job
2. Send an OBIT notice to the server with resource usage information
3. 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.

2.6.11 Hold Job Request

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

- The user of the client is authorized to add any of the specified holds
- The batch server manages the specified job

When the server accepts the *Hold Job* request, it will add each specified hold 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:

If checkpoint / restart is supported by the host system, placing a hold on a running job will cause:

a. The job is checkpointed

b. The resources assigned to the job will be released

c. 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 type(s). This will have no effect unless the job is rerun or restarted.

2.6.12 Release Job Request

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

- The user of the client is authorized to add (remove) any of the specified holds.
- The batch server manages the specified job.

When the server accepts the *Release Job* request, it will remove each specified type of hold 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 all holds have been removed, but the Execution_Time attribute specifies a time in the future, the job is placed in the *waiting* state.

2.6.13 Move Job Request

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

- The user of the client is authorized to remove the designated job from the queue in which the job resides
- The user of the client is authorized to submit a job to the new destination
- The designated job is owned by the server
- The designated job is in the *queued*, *held*, or *waiting* state
- The new destination is enabled
- The new destination is accessible. 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 ensure that a job is neither lost nor duplicated.

2.6.14 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 attribute and resource relationships listed in the request restrict the selection of jobs. Only jobs which have attributes and resources that meet the specified criteria 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.

2.6.15 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 accept a *Signal Job* request if all of the following conditions are true:

- The user of the client is authorized to signal the job
- The job is in the *running* state, except for the special signal "resume" when the job must be in the *Suspended* state
- The server owns the designated job
- The requested signal is 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.

2.6.16 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 accept a *Status Job* request if all of the following conditions are true:

- The user of the client is authorized to query the status of the designated job
- The designated job is owned by the server

When the server accepts the request, it will return a *Job Status* message to the client. See the `qstat` command 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.

2.7 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.

2.7.1 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.

2.7.2 Job Dependency

PBS supports job dependencies. 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 via the `qsub` command with the `-W depend=<dependency list>` option.

See "qsub" on page 216 of the PBS Professional Reference Guide for the complete specification of the dependency list, and "Using Job Dependencies", on page 109 of the PBS Professional User's Guide for how to use them.

When a server queues a job with a dependency type of *after*, *afterok*, *afternotok*, or *afterany* in an execution queue, the server will send a *Register Dependent Job* request to the server managing the job specified by the associated job identifier. The request will specify that the server is to register the dependency. This actually creates a corresponding *before* type dependency attribute entry on the parent (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, starts, or terminates, 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 corre-

sponding *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 so 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.

2.8 Deferred Services

The PBS server uses an internal mechanism of deferred services to handle some work asynchronously.

Servers use deferred services for these job-related tasks:

- File staging
- Job selection
- Job initiation
- Job routing
- Job exit
- Job abort
- Rerunning jobs after a server restart

The following rules apply to deferred services used for jobs:

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

2.8.1 Job Scheduling

PBS has a default scheduler; if you want to schedule individual partitions separately, you can add any number of additional schedulers, called *multischeds*. Each PBS scheduler follows its own scheduling policy.

Each scheduler daemon implements a policy that you define that controls when each job is run and on which resources. See "About Schedulers" on page 91 in the PBS Professional Administrator's Guide.

2.8.1.1 Connection Between Scheduler and Server

Each scheduler is persistently connected to the server via `pbs_connect()`. If the scheduler does not have a connection to the server, it continues trying every 2 seconds until it succeeds.

Each scheduler registers itself as a client with the server by sending a PBS_BATCH_RegisterSched batch request. The scheduler passes its name to the server along with this batch request. The administrator must set the scheduler hostname in the sched_host scheduler attribute when creating the scheduler object so that the server can verify that the incoming client request is from the specified scheduler host. The server authenticates the scheduler, and marks this authenticated client as a scheduler.

The scheduler uses `pbs_connect()` to make two connections to the server. After each call to `pbs_connect()`, the scheduler sends a PBS_BATCH_RegisterSched batch request along with its name, and waits for an ACK from the server. The scheduler first establishes a primary connection to the server, and uses it for data and IFL calls. The scheduler then establishes a secondary connection to the server, and uses it to get scheduling commands from the server and to send end-of-cycle notifications to the server. After the scheduler receives an ACK for both connections, it waits for scheduling commands from the server. The server waits for each end-of-cycle notification before sending the next scheduling command.

2.8.1.1.i Process for Server to Accept Scheduler Connection Request

1. When the server receives a request to register a scheduler, the server verifies that the connection is authenticated and that the connection is coming from a daemon.
2. Next, the server validates the scheduler name via find_sched(); if no scheduler is found with the specified name, the request is rejected.
3. If the scheduler is already connected, the server rejects the request.
4. If a scheduler with the specified name is found, temporarily store the current connection file descriptor in the scheduler object; later this will become the primary connection and be used to accept the current request.
5. When the second register scheduler batch request is received for same scheduler object, the server again authenticates the connection and validates the scheduler name and user.
6. The server checks for a temporary stored connection; if there is no temporary stored connection found, the server rejects the request.
7. The server validates that the second connection is not the same as the first connection; if both are the same connection, the server rejects the request.
8. The server validates that both connections come from the same host; if connections are not from the same host, the server rejects the request.
9. The server checks that the connection host is identical to the value of sched_host set by the administrator on the scheduler object; if they are not identical, the server rejects the request.
10. If all tests are passed, the server accepts the request, promotes the client as a scheduler, and assigns the first connection as primary connection and the second connection as secondary connection to the scheduler object.

2.8.1.2 Scheduling Cycle

If a scheduler's scheduling attribute is *True*, the server sends scheduling commands to that scheduler.

A scheduler runs in a loop. Inside each loop, it starts up, performs all of its work, and then stops. The scheduling cycle is triggered by a timer and by several possible events.

When there are no events to trigger the scheduling cycle, it is started by a timer. The time between starts is set in each scheduler's scheduler_iteration server attribute. The default value is 10 minutes.

The maximum duration of the cycle is set in each scheduler's sched_cycle_length attribute. A scheduler will terminate its cycle if the duration of the cycle exceeds the value of the attribute. The default value for the length of the scheduling cycle is 20 minutes. A scheduler does not include the time it takes to query dynamic resources in its cycle measurement.

2.8.1.3 Triggers for Scheduling Cycle

A scheduler starts when the following happen:

- The specified amount of time has passed since the previous start
- A job is submitted
- A job finishes execution.
- A new reservation is created
- A reservation starts
- Scheduling is enabled
- The server comes up
- A job is qrun
- A queue is started
- A job is moved to a local queue
- Eligible wait time for jobs is enabled
- A reservation is re-confirmed after being degraded
- A hook restarts the scheduling cycle

2.8.2 File Staging

PBS provides staging in before execution and staging out after execution. These services are requested via the `-W` option, which sets the stagein and stageout job attributes. The attributes specify the files to be staged:

-W stagein=<execution path>@<input file storage host>:<input file storage path>[,...]

-W stageout=<execution path>@<output file storage host>:<output file storage path>[,...]

The name *execution path* is the name of the file in the job's staging and execution directory (on the execution host). The *execution path* can be relative to the job's staging and execution directory, or it can be an absolute path.

The '@' character separates the execution specification from the storage specification.

The name *storage path* is the file name on the host specified by *storage host*. For stagein, this is the location where the input files come from. For stageout, this is where the output files end up when the job is done. The user must specify a hostname. The name can be absolute, or it can be relative to your home directory on the machine named *storage host*.

For stagein, the direction of travel is **from** *storage path* **to** *execution path*.

For stageout, the direction of travel is **from** *execution path* **to** *storage path*.

A request to stage in a file tells the server to direct MoM to copy a file from the storage location to the execution location. The user must have authority to access the file under the same username under which the job will be run. The storage 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 in the wait state and mail is sent to the job owner.

A request to stage out a file tells the server to direct MoM to move a file from the execution location to the storage location. This service is performed after the job has completed execution and regardless of job 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.

For file copy mechanism information, see "Setting File Transfer Mechanism" on page 441 in the PBS Professional Administrator's Guide.

2.8.3 Job Start

The server receives *Run Job* requests from a PBS scheduler and the `qrun` command. If a request is authenticated, the server forwards the *Run Job* request to the primary MoM for the job; the primary MoM is chosen by the scheduler or specified in the *Run Job* request.

See the sequence of events in "Sequence of Events for Start of Job" on page 477 in the PBS Professional Administrator's Guide.

The primary MoM creates a session leader that runs the shell program specified in the job's Shell_Path_List attribute.

The pathname of the script and any script arguments are passed as parameters to the shell. If the pathname of the shell is relative, the MoM searches its execution path, $PATH, for the shell. If the pathname of the shell is omitted or is the null string, the MoM uses the login shell for the job owner.

The MoM determines the job owner using the following rules:

1. Choose the username in the User_List job attribute whose hostname matches the execution host.
2. Choose the username in the User_List job attribute which has no associated hostname.
3. Use the username from the Job_Owner job attribute.

The MoM creates and sets the following environment variables in the environment of the session leader of the job:

- PBS_ENVIRONMENT; value set to the string "PBS_BATCH"
- PBS_QUEUE; value set to the name of the execution queue

PBS provides each job with environment variables where the job runs. PBS takes some from the submission environment, and creates others. Job submitters can create environment variables for their jobs. The environment variables created by PBS begin with "PBS_". The environment variables that PBS takes from the job submission environment begin with "PBS_O_".

For example, here are a few of the environment variables that accompany a job submitted by user1, with typical values:

```
PBS_O_HOME=/u/user1
PBS_O_LOGNAME=user1
PBS_O_PATH=/usr/bin:/usr/local/bin:/bin
PBS_O_SHELL=/bin/tcsh
PBS_O_HOST=host1
PBS_O_WORKDIR=/u/user1
PBS_JOBID=16386.server1
```

For a complete list of PBS environment variables, "PBS Environment Variables" on page 397 of the PBS Professional Reference Guide.

The MoM puts all of the variables found in the job's Variable_List attribute, with their corresponding values, into the environment of the job's session leader.

The MoM places the specified limits on host-level resources.

If the job has been run before and is now being rerun, the MoM will ensure 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 at an interval less than the minimum interval for the queue, then checkpoint will be set for an interval given by the queue attribute checkpoint_min.

The MoM will set up the standard output stream and the standard error stream of the job according to the table labeled "How k, sandbox, o, and e Options to qsub Affect stdout and stderr", on page 43 of the PBS Professional User's Guide.

2.8.4 Job Routing

The PBS server performs all job routing tasks. Job routing is described in "Routing Queues" on page 27 in the PBS Professional Administrator's Guide.

If the routing destination is at another server, the current server uses a *Queue Job* request to move the job to the new destination.

2.8.5 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.
- Manage the output and error streams of the job, according to "How k, sandbox, o, and e Options to qsub Affect stdout and stderr", on page 43 of the PBS Professional User's Guide.
- If the Mail_Points job attribute contains the value e (*EXIT)*, the server will send mail to the users listed in the Mail_Users job attribute.
- Files are staged out
- Frees 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.

2.8.6 Aborting Job

If the server aborts a job and the Mail_Points job attribute contains the value *a (ABORT)*, the server will send mail to the users listed in the Mail_Users job attribute. The mail message will contain the reason the job was aborted.

2.8.7 Timed Events

The server performs certain events at a specified time or after a specified time delay. Examples:

- A job may have its 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.

2.8.8 Event Logging

The PBS server maintains an event logfile, the format and contents of which are documented in "Event Logging" on page 428 in the PBS Professional Administrator's Guide.

2.8.9 Accounting

The PBS server maintains an accounting file, the format and contents of which are documented in "Accounting" on page 529 in the PBS Professional Administrator's Guide.

2.9 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.

2.9.1 Resource Limits

A job submitter can specify limits for resources used by their job, by requesting those amounts. If the job exceeds those limits, it is aborted. The administrator can specify default limits for resource use by jobs. Defaults are specified at the server and at queues. Defaults are applied when limits are not specified by the submitter. The administrator can also use hooks to set resource requests, and thereby limits, in whatever way is useful. See "Allocating Default Resources to Jobs" on page 244 in the PBS Professional Administrator's Guide and the *PBS Professional Plugins (Hooks) Guide*.

If the submitter does not specify a limit for a resource and there is no default, the job can use an unlimited amount of the resource.

2.9.2 Resource Names

For additional information, see "List of Built-in Resources" on page 259 of the PBS Professional Reference Guide where all resource names are documented.

2.10 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.

2.10.1 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.

3 Developer Headers and Libraries

3.1 Location of API Libraries

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

3.2 Location of Header Files

Header files used by your code are found in `$PBS_EXEC/include`.

3.3 Developer Package

We provide a development package as an RPM package. The files in this package are useful only for developing and compiling software that interfaces with PBS. They are not required to run PBS.

The development package is named `pbspro-devel` and contains the following headers and libraries:

```
/opt/pbs/include/pbs_error.h
/opt/pbs/include/pbs_ifl.h
/opt/pbs/include/rm.h
/opt/pbs/include/tm.h
/opt/pbs/include/tm_.h
/opt/pbs/lib/libattr.a
/opt/pbs/lib/liblog.a
/opt/pbs/lib/libnet.a
/opt/pbs/lib/libpbs.a
/opt/pbs/lib/libpbs_sched.a
/opt/pbs/lib/libsite.a
```

These files were previously in the `pbspro-server`, `pbspro-client` and `pbspro-execution` packages.

The `pbspro-devel` package also contains the `README` file, like the other PBS Professional RPM packages:

```
/usr/share/doc/pbspro-devel-19.0.0/README.md
```

You can install the `pbspro-devel` package separately from the other PBS packages. This package does not conflict with other PBS packages.

3.4 Batch Interface Library

The primary external application programming interface to PBS is the Batch Interface Library, or IFL. This library provides all of the batch service requests used for PBS. The IFL provides a user-callable function corresponding to each batch client command in PBS Professional. Each command generates its own batch service request. You request service from a batch server by calling the appropriate library routine and passing it the required arguments.

The user-callable routines are declared in the header file `PBS_ifl.h`.

We describe the Batch Interface Library in section , “Batch Interface Library (IFL)”, on page 21.

3.4.1 Error Codes

Error codes are available in the header file `PBS_error.h`.

3.4.2 Windows Requirement

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.

3.5 Example Compilation Line

A compile command might look like the following:

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

4
Batch Interface Library (IFL)

You can use the commands in this library to build your new batch clients. For example, you can customize your job status display instead of using `qstat`, build new control commands, or use these commands to build jobs that can get their own status or spawn new jobs.

4.1 Interface Library Overview

The primary external application programming interface to PBS is the Batch Interface Library, or IFL. This library provides all of the batch service requests used for PBS. The IFL provides a user-callable function corresponding to each batch client command in PBS Professional. Each command generates its own batch service request. You request service from a batch server by calling the appropriate library routine and passing it the required arguments.

The user-callable routines are declared in the header file `PBS_ifl.h`.

Error codes are available in the header file `PBS_error.h`.

4.1.1 Connection to Server

We provide network connection management routines to be used with our API commands.

You open a connection with a batch server via a call to `pbs_connect()`, which returns a connection handle to the desired server. You can open multiple connections, and you can use each connection for multiple service requests.

When you are finished using a connection to the server, close it via a call to `pbs_disconnect()`.

4.1.2 Authentication

Before it establishes a connection, `pbs_connect()` `fork()`s and `exec()`s a `pbs_iff` process. The `pbs_iff` process provides a credential which validates the user's identity, and prevents a user from spoofing another user's identity. This credential is included in each batch request sent to the server, and consists of the following:

- The user's name from the password file based on running `pbs_iff's` "real uid" value
- The unprivileged, client-side port value associated with the original `pbs_connect()` request message to the server.

The server checks the entries in its connection table for a matching entry which is not yet marked authenticated. The server requires that the matching entry came from a privileged, remote-end, port value.

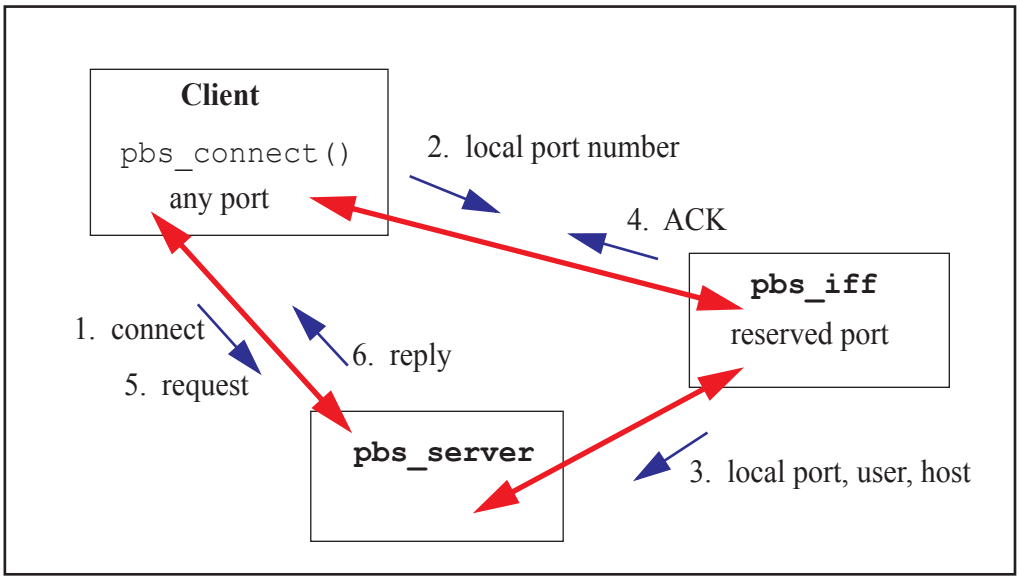

Figure 4-1: Interface Between Client, IFF, and Server

4.1.3 Windows Requirement

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

4.2 Batch Library Routines

4.3 pbs_alterjob . . . 24
4.4 pbs_asyrunjob . . . 26
4.5 pbs_confirmresv . . . 28
4.6 pbs_connect . . . 30
4.7 pbs_default . . . 32
4.8 pbs_deljob . . . 33
4.9 pbs_delresv . . . 35
4.10 pbs_disconnect . . . 36
4.11 pbs_geterrmsg . . . 37
4.12 pbs_holdjob . . . 38
4.13 pbs_locjob . . . 39
4.14 pbs_manager . . . 41
4.15 pbs_modify_resv . . . 45
4.16 pbs_movejob . . . 47
4.17 pbs_msgjob . . . 49
4.18 pbs_orderjob . . . 51

4.19 pbs_preempt_jobs . . . 52
4.20 pbs_relnodesjob . . . 54
4.21 pbs_rerunjob . . . 56
4.22 pbs_rlsjob . . . 57
4.23 pbs_runjob . . . 58
4.24 pbs_selectjob . . . 60
4.25 pbs_selstat . . . 63
4.26 pbs_sigjob . . . 67
4.27 pbs_statfree . . . 69
4.28 pbs_stathost . . . 70
4.29 pbs_statjob . . . 72
4.30 pbs_statnode . . . 75
4.31 pbs_statque . . . 77
4.32 pbs_statresv . . . 79
4.33 pbs_statrsc . . . 81
4.34 pbs_statsched . . . 83
4.35 pbs_statserver . . . 85
4.36 pbs_statvnode . . . 87
4.37 pbs_submit . . . 89
4.38 pbs_submit_resv . . . 91
4.39 pbs_terminate . . . 93

4.3 pbs_alterjob

alter a PBS batch job

4.3.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*int pbs_alterjob(int connect, char *jobID, struct attropl *change_list, char *extend)*

4.3.2 Description

Issues a batch request to alter a batch job.

This command generates a *Modify Job* (11) batch request and sends it to the server over the connection specified by connect.

Job state may affect which attributes can be altered. See "qalter" on page 130 of the PBS Professional Reference Guide.

4.3.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection over which to send batch request to server.

jobID

ID of job or job array to be altered. Format for a job:

<sequence number>.<server name>

Format for an array job:

<sequence number>[].<server name>

change_list

Pointer to a list of attributes to change. Each attribute is described in an attropl structure, defined in `pbs_ifl.h` as:

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

extend

Character string for extensions to command. Not currently used.

4.3.3.1 Members of attropl Structure

next

Points to next attribute in list. A null pointer terminates the list.

name

Points to a string containing the name of the attribute.

resource

Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a pointer to a null string.

If the resource is already present in the job's Resource_List attribute, the value is altered as specified. Otherwise the resource is added.

value

Points to a string containing the value of the attribute or resource.

op

Defines the operation to perform on the attribute or resource. For this command, operators are *SET*, *UNSET*, *INCR*, *DECR.*

4.3.4 Return Value

The routine returns *0* (*zero*) on success.

If an error occurred, the routine returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

4.3.5 See Also

qalter, qhold, qrls, qsub, pbs_connect, pbs_holdjob, pbs_rlsjob

4.4 pbs_asyrunjob

run an asynchronous PBS batch job

4.4.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*int pbs_asyrunjob(int connect, char *jobID, char *location, char *extend)*

4.4.2 Description

Issues a batch request to run a batch job.

Generates an *Asynchronous Run Job* (23) request and sends it to the server over the connection specified by connect.

The server validates the request and replies before initiating the execution of the job.

You can use this version of the call to reduce latency in scheduling, especially when the scheduler must start a large number of jobs.

4.4.3 Required Privilege

You must have Manager or Operator privilege to use this command.

4.4.4 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

jobID

ID of job to be run.

Format for a job:

<sequence number>.<server name>

Format for a job array:

<sequence number>[].<server name>

location

Location where job should run, and optionally resources to use. Same as `qrun -H`:

-H <vnode specification without resources>

The *vnode specification without resources* has this format:

(<vchunk>)[+(<vchunk>) ...]

where *vchunk* has the format

<vnode name>[+<vnode name> ...]

Example:

```
-H (VnodeA+VnodeB)+(VnodeC)
```

PBS applies one requested chunk from the job's selection directive in round-robin fashion to each *vchunk* in the list. Each *vchunk* must be sufficient to run the job's corresponding chunk, otherwise the job may not execute correctly.

-H <vnode specification with resources>

The *vnode specification with resources* has this format:

(<vchunk>)[+(<vchunk>) ...]

where *vchunk* has the format

<vnode name>:<vnode resources>[+<vnode name>:<vnode resources> ...]

and where *vnode resources* has the format

<resource name>=<value>[:<resource name>=<value> ...]

Example:

```
-H (VnodeA:mem=100kb:ncpus=1) +(VnodeB:mem=100kb:ncpus=2+VnodeC:mem=100kb)
```

PBS creates a new selection directive from the *vnode specification with resources*, using it instead of the original specification from the user. Any single resource specification results in the job's original selection directive being ignored. Each *vchunk* must be sufficient to run the job's corresponding chunk, otherwise the job may not execute correctly.

If the job being run requests `-l place=exclhost`, take extra care to satisfy the `exclhost` request. Make sure that if any vnodes are from a multi-vnoded host, all vnodes from that host are allocated. Otherwise those vnodes can be allocated to other jobs.

extend

Character string for extensions to command. Not currently used.

4.4.5 Return Value

The routine returns *0* (*zero*) on success.

If an error occurred, the routine returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

4.4.6 See Also

qrun, pbs_connect, pbs_runjob

4.5 pbs_confirmresv

confirm a PBS reservation

4.5.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*int pbs_confirmresv(int connect, char *reservationID, char *location, unsigned long start_time, char *extend)*

4.5.2 Description

Issues a batch request to confirm a PBS advance, standing, or maintenance reservation.

This function generates a *Confirm Reservation* (75) batch request and sends it to the server over the connection specified by connect.

4.5.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection over which to send batch request to server.

reservationID

Reservation to be confirmed.

Format for advance reservation:

R<sequence number>.<server name>

Format for standing reservation:

S<sequence number>.<server name>

Format for maintenance reservation:

M<sequence number>.<server name>

location

String describing vnodes and resources to be used for reservation. Format:

(<vchunk>)[+(<vchunk>) ...]

where *vchunk* has the format

<vnode name>:<vnode resources>[+<vnode name>:<vnode resources> ...]

and where *vnode resources* has the format

<resource name>=<value>[:<resource name>=<value> ...]

Example:

```
-H (VnodeA:mem=100kb:ncpus=1) +(VnodeB:mem=100kb:ncpus=2+VnodeC:mem=100kb)
```

start_time

Unsigned long containing start time in seconds since epoch. Used only for ASAP reservations (reservations created by using `pbs_rsub -W qmove=<jobID>` on an existing job).

extend

Character string for specifying confirmation/non-confirmation action:

- To confirm a normal reservation, pass in *PBS_RESV_CONFIRM_SUCCESS*.
- To have an unconfirmed reservation deleted, pass in *PBS_RESV_CONFIRM_FAIL*.
- To have the scheduler set the time when it will try to reconfirm a degraded reservation, pass in *PBS_RESV_CONFIRM_FAIL*.

4.5.4 Return Value

The routine returns *0* (*zero*) on success.

If an error occurred, the routine returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

4.5.5 See Also

pbs_rsub, pbs_connect

4.6 pbs_connect

return a connection handle from a PBS batch server

4.6.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*int pbs_connect(char *server)*

4.6.2 Description

This function establishes a virtual stream (TCP/IP) connection with the specified batch server.

Returns a connection handle.

`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.

4.6.3 Arguments

server

Specifies name of server to connect to. Format:

<hostname>[:<port>]

If you do not specify a port, PBS uses the default.

If *server* is a null pointer or a null string, this function opens a connection to the default server. The default server is specified in the PBS_DEFAULT environment variable or the PBS_SERVER parameter in `/etc/pbs.conf`.

4.6.4 Usage

Use this function to establish a connection handle to the desired server before calling any of the other `pbs_*` API functions. They will send their batch requests over the connection established by this function. You can send multiple requests over one connection.

4.6.5 Cleanup

After you are done using the connection handle, close the connection via a call to `pbs_disconnect()`.

4.6.6 Side Effects

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

4.6.7 Windows Requirement

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.

4.6.8 Return Value

On success, the routine returns a connection handle which is a non-negative integer.

If an error occurred, the routine returns -1, and the error number is available in the global integer pbs_errno.

4.6.9 See Also

qsub, pbs_alterjob, pbs_deljob, pbs_disconnect, pbs_geterrmsg, pbs_holdjob, pbs_locjob, pbs_manager, pbs_modify_resv, pbs_movejob, pbs_msgjob, pbs_rerunjob, pbs_rlsjob, pbs_runjob, pbs_selectjob, pbs_selstat, pbs_sigjob, pbs_statjob, pbs_statque, pbs_statresv, pbs_statsched, pbs_statserver, pbs_submit, pbs_submit_resv, pbs_terminate, pbs_server

4.7 pbs_default

return the name of the default PBS server

4.7.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*char *pbs_default()*

4.7.2 Description

Returns a pointer to a character string containing the name of the default PBS server.

The default server is specified in the PBS_DEFAULT environment variable or the PBS_SERVER parameter in `/etc/pbs.conf`.

4.7.3 Return Value

On success, returns a pointer to a character string containing the name of the default PBS server. You do not need to free the character string.

Returns null if it cannot determine the name of the default server.

4.8 pbs_deljob

delete a PBS batch job

4.8.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

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

4.8.2 Description

Issues a batch request to delete a batch job.

This function generates a *Delete Job* (6) batch request and sends it to the server over the connection specified by connect.

If the batch job is running, the MoM sends the SIGTERM signal followed by SIGKILL.

If the batch job is deleted by a user other than the job owner, PBS sends mail to the job owner.

4.8.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection over which to send batch request to server.

jobID

ID of job, job array, subjob, or range of subjobs to be deleted.

Format for a job:

<sequence number>.<server name>

Format for an array job:

<sequence number>[].<server name>

Format for a subjob:

<sequence number>[<index>][.<server name>]

Format for a range of subjobs:

<sequence number>[<first>-<last>][.<server name>]

extend

Character string for extensions to command. If the string is not null, it is appended to the message mailed to the job owner.

4.8.4 Return Value

The routine returns *0* (*zero*) on success.

On error, the routine returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

4.8.5 See Also

qdel, pbs_connect

4.9 pbs_delresv

delete a reservation

4.9.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*int pbs_delresv(int connect, char *reservationID, char *extend)*

4.9.2 Description

Issues a batch request to delete a reservation.

This function generates a *Delete Reservation* (72) batch request and sends it to the server over the connection specified by connect.

If the reservation is in state *RESV_RUNNING*, and there are jobs in the reservation queue, those jobs are deleted before the reservation is deleted.

4.9.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection over which to send batch request to server.

reservationID

Reservation to be deleted.

Format for advance reservation:

R<sequence number>.<server name>

Format for standing reservation:

S<sequence number>.<server name>

Format for maintenance reservation:

M<sequence number>.<server name>

extend

Character string for extensions to command. Not currently used.

4.9.4 Return Value

The routine returns *0* (*zero*) on success.

On error, the routine returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

4.9.5 See Also

pbs_rdel, pbs_connect

4.10 pbs_disconnect

disconnect from a PBS batch server

4.10.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

int pbs_disconnect(int connect)

4.10.2 Description

Closes the virtual stream connection to a PBS batch server. Connection was previously returned from a call to `pbs_connect()`.

4.10.3 Arguments

connect

Connection handle to be closed. Return value of `pbs_connect()`. Specifies connection used earlier to send batch requests to server.

4.10.4 Return Value

The routine returns *0* (*zero*) after successfully closing the connection.

If an error occurred, the routine returns -1, and the error number is available in the global integer pbs_errno.

4.10.5 See Also

pbs_connect

4.11 pbs_geterrmsg

get error message for most recent PBS batch operation

4.11.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*char *pbs_geterrmsg(int connect)*

4.11.2 Description

Returns most recent error message text associated with a batch server request.

If a preceding batch interface library call over the connection specified by connect returned an error from the server, the server may have created an associated text message. If there is a text message, this function returns a pointer to the text message.

4.11.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to request error message from server.

4.11.4 Return Value

If the server returned an error and created an error text string in reply to a previous batch request, this function returns a pointer to the text string. The text string is null-terminated.

If the server does not have an error text string, this function returns a null pointer.

The text string is a global variable; you do not need to free it.

4.11.5 See Also

pbs_connect

4.12 pbs_holdjob

place a hold on a PBS batch job

4.12.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

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

4.12.2 Description

Issues a batch request to place a hold on a job or job array.

This function generates a *Hold Job* (7) batch request sends it to the server over the connection specified by connect.

4.12.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection over which to send batch request to server.

jobID

ID of job which is to be held.

Format for a job:

<sequence number>.<server name>

Format for a job array:

<sequence number>[].<server name>

hold_type

Type of hold to apply to job or job array. Valid values are defined in `pbs_ifl.h`. If *hold_type* is a null pointer or points to a null string, PBS applies a *User* hold to the job or job array.

extend

Character string for extensions to command. Not currently used.

4.12.4 Return Value

The routine returns *0* (*zero*) on success.

If an error occurred, the routine returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

4.12.5 See Also

qhold, pbs_connect, pbs_alterjob, pbs_rlsjob

4.13 pbs_locjob

return current location of a PBS batch job

4.13.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*char *pbs_locjob(int connect, char *jobID, char *extend)*

4.13.2 Description

Issues a batch request to locate a batch job or job array.

This function generates a *Locate Job* (8) batch request and sends it to the server over the connection specified by connect.

If the server currently manages the batch job, or knows which server does currently manage the job, the server returns the location of the job.

4.13.3 Arguments

connect
: Return value of `pbs_connect()`. Specifies connection over which to send batch request to server.

jobID
: ID of job to be located.

Format for a job:

<sequence number>.<server name>

Format for a job array:

<sequence number>[].<server name>

extend
: Character string for extensions to command. Not currently used.

4.13.4 Cleanup

The character string returned by `pbs_locjob()` is allocated by `pbs_locjob()`. You must free it via a call to `free()`.

4.13.5 Return Value

On success, returns a pointer to a character string containing current location. Format:

<server name>

On failure, returns a null pointer, and the error number is available in the global integer pbs_errno.

4.13.6 See Also

pbs_connect

4.14 pbs_manager

modify a PBS batch object

4.14.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*int pbs_manager(int connect, int command, int object_type, char *object_name, struct attropl *attrib_list, char *extend)*

4.14.2 Description

Issues a batch request to perform administrative functions at a server.

Generates a *Manager* (9) batch request and sends it to the server over the connection specified by connect.

You can use this to create, delete, and set attributes of objects such as queues.

4.14.3 Required Privilege

This function requires Manager or Operator privilege depending on the operation, and root privilege when 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.
- Hook commands are run at the server host.

4.14.4 Arguments

connect
: Return value of `pbs_connect()`. Specifies connection over which to send batch request to server.

command
: Operation to be performed. Valid values are specified in `pbs_ifl.h`.

object_type
: Specifies type of object on which command is to operate. Valid values are specified in `pbs_ifl.h`.

object_name
: Name of object on which to operate.

attrib_list

Pointer to a list of attributes to be operated on. Each attribute is described in an attropl structure, defined in pbs_ifl.h as:

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

extend

Character string for extensions to command. Not currently used.

4.14.4.1 Members of attropl Structure

next

Points to next attribute in list. A null pointer terminates the list.

name

Points to a string containing the name of the attribute.

resource

Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

If the resource is already present in the object's attribute, the value is altered as specified. Otherwise the resource is added.

value

Points to a string containing the new value of the attribute or resource. For parameterized limit attributes, this string contains all parameters for the attribute.

op

Defines the manner in which the new value is assigned to the attribute or resource. The operators used for this function are *SET*, *UNSET*, *INCR*, *DECR*.

4.14.5 Usage for Hooks

When importing a hook or hook configuration file:

- Set *command* to *MGR_CMD_IMPORT*
- Set *object_type* to *SITE_HOOK* (or *PBS_HOOK* if you are importing a configuration file for a built-in hook; you cannot import a built-in hook)
- Set *object_name* to the name of the hook
- In one attropl structure:
  - Set *name* to "*content-type*"
  - Set *value* to "*application/x-python*" for a hook, or "*application/x-config*" for a configuration file
- In another attropl structure:
  - Set *name* to "*content-encoding*"
  - Set *value* to "*default*" or "*base64*"
- In a third attropl structure:
  - Set *name* to "*input-file*"
  - Set *value* to the name of the input file
- Set *op* to *SET*

When exporting a hook or hook configuration file:

- Set *command* to *MGR_CMD_EXPORT*
- Set *object_type* to *SITE_HOOK* (or *PBS_HOOK* if you are exporting a configuration file for a built-in hook; you cannot export a built-in hook)
- Set *object_name* to the name of the hook
- In one attropl structure:
  - Set *name* to "*content-type*"
  - Set *value* to "*application/x-python*" for a hook, or "*application/x-config*" for a configuration file
- In another attropl structure:
  - Set *name* to "*content-encoding*"
  - Set *value* to "*default*" or "*base64*"
- In a third attropl structure:
  - Set *name* to "*output-file*"
  - Set *value* to the name of the output file
- Set *op* to *SET*

See the *PBS Professional Hooks Guide*.

4.14.6 Return Value

The routine returns *0* (*zero*) on success.

If an error occurred, the routine returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

4.14.7 See Also

qmgr, pbs_connect

4.15 pbs_modify_resv

modify a PBS reservation

4.15.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*char *pbs_modify_resv(int connect, char *reservationID, struct attropl *attrib_list, char *extend)*

4.15.2 Description

Issues a batch request to modify a reservation.

Generates a *Modify Reservation* (91) batch request and sends it to the server over the connection specified by connect.

4.15.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection over which to send batch request to server.

reservationID

Reservation to be modified.

Format for advance reservation:

R<sequence number>.<server name>

Format for standing reservation:

S<sequence number>.<server name>

attrib_list

Pointer to a list of attributes to modify. Each attribute is described in an attropl structure, defined in `pbs_ifl.h` as:

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

For any attribute that is not specified or that is a null pointer, PBS takes the default action for that attribute. The default action is to assign the default value or to not pass the attribute with the reservation; the action depends on the attribute.

extend

Character string for extensions to command. Not currently used.

4.15.3.1 Members of attropl Structure

next

Points to next attribute in list. A null pointer terminates the list.

name

Points to a string containing the name of the attribute.

resource

Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

If the resource is already present in the reservation's Resource_List attribute, the value is altered as specified. Otherwise the resource is added.

value

Points to a string containing the value of the attribute or resource.

op

Operator. The only allowed operator for this function is *SET*.

4.15.4 Return Value

On success, returns a character string containing the reservation ID assigned by the server.

On failure, returns a null pointer, and the error number is available in the global integer pbs_errno.

4.15.5 Cleanup

The space for the reservation ID string is allocated by `pbs_modify_resv()`.

Release the reservation ID via a call to `free()` when no longer needed.

4.15.6 See Also

pbs_rsub, pbs_connect

4.16 pbs_movejob

move a PBS batch job to a new destination

4.16.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*int pbs_movejob(int connect, char *jobID, char *destination, char *extend)*

4.16.2 Description

Issues a batch request to move a job or job array to a new destination.

Generates a *Move Job* (12) batch request and sends it to the server over the connection specified by connect.

Moves specified job or job array from its current queue and server to the specified queue and server.

You cannot move a job in the Running, Transiting, or Exiting states.

4.16.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection over which to send batch request to server.

jobID

ID of job to be moved.

Format for a job:

<sequence number>.<server name>

Format for a job array:

<sequence number>[].<server name>

destination

New location for job or job array. Formats:

<queue name>@<server name>

Specified queue at specified server

<queue name>

Specified queue at default server

@<server name>

Default queue at specified server

@default

Default queue at default server

(null pointer or null string)

Default queue at default server

extend

Character string for extensions to command. Not currently used.

4.16.4 Return Value

The routine returns *0* (*zero*) on success.

If an error occurred, the routine returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

4.16.5 See Also

qmove, pbs_connect

4.17 pbs_msgjob

record a message for a running PBS batch job

4.17.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*int pbs_msgjob(int connect, char *jobID, int file, char *message, char *extend)*

4.17.2 Description

Issues a batch request to write a message in one or more output files of a batch job.

Generates a *Message Job* (10) batch request and sends it to the server over the connection specified by connect.

You can write a message into a job's `stdout` and/or `stderr` files. Can be used on jobs or subjobs, but not job arrays or ranges of subjobs.

4.17.3 Arguments

connect
: Return value of `pbs_connect()`. Specifies connection over which to send batch request to server.

jobID
: ID of job into whose output file(s) to write.

Format for a job:

<sequence number>.<server name>

Format for a subjob:

<sequence number>[<index>].<server name>

file
: Indicates whether to write to `stdout`, `stderr`, or both:

1
: Writes to `stdout`

2
: Writes to `stderr`

3
: Writes to `stdout` and `stderr`

message
: Character string to be written to output file(s).

extend
: Character string for extensions to command. Not currently used.

4.17.4 Return Value

The routine returns *0* (*zero*) on success.

If an error occurred, the routine returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

4.17.5 See Also

qmsg, pbs_connect

4.18 pbs_orderjob

swap positions of two PBS batch jobs

4.18.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*int pbs_orderjob(int connect, char *jobID1, char *jobID2, char *extend)*

4.18.2 Description

Issues a batch request to swap the positions of two jobs.

Generates an *Order Job* (50) batch request and sends it to the server over the connection specified by connect.

Can be used on jobs and job arrays. Can be used on jobs in different queues. Both jobs must be at the same server.

You cannot swap positions of jobs that are running.

4.18.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection over which to send batch request to server.

jobID1, jobID2

IDs of jobs to be swapped.

Format for a job:

<sequence number>.<server name>

Format for a job array:

<sequence number>[].<server name>

extend

Character string for extensions to command. Not currently used.

4.18.4 Return Value

The routine returns *0* (*zero*) on success.

If an error occurred, the routine returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

4.18.5 See Also

qmove, qorder, qsub, pbs_connect

4.19 pbs_preempt_jobs

preempt a list of jobs

4.19.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*preempt_job_info *pbs_preempt_jobs(int connect, char **jobID_list)*

4.19.2 Description

Sends the server a list of jobs to be preempted.

Sends a *Preempt Jobs* (93) batch request to the server over the connection specified by connect.

Returns a list of preempted jobs along with the method used to preempt each one.

4.19.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

jobID_list

List of job IDs to be preempted, as a null-terminated array of pointers to strings.

Format for a job ID:

<sequence number>.<server name>

Format for a job array ID:

<sequence number>[].<server name>

For example:

```
const char *joblist[3];
joblist[0]="123.myserver";
joblist[1]="456.myserver";
joblist[2]=NULL;
```

4.19.4 Return Value

Returns a list of preempted jobs. Each job is represented in a preempt_job_info structure, which has the following fields:

job_id

The job ID, in a char*

preempt_method

How the job was preempted, in a char:

S

The job was preempted using suspension.

C

The job was preempted using checkpointing.

R
: The job was preempted by being requeued.

D
: The job was preempted by being deleted.

0 (zero)
: The job could not be preempted.

4.19.5 Cleanup

You must free the list of preempted jobs by passing it directly to `free()`.

4.20 pbs_relnodesjob

release some or all of the non-primary-execution-host vnodes assigned to a PBS job

4.20.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*int pbs_relnodesjob (int connect, char *jobID, char *vnode_list, char *extend)*

4.20.2 Description

Issues a batch request to release some or all of the vnodes of a batch job. Generates a *RelnodesJob* (90) batch request and sends it to the server over the connection specified by connect.

You cannot release vnodes on the primary execution host.

Do not use when MPI processes are running on a host managed by the cgroups hook. Use when MPI processes are not running.

You can use this on jobs and subjobs, but not on job arrays or ranges of subjobs.

4.20.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

jobID

ID of job or subjob whose vnodes are to be released.

Format for a job:

<sequence number>.<server name>

Format for a subjob:

<sequence number>[<index>].<server name>

vnode_list

List of vnode names separated by plus signs ("+").

If *vnode_list* is a null pointer, this specifies that all the vnodes assigned to the job that are not on the primary execution host are to be released.

extend

Character string for extensions to command. Not currently used.

4.20.4 Return Value

On success, returns *0* (*zero*).

On error, returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

4.20.5 See Also

pbs_connect

4.21 pbs_rerunjob

requeue a PBS batch job

4.21.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*int pbs_rerunjob(int connect, char *jobID, char *extend)*

4.21.2 Description

Issues a batch request to requeue a batch job, job array, subjob, or range of subjobs.

Generates a *Rerun Job* (14) batch request and sends it to the server over the connection specified by connect.

You cannot requeue a job that is marked as not rerunnable (the Rerunable attribute is *False*).

4.21.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

jobID

ID of job to be requeued.

Format for a job:

<sequence number>.<server name>

Format for a job array:

<sequence number>[].<server name>

Format for a subjob:

<sequence number>[<index>].<server name>

Format for a range of subjobs:

<sequence number>[<index start>-<index end>].<server name>

extend

Character string for extensions to command. Not currently used.

4.21.4 Return Value

The routine returns *0* (*zero*) on success.

If an error occurred, the routine returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

4.21.5 See Also

qrerun, pbs_connect

4.22 pbs_rlsjob

release a hold on a PBS batch job

4.22.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*int pbs_rlsjob(int connect, char *jobID, char *hold_type, char *extend)*

4.22.2 Description

Issues a batch request to release a hold on a job or job array.

Generates a *Release Job* (13) batch request and sends it to the server over the connection specified by connect.

4.22.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection over which to send batch request to server.

jobID

ID of job which is to have a hold released.

Format for a job:

<sequence number>.<server name>

Format for a job array:

<sequence number>[].<server name>

hold_type

Type of hold to remove from job or job array. Valid values are defined in `pbs_ifl.h`. If *hold_type* is a null pointer or points to a null string, PBS removes a *User* hold from the job or job array.

extend

Character string for extensions to command. Not currently used.

4.22.4 Return Value

The routine returns *0* (*zero*) on success.

If an error occurred, the routine returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

4.22.5 See Also

qhold, qrls, pbs_connect, pbs_holdjob

4.23 pbs_runjob

run a PBS batch job

4.23.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

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

4.23.2 Description

Issues a batch request to run a batch job.

Generates a *Run Job* (15) batch request and sends it to the server over the connection specified by connect.

If no file stagein is required, the server replies when the job has started execution. If file stagein is required, the server replies when staging is started.

4.23.3 Required Privilege

You must have Operator or Administrator privilege to use this command.

4.23.4 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

jobID

ID of job to run.

Format for a job:

<sequence number>.<server name>

Format for a job array:

<sequence number>[].<server name>

location

Location where job should run, and optionally resources to use. Same as `qrun -H`:

-H <vnode specification without resources>

The *vnode specification without resources* has this format:

(<vchunk>)[+(<vchunk>) ...]

where *vchunk* has the format

<vnode name>[+<vnode name> ...]

Example:

```
-H (VnodeA+VnodeB)+(VnodeC)
```

PBS applies one requested chunk from the job's selection directive in round-robin fashion to each *vchunk* in the list. Each *vchunk* must be sufficient to run the job's corresponding chunk, otherwise the job may not execute correctly.

-H <vnode specification with resources>

The *vnode specification with resources* has this format:

(<vchunk>)[+(<vchunk>) ...]

where *vchunk* has the format

<vnode name>:<vnode resources>[+<vnode name>:<vnode resources> ...]

and where *vnode resources* has the format

<resource name>=<value>[:<resource name>=<value> ...]

Example:

```
-H (VnodeA:mem=100kb:ncpus=1) +(VnodeB:mem=100kb:ncpus=2+VnodeC:mem=100kb)
```

PBS creates a new selection directive from the *vnode specification with resources*, using it instead of the original specification from the user. Any single resource specification results in the job's original selection directive being ignored. Each *vchunk* must be sufficient to run the job's corresponding chunk, otherwise the job may not execute correctly.

If the job being run requests `-l place=exclhost`, take extra care to satisfy the `exclhost` request. Make sure that if any vnodes are from a multi-vnoded host, all vnodes from that host are allocated. Otherwise those vnodes can be allocated to other jobs.

extend

Character string for extensions to command. Not currently used.

4.23.5 Return Value

The routine returns *0* (*zero*) on success.

If an error occurred, the routine returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

4.23.6 See Also

qrun, pbs_asyrunjob, pbs_connect

4.24 pbs_selectjob

select PBS batch jobs according to specified criteria

4.24.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*char **pbs_selectjob(int connect, struct attropl *criteria_list, char *extend)*

4.24.2 Description

`pbs_selectjob()` issues a batch request to select jobs that meet specified criteria, and returns an array of job IDs that meet the specified criteria.

This command generates a *Select Jobs* (16) batch request and sends it to the server over the connection handle specified by connect.

By default, `pbs_selectjob()` returns all batch jobs for which the user is authorized to query status. You filter the jobs by specifying values for job attributes and resources. You send a linked list of attributes with associated values and operators. Job attributes are listed in "Job Attributes" on page 327 of the PBS Professional Reference Guide.

Returns a list of jobs that meet all specified criteria.

4.24.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

criteria_list

Pointer to a list of attributes to use as selection criteria. Each attribute is described in an attropl structure, defined in `pbs_ifl.h` as:

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

If criteria_list itself is null, you are not using attributes or resources as selection criteria.

extend

Character string where you can specify limits or extensions of your search.

4.24.3.1 Members of attropl Structure

next

Points to next attribute in list. A null pointer terminates the list.

name

Points to a string containing the name of the attribute.

resource

Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

value

Points to a string containing the value of the attribute or resource.

op

Defines the operator in the logical expression:

<existing value> <operator> <specified limit>

Jobs for which the logical expression evaluates to *True* are selected.

For this command, op can be *EQ, NE, GE, GT, LE, LT.*

4.24.4 Querying States

You can select jobs in more than one state using a single request, by listing all states you want returned. For example, to get jobs in *Held* and *Waiting* states:

- Fill in criteria_list->name with "*job_state*"
- Fill in criteria_list->value with "*HW*" for *Held* and *Waiting*

4.24.5 Extending Your Query

You can use the following characters in the extend parameter:

T, t

Extends query to include subjobs. Job arrays are not included.

x

Extends query to include finished and moved jobs.

4.24.5.1 Querying Finished and Moved Jobs

To get information on finished or moved jobs, as well as current jobs, add an '*x*' character to the extend parameter (set one character to be the '*x*' character). For example:

```
pbs_selectjob ( ..., ..., <extend characters>) ...
```

To get information on finished jobs only:

- Add the '*x*' character to the extend parameter
- Fill in criteria_list->name with "*ATTR_state*"
- Fill in criteria_list->value with "*FM*" for *Finished* and *Moved*

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

4.24.5.2 Querying Job Arrays and Subjobs

To query only job arrays (not jobs or subjobs):

- Fill in criteria_list->name with "*array*"
- Fill in criteria_list->value with "*True*"

To query only job arrays and subjobs (not jobs):

- Fill in criteria_list->name with "*array*"
- Fill in criteria_list->value with "*True*"
- Add the '*T*' or '*t*' character to the extend parameter

To query only jobs and subjobs (not job arrays), add the '*T*' or '*t*' character to the extend parameter.

4.24.6 Return Value

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 ID in the form:

<sequence number>.<server>@<server>

If no jobs met the criteria, the first pointer in the array is null.

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

4.24.7 Cleanup Required

The returned array of character pointers is malloc()'ed by `pbs_selectjob()`. When the array is no longer needed, you must free it via a call to `free()`.

4.24.8 See Also

pbs_alterjob, pbs_connect, qsig

4.25 pbs_selstat

get status of selected PBS batch jobs

4.25.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*struct batch_status *pbs_selstat(int connect, struct attropl *criteria_list, struct attrl *output_attribs, char *extend)*

4.25.2 Description

Issues a batch request to get the status of jobs which meet the specified criteria.

Generates a *Select Status* (51) batch request and sends it to the server over the connection specified by connect.

Returns a list of batch_status structures for jobs that meet the selection criteria.

This function is a combination of `pbs_selectjob()` and `pbs_statjob()`.

By default this gives status for all jobs for which you are authorized to query status. You can filter the results by specifying selection criteria.

4.25.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

criteria_list

Pointer to a list of selection criteria, which are attributes and resources with required values. If this list is null, you are not filtering your results via selection criteria. Each attribute or resource is described in an attropl structure, defined in `pbs_ifl.h` as:

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

If criteria_list itself is null, you are not using attributes or resources as selection criteria.

output_attribs

Pointer to a list of attributes to return. If this list is null, all attributes are returned. Each attribute is described in an attrl structure, defined in `pbs_ifl.h` as:

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

extend

Character string where you can specify limits or extensions of your selection.

4.25.3.1 Members of attropl Structure

next

Points to next attribute in list. A null pointer terminates the list.

name

Points to a string containing the name of the attribute.

resource

Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

value

Points to a string containing the value of the attribute or resource. For parameterized limit attributes, this string contains all parameters for the attribute.

op

Specifies the test to be applied to the attribute or resource. The operators are EQ, NE, GE, GT, LE, LT.

4.25.3.2 Members of attrl Structure

name

Points to a string containing the name of the attribute.

resource

Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

value

Points to a string containing the value of the attribute or resource. Should always be null.

next

Points to next attribute in list. A null pointer terminates the list.

4.25.4 Querying States

You can select jobs in more than one state using a single request, by listing all states you want returned. For example, to get jobs in *Held* and *Waiting* states:

- Fill in criteria_list->name with "*job_state*"
- Fill in criteria_list->value with "*HW*" for *Held* and *Waiting*

4.25.5 Extending Your Query

You can use the following characters in the extend parameter:

T, t

Extends query to include subjobs. Job arrays are not included.

x

Extends query to include finished and moved jobs.

4.25.5.1 Querying Finished and Moved Jobs

To get information on finished or moved jobs, as well as current jobs, add an '*x*' character to the extend parameter (set one character to be the '*x*' character). For example:

```
pbs_selstat ( ..., ..., <extend characters>) ...
```

To get information on finished jobs only:

- Add the '*x* character to the extend parameter
- Fill in criteria_list->name with "*ATTR_state*"
- Fill in criteria_list->value with "*FM*" for *Finished* and *Moved*

For example:

```
criteria_list->name = ATTR_state;
criteria_list->value = "FM";
criteria_list->op = EQ;
pbs_selstat ( ..., criteria_list, ..., extend) ...
```

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

4.25.5.2 Querying Job Arrays and Subjobs

To query only job arrays (not jobs or subjobs):

- Fill in criteria_list->name with "*array*"
- Fill in criteria_list->value with "*True*"

To query only job arrays and subjobs (not jobs):

- Fill in criteria_list->name with "*array*"
- Fill in criteria_list->value with "*True*"
- Add the '*T*' or '*t*' character to the extend parameter

To query only jobs and subjobs (not job arrays), add the '*T*' or '*t*' character to the extend parameter.

4.25.6 Return Value

Returns a pointer to a list of batch_status structures for jobs that meet the selection criteria. If no jobs meet the criteria or can be queried for status, returns the null pointer.

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

4.25.6.1 The 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;
}
```

4.25.7 Cleanup

You must free the list of batch_status structures when no longer needed, by calling `pbs_statfree()`.

4.25.8 See Also

qsig, qstat, pbs_connect, pbs_selectjob, pbs_statfree, pbs_statjob

4.26 pbs_sigjob

send a signal to a PBS batch job

4.26.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*int pbs_sigjob(int connect, char *jobID, char *signal, char *extend)*

4.26.2 Description

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

Generates a *Signal Job* (18) batch request and sends it to the server over the connection specified by connect.

You can send a signal to a job, job array, subjob, or range of subjobs.

The batch server sends the job the specified signal.

The job must be in the running or suspended state.

4.26.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

jobID

ID of job to be signaled.

Format for a job:

<sequence number>.<server name>

Format for a job array:

<sequence number>[].<server name>

Format for a subjob:

<sequence number>[<index>].<server name>

Format for a range of subjobs:

<sequence number>[<index start>-<index end>].<server name>

signal

Name of signal to send to job. Can be alphabetic, with or without SIG prefix. Can be signal number.

The following special signals are all lower-case, and have no associated signal number:

admin-suspend

Suspends a job and puts its vnodes into the *maintenance* state. The job is put into the *S* state and its processes are suspended.

admin-resume

Resumes a job that was suspended using the *admin-suspend* signal, without waiting for scheduler. Cannot be used on jobs that were suspended with the *suspend* signal. When the last admin-suspended job has been admin-resumed, the vnode leaves the maintenance state.

suspend

Suspends specified job(s). Job goes into *suspended (S)* state.

resume
Marks specified job(s) for resumption by scheduler when there are sufficient resources. Cannot be used on jobs that were suspended with the *admin_suspend* signal.

If the signal is not recognized on the execution host, no signal is sent and an error is returned.

extend
Character string for extensions to command. Not currently used.

4.26.4 Return Value

The routine returns *0* (*zero*) on success.

If an error occurred, the routine returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

4.26.5 See Also

qsig, pbs_connect

4.27 pbs_statfree

free a PBS status object

4.27.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*void pbs_statfree(struct batch_status *psj)*

4.27.2 Description

Frees the specified PBS status object returned by PBS API routines such as `pbs_statque()`, `pbs_statserver()`, `pbs_stathook()`, etc.

4.27.3 Arguments

psj

Pointer to the batch_status structure to be freed.

4.27.3.1 The 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;
}
```

4.27.4 Return Value

No return value.

4.28 pbs_stathost

get status of PBS execution host(s)

4.28.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*void pbs_statfree(struct batch_status *psj)*

*struct batch_status *pbs_stathost(int connect, char *target, struct attrl *output_attribs, char *extend)*

4.28.2 Description

Issues a batch request to get the status of PBS execution hosts.

Generates a *Status Node* (58) batch request and sends it to the server over the connection specified by connect.

Returns specified attributes or all attributes of specified execution host or all execution hosts. If an execution host has multiple vnodes, this command reports aggregated information from the vnodes for that host.

4.28.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

target

Name of execution host whose attributes are to be reported. If this argument is a null pointer or points to a null string, returns attributes of all execution hosts known to the server.

output_attribs

Pointer to a list of attributes to return. If this argument is null, returns all attributes. Each attribute is described in an attrl structure, defined in `pbs_ifl.h` as:

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

extend

Character string for extensions to command. Not currently used.

4.28.3.1 Members of attrl Structure

name

Points to a string containing the name of the attribute.

resource

Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

value
: Points to a string containing the value of the attribute or resource.

next
: Points to next attribute in list. A null pointer terminates the list.

4.28.4 Return Value

Returns a pointer to a list of batch_status structures describing the execution host(s).

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

4.28.4.1 The 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;
}
```

4.28.5 Cleanup

You must free the list of batch_status structures when no longer needed, by calling pbs_statfree().

4.28.6 See Also

qstat, pbs_connect, pbs_statfree

4.29 pbs_statjob

get status of PBS batch jobs

4.29.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*void pbs_statfree(struct batch_status *psj)*

*struct batch_status *pbs_statjob(int connect, char *ID, struct attrl *output_attribs, char *extend)*

4.29.2 Description

Issues a batch request to get the status of a specified batch job, a list of batch jobs, or the batch jobs at a queue or server.

Generates a *Status Job* (19) batch request and sends it to the server over the connection specified by connect.

You can query status of jobs, job arrays, subjobs, and ranges of subjobs.

Queries all specified jobs that the user is authorized to query.

4.29.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

ID

Job ID, list of job IDs, queue, server, or null.

If *ID* is a null pointer or points to a null string, gets status of jobs at connected server.

Format for a job:

<sequence number>.<server name>

Format for a job array:

<sequence number>[].<server name>

Format for a subjob:

<sequence number>[<index>].<server name>

Format for a range of subjobs:

<sequence number>[<index start>-<index end>].<server name>

Format for a list of jobs: comma-separated list of job IDs in a single string. White space is ignored. No limit on length:

"<job ID>,<job ID>,<job ID>, ..."

Format for a queue:

<queue name>@<server name>

Format for a server:

<server name>

output_attribs

Pointer to a list of attributes to return. If this list is null, all attributes are returned. Each attribute is described in an attrl structure, defined in pbs_ifl.h as:

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

extend

Character string where you can specify limits or extensions of your search. Order of characters is not important.

4.29.3.1 Members of attrl Structure

name

Points to a string containing the name of the attribute.

resource

Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

value

Points to a string containing the value of the attribute or resource.

next

Points to next attribute in list. A null pointer terminates the list.

4.29.4 Querying Job Arrays and Subjobs

You can query status of job arrays and their subjobs, or just the parent job arrays only.

To query status of job arrays and their subjobs, include the job array IDs in the ID argument, and include the '*t*' character in the extend argument. The function returns the status of each parent job array followed by status of each subjob in that job array.

To query status of one or more parent job arrays only, but not their subjobs, include their job IDs in the ID argument, but do not include anything in the extend argument.

4.29.5 Querying the Jobs at a Queue or Server

To query status of all jobs at a queue, give the queue name in the ID argument.

To query status of all jobs at a server, give the server name in the ID argument. If you give a null ID argument, the function queries the default server.

4.29.6 Extending Your Query

You can use the following characters in the extend parameter:

T, t

Extends query to include subjobs. Job arrays are not included.

x

Extends query to include finished and moved jobs.

4.29.6.1 Querying Finished and Moved Jobs

To get status for finished or moved jobs, as well as current jobs, add an '*x*' character to the extend parameter (set one character to be the '*x*' character). For example:

```
pbs_statjob ( ..., ..., <extend characters>) ...
```

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

4.29.7 Return Values

For a single job, if the job can be queried, returns a pointer to a batch_status structure containing the status of the specified job. If the job cannot be queried, returns a NULL pointer, and pbs_errno is set to an error number indicating the reason the job could not be queried.

For a list of jobs, if any of the specified jobs can be queried, returns a pointer to a batch_status structure containing the status of all the queryable jobs. If none of the jobs can be queried, returns a NULL pointer, 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, returns a pointer to a batch_status structure containing the status of all the queryable jobs in the queue. If the queue does not exist, returns a NULL pointer, and pbs_errno is set to *PBSE_UNKQUE* (*15018*). If the queue exists but contains no queryable jobs, returns a NULL pointer, 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, returns 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, returns a NULL pointer, and pbs_errno is set to *PBSE_NONE* (*0*).

4.29.7.1 The 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;
}
```

4.29.8 Cleanup

You must free the list of batch_status structures when no longer needed, by calling `pbs_statfree()`.

4.29.9 See Also

qstat, pbs_connect

4.30 pbs_statnode

get status of PBS execution host(s)

4.30.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*void pbs_statfree(struct batch_status *psj)*

*struct batch_status *pbs_statnode(int connect, char *target, struct attrl *output_attribs, char *extend)*

4.30.2 Description

Issues a batch request to get the status of PBS execution hosts.

Generates a *Status Node* (58) batch request and sends it to the server over the connection specified by connect.

Returns specified attributes or all attributes of specified execution host or all execution hosts. If an execution host has multiple vnodes, this command reports aggregated information from the vnodes for that host.

Identical to `pbs_stathost()`; retained for backward compatibility.

4.30.3 Arguments

connect
: Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

target
: Name of execution host whose attributes are to be reported. If this argument is null, returns attributes of all execution hosts known to the server.

output_attribs
: Pointer to a list of attributes to return. If this argument is a null pointer or points to a null string, returns all attributes. Each attribute is described in an attrl structure, defined in `pbs_ifl.h` as:

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

extend
: Character string for extensions to command. Not currently used.

4.30.3.1 Members of attrl Structure

name
: Points to a string containing the name of the attribute.

resource
: Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

value
: Points to a string containing the value of the attribute or resource.

next
: Points to next attribute in list. A null pointer terminates the list.

4.30.4 Return Value

Returns a pointer to a list of batch_status structures describing the host(s).

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

4.30.4.1 The 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;
}
```

4.30.5 Cleanup

You must free the list of batch_status structures when no longer needed, by calling `pbs_statfree()`.

4.30.6 See Also

qstat, pbs_connect

4.31 pbs_statque

get status of PBS queue(s)

4.31.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*struct batch_status *pbs_statque(int connect, char *target, struct attrl *output_attribs, char *extend)*

4.31.2 Description

Issues a batch request to get the status of PBS queues.

Generates a *Status Queue* (20) batch request and sends it to the server over the connection specified by connect.

Returns specified attributes or all attributes of specified queue or all queues.

4.31.3 Arguments

connect
: Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

target
: Name of queue whose attributes are to be reported. If this argument is null, returns attributes of all queues known to the server.

output_attribs
: Pointer to a list of attributes to return. If this argument is a null pointer or points to a null string, returns all attributes. Each attribute is described in an attrl structure, defined in `pbs_ifl.h` as:

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

extend
: Character string for extensions to command. Not currently used.

4.31.3.1 Members of attrl Structure

name
: Points to a string containing the name of the attribute.

resource
: Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

value
: Points to a string containing the value of the attribute or resource.

next

Points to next attribute in list. A null pointer terminates the list.

4.31.4 Return Value

Returns a pointer to a list of batch_status structures describing the queue(s).

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

4.31.4.1 The 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;
}
```

4.31.5 Cleanup

You must free the list of batch_status structures when no longer needed, by calling pbs_statfree().

4.31.6 See Also

qstat, pbs_connect, pbs_statfree

4.32 pbs_statresv

get status of PBS reservation(s)

4.32.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*void pbs_statfree(struct batch_status *psj)*

*struct batch_status *pbs_statresv(int connect, char *target, struct attrl *output_attribs, char *extend)*

4.32.2 Description

Issues a batch request to get the status of PBS reservation(s).

Generates a *Status Reservation* (71) batch request and sends it to the server over the connection specified by connect.

Returns specified attributes or all attributes of specified reservation or all reservations.

4.32.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

target

ID of reservation whose attributes are to be reported. If this argument is a null pointer or points to a null string, returns attributes of all reservations the user is authorized to query.

Format for advance reservation:

R<sequence number>.<server name>

Format for standing reservation:

S<sequence number>.<server name>

Format for maintenance reservation:

M<sequence number>.<server name>

output_attribs

Pointer to a list of attributes to return. If this argument is null, returns all attributes. Each attribute is described in an attrl structure, defined in `pbs_ifl.h` as:

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

extend

Character string for extensions to command. Not currently used.

4.32.3.1 Members of attrl Structure

name
: Points to a string containing the name of the attribute.

resource
: Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

value
: Points to a string containing the value of the attribute or resource.

next
: Points to next attribute in list. A null pointer terminates the list.

4.32.4 Return Value

Returns a pointer to a list of batch_status structures describing the reservation(s).

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

4.32.4.1 The 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;
}
```

4.32.5 Cleanup

You must free the list of batch_status structures when no longer needed, by calling pbs_statfree().

4.32.6 See Also

qstat, pbs_connect, pbs_statfree

4.33 pbs_statrsc

get status of PBS resources

4.33.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*void pbs_statfree(struct batch_status *psj)*

*struct batch_status *pbs_statrsc(int connect, char *rescname, struct attrl *output_attribs, char *extend)*

4.33.2 Description

Issues a batch request to query and return the status of a specified resource, or a set of resources at a server.

Generates a *Status Resource* (82) batch request and sends it to the server over the connection specified by connect.

4.33.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

rescname

Name of resource to be queried. If this is null, queries all resources at the server.

output_attribs

Pointer to a list of attributes to return. If this argument is a null pointer or points to a null string, returns all attributes. Each attribute is described in an attrl structure, defined in `pbs_ifl.h` as:

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

extend

Character string for extensions to command. Not currently used.

4.33.3.1 Members of attrl Structure

name

Points to a string containing the name of the attribute.

resource

Points to a string containing the name of a resource. Should be a null pointer.

value

Points to a string containing the value of the attribute or resource. Should always be a pointer to a null string.

next

Points to next attribute in list. A null pointer terminates the list.

4.33.4 Querying Resources at Server

Use the pbs_connect() command to get a connection handle at the server.

To query all resources at the server, pass a null pointer as the name of the resource.

4.33.5 Return Value

For a single resource, if the resource can be queried, returns a pointer to a batch_status structure containing the status of the specified resource.

If the resource cannot be queried, the routine returns a null pointer, and the error number is available in the global integer pbs_errno.

When querying a server, the connection to the server is already established by pbs_connect(). If there are resources at the server, returns a pointer to a batch_status structure describing the queryable resource(s) at the server.

In the unlikely event that the server does not contain any queryable resources because the user is unprivileged and all resources are marked as invisible (the i flag is set), returns a NULL pointer, and pbs_errno is set to *PBSE_NONE* (*0*).

4.33.5.1 The 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;
}
```

4.33.6 Cleanup

You must free the list of batch_status structures when no longer needed, by calling pbs_statfree().

4.33.7 See Also

qstat, pbs_connect

4.34 pbs_statsched

get status of PBS schedulers

4.34.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*void pbs_statfree(struct batch_status *psj)*

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

4.34.2 Description

Issues a batch request to get the status of the PBS schedulers.

Generates a *Status Scheduler (81)* batch request and sends it to the server over the connection specified by connect.

This command returns status of the default scheduler and all multischeds.

4.34.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

output_attribs

Pointer to a list of attributes to return. If this argument is a null pointer or points to a null string, returns all attributes. Each attribute is described in an attrl structure, defined in `pbs_ifl.h` as:

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

extend

Character string for extensions to command. Not currently used.

4.34.3.1 Members of attrl Structure

name

Points to a string containing the name of the attribute.

resource

Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

value

Points to a string containing the value of the attribute or resource. Should always be a pointer to a null string.

next

Points to next attribute in list. A null pointer terminates the list.

4.34.4 Return Value

Returns a pointer to a list of batch_status structures describing the default scheduler and all multischeds.

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

4.34.4.1 The 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;
}
```

4.34.5 Cleanup

You must free the list of batch_status structures when no longer needed, by calling pbs_statfree().

4.34.6 See Also

qstat, pbs_connect

4.35 pbs_statserver

get status of a PBS batch server

4.35.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*void pbs_statfree(struct batch_status *psj)*

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

4.35.2 Description

Issues a batch request to get the status of a batch server.

Generates a *Status Server* (21) batch request and sends it to the server over the connection specified by connect.

4.35.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

output_attribs

Pointer to a list of attributes to return. If this argument is a null pointer or points to a null string, returns all attributes. Each attribute is described in an attrl structure, defined in `pbs_ifl.h` as:

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

extend

Character string for extensions to command. Not currently used.

4.35.3.1 Members of attrl Structure

name

Points to a string containing the name of the attribute.

resource

Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

value

Points to a string containing the value of the attribute or resource. Should always be a pointer to a null string.

next

Points to next attribute in list. A null pointer terminates the list.

4.35.4 Return Value

Returns a pointer to a batch_status structure describing the server.

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

4.35.4.1 The 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;
}
```

4.35.5 Cleanup

You must free the batch_status structure when no longer needed, by calling pbs_statfree().

4.35.6 See Also

qstat, pbs_connect, pbs_statfree

4.36 pbs_statvnode

get status of PBS vnode(s) on execution hosts

4.36.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*void pbs_statfree(struct batch_status *psj)*

*struct batch_status *pbs_statvnode(int connect, char *target, struct attrl *output_attribs, char *extend)*

4.36.2 Description

Issues a batch request to get the status of PBS vnodes on execution hosts.

Generates a *Status Node* (58) batch request and sends it to the server over the connection specified by connect.

Returns specified attributes or all attributes of specified execution host vnode or all execution host vnodes.

4.36.3 Arguments

connect
: Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

target
: Name of execution host vnode whose attributes are to be reported. If this argument is null, returns attributes of all execution host vnodes known to the server.

output_attribs
: Pointer to a list of attributes to return. If this argument is a null pointer or points to a null string, returns all attributes. Each attribute is described in an attrl structure, defined in `pbs_ifl.h` as:

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

extend
: Character string for extensions to command. Not currently used.

4.36.3.1 Members of attrl Structure

name
: Points to a string containing the name of the attribute.

resource
: Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

value
: Points to a string containing the value of the attribute or resource.

next
: Points to next attribute in list. A null pointer terminates the list.

4.36.4 Return Value

Returns a pointer to a list of batch_status structures describing the vnode(s).

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

4.36.4.1 The 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;
}
```

4.36.5 Cleanup

You must free the list of batch_status structures when no longer needed, by calling pbs_statfree().

4.36.6 See Also

qstat, pbs_connect, pbs_statfree

4.37 pbs_submit

submit a PBS batch job

4.37.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*char *pbs_submit(int connect, struct attropl *attrib_list, char *jobscript, char *destqueue, char *extend)*

4.37.2 Description

Issues a batch request to submit a new batch job.

Generates a *Queue Job* (1) batch request and sends it to the server over the connection specified by connect.

Submits job to specified queue at connected server, or if no queue is specified, to default queue at connected server.

4.37.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

attrib_list

Pointer to a list of attributes explicitly requested for job. Each attribute is described in an attropl structure, defined in `pbs_ifl.h` as:

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

For any attribute that is not specified or that is a null pointer, PBS takes the default action for that attribute. The default action is to assign the default value or to not pass the attribute with the job; the action depends on the attribute.

jobscript

Pointer to path to job script. Can be absolute or relative. Relative path begins with the directory where the user submits the job.

If null pointer or pointer to null string, no script is passed with job.

destqueue

Pointer to name of destination queue at connected server. If this is a null pointer or points to a null string, the job is submitted to the default queue at the connected server.

extend

Character string for extensions to command. Not currently used.

4.37.3.1 Members of attropl Structure

next
: Points to next attribute in list. A null pointer terminates the list.

name
: Points to a string containing the name of the attribute.

resource
: Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

value
: Points to a string containing the value of the attribute or resource.

op
: Operation to perform on the attribute or resource. In this command, the only allowed operator is SET.

4.37.4 Return Value

Returns a pointer to a character string containing the job ID assigned by the server.

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

4.37.5 Cleanup

The space for the job ID returned by `pbs_submit()` is allocated by `pbs_submit()`. Free it via a call to `free()` when you no longer need it.

4.37.6 See Also

qsub, pbs_connect

4.38 pbs_submit_resv

submit a PBS reservation

4.38.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*char *pbs_submit_resv(int connect, struct attropl *attrib_list, char *extend)*

4.38.2 Description

Issues a batch request to submit a new reservation.

Generates a *Submit Reservation* (70) batch request and sends it to the server over the connection specified by connect.

Returns a pointer to the reservation ID.

4.38.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection over which to send batch request to server.

attrib_list

Pointer to a list of attributes to set, with values. Each attribute is described in an attropl structure, defined in `pbs_ifl.h` as:

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

For any attribute that is not specified or that is a null pointer, PBS takes the default action for that attribute. The default action is to assign the default value or to not pass the attribute with the reservation; the action depends on the attribute.

extend

Character string for extensions to command. Not currently used.

4.38.3.1 Members of attropl Structure

next

Points to next attribute in list. A null pointer terminates the list.

name

Points to a string containing the name of the attribute.

resource

Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

value

Points to a string containing the value of the attribute or resource.

op

Operator. The only allowed operator for this function is SET.

4.38.4 Return Value

Returns a pointer to a character string containing the reservation ID assigned by the server.

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

4.38.5 Cleanup

The space for the reservation ID returned by `pbs_submit_resv()` is allocated by `pbs_submit_resv()`. Free it via a call to `free()` when you no longer need it.

4.38.6 See Also

pbs_rsub, pbs_connect

4.39 pbs_terminate

shut down a PBS batch server

4.39.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

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

4.39.2 Description

Issues a batch request to shut down a batch server.

Generates a *Server Shutdown* (17) batch request and sends it to the server over the connection specified by connect.

The `pbs_terminate()` command exits after the server has completed its shutdown procedure.

4.39.3 Required Privilege

You must have Operator or Manager privilege to run this command.

4.39.4 Arguments

connect
: Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

manner
: Manner in which to shut down server. The available manners are defined in `pbs_ifl.h`. Valid values: *SHUT_IMMEDIATE, SHUT_DELAY, SHUT_QUICK*. See "qterm" on page 236 of the PBS Professional Reference Guide for information on manner in which to shut down server.

extend
: Character string for extensions to command. Not currently used.

4.39.5 Return Value

The routine returns *0* (*zero*) on success.

If an error occurred, the routine returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

4.39.6 See Also

qterm, pbs_connect

5
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.

5.1 TM Library Routines

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

5.2 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

5.2.1 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()
```

5.2.2 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 an there are no events to report, the function will block waiting for an event. If no local error takes place, TM_SUCCESS is returned. If an error is reported by MoM for an event, then the argument tm_errno will be set to an error code.

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

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

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

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

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

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

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

5.2.3 See Also

pbs_mom(8B), pbs_sched(8B)

6
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.

6.1 RM Library Routines

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

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

resource monitor API

6.2.1 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;
```

6.2.2 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 TPP 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 tpp_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.

6.2.3 See Also

tcp(4P), udp(4P)

7
TCL/tk Interface

As of version 19.4.1, the PBS TCL API is **deprecated**.

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.

7.1 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 **obsolete**. Any attempt to use it will result in a PBSE_NOSUPPORT error being returned.

7.2 pbs_tclapi

PBS TCL Application Programming Interface

7.2.1 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.

7.2.2 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.

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.

7.2.3 See Also

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

8 Hooks

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

8.1 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)`

8.2 How Hooks Work

8.2.1 Hook Contents and Permissions

A hook contains a Python script. The script is evaluated by a Python 3 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.

8.2.2 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()".

8.2.3 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()
```

8.2.4 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.

8.3 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.

8.3.1 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.

8.3.1.1 Description of pbs Module

8.4 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.

8.4.0.1 pbs Module Objects

See "The pbs Module" on page 82 in the PBS Professional Hooks Guide.

8.4.0.2 pbs Module Global Attribute Creation Methods

See "PBS Types and Their Methods" on page 168 in the PBS Professional Hooks Guide.

8.4.0.3 Attributes and Resources

See "Using Attributes and Resources in Hooks" on page 45 in the PBS Professional Hooks Guide.

8.4.0.4 Exceptions

See "Table of Exceptions" on page 44 in the PBS Professional Hooks Guide and "Hook Alarm Calls and Unhandled Exceptions" on page 44 in the PBS Professional Hooks Guide.

8.4.0.5 See Also

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

8.4.1 The pbs_manager() API

The `pbs_manager()` API is described in "pbs_manager" on page 41.

The `pbs_manager()` API contains the following:

- An obj_name called "hook" defined as MGR_OBJ_HOOK, for use with non-built-in hooks
- An obj_name called "pbshook" defined as MGR_OBJ_PBS_HOOK, for use with built-in hooks.
- 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".

8.4.1.1 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()` shows:

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

8.4.1.2 Privilege for Hooks

To run, hooks require root privilege on Linux, and local Administrators privilege on Windows.

8.4.1.3 Examples of Using pbs_manager()

Example 8-1: 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 8-2: 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 8-3: 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 8-4: 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);
```

8.4.2 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.

8.4.2.1 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
```

8.5 pbs_stathook(3B)

get status information about PBS site hooks

8.5.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*struct batch_status *pbs_stathook(int connect, char *hook_id, struct attrl *output_attribs, char *extend)*

*void pbs_statfree(struct batch_status *psj)*

8.5.2 Description

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

Generates a *Status Hook* batch request and sends it to the server over the connection specified by connect.

8.5.2.1 Required Privilege

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

8.5.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

hook_id

Hook name, null string, or null pointer.

If *hook_id* specifies a name, the attribute-value list for that hook is returned.

If *hook_id* is a null string or a null pointer, the status of all hooks at the current server is returned.

output_attribs

Pointer to a list of attributes to return. If this list is null, all attributes are returned. Each attribute is described in an attrl structure, defined in `pbs_ifl.h` as:

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

extend

Character string where you can specify limits or extensions of your selection.

8.5.3.1 Members of attrl Structure

name

Points to a string containing the name of the attribute.

resource
: Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

value
: Should always be a pointer to a null string.

next
: Points to next attribute in list. A null pointer terminates the list.

8.5.4 Return Value

Returns a pointer to a list of batch_status structures for the specified site hook. If no site hook can be queried for status, returns the null pointer.

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

8.5.4.1 The 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;
}
```

8.5.5 Cleanup

You must free the list of batch_status structures when no longer needed, by calling pbs_statfree().

8.5.6 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 "Tables: Reading & Setting Built-in Job Resources in Hooks" on page 64 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)"
```

8.5.7 See Also

pbs_connect, pbs_statfree, “Hook Attributes” on page 349 of the PBS Professional Reference Guide

9
Custom Authentication and Encryption Library APIs

This chapter describes LibAuth, which contains APIs you can use to create your own custom authentication or encryption library.

Call the `pbs_auth_set_config` API first before calling any other LibAuth API.

Table of Authentication and Encryption APIs

9.1 pbs_auth_set_config . . . 124
9.2 pbs_auth_create_ctx . . . 125
9.3 pbs_auth_destroy_ctx . . . 127
9.4 pbs_auth_get_userinfo . . . 128
9.5 pbs_auth_process_handshake_data . . . 130
9.6 pbs_auth_encrypt_data . . . 132
9.7 pbs_auth_decrypt_data . . . 133

9.1 pbs_auth_set_config

specify configuration for authentication library

9.1.1 Synopsis

*void pbs_auth_set_config(const pbs_auth_config_t *auth_config)*

9.1.2 Description

Specifies configuration for the authentication library. Use this to specify logging method, where to find required credentials, etc.

Call this API first before calling any other LibAuth API.

9.1.3 Arguments

const pbs_auth_config_t *auth_config

Pointer to a configuration structure

9.1.4 Configuration Structure

typedef struct pbs_auth_config {

char *pbs_home_path;

Path to PBS_HOME directory (aka same value as PBS_HOME in pbs.conf). This must be a null-terminated string.

char *pbs_exec_path;

Path to PBS_EXEC directory (aka same value as PBS_EXEC in pbs.conf). This must be a null-terminated string.

char *auth_method;

Name of authentication method (aka same value as PBS_AUTH_METHOD in pbs.conf). This must be a null-terminated string.

char *encrypt_method;

Name of encryption method (aka same value as PBS_ENCRYPT_METHOD in pbs.conf). This must be a null-terminated string

void (*logfunc)(int type, int objclass, int severity, const char *objname, const char *text);

Function pointer to the logging method with the same signature as log_event from Liblog.

With this, the user of the authentication library can redirect logs from the authentication library into respective log files or stderr in case no log files.

If func is set to NULL then logs will be written to stderr (if available, else no logging at all).

} pbs_auth_config_t;

9.1.5 Return Value

None return value

9.2 pbs_auth_create_ctx

create authentication context

9.2.1 Synopsis

*int pbs_auth_create_ctx(void **ctx, int mode, int conn_type, char *hostname)*

9.2.2 Description

Creates an authentication context for a given mode and connection type. Context is used by other LibAuth APIs for authentication, encryption, and decryption of data.

9.2.3 Arguments

void **ctx

Pointer to auth context to be created

int mode

Specifies type of context to be created. Should be one of *AUTH_CLIENT* or *AUTH_SERVER*.

Use AUTH_CLIENT for client-side (who is initiating authentication) context

Use AUTH_SERVER for server-side (who is authenticating incoming user/connection) context

```
enum AUTH_ROLE {
    AUTH_ROLE_UNKNOWN = 0,
    AUTH_CLIENT,
    AUTH_SERVER,
    AUTH_ROLE_LAST
};
```

int conn_type

Specifies type of connection is for context to be created. Should be one of *AUTH_USER_CONN* or *AUTH_SERVICE_CONN*

Use AUTH_USER_CONN for user-oriented connection (such as when PBS client is connecting to PBS server)

Use AUTH_SERVICE_CONN for service-oriented connection (such as when PBS MoM is connecting to PBS server via PBS comm)

```
enum AUTH_CONN_TYPE {
    AUTH_USER_CONN = 0,
    AUTH_SERVICE_CONN
};
```

char *hostname

The null-terminated hostname of another authenticating party

9.2.4 Return Value

Integer.

0
: On Success

1
: On Failure

9.2.5 Cleanup

When a context created using this API is no longer required, destroy it via `auth_destroy_ctx.`

9.3 pbs_auth_destroy_ctx

destroy an authentication context created using `auth_create_ctx`

9.3.1 Synopsis

*void pbs_auth_destroy_ctx(void *ctx)*

9.3.2 Description

Destroys the authentication context created using `auth_create_ctx`

9.3.3 Arguments

void *ctx
: Pointer to authentication context to be destroyed

9.3.4 Return Value

No return value

9.4 pbs_auth_get_userinfo

extract username with its realm, and hostname of connecting party from authentication context

9.4.1 Synopsis

*int pbs_auth_get_userinfo(void *ctx, char **user, char **host, char **realm)*

9.4.2 Description

Extracts username with its realm, and hostname of the connecting party from the given authentication context.

The extracted user, host, and realm values are null-terminated strings.

This API is mostly useful for authenticating on the server side to get information about an authenticating client.

The authenticating server can use this information from the auth library to match against the actual username/realm/hostname provided by the connecting party.

9.4.3 Arguments

void *ctx
: Pointer to auth context from which information will be extracted

char **user
: Pointer to a buffer in which this API will write the username

char **host
: Pointer to a buffer in which this API will write hostname

char **realm
: Pointer to a buffer in which this API will write the realm

9.4.4 Return Value

Integer

0
: On success

1
: On failure

9.4.5 Cleanup

When the returned user, host, and realm are no longer required, free them using `free()`, since they use allocated heap memory.

9.4.6 Example

This example shows what the values of user, host, and realm will be. Let's use an example with GSS/Kerberos authentication, where the authentication client hostname is "xyz.abc.com", the username is "test", and in the Kerberos configuration, the domain realm is "PBSPRO". When the client authenticates to the server using the Kerberos authentication method, it is authenticated as "test@PBSPRO", and this API returns user = test, host = xyz.abc.com, and realm = PBSPRO.

9.5 pbs_auth_process_handshake_data

handle and generate handshakes, and generate handshake data

9.5.1 Synopsis

*int pbs_auth_process_handshake_data(void *ctx, void *data_in, size_t len_in, void **data_out, size_t *len_out, int *is_handshake_done)*

9.5.2 Description

Process incoming handshake data and do the handshake. If required generate handshake data which to be sent to another party. If there is no incoming data then initiate a handshake and generate initial handshake data to be sent to the authentication server.

9.5.3 Arguments

void *ctx
: Pointer to authentication context for which handshake is happening

void *data_in
: Incoming handshake data to process, if any. A NULL value indicates that this API should initiate a handshake and generate initial handshake data to be sent to the authentication server.

size_t len_in
: Length of incoming handshake data, if any, else 0

void **data_out
: Outgoing handshake data to be sent to another authentication party.

A NULL value indicates that the handshake is completed and no further data needs to be sent.

When this API returns 1 (failure), `data_out` contains the error message.

size_t *len_out
: Length of outgoing handshake/auth error data, if any, else 0

int *is_handshake_done
: Indicates whether handshake is completed or not.

0 means that the handshake is not completed.

1 means that the handshake is completed.

9.5.4 Return Value

Integer

0
: On success

1
: On failure

On failure, the value of `data_out` is the error data/message, to be sent to another authentication party as authentication error data.

9.5.5 Cleanup

When the returned `data_out` (if any) is no longer required, free it using `free()`, since it uses allocated heap memory.

9.6 pbs_auth_encrypt_data

encrypt data using specified authentication context

9.6.1 Synopsis

*int pbs_auth_encrypt_data(void *ctx, void *data_in, size_t len_in, void **data_out, size_t *len_out)*

9.6.2 Description

Encrypt given unencrypted data using the specified authentication context.

9.6.3 Arguments

void *ctx
: Pointer to auth context which will be used while encrypting given unencrypted data

void *data_in
: unencrypted data to encrypt

size_t len_in
: Length of unencrypted data

void **data_out
: Encrypted data

size_t *len_out
: Length of encrypted data

9.6.4 Return Value

Integer

0
: Success

1
: Failure

9.6.5 Cleanup

When the returned `data_out` is no longer required, free it using `free()`, since it uses allocated heap memory.

9.7 pbs_auth_decrypt_data

decrypt data

9.7.1 Synopsis

*int pbs_auth_decrypt_data(void *ctx, void *data_in, size_t len_in, void **data_out, size_t *len_out)*

9.7.2 Description

Decrypt encrypted data using the specified authentication context

9.7.3 Arguments

void *ctx
: Pointer to authentication context which will be used while decrypting given encrypted data

void *data_in
: Encrypted data to decrypt

size_t len_in
: Length of encrypted data

void **data_out
: Unencrypted data

size_t *len_out
: Length of unencrypted data

9.7.4 Return Value

Integer

0
: On success

1
: On failure

9.7.5 Cleanup

When the returned `data_out` is no longer required, free it using `free()`, since it uses allocated heap memory.

Index

A
activereq PG-102
addreq PG-102
allreq PG-102

C
closerm PG-102
commands PG-4
configrm PG-102
credential PG-21

D
downrm PG-102

E
executor PG-4

F
flushreq PG-102
fullresp PG-102

G
getreq PG-102

J
job
    executor (MoM) PG-4

M
MoM PG-3, PG-4

O
openrm PG-102

P
pbs_alterjob PG-24
pbs_asyrunjob PG-26, PG-58
pbs_auth_create_ctx PG-125
pbs_auth_decrypt_data PG-133
pbs_auth_destroy_ctx PG-127
pbs_auth_encrypt_data PG-132
pbs_auth_get_userinfo PG-128
pbs_auth_process_handshake_data PG-130
pbs_auth_set_config PG-124
pbs_connect PG-21, PG-30
pbs_default PG-32
pbs_deljob PG-33
pbs_delresv PG-35
pbs_disconnect PG-36
pbs_geterrmsg PG-37
pbs_holdjob PG-38
pbs_iff PG-21
pbs_locjob PG-39
pbs_manager PG-41
pbs_module PG-113
pbs_mom PG-3, PG-4
pbs_movejob PG-47
pbs_msgjob PG-49
pbs_orderjob PG-51
pbs_preempt_jobs PG-52
pbs_relnodesjob PG-54
pbs_rerunjob PG-56
pbs_rlsjob PG-57
pbs_runjob PG-26, PG-58
pbs_sched PG-2, PG-3
pbs_selectjob PG-60
pbs_selstat PG-63
pbs_server PG-2, PG-3
pbs_sigjob PG-67
pbs_statfree PG-69
pbs_stathook(3B) PG-119
pbs_stathost PG-70
pbs_statjob PG-72
pbs_statnode PG-75
pbs_statque PG-77
pbs_statresv PG-79
pbs_statrsc PG-81
pbs_statsched PG-83
pbs_statserver PG-85
pbs_statvnode PG-87
pbs_submit PG-89
pbs_submit_resv PG-91
pbs_tclapi PG-106
pbs_tclsh PG-105
pbs_terminate PG-93
pbs_wish PG-105

S
scheduler PG-3

server PG-3

T

TCL PG-105
tm_atnode PG-96
tm_attach PG-96
tm_finalize PG-96
tm_init PG-96
tm_kill PG-96
tm_nodeinfo PG-96
tm_notify PG-96
tm_obit PG-96
tm_poll PG-96
tm_publish PG-96
tm_rescinfo PG-96
tm_spawn PG-96
tm_subscribe PG-96
tm_taskinfo PG-96