Altair PBS Professional 2021.1.3

Plugins (Hooks) Guide

You are reading the Altair PBS Professional 2021.1.3

Hooks Guide (HG)

Updated 4/7/22

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

About PBS Documentation vii

1 New Hook Features 1

1.1 Changes in Previous Releases . . . 1
1.2 Deprecations and Removals . . . 3

2 Introduction to Hooks 5

2.1 Introduction to Hooks . . . 5
2.2 Glossary . . . 5
2.3 Prerequisites and Requirements for Hooks . . . 7
2.4 Uses for Hooks . . . 7

3 Quick Start with Hooks 11

3.1 Simple How-to for Writing Hooks . . . 11
3.2 Writing Hooks: Basic Hook Structure . . . 11
3.3 Example of Simple Hook . . . 12
3.4 Importing Hook Configuration File . . . 13
3.5 Creating and Importing Your Hook . . . 13
3.6 Setting Attributes for Your Hook . . . 13

4 Hook Basics 15

4.1 Hook Basics . . . 15
4.2 Viewing Hook Information . . . 22
4.3 Restarting the Python Interpreter . . . 23
4.4 Attributes and Parameters Affecting Hooks . . . 24
4.5 Python Modules and PBS . . . 24
4.6 See Also . . . 26

5 Creating and Configuring Hooks 29

5.1 Creating and Configuring Hooks . . . 29
5.2 Writing Hook Scripts to Operate on PBS Elements . . . 40
5.3 Advice and Caveats for Writing Hooks . . . 68

6 Hook Objects and Methods 75

6.1 The pbs Module . . . 76
6.2 PBS Interface Objects . . . 76
6.3 Event Objects . . . 87
6.4 Server Objects . . . 118
6.5 Queue Objects . . . 121
6.6 Job Objects . . . 122
6.7 The exec_vnode Object . . . 129
6.8 Chunk Objects . . . 131
6.9 Reservation Objects . . . 131
6.10 Vnode Objects . . . 133
6.11 Configuration File Objects . . . 135
6.12 Constant Objects . . . 140
6.13 Object Members and Methods . . . 141

7 Built-in Hooks 155

7.1 Managing Built-in Hooks . . . 155
7.2 Prerequisites . . . 155
7.3 Allowed Operations . . . 155
7.4 Viewing Built-in Hooks . . . 155
7.5 Setting Attributes of Built-in Hooks . . . 156
7.6 Editing and Importing Configuration Files for Built-in Hooks . . . 156
7.7 Restrictions . . . 156
7.8 Replacing a Built-in Hook with Your Own Hook . . . 156
7.9 Errors and Logging when Operating on Built-in Hooks . . . 157

8 Debugging Hooks 159

8.1 The `pbs_python` Hook Debugging Tool . . . 159
8.2 Files for Debugging . . . 159
8.3 Steps to Debug a Hook Using pbs_python . . . 165
8.4 Caveats and Restrictions for pbs_python . . . 165
8.5 Examples of Using `pbs_python` to Debug Hooks . . . 166
8.6 Using Log Messages to Debug Hook Scripts . . . 174
8.7 Checking Hook Syntax using Python . . . 174
8.8 Examples of Debugging Files . . . 174
8.9 Interactive Debugging using pbs_python . . . 221
8.10 Error Reporting and Logging . . . 221

9 Hook Examples 231

Index 289

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 New Hook Features

This chapter briefly lists new features by release, with the most recent listed first. This chapter also lists deprecated elements, such as options, keywords, etc.

The *Release Notes* included with this release of PBS Professional list all new features in this version of PBS Professional, and any warnings or caveats. Be sure to review the Release Notes, as they may contain information that was not available when this book was written.

1.1 Changes in Previous Releases

Improved Cgroups Hook (2020.1)

The cgroups hook is improved for 2020.1. See "Configuring and Using PBS with Cgroups" on page 571 in the PBS Professional Administrator's Guide.

Faster Read of Custom Job Resources by Execution Hooks (2020.1)

You can specify which custom job resources are cached at MoMs so that execution hooks can read them faster. See "Allowing Execution Hooks to Read Custom Job Resources Faster" on page 263 in the PBS Professional Administrator's Guide.

New Post-suspend and Pre-resume Hooks (2020.1)

PBS has two new hook events for just after suspending a job and just before resuming it. See section 6.3.1, "Event Types", on page 87.

Python Version Changed to 3.6 (19.4.1)

PBS 19.4.1 uses Python version 3.6.

Hooks Support Reliable Job Startup and Run (19.2)

Hooks have been enhanced to allow you to provide jobs with extra vnodes in case of vnode failure. See "Vnode Fault Tolerance for Job Start and Run" on page 437 in the PBS Professional Administrator's Guide.

New Reservation End Hook (19.2)

You can create hooks for the end of a reservation. See "resv_end: Event when Reservation Ends" on page 90 in the PBS Professional Hooks Guide.

Python Version Changed to 2.7.1 (18.2.3)

PBS 18.2.3 uses Python 2.7.1. The use of Python 2.5.1 is deprecated.

Periodic Server Hook (18.2.3)

PBS has a periodic hook that runs at the server. See section 6.3.1.7, "periodic: Periodic Event at Server Host", on page 95.

Hook to Run Job Start Time Estimator (18.2.3)

PBS has a built-in hook named PBS_est that can run the job start time estimator. See "Estimating Job Start Time" on page 133 in the PBS Professional Administrator's Guide.

Configurable Python Interpreter Restarts (18.2.3)

You can configure how often you want the Python interpreter to restart. See section 4.3, "Restarting the Python Interpreter", on page 23.

PBS Can Report Custom Resources Set in Hooks (18.2.3)

MoM can accumulate and report custom resources that are set in a hook. See section 5.2.4.12, “Setting Job Resources in Hooks”, on page 49.

The execjob_prologue Hook Runs on All Sister MoMs (18.2.3)

The execjob_prologue hook runs on all sister MoMs. See section 6.3.1.9, “execjob_prologue: Event Just Before Execution of Top-level Job Process”, on page 97.

New Hook Events (13.0)

PBS provides three new hook events:

- An execjob_launch hook runs just before MoM runs the user's program
- An execjob_attach hook runs when pbs_attach is called
- An exechost_startup hook runs when MoM starts up or is HUPed

See section 4.1.2, “When Hooks Run”, on page 15, section 6.3.1.10, “execjob_launch: Event when Execution Host Receives Job”, on page 98, section 6.3.1.11, “execjob_attach: Event when pbs_attach() runs”, on page 100, and section 6.3.1.17, “exechost_startup: Event When Execution Host Starts Up”, on page 106.

Configuration Files for Hooks (13.0)

You can use configuration files with hooks. See section 5.1.6, “Using Hook Configuration Files”, on page 32.

Configuring Vnodes in Hooks (13.0)

You can use hooks to configure vnode attributes and resources. See section 5.2.4.11, “Setting and Unsetting Vnode Resources and Attributes”, on page 48.

Adding Custom Resources in Hooks (13.0)

You can use hooks to add custom non-consumable host-level resources. See section 5.2.7, “Adding Custom Non-consumable Host-level Resources”, on page 64.

Node Health Hook Features (13.0)

PBS has node health checking features for hooks. You can offline and clear vnodes, and restart the scheduling cycle. See section 5.2.12.4, “Offlining and Clearing Vnodes Using the fail_action Hook Attribute”, on page 66 and section 5.2.6, “Restarting Scheduler Cycle After Hook Failure”, on page 63.

Hook Debugging Enhancements (13.0)

You can get hooks to produce debugging information, and then read that information in while debugging hooks. See Chapter 8, "Debugging Hooks", on page 159.

Managing Built-in Hooks (13.0)

You can enable and disable built-in hooks. See Chapter 7, "Built-in Hooks", on page 155.

Scheduler Does not Trigger modifyjob Hooks (13.0)

The scheduler does not trigger modifyjob hooks. See Chapter 5, "Creating and Configuring Hooks", on page 29.

runjob Hook can Modify Job Attributes (12.2)

The runjob hook can modify a job's attributes and resources. See section 5.2.4, “Using Attributes and Resources in Hooks”, on page 44.

Execution Event and Periodic Hooks (12.0)

You can write hooks that run at the execution host when the job reaches the execution host, when the job starts, ends, is killed, and is cleaned up. You can also write hooks that run periodically on all execution hosts. See Chapter 5, "Creating and Configuring Hooks", on page 29.

Vnode Access for Hooks (11.0)

Hooks have access to vnode attributes and resources. See Chapter 5, "Creating and Configuring Hooks", on page 29.

Provisioning (10.2)

PBS provides automatic provisioning of an OS or application on vnodes that are configured to be provisioned. When a job requires an OS that is available but not running, or an application that is not installed, PBS provisions the vnode with that OS or application. See Chapter 7, "Provisioning", on page 329.

New Hook Type (10.2)

PBS has a new hook type which can be triggered when a job is to be run. See "Creating and Configuring Hooks" on page 29.

Hooks (10.0)

Hooks are custom executables that can be run at specific points in the execution of PBS. They accept, reject, or modify the upcoming action. This provides job filtering, patches or workarounds, and extends the capabilities of PBS, without the need to modify source code. See Chapter 5, "Creating and Configuring Hooks", on page 29.

1.2 Deprecations and Removals

The use of Python 2.x is **deprecated**. PBS now uses Python 3.6. (19.4.1)

2 Introduction to Hooks

Hooks are custom executables that can be run at specific points in the execution of PBS. They accept, reject, or modify the upcoming action. This provides job filtering, patches, MoM startup checks, workarounds, etc., and extends the capabilities of PBS, without the need to modify source code.

This chapter describes how hooks can be used, how they work, the interface to hooks provided by the pbs module, how to create and deploy hooks, and how to get information about hooks.

Please read the entire chapter, and the "Special Notes (Hooks)" section of the release notes, before writing any hooks.

2.1 Introduction to Hooks

A hook is a block of Python code that PBS executes at certain events, for example, when a job is queued. As long as the Python code conforms to the rules we describe, you can have it do whatever you want. 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. The hook can use a configuration file that you provide. PBS provides an interface for use in hooks. This interface allows hooks to read and/or modify things such as job, server, vnode, and queue attributes, and the event that triggered the hook.

2.1.1 Built-in Hooks

Some functions of standard PBS are accomplished through built-in hooks. We use the keyword *pbshook* with these hooks. These hooks are not designed to be altered, so they have some restrictions placed on them. See Chapter 7, "Built-in Hooks", on page 155.

2.2 Glossary

Accept an action

The hook allows the action to take place.

Action

A PBS operation or state transition. Also called an *event*. For a list of events, see section 6.3.1, “Event Types”, on page 87.

Built-in hook

A hook that is supplied as part of PBS. These hooks cannot be created or deleted by administrators.

Creating a hook

When you "create a hook" using `qmgr`, you're telling PBS that you want it to make you an empty hook object that has no characteristics other than a name.

Event

A PBS operation or state transition. Also called *action*. For a list of events, see section 6.3.1, “Event Types”, on page 87.

Execution event hook, MoM hook

A hook that runs at an execution host. These hooks run after a job is received by MoM. Execution event hooks have names prefixed with "*execjob_*".

Failure action

The action taken when a hook fails to execute. Specified in the fail_action hook attribute. See section 5.1.9.2, "Using the fail_action Hook Attribute", on page 37.

Hook configuration file

Configuration file specific to a particular hook. See section 5.1.6, "Using Hook Configuration Files", on page 32.

Importing a hook

When you "import a hook" using `qmgr`, you're telling PBS which Python script to run when the hook is triggered.

Importing a hook configuration file

When you "import a hook configuration file" using `qmgr`, you're telling PBS which file should be stored as the configuration file for the specified hook.

MoM hook, execution event hook

A hook that runs at an execution host. These hooks run after a job is received by MoM. Execution event hooks have names prefixed with "*execjob_*"

Non-job event hook

A hook that is not directly related to a specific job. Non-job event hooks are periodic hooks, startup hooks, provisioning hooks, and reservation creation hooks.

pbshook

PBS keyword for a built-in hook.

pbs module

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

Pre-execution event hook, server hook

A hook that runs at the PBS server. A server hook runs before the job is sent to MoM. These hooks do not run on execution hosts. Pre-execution event hooks are for job submission, moving a job, altering a job, or just before sending a job to an execution host.

Reject an action

The hook prevents the action from taking place. For example, if a runjob hook rejects a job, the job is requeued.

Server hook, pre-execution event hook

A hook that runs at the PBS server. A server hook runs before the job is sent to MoM. These hooks do not run on execution hosts. Pre-execution event hooks are for job submission, moving a job, altering a job, or just before sending a job to an execution host.

2.3 Prerequisites and Requirements for Hooks

- To create a hook under Linux, you must be logged into the primary or secondary server host as root. You must create any hooks at the primary or secondary server host.
- When creating hooks, make sure that each execution host where execution or periodic hooks should run has the $reject_root_scripts MoM parameter set to *False*. The default for this parameter is *False*.
- In order for execution event hooks to function, either the query_other_jobs server attribute must be set to *True*, or root at every execution host must be added to the managers list (root@hostname must be added to the managers server attribute). If you have any hooks running with user set to *pbsuser*, you will have to set query_other_jobs to *True* (you probably don't want to add *pbsuser* to managers).

A normal, non-privileged, user cannot circumvent, disable, add, delete, or modify hooks or the environment in which the hooks are run.

2.4 Uses for Hooks

2.4.1 Routing Jobs

- Route jobs into specific queues or between queues:
  - Automatically route interactive jobs into a particular execution queue
  - Move a job to another queue; for example, if project allocation is used up, move job to "background" queue
- Reject job submissions that do not specify a valid queue, printing an error message explaining the problem
- Enable project-based ACLs for queues to make sure the appropriate job runs in the correct queue

2.4.2 Managing Resource Requests and Usage

- Reject improperly specified jobs:
  - Reject jobs which do not specify walltime
  - Reject jobs that request a number of processors that is not a multiple of 8
  - Reject jobs requesting a specific queue, but not requesting memory
  - Reject jobs whose processors per node is not specified or is not numeric
- Modify job resource requests:
  - Apply default memory limit to jobs that request a specific queue
  - Check on requested CPU and memory and modify these or supply them if missing
  - Adjust for the fact that users ask for 2GB on a machine that has 2GB physical memory, but only 1.8 GB available memory, by changing the memory request to 1.8GB
- Reject parallel jobs for some queues.
- Set default properties, for example, if "myri" is not set, set it to "*False*" to ensure Myrinet is used only for Myrinet jobs.
- Convert from ALPS-specific resource request strings into PBS-specific job requirements.
- Automatically translate old syntax to new syntax.
- Compensate for dissimilar system capabilities; for example, allow users to use more CPUs only if they use old, slow machines.
- Limit reservations submitted by users to a maximum amount of resources and walltime, but do not limit reservations submitted by PBS administrators.
- Define resources and set values.

2.4.3 Ensuring that Jobs Run Properly

- Make sure that jobs, or all jobs in a queue, request exclusive access (`-l place=excl`).
- Reject multi-host jobs, restricting each job to a single machine.
- Put a hold on the job if there isn't enough scratch space when the job is submitted.
- Reject jobs that could cause problems, based on the user and type of job that have caused previous problems. For example, if Bill's Abaqus jobs crash the system, reject new Abaqus jobs from Bill.
- Validate an input deck before the job is submitted.
- Modify a job's dependency list when the job is rejected.
- Modify a job's list of environment variables before it gets to the execution host(s).

2.4.4 Managing Job Output

- Manage where output goes by modifying a job's output path with the job's ID.

2.4.5 Controlling Interactive Jobs

- Control interactive job submission; for example, enable or disable interactive jobs at the server or queue level

2.4.6 Helping Schedule Jobs

- Increase the priority of an array job once the first subjob runs, by modifying the value of a job resource used in the job sorting formula
- Change scheduling according to user and job:
  - Set initial user-dependent coefficients for the scheduling formula. For example, set values of custom resources based on job attributes and user
  - Set whether or not the job is rerunnable, based on user
  - Calculate CPH (CPH == total ncpus * walltime in hours) and set a custom CPH job resource to the value
- Set initial priorities for jobs
- Periodically run the job start time estimator named `pbs_est` at the server. See "Estimating Job Start Time" on page 133 of the PBS Professional Administrator's Guide.

2.4.7 Communicating Information to Users

- Report useful error messages back to the user, e.g., `"You do not have sufficient walltime left to run your job for 1:00:00. Your walltime balance is 00:30:00."`

2.4.8 Managing User Activity

- Reject jobs from blacklisted users
- Prevent users from using `qalter` to change their jobs in any way, allowing only administrators to `qalter` jobs
- Prevent users from bypassing controls: disallow a job being submitted to queueA in a held state and then being moved to queueB where the job would not have passed hook checks for queueB initially. For example, if a queuejob hook disallows interactive jobs for queueB, the administrator also needs to ensure that an interactive job is not initially submitted to queueA and later moved to queueB
- Prevent users from overriding node_group_key with `qsub -lplace = group = X`, or with `qalter`
- Restrict the ability to submit a reservation to PBS administrators only

2.4.9 Enabling Accounting and Validation

- Make sure correct project designation is used: if no project or account string is found, look up username in database to find appropriate project to use and add it as project or account string before submission
- Submit job to correct queue based on project: check for project number and submit job to queues based on project type, e.g. project number 1234 jobs get submitted into "challenge" queue; similarly for "standard" queue, etc
- Validate project before the job executes; if validation fails, do not start job, and print error message. Validation can be based on project name, or for example requested resources, such as CPU hours

2.4.10 Allocation Management

- You can use a job submission (queuejob) hook to check whether an entity has enough resources allocated to accept the job.
- You can use a hook that runs just before the job is sent to the execution host (runjob) to perform allocation management tasks such as deducting requested amounts of resources from an entity's allocation.
- You can use a hook that runs after a job finishes (execjob_epilogue) to perform final allocation management tasks such as allocation reconciliation.

2.4.11 Managing Job Execution

Hooks that run periodically at execution hosts can do the following:

- Modify job environment variables
- Check vnode health
- Report I/O wait time
- Report memory usage integral (MB*time used)
- Report energy usage to run a given job, if you have power sensors on vnodes
- Report actual usage of accelerator hardware (FPGAs, GPUs, etc)
- Interrogate HW performance counters so that you can flag codes that are not running efficiently (e.g. FLOPS < 5% of peak FLOPS)
- Record how much disk space a job has accumulated in PBS_JOBDIR
- Record power usage, energy usage, and disk space usage

Hooks that run just before the user's program executes can do the following:

- Change the job shell or executable
- Change the job shell or executable arguments
- Change the job's environment variables

2.4.12 Configuring Vnodes

Hooks that run when an execution host starts can do the following:

- Configure vnodes on the local host
- Create custom resources for vnodes
- Offline vnodes that are not ready for use
- Return vnodes to use that have been offlined

2.4.13 Provisioning Vnodes

- Provision a vnode with a new AOE. See Chapter 7, "Provisioning", on page 329.

2.4.14 Accepting or Rejecting Job Task Attachment

- Allow or disallow action when MoM is about to attach a process for a job

3
Quick Start with Hooks

3.1 Simple How-to for Writing Hooks

We will go into the details of what goes into a hook later in the chapter, but here we show the basics of how to create a hook. Steps for creating a hook:

1. Log into the server host as root
2. Write the hook script
3. Create an empty hook via `qmgr`
4. Set the attributes of the hook so that it triggers when you want, etc
5. If the hook will use a configuration file:
   a. Write the hook configuration file
   b. Import the hook configuration file
6. Import the hook script into the empty hook. You do not need to restart the MoM, unless it's an exechost_startup hook. Since exechost_startup hooks run only when MoM starts up or is HUPed, if you want the hook to run now, restart or `kill -HUP` the MoM.

3.2 Writing Hooks: Basic Hook Structure

- Import the pbs and sys modules:
  ```
  import pbs
  import sys
  ```
- Use the try... except construction, where you test for conditions in the try block, and accept or reject the event:
  ```
  try:
      …
  except:
  ```
  Consider either rerunning the job or deleting the job inside the `except:` block.
- Treat the SystemExit exception as a normal occurrence, and pass if it occurs:
  ```
  except SystemExit:
      pass
  ```
- Reject the event, or rerun or delete the job, if any other exception occurs:
  ```
  except:
      pbs.event().reject("%s hook failed with %s")
  ```
- If the requestor is the scheduler, and where appropriate, the server or MoM, allow the action to take place:
  ```
  if pbs.event().requestor in ["PBS_Server", "Scheduler", "pbs_mom"]:
      pbs.event().accept()
  ```

The following code fragment is a basic hook skeleton:

```
import pbs
import sys

e=pbs.event()
j=e.job
try:
    if e.requestor in ["Scheduler"]:
    e.accept()
    …

except SystemExit:
    pass

except:
    j.rerun()
    e.reject("%s hook failed with %s. Please contact Admin" % (e.hook_name, sys.exc_info()[:2]))
```

3.3 Example of Simple Hook

Example 3-1: Set job priority

Set a job's priority

```
import pbs
import sys

    e = pbs.event()

try:
    # Get the hook event information and parameters
    # This will be for the 'modifyjob' event type.

    # Ignore requests from scheduler or server
    if e.requestor in ["PBS_Server", "Scheduler"]:
        e.accept()

    # Get the information for the job being queued
    j = e.job
    # Set the job's priority
    j.Priority = 7
    # accept the event
    e.accept()
except SystemExit:
    pass
except:
    e.reject("Failed to set job priority")
```

3.4 Importing Hook Configuration File

If you want your hook to use a configuration file, you can import the configuration file. A configuration file is not required.

Syntax for importing a configuration file:

```
Qmgr: import hook <hook_name> application/x-config <content-encoding>
   <input_config_file>
```

Here, <*content-encoding*> can be "*default*" (7-bit) or "*base64*".

See section 5.1.6, "Using Hook Configuration Files", on page 32.

3.5 Creating and Importing Your Hook

When you "create a hook" using `qmgr`, you're telling PBS that you want it to make you an empty hook object that has no characteristics other than a name. When you "import a hook" using `qmgr`, you're telling PBS which Python script to run when the hook is triggered.

Syntax for creating a hook:

```
Qmgr: create hook <hook name>
```

Simple syntax for importing a hook:

```
Qmgr: import hook <hook name> application/x-python <content-encoding> <input_file>
```

This uses the script named <*input_file*> as the contents of your hook.

- The <*input_file*> must be encoded with <*content-encoding*>.
- The allowed values for <*content-encoding*> are "*default*" (7 bit) and "*base64*".
- <*input_file*> must be locally accessible to both `qmgr` and the batch server.
- A relative path in <*input_file*> is relative to the directory where `qmgr` was executed.
- If your hook already has a content script, then that is overwritten by this import call.
- If the name of <*input_file*> contains spaces, <*input file*> must be quoted.

3.6 Setting Attributes for Your Hook

Hooks have attributes that control their behavior, such as which events trigger the hook, the time to allow the hook to execute, etc. The only attribute you must set for a simple hook is the event(s) that will trigger the hook. Choose your hook type according to the event you want, by looking in Table 5-1, "Hook Trigger Events," on page 31.

Syntax for setting the hook event(s):

```
Qmgr: set hook <hook name> event = <event name>
Qmgr: set hook <hook name> event = "<event name>, <event name>"
```

For more details on setting hook trigger events, see section 5.1.5, "Setting Hook Trigger Events", on page 31.

You can set the rest of the hook's attributes if you wish. To set a hook attribute:

```
Qmgr: set hook <hook name> <attribute> = <value>
```

For a list of all the hook attributes, see section 5.1.9.3, "List of Hook Attributes", on page 37.

4
Hook Basics

4.1 Hook Basics

4.1.1 Accepting or Rejecting Actions

Hooks accept (allow) or reject (prevent) actions, modify input parameters, modify job attributes, environment variables, programs, program arguments, and change internal or external values.

Each action can have zero or more hooks. Each hook must either accept or reject its action. All of an action's hooks are run when that action is to be performed. For PBS to perform an action, all hooks enabled for that action must accept the action. If any hook rejects the action, the action is not performed by PBS. If a hook script doesn't call accept() or reject(), and it doesn't encounter an exception, PBS behaves as if the hook accepts the action. An action is always accepted, unless:

- pbs.event().reject() is called
- An unhandled exception is encountered
- The hook alarm has been triggered due to hook timeout being reached

When PBS executes the hooks for an action, it stops processing hooks at the first hook that rejects the action.

4.1.1.1 Examples of Accepting and Rejecting Actions

Example 4-1: Accepting an action: In this example, userA submits a job to queue Queue1, and the job submission action has two hooks: hook1 disallows jobs submitted by UserB, and hook2 disallows jobs being submitted directly to Queue2. Both hook1 and hook2 accept userA's job submission to Queue1, so the submission goes ahead.

Example 4-2: Rejecting an action: In this example, userA uses the `qmove` command to try to move jobA from Queue1 to Queue2. The job move action has two hooks: hook3 disallows jobs being moved into Queue2, and hook4 disallows userB moving jobs out of Queue1. In this example, hook3 rejects the action, so the move operation is disallowed, even though hook4 would have accepted the action.

4.1.2 When Hooks Run

Each type of event has a corresponding type of hook. The following are the events where you can run hooks, with the hook type:

4.1.2.1 Job-related Hooks that Run Before Execution

Hooks that run before a job is received by an execution host (pre-execution event hooks):

queuejob: Queueing a job

modifyjob: Modifying a job, except when scheduler makes the modification (can also run after job is received by execution host)

movejob: Moving a job

runjob: Just before a job is sent to an execution host

4.1.2.2 Job-related Hooks that Run at Execution Host

Hooks that run after a job is received by an execution host (execution event hooks):

execjob_begin: When a job is received by an execution host, after stagein

execjob_prologue: Just before starting a job's shell

execjob_launch: Just before starting the user's program

execjob_attach: When running `pbs_attach()`

execjob_preterm: Just before killing a job

execjob_epilogue: Just after executing or killing a job, but before job is cleaned up

execjob_end: Just after cleaning a job up

execjob_postsuspend: Just after suspending a job

execjob_preresume: Just before resuming a job

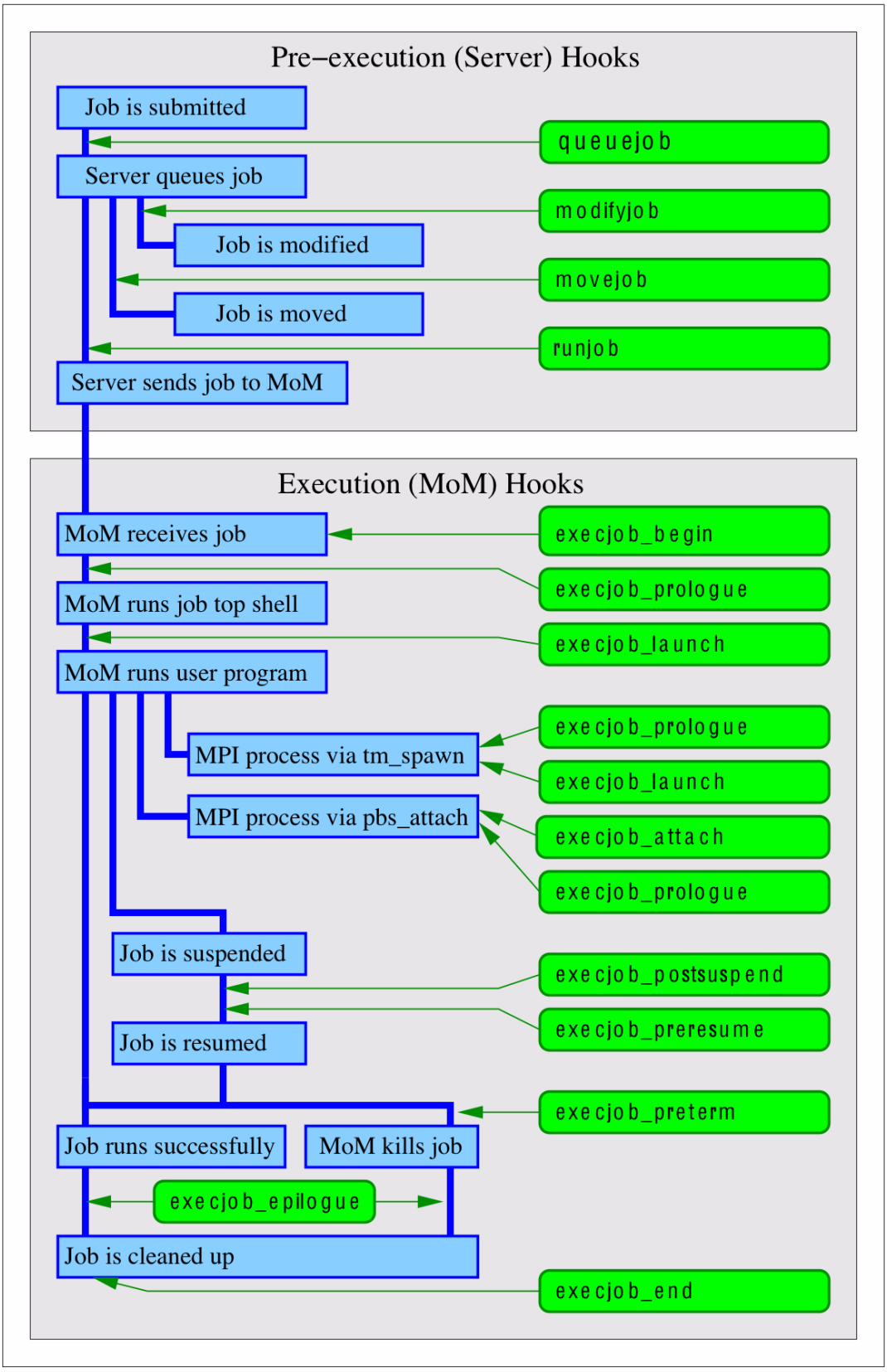

Figure 4-1:Simplified view of trigger timing for job-related hooks

4.1.2.3 Non-job-related Hooks

Hooks that are not directly related to a specific job (non-job event hooks):

resvsub: Submitting a PBS reservation

resv_end: When a PBS reservation ends

provision: Provisioning a vnode

exechost_periodic: Periodically on all execution hosts

exechost_startup: When an execution host is started or receives a HUP

periodic: periodically at the server

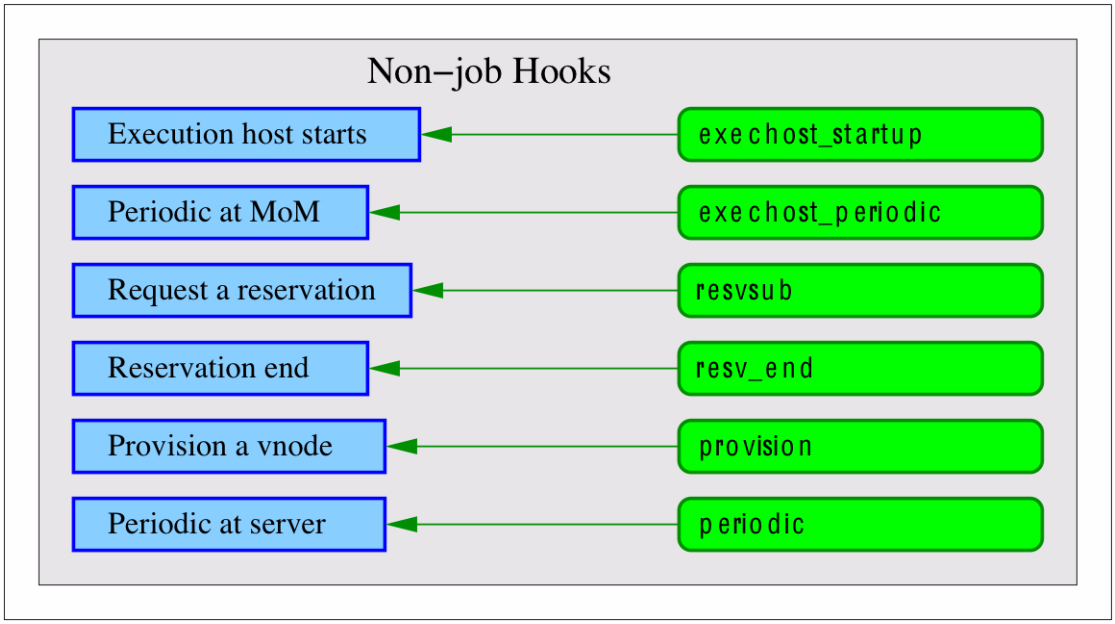

Figure 4-2:Simplified view of trigger timing for non-job-related hooks

4.1.2.4 Each Triggering Event Runs One Hook Instance

Each time an event triggers a hook, the hook runs for that instance of the event. If you have written a hook that runs at job submission, this hook will run for each job that is submitted to this server. Each MoM runs one copy of each of her execution hooks per job. Execution hooks run one per job at the MoM, not one per vnode. For a job that runs on four vnodes of a multi-vnoded machine where all the vnodes are managed by one MoM, where you have written one execution hook, only one instance of the hook runs for that job.

Each time a job goes through a triggering event, PBS runs any relevant hooks. This means that if you run a job, that triggers a runjob hook. If the job is killed and requeued and runs again, the runjob hook runs again.

If the scheduler modifies a job, any modifyjob hooks are not triggered.

When you are using peer scheduling, and a job is pulled from one complex to another, the pulling complex applies its hooks as if the job had been submitted locally, and the furnishing complex applies its movejob hooks. Figure 4-3 shows an example of the hooks that are triggered when a job is moved from a complex containing a movejob hook to a complex containing a queuejob hook.

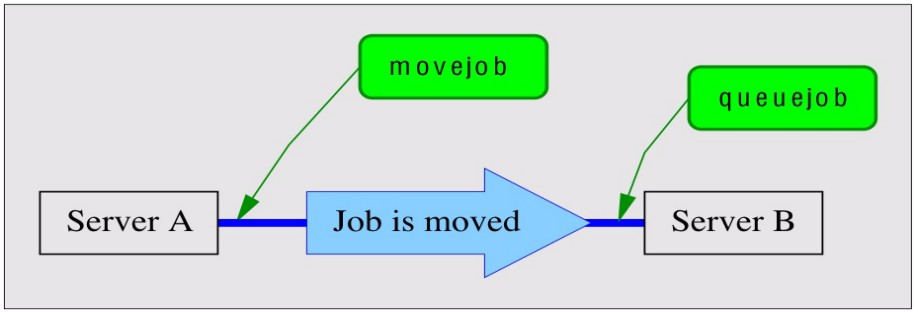

Figure 4-3:Hooks that run when job is moved

4.1.2.5 Execution Event Hook Triggers in Lifecycle of Job

The hooks triggered for an MPI job depend on whether MPI processes are spawned using the PBS TM interface via `tm_spawn()`, or are spawned using `pbs_attach()`. When a process is spawned using `tm_spawn()`, MoM starts the process. When a process uses `pbs_attach()`, `pbs_attach()` starts the process and informs MoM of the process ID.

The following shows where execution event hooks are triggered in the lifecycle of a normal, successful job. We show the timing for hooks on the primary execution host, on a sister vnode where a process is spawned using `tm_spawn()`, and on a sister vnode where a process is spawned using `pbs_attach()`.

Table 4-1: Execution Event Hook Timing

Job Lifecycle	Hooks Are Triggered		
	Primary Execution Host	Sister (`tm_spawn`)	Sister (`pbs_attach`)
Application licenses are checked out			
Any required job-specific staging and execution directories are created			
PBS_JOBDIR and job's jobdir attribute are set to pathname of staging and execution directory			
Files are staged in			
	execjob_begin	execjob_begin	execjob_begin
Job is sent to MoM			
	execjob_prologue If there is no execjob_prologue hook, the prologue script runs		
Server writes accounting log "S" record			
Primary execution host tells sister MoMs they will run job task(s)			

Table 4-1: Execution Event Hook Timing

Job Lifecycle	Hooks Are Triggered		
	Primary Execution Host	Sister (`tm_spawn`)	Sister (`pbs_attach`)
If necessary, MoM creates work directory			
MoM creates temporary directory for job			
MoM sets TMPDIR, JOBDIR, and other environment variables in job's environment			
MoM performs hardware-dependent setup: The job's cpusets are created, ALPS reservations are created			
	execjob_launch		
The job script starts			
Job starts an MPI process on sister vnode			
		execjob_prologue	execjob_prologue
		execjob_launch, for all tasks on this sister	
			execjob_attach, for all tasks on this sister
Job is suspended			
	execjob_postsuspend	execjob_postsuspend	execjob_postsuspend
	execjob_preresume		
		execjob_preresume (if successful on primary MoM)	execjob_preresume (if successful on primary MoM)
Job is resumed			
The job script finishes			
	execjob_epilogue If there is no execjob_epilogue hook, the epilogue script runs	execjob_epilogue	execjob_epilogue
The obit is sent to the server			
Server writes accounting log "E" record			
Any specified file staging out takes place, including `stdout` and `stderr`			
Files staged in or out are deleted			
Any job-specific staging and execution directories are removed			

Table 4-1: Execution Event Hook Timing

Job Lifecycle	Hooks Are Triggered		
	Primary Execution Host	Sister (`tm_spawn`)	Sister (`pbs_attach`)
The job's cpusets are destroyed			
Job files are deleted			
	execjob_end	execjob_end	execjob_end
Application licenses are returned to pool			

4.1.3 Account Under Which Hooks Run

A hook runs as the administrator or as the job owner, depending on the value of the hook's user attribute. If this is set to *pbsadmin*, the hook runs as the Administrator. If this is set to *pbsuser*, the hook runs as the job owner.

4.1.4 Where Hooks Run

Pre-execution event, periodic, provision, and reservation hooks run on the primary server host, or the secondary server host during failover. For most Linux systems, execution event, startup, and exechost_periodic hooks run on the execution host(s). However, on Cray, execution event, startup, and exechost_periodic hooks run on the host(s) where MoM runs.

4.1.5 Permissions and Location for Hook Creation and Modification

Hooks can be created or modified only by the administrator, and only at the hosts on which the primary and secondary servers run.

4.1.6 Failover

The secondary server uses the same filesystem as the primary server. Any hooks created are stored in the same place and are accessible by both servers, whether the primary or the secondary server is running.

When the secondary server takes over for the primary server after the primary's host has gone down or becomes inaccessible, any hooks created at the primary server continue to function under the secondary server.

If the you create a new hook while the secondary server has control, that hook will persist once the primary server takes over: if the primary server comes back up and takes over, hooks created while the secondary server had control continue to function.

4.1.7 What Hooks Cannot Access or Do

- Hooks cannot read or modify anything not presented in the PBS hook interface
- Hooks cannot modify the server or any queues
- Pre-execution event hooks cannot read or set vnode attributes or resources, except that the runjob hook can set the state attribute for any vnode to be used by the job
- Hooks do not have access to other servers besides the default server:
  - Hooks cannot change the destination server to a non-default server
  - Hooks can allow a job submission or a `qmove` to a non-default server, and can change the destination server from a remote server to the default server
- Hooks cannot directly print to `stdout` or `stderr` or read from `stdin`.
- movejob hooks do not run on `pbs_rsub -Wqmove=<job ID>`

4.1.8 What Hooks Should Not Do

- Hooks should not edit configuration files directly, meaning hooks should not edit the following:

  `PBS_HOME/sched_priv/sched_config`

  `PBS_HOME/sched_priv/fairshare`

  `PBS_HOME/sched_priv/dedicated`

  `PBS_HOME/sched_priv/holidays`

  `/etc/pbs.conf`

  `PBS_HOME/server_priv/resourcedef`

  `PBS_HOME/mom_priv/config`
- Hooks should not execute PBS commands

4.2 Viewing Hook Information

4.2.1 Listing Hooks

To list one hook and its attributes on the current server:

```
Qmgr: list hook <hook name>
```

To list all hooks and their attributes on the current server:

```
Qmgr: list hook
```

4.2.2 Viewing Hook Contents

To view the contents of a hook, export the hook's contents:

qmgr -c "export hook <hook_name> <content-type> <content-encoding>" > [<output_file>]

You cannot export the contents of a built-in hook.

4.2.3 Printing Hook Creation Commands

To view the commands to create one hook, including any configuration file:

```
Qmgr: print hook <hook name>
```

To view the commands to create all the hooks on the default server, including their configuration files:

```
Qmgr: print hook
```

or

```
qmgr -c "print hook"
```

For example, to see the commands used to create hook1 and hook2:

```
# qmgr -c "print hook"
create hook hook1
import hook hook1 application/x-python base64 - cHJpbnQgImhlbGxvLCB3b3JsZCIK

set hook hook1 event=movejob
set hook hook1 alarm=10
set hook hook1 order=5
create hook hook2
import hook hook2 application/x-python base64 - servaJLSDFSESF

set hook hook2 event=queuejob
set hook hook2 alarm=15
set hook hook2 order=60
…
```

4.2.4 Re-creating Hooks

To re-create a hook, including its configuration file, you feed `qmgr` hook descriptions back into `qmgr`. These hook descriptions are the same information that `qmgr` prints out. To print out the statements needed to recreate a hook, use the `print hook` or `print hook <hook name>` `qmgr` commands.

For example, to save information for hook1 and hook2:

```
# qmgr -c "print hook" > hookInfo
```

To re-create hook1 and hook2, with their configuration files:

```
# qmgr < hookInfo
```

4.3 Restarting the Python Interpreter

PBS keeps track of the number of hooks serviced, the number of objects created, and the time since the Python interpreter was last restarted. You can set a limit for the number of hooks created in the python_restart_max_hooks server attribute, a limit for the number of objects created in the python_restart_max_objects server attribute, and a limit for the minimum time interval at which to restart the Python interpreter in the python_restart_min_interval server attribute.

python_restart_max_hooks

The maximum number of hooks to be serviced before the Python interpreter is restarted. If this number is exceeded, and the time limit set in python_restart_min_interval has elapsed, the Python interpreter is restarted.

Type: integer

Default: *100*

Python type: int

python_restart_max_objects

The maximum number of objects to be created before the Python interpreter is restarted. If this number is exceeded, and the time limit set in python_restart_min_interval has elapsed, the Python interpreter is restarted.

Type: integer

Default: *1000*

Python type: int

python_restart_min_interval

The minimum time interval before the Python interpreter is restarted. If this interval has elapsed, and either the maximum number of hooks to be serviced (set in python_restart_max_hooks) has been exceeded or the maximum number of objects to be created (set in python_restart_max_objects) has been exceeded, the Python interpreter is restarted.

Type: integer seconds or [[HH:]MM:]SS

Default: *30*

Python type: pbs.duration

4.4 Attributes and Parameters Affecting Hooks

- Each hook's attributes affect the behavior of that hook. Hook attributes are listed in "Hook Attributes" on page 352 of the PBS Professional Reference Guide.
- The $reject_root_scripts MoM parameter controls whether MoM accepts new hook scripts.
- The server attributes that control when the Python interpreter is restarted are listed in Chapter 4, "Restarting the Python Interpreter", on page 23.

4.5 Python Modules and PBS

When you run a hook inside `pbs_python`, the hook has access to modules here:

- In `PBS_EXEC/python`
- In `PBS_EXEC/lib/python/altair`

Your hook can use other modules if you specify them in the hook.

The PBS_EXEC/python modules are in the following directories:

```
PBS_EXEC/python/lib/python36.zip
PBS_EXEC/python/lib/python3.6
PBS_EXEC/python/lib/python3.6/plat-linux2
PBS_EXEC/python/lib/python3.6/lib-tk
PBS_EXEC/python/lib/python3.6/lib-dynload
PBS_EXEC/python/lib/python3.6/site-packages
```

4.5.1 Python Module Caveats

In order to use PBS_EXEC/python/lib/python3.6/site-packages, you must first call the following:

```
import site
site.main()
```

4.5.2 Modifying Python Modules

If you need to use other modules, we recommend that you put the modules in a different directory from PBS_EXEC/lib/python.

To use other modules besides the ones in PBS_EXEC/lib/python, specify the path in the hook.

If you are adding modules that are not in PBS_EXEC/lib/python, you can do this:

```
import sys
if '/usr/lib64/python3.6' not in sys.path:
    sys.path.append('/usr/lib64/python3.6')
import pbs
```

If you need to include user-defined paths ahead of the default modules, you can do the following. For example, if you put a module in /usr/lib64/python3.6, in /usr/local/lib64/python3.6, and in /usr/local/lib64/custom/python and you want to load them before the PBS-provided modules, add them to your hook this way:

```
import sys
my_paths = ['/usr/lib64/python3.6',
'/usr/local/lib64/python3.6',
'/usr/local/lib64/custom/python']
for my_path in my_paths:
    if my_path not in sys.path:
        sys.path.insert(0, my_path)
import pbs
```

4.5.2.1 Caveats for Modifying Python Modules

If you change a Python module in a pre-execution event hook (queuejob, movejob, modifyjob, runjob), you must restart the server in order to use the new module, because "import" is cached.

4.5.3 List of Modules in pbs_python

The following are the modules that are available via PBS_EXEC/lib/python:

Table 4-2: Modules in pbs_python

Module Name			
cElementTree	handler	pulldom	stringprep
collections	handlers	pydoc	symbol
config	itertools	re	time
copy	linecache	repr	token
copy_reg	math	saxutils	tokenize
cStringIO	md5	select	types
datetime	minicompat	sre	UserDict
dis	minidom	sre_compile	UserList
domreg	NodeFilter	sre_constants	UserString
ElementInclude	opcode	sre_parse	uuid
ElementPath	operator	stat	xmlbuilder
ElementTree	os	statvfs	xmlreader
expat	parser	string	zipfile
expatbuilder	pkgutil	StringIO	_exceptions
expatreader	posixpath	stringold	_socket

4.6 See Also

For a description of the PBS hook APIs, see the PBS Professional Programmer's Guide. Each PBS object's attribute's Python type is listed in its description in "Attributes" on page 279 of the PBS Professional Reference Guide. For example, "Server Attributes" on page 283 of the PBS Professional Reference Guide lists the Python type for the job_sort_formula server attribute.

The following man pages and equivalent sections contain useful information:

Table 4-3: See Also

Man Page	Guide Section
pbs_module(7B)	section 8.3.1, "The pbs Module", on page 112 of the PBS Professional Programmers Guide
pbs_stathook(3B)	section 8.4.2, "The pbs_stathook() API", on page 118 of the PBS Professional Programmers Guide
pbs_hook_attributes(7B)	"Hook Attributes" on page 352 of the PBS Professional Reference Guide
pbs_job_attributes(7B)	"Job Attributes" on page 330 of the PBS Professional Reference Guide

Table 4-3: See Also

Man Page	Guide Section
pbs_server_attributes(7B)	"Server Attributes" on page 283 of the PBS Professional Reference Guide
pbs_queue_attributes(7B)	"Queue Attributes" on page 313 of the PBS Professional Reference Guide
pbs_node_attributes(7B)	"Vnode Attributes" on page 322 of the PBS Professional Reference Guide
qmgr(1B)	"qmgr" on page 151 of the PBS Professional Reference Guide
qsub(1B)	"qsub" on page 215 of the PBS Professional Reference Guide
qmove(1B)	"qmove" on page 174 of the PBS Professional Reference Guide
qalter(1B)	"qalter" on page 129 of the PBS Professional Reference Guide
pbs_rsub(1B)	"pbs_rsub" on page 95 of the PBS Professional Reference Guide
pbs_manager(3B)	"pbs_manager" on page 41 of the PBS Professional Programmers Guide

5 Creating and Configuring Hooks

5.1 Creating and Configuring Hooks

In this chapter we describe how to create and configure site-defined hooks. For information about operating on built-in hooks, see Chapter 7, "Built-in Hooks", on page 155.

5.1.1 Introduction to Creating and Configuring Hooks

Hooks can only be created, run, or modified by the Administrator, and only on the host(s) on which the primary or secondary server runs.

You create hooks using the `qmgr` command to create, delete, import, or export the hook. The `qmgr` command operates on the *hook* object.

Syntax for operating on hooks:

qmgr -c "<command> hook <hook name> [<arguments to command>]"

where

command is *create, delete, set, unset, list, print, import, export*

5.1.1.1 Hook Name Restrictions

- Each hook must have a unique name.
- The name must be alphanumeric, and start with an alphabetic character.
- The name must not begin with "*PBS*".
- The name of a hook can be a legal PBS object name, such as the name of a queue.
- Hook names are case-sensitive.

5.1.2 Overview of Creating and Configuring a Hook

The following is an overview of the steps to create a hook. Each step is described in the following sections. You must be logged into the primary or secondary server host as root.

1. Use the `create hook` `qmgr` command to create an empty hook with the name you specify
2. Import the contents of a hook script into the hook
3. Set the hook's trigger event
4. If the hook will use a configuration file, write and import the configuration file
5. Set the hook's order of execution, if there is another hook for the same event
6. Optionally, set the hook's timeout
7. Make sure that the $reject_root_scripts MoM configuration parameter is set to *False* on all execution hosts where you want hooks to run. The default for this parameter is *False*.

You do not need to restart the MoM.

5.1.2.1 Example of Creating and Configuring a Hook

Create the hook:

```
Qmgr: create hook hook1
```

Import the hook script named `hook1_script.py` into the hook:

```
Qmgr: import hook hook1 application/x-python default /hooks/hook1_script.py
```

Make hook1 a queuejob hook:

```
Qmgr: set hook hook1 event = queuejob
```

Make this the second queuejob hook:

```
Qmgr: set hook hook1 order = 2
```

Set the hook to time out after 60 seconds:

```
Qmgr: set hook hook1 alarm = 60
```

Look at the $reject_root_scripts MoM configuration parameter where you want the hook to run, and make sure it is set to *False*.

5.1.3 Creating Empty Hooks

To create a hook, use the `create hook` command in `qmgr` to create an empty hook with the name you specify:

The create hook `qmgr` command creates an empty hook.

Syntax for creating a hook:

Qmgr: create hook <hook name>

5.1.3.1 Example of Creating an Empty Hook

To create the hook named "hook1", specify a filename, for example `"/hooks/hook1.py"`, that is locally accessible to `qmgr` and the PBS server:

```
Qmgr: create hook hook1
```

5.1.4 Deleting Hooks

To delete a hook, you use the `delete hook` command in `qmgr`.

Syntax for deleting a hook:

```
Qmgr: delete hook <hook name>
```

5.1.4.1 Example of Deleting a Hook

To delete hook hook1:

```
Qmgr: delete hook hook1
```

5.1.5 Setting Hook Trigger Events

To set the events that will cause a hook to be triggered, use the `set hook <hook name> event` command in `qmgr`. You can add triggering events to a hook.

To set one event:

```
Qmgr: set hook <hook name> event = <event name>
```

Designate triggers for a hook by setting <*event name*> to one of the following events:

Table 5-1: Hook Trigger Events

Action (Event)	Event Name
Accepting job into queue	*queuejob*
Modifying job, except when scheduler makes modification	*modifyjob*
Moving job	*movejob*
Before a job is sent to an execution host	*runjob*
Periodically on server host	*periodic*
When a job is received by an execution host, after stagein	*execjob_begin*
When `pbs_attach()` is called	*execjob_attach*
Just before executing a job's top shell	*execjob_prologue*
Just before executing the user's program	*execjob_launch*
Just after suspending a job	*execjob_postsuspend*
Just before resuming a job	*execjob_preresume*
Just after executing or killing a job, but before job is cleaned up	*execjob_epilogue*
Just before killing a job	*execjob_preterm*
Just after cleaning up a job that has finished or been killed	*execjob_end*
When an execution host starts up or receives a HUP	*exechost_startup*
Periodically on all execution hosts	*exechost_periodic*

Table 5-1: Hook Trigger Events

Action (Event)	Event Name
Provisioning a vnode	*provision*
Submitting reservation	*resvsub*
When reservation ends	*resv_end*

To add an event:

Qmgr: set hook <hook name> event += <event name>

For a detailed description of each event, see section 6.3.1, “Event Types”, on page 87.

5.1.5.1 Example of Setting Hook Trigger Events

To set an event that will cause hook "UserFilter" to be triggered:

```
Qmgr: set hook UserFilter event = queuejob
```

Add another event:

```
Qmgr: set hook UserFilter event += modifyjob
```

Set two events at once:

```
Qmgr: set hook UserFilter event = "queuejob, modifyjob"
```

You must enclose the value in double quotes if it contains a comma.

5.1.6 Using Hook Configuration Files

You can customize the behavior of a hook by providing a configuration file for the hook. You write the hook so that it reads and acts on its configuration file. Hooks are not required to use configuration files. A configuration file can contain whatever information is useful to the hook. A configuration file is just a file of whatever information you want; the way the hook reads and uses the contents of a configuration file is up to you. The hook itself processes the configuration file.

5.1.6.1 Format of Configuration File

PBS supports several file formats for configuration files. The format of the file is specified in its suffix. Formats can be specified in any of the following ways:

- `.ini`
- `.json`
- `.txt` (generic, no special format)
- `.xml`
- No suffix: treat the input file as if it is a `.txt` file
- The dash (-) symbol: configuration file content will be taken from `STDIN`. The content is treated as if it is a `.txt` file.

For example, to import a configuration file in `.json` format:

```
# qmgr -c "import hook <hook_name> application/x-config default input_file.json"
```

5.1.6.2 Importing Configuration File

To provide a configuration file for a hook, you import the configuration file into the hook. The import command is the same as for a hook, except that you set <*content-type*> to "*application/x-config*". Syntax for importing a configuration file:

```
Qmgr: import hook <hook_name> application/x-config <content-encoding>
   <input_config_file>
```

or

 # qmgr -c "import hook <hook_name> application/x-config <content-encoding> <input_config_file>"

where <*content-encoding*> is "*default*" (7-bit) or "*base64*".

This uses the contents of <*input_config_file*> or `stdin` (-) as the contents of configuration file for hook <*hook_name*>.

- The <*input_config_file*> or `stdin` (-) data must have a format <*content-type*> and must be encoded with <*content-encoding*>.
- The allowed values for <*content-encoding*> are "*default*" (7bit) and "*base64*".
- If the source of input is `stdin` (-) and <*content-encoding*> is "*default*", then `qmgr` expects the input data to be terminated by EOF.
- If the source of input is `stdin` (-) and <*content-encoding*> is "base64", then `qmgr` expects input data to be terminated by a blank line.
- <*input_config_file*> must be locally accessible to both `qmgr` and the requested batch server.
- A relative path <*input_config_file*> is relative to the directory where `qmgr` was executed.
- If a hook already has a configuration file, then that is overwritten by this import call.
- If <*input_config_file*> name contains spaces, <*input_config_file*> must be quoted.
- There is no restriction on the size of the hook configuration file.

5.1.6.2.i Examples of Importing Configuration Files

Importing a Python configuration file:

```
# qmgr -c 'import hook hook1 application/x-config default hello.py'
```

Importing a JSON configuration file:

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

5.1.6.3 Exporting Configuration Files

To edit or display the content of a hook configuration file associated with the hook named <hook_name>, export the configuration file. Use the export command:

qmgr -c "export hook <hook_name> application/x-config default" > <output file>

5.1.6.4 How Hooks Find Configuration Files

There are two ways to retrieve a configuration file in a hook.

- PBS puts the configuration file in a location that can be read by the hook, and sets the PBS_HOOK_CONFIG_FILE environment variable to that path. Your hook script can use this path:

```
import os
import ConfigParser

if "PBS_HOOK_CONFIG_FILE" in os.environ:
    config_file = os.environ["PBS_HOOK_CONFIG_FILE"]
    config = ConfigParser.RawConfigParser()
    config.read(os.environ["PBS_HOOK_CONFIG_FILE")
```

- Your hook can use the pbs.hook_config_filename variable, which contains the path to the configuration file. See "pbs.hook_config_filename" on page 135.

  If there is no configuration file, this variable returns *None*.

5.1.6.5 Changing a Hook Configuration File

To replace the content of a hook configuration file, export the file, edit it, and issue another "`import`" hook command with updated <*input_config_file*> content.

5.1.6.6 Validation and Errors

- PBS pre-validates <*input_config_file*> according to its file format, and returns an error in `qmgr`'s `STDERR` if validation fails. For example:

```
# qmgr -c "import hook submit application/x-config default file.json"
"Failed to validate config file, hook 'submit' config file not overwritten"
```

- If the input configuration file given is of unrecognized suffix, the following message is returned in `qmgr`'s `STDERR`.

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

- If you import a configuration file and PBS cannot open the file because it is non-existent, has permission problems, or has another system-related error, the following error message is printed in `STDERR`:

```
"qmgr: hook error: failed to open <filename> - <error message>"
```

- If you attempt to export a hook configuration file, but the file is unwriteable due to ownership or permission problems, the following error message is printed to `STDERR`:

```
"qmgr: hook error: <output_file> permission denied"
```

5.1.7 Importing Hooks

To import a hook, you import the contents of a hook script into the hook. You must specify a filename that is locally accessible to `qmgr` and the PBS server.

Syntax for importing a hook:

`Qmgr: import hook <hook_name> <content-type> <content-encoding> {<input_file>|-}`

This uses the contents of <*input_file*> or stdin (-) as the contents of hook <*hook_name*>.

- The <*input_file*> or stdin (-) data must have a format <*content-type*> and must be encoded with <*content-encoding*>.
- For script files, the only <*content-type*> currently supported is "*application/x-python*".
- The allowed values for <*content-encoding*> are "*default*" (7 bit) and "*base64*".
- If the source of input is stdin (-) and <*content-encoding*> is "*default*", then qmgr expects the input data to be terminated by EOF.
- If the source of input is stdin (-) and <*content-encoding*> is "*base64*", then qmgr expects input data to be terminated by a blank line.
- <*input_file*> must be locally accessible to both qmgr and the requested batch server.
- A relative path in <*input_file*> is relative to the directory where qmgr was executed.
- If a hook already has a content script, then that is overwritten by this import call.
- If the name of <*input_file*> contains spaces, <*input file*> must be quoted.
- There is no restriction on the size of the hook script.

5.1.7.1 Examples of Importing Hooks

Example 5-1: Given a Python script in ASCII text file "hello.py", this makes its contents into the script contents of hook1:

```
#cat hello.py
import pbs
pbs.event().job.comment="Hello, world"
# qmgr -c 'import hook hook1 application/x-python default hello.py'
```

Example 5-2: Given a base64-encoded file "hello.py.b64", qmgr unencodes the file's contents, and then makes this script the contents of hook1:

```
# cat hello.py.b64
cHJpbnQgImhlbGxvLCB3b3JsZCIK
# qmgr -c 'import hook hook1 application/x-python base64 hello.py.b64'
```

Example 5-3: Read stdin for text containing data until EOF, and make this into the script contents of hook1:

```
# qmgr -c 'import hook hook1 application/x-python default -'
import pbs
pbs.event().job.comment="Hello from stdin"
Ctrl-D
```

Example 5-4: Read stdin for a base64-encoded string of data terminated by a blank line. PBS unencodes the data and makes this script the contents of hook1.

```
# qmgr -c 'import hook hook1 application/x-python base64 -'
cHJpbnQgImhlbGxvLCB3b3JsZCIK
Ctrl-D
```

5.1.8 Exporting Hooks

Syntax for exporting a hook:

qmgr -c "export hook <hook_name> <content-type> <content-encoding>" > <output_file>

This dumps the script contents of hook <*hook_name*> into <*output_file*>, or stdout if <*output_file*> is not specified.

- The resulting <*output_file*> or stdout data is of <*content-type*> and <*content-encoding*>.
- The only <*content-type*> currently supported for scripts is "*application/x-python*".
- The allowed values for <*content-encoding*> are "*default*" (7bit) and "*base64*".
- <*output_file*> must be a path that can be created by qmgr.
- Any relative path in <*output_file*> is relative to the directory where qmgr was executed.
- If <*output_file*> already exists it is overwritten. If PBS is unable to overwrite the file due to ownership or permission problems, then an error message is displayed in stderr.
- If the <*output_file*> name contains spaces, <*output file*> must be enclosed in quotes.

5.1.8.1 Examples of Exporting Hooks

Example 5-5: Dumps hook1's script contents directly into the file "hello.py.out":

```
# qmgr -c "export hook hook1 application/x-python default" > hello.py
# cat hello.py
import pbs
pbs.event().job.comment="Hello, world"
```

Example 5-6: To dump the script contents of a hook 'hook1' into a file in "\My Hooks\hook1.py":

```
qmgr -c "export hook hook1 application/x-python default" > "\My Hooks\hook1.py"
```

Example 5-7: Dump hook1's script contents base64-encoded into a file called "hello.py.b64":

```
# qmgr -c "export hook hook1 application/x-python base64" > hello.py.b64
# cat hello.py.b64
cHJpbnQgImhlbGxvLCB3b3JsZCIK
```

Example 5-8: Dump hook1's script contents directly to stdout:

```
# qmgr -c "export hook hook1 application/x-python default"
import pbs
pbs.event().job.comment="Hello, world"
```

Example 5-9: Dump hook1's script contents base64-encoded into stdout:

```
# qmgr -c "export hook hook1 application/x-python base64"
 cHJpbnQgImhlbGxvLCB3b3JsZCIK
```

5.1.9 Setting and Unsetting Hook Attributes

You configure a hook using the qmgr command to set or unset its attributes. An unset hook attribute takes the default value for that attribute.

Hook attributes can be viewed via qmgr:

```
Qmgr: list hook <hook name>
```

To set a hook attribute:

```
Qmgr: set hook <hook name> <attribute> = <value>
```

To unset a hook attribute:

```
Qmgr: unset hook <hook name> <attribute>
```

For example, to unset hook1's alarm attribute, causing its value to revert to its default value:

```
Qmgr: unset hook hook1 alarm
```

This causes hook1's alarm to revert to the default of *30 seconds*.

5.1.9.1 Caveats for Setting Hook Attributes

You cannot set the type attribute for a built-in hook.

5.1.9.2 Using the fail_action Hook Attribute

The fail_action hook attribute is a string_array and can take on multiple values:

None
: No action is taken.

offline_vnodes
: After unsuccessful hook execution, offlines the vnodes managed by the MoM executing the hook. Can be set for execjob_begin, execjob_prologue, and exechost_startup hooks only.

clear_vnodes_upon_recovery
: After successful hook execution, clears vnodes previously offlined via *"offline_vnodes"* fail action. Can be set for exechost_startup hooks only.

scheduler_restart_cycle
: After unsuccessful hook execution, restarts scheduling cycle. Can be set for execjob_begin and execjob_prologue hooks only.

Default value: "*None*"

If you specify offlining or clearing vnodes in addition to restarting the scheduler, the scheduler restart happens last. The order of the values is not important.

To set the attribute:

```
# qmgr -c "set hook <hook_name> fail_action = <fail_action value>"
# qmgr -c "set hook <hook_name> fail_action = '<fail_action value>,<fail_action value>'"
```

To add a value to the list of values:

```
# qmgr -c "set hook <hook_name> fail_action += <fail_action value>"
```

To remove a value from the list of values:

```
# qmgr -c "set hook <hook_name> fail_action -=  <fail_action value>"
```

To find out what the values are:

```
# qmgr -c "list hook <hook_name> fail_action"
<hook_name>
    fail_action = <fail_action value>
```

To unset the attribute:

```
# qmgr -c "unset hook <hook_name> fail_action"
```

See section 5.2.12.4, “Offlining and Clearing Vnodes Using the fail_action Hook Attribute”, on page 66 and section 5.2.6, “Restarting Scheduler Cycle After Hook Failure”, on page 63.

5.1.9.3 List of Hook Attributes

Hook attributes are listed in “Hook Attributes” on page 352 of the PBS Professional Reference Guide.

5.1.10 Enabling and Disabling Hooks

A hook is either *enabled*, and will run when its action happens, or is *disabled*, and will not run. Hooks are enabled by default.

Syntax to enable a hook:

```
Qmgr: set hook <hook name> enabled=True
```

Syntax to disable a hook:

```
Qmgr: set hook <hook name> enabled=False
```

5.1.10.1 Example of Enabling and Disabling Hooks

To enable hook1:

```
Qmgr: set hook hook1 enabled=True
```

To disable hook1:

```
Qmgr: set hook hook1 enabled=False
```

5.1.11 Setting the Relative Order of Hook Execution

When there are multiple hooks of the same type for one action, you may wish to specify the order in which these hooks are run. The order in which the hooks for an action are run is determined by each hook's order attribute. Hooks with a lower value for order will run before hooks with a higher value. To set the relative order in which the hooks for an action will be run, set each hook's order attribute.

Syntax:

```
Qmgr: set hook <hook name> order=<ordering>
```

<*ordering*> is an integer. Hooks with lower values for <*ordering*> run before those with higher values; a hook with *order=1* runs before a hook with *order=2*.

Valid values for order:

- Built-in hooks can be from *-1000* to *2000*
- Site hooks can be from *1* to *1000*

The order in which hooks of the same type for unrelated actions execute is undefined. For example, there are two queuejob hooks, Hook1 and Hook2, and userA submits jobA and userB submits jobB. While Hook1 always runs before Hook2 for the same job, the order of execution is undefined for different jobs. So the order could be:

Hook1 (jobB)

Hook1 (jobA)

Hook2 (jobA)

Hook2 (jobB)

5.1.11.1 Example of Setting Relative Order of Hook Execution

To set hookA to run first and hookB to run second:

```
Qmgr: set hook hookA order=2
Qmgr: set hook hookB order=5
```

5.1.11.2 Caveats for Setting Relative Order of Hooks

The order attribute is ignored for exechost_periodic and periodic hooks.

5.1.12 Setting Hook Timeout

You may wish to specify how long PBS should wait for a hook to run. Execution for each hook times out after the number of seconds specified in the hook's alarm attribute. If the hook does not run in the specified time, PBS aborts the hook and rejects the hook's action.

Syntax:

```
Qmgr: set hook <hook name> alarm=<timeout>
```

<*timeout*> is the number of seconds PBS will allow the hook to run.

When a hook timeout is triggered, the hook script gets a Python KeyboardInterrupt from the PBS server. The server logs show:

```
06/17/2008 17:57:16;0001;Server@host2;Svr;Server@host2;PBS server internal error (15011) in
    Python script received a KeyboardInterrupt, <type 'exceptions.KeyboardInterrupt'>
```

5.1.12.1 Example of Setting Hook Timeout

To set the number of seconds that PBS will wait for hook hook1 to execute before aborting the hook and reject the action:

```
Qmgr: set hook hook1 alarm=20
```

5.1.13 Setting Hook Interval (Frequency)

You can specify the interval at which a periodic hook runs. You can do this only for hooks whose event type is exechost_periodic or periodic.

Syntax:

```
Qmgr: set hook <hook name> freq=<interval>
```

<*interval*> is the number of seconds elapsed between calls to this hook.

5.1.13.1 Example of Setting Hook Interval (Frequency)

To set the number of seconds between calls to an exechost_periodic or periodic hook:

```
Qmgr: set hook hook1 freq=200
```

5.1.14 Setting Hook User Account

You can specify the account under which a hook runs.

Syntax:

```
Qmgr: set hook <hook name> user=<pbsadmin | pbsuser>
```

pbsadmin specifies that the hook runs as root or as administrator.

pbsuser specifies that the hook runs as the job owner.

You can specify that a hook runs as the job owner only for execjob_prologue, execjob_epilogue, and execjob_preterm hooks.

If you do not set the account, it defaults to *pbsadmin*.

5.1.14.1 Example of Setting Hook User Account

To set the account under which a hook runs:

```
Qmgr: set hook hook1 user=pbsuser
```

5.2 Writing Hook Scripts to Operate on PBS Elements

5.2.1 How We Define and Refer to Objects and Methods

5.2.1.1 Scope of Object or Method

When we define an object or method, we show the scope of the object or method. For example, the scope of a job is the pbs module, so we call it a *pbs.job*, and a server has the same scope, so it is a *pbs.server*. Similarly, the logjobmsg() method has module-wide scope, and is defined as *pbs.logjobmsg()*.

However, the scope of a job ID object is the job, not the module, so it is defined as a *pbs.job.id*, and the scope of the job's is_checkpointed() method is the job, so it is defined as *pbs.job.is_checkpointed()*.

5.2.1.2 Referring to Objects

In a hook, you refer to the triggering event using pbs.event(). In a hook that is triggered by a job-related event, such as a movejob or execjob_begin hook, the event has an associated pbs.job object representing the job that triggered the event, and you refer to it using *pbs.event().job*. You can refer to members of that job object using *pbs.event().job.<member>*. For example, to refer to the ID of the job associated with the event, you use *pbs.event().job.id*. To use the is_checkpointed() method on the job associated with the event, you use *pbs.event().job.is_checkpointed()*. You can use shortcuts:

```
e = pbs.event()
j = e.job
c = j.is_checkpointed()
```

5.2.1.3 How to Retrieve Objects: Event vs. Server

Each event has access to specific objects, listed in Table 6-3, “Using Event Object Members in Events,” on page 108. You can manipulate many of these objects through the event. To retrieve the job that triggered an event, you refer to it this way: *pbs.event().job*.

The server has read access to all objects in the pbs module. You refer to these objects through the server. For example, to retrieve a job whose ID is "*1234*" through the server, you use *pbs.server().job("1234")*. **You cannot manipulate an object that is retrieved through the server**.

5.2.1.3.i Retrieving Jobs

The way you retrieve a job determines how much access you have to that job. You can retrieve a job either through the event, via pbs.event().job, or through the server, via pbs.server().job().

If you retrieve a job through an event, the event gives you the job itself, represented as an object. You can see and alter some job attributes for an event-retrieved job object. To get the job object representing the job associated with the current event, on which you can operate, use pbs.event().job. We show which hooks can see and set each job attribute in Table 5-6, “Job Attributes Readable & Settable via Events,” on page 55.

However, if you retrieve a job through the server, the server gives you an instantiated job object that contains a **copy** of the job. You cannot set any job attributes for a server-retrieved job object, and trying to operate on a server-retrieved copy of the job causes an exception. In order to get read-only information about a particular job with ID <*id*>, use *pbs.server().job('<job ID>')*. This returns a read-only copy of the job.

You can see all of the attributes for a server-retrieved job object, except in a queuejob hook. In a queuejob hook, the event gives you the job as it exists before the server sees it, but the server cannot retrieve it, because the job has not yet made it to the server.

5.2.1.3.ii Retrieving Vnodes

Vnode objects behave like job objects. If you retrieve a vnode object through an event, via pbs.event().vnode_list[], you can see some of the vnode's attributes, and set vnode attributes. We show which hooks can see and set each vnode attribute in Table 5-7, "Vnode Attributes Readable & Settable via Events," on page 57.

If you retrieve a vnode object through the server, via pbs.server().vnode(), you have a **copy** of the vnode, and you can see all of the vnode's attributes, but you cannot set any of them.

5.2.1.3.iii Retrieving Queues

You can retrieve queues through the server only, using *pbs.server().queue("<queue name>")*, or using *pbs.server().queues()*. You cannot make any changes to queue objects in hooks. These are read-only.

You can change a job's destination queue, but only to a queue at the local server. Hooks have access only to the local server. Hooks can allow a job submission to a remote server, but they **cannot** specify a remote server. See section 5.3.9.1, "Local Server Only", on page 72. Hooks can specify the destination queue at a local server for a queuejob or movejob event, whether the original destination queue was at the local server or a remote server.

To specify a destination queue at the local server:

```
pbs.event().job.queue = pbs.server().queue("<local_queue>")
```

Do not specify a queue at a remote server in a hook script.

5.2.1.3.iv Retrieving Reservations

In order to get information about a reservation being created in a resvsub event, use *pbs.event().resv*. pbs.server() cannot return information about the reservation, because the reservation has not yet been created.

5.2.2 Recommended Hook Script Structure

5.2.2.1 Catch Exceptions

Your hook script should catch all exceptions except for SystemExit. We recommend that you catch exceptions via try... except and accompany them with a call to pbs.event().reject().

It is helpful if it displays a useful error message in the `stderr` of the command triggering the hook. The error message should show the type of the error and should describe the error.

Here is the recommended script structure:

```
import pbs
import sys

try:
    …

except SystemExit:
    pass

except:
    pbs.event().job.rerun()
    pbs.event().reject("%s hook failed with %s. Please contact \
        Admin" % (pbs.event().hook_name, sys.exc_info()[:2]))
```

5.2.2.1.i Example of Catching Exceptions

This example shows how a coding error in the hook is caught with the except statement, and an appropriate error message is generated. In line 7, the statement k=5/0 generates a divide-by-zero error. The hook script is designed to reject interactive jobs that are submitted to queue "nointer".

```
import pbs
import sys

try:

    batchq = "nointer"
    e = pbs.event()
    j = e.job
    k = 5/0
    if j.queue and j.queue.name == batchq and j.interactive:
        e.reject("Can't submit an interactive job in '%s' queue" %
        (batchq))

except SystemExit:

    pass

except:
    e.reject("%s hook failed with %s. Please contact Admin" % (e.hook_name, sys.exc_info()[:2]))
```

The hook is triggered:

```
% qsub job.scr
qsub: c1 hhook failed with (<class 'ZeroDivisionError'>, ZeroDivisionError('division by zero',)).
    Please contact Admin
```

5.2.2.1.ii Table of Exceptions

The following exceptions may be raised when using the pbs.* objects:

Table 5-2: Exceptions Raised When Using pbs.* Objects

Object	Exception
pbs.BadAttributeValueError	Raised when setting member value of a pbs.* object to an invalid value.
pbs.BadAttributeValueTypeError	Raised when setting member value of a pbs.* object to an invalid type.
pbs.BadResourceValueError	Raised when setting resource value of a pbs.* object to an invalid value.
pbs.BadResourceValueTypeError	Raised when setting resource value of a pbs.* object to an invalid type.
pbs.EventIncompatibleError	Raised when referencing a nonexistent member in pbs.event. Example: calling pbs.event().resv for pbs.event().type of pbs.QUEUEJOB
pbs.UnsetAttributeNameError	Raised when referencing a non-existent member name of a pbs.* object.
pbs.UnsetResourceNameError	Raised when referencing a non-existent resource name of a pbs.* object.
SystemExit	1. Raised when pbs.event().reject() terminates hook execution. 2. Raised when pbs.event().accept() terminates hook execution.

5.2.3 Hook Alarm Calls and Unhandled Exceptions

- An execjob_begin or exechost_startup hook can cause a failure action to take place when the hook script fails due to an alarm call or an unhandled exception. Otherwise, the following happens:

If a pre-execution event or execution event hook encounters an unhandled exception:

- PBS rejects the corresponding action. The command that initiates the action results in the following message in stderr:

  "<command_name>: request rejected as filter hook <hook_name> encountered an exception. Please inform Admin"

- The following message appears in the appropriate PBS daemon log, logged under PBSEVENT_DEBUG2 event class:

  "<request type> hook <hook_name> encountered an exception, request rejected"

- The job is left unmodified.

- If an exechost_startup hook script encounters an unexpected error causing an unhandled exception, vnode changes do not take effect, but MoM continues to run, and the following message appears at level PBSEVENT_DEBUG2 in mom_logs:

  "exechost_startup hook <hook_name> encountered an exception, request rejected"

  - The following statements will cause an unhandled exception if they appear in a hook script as is:
  - ZeroDivisionError exception raised:

    ```
    val = 5/0
    ```

  - BadAttributeValueError exception raised; pbs.hold_types and strings don't mix:

    ```
    pbs.event().job.Hold_Types = "z"
    ```

  - EventIncompatibleError exception raised for the following runjob event; runjob event has job attribute, not resv attribute:

    ```
    r = pbs.event().resv
    ```

- You can use execjob_begin and exechost_startup hooks to offline vnodes when those hooks encounter alarm calls or unhandled exceptions. See "Offlining and Clearing Vnodes Using the fail_action Hook Attribute" on page 66 of the PBS Professional Reference Guide. You can then clear the offline state from those vnodes later when an exechost_startup hook runs successfully.

- You can use an execjob_begin hook restart the scheduler cycle when the hook encounters an alarm call or unhandled exception. See "Restarting Scheduler Cycle After Hook Failure" on page 63 of the PBS Professional Reference Guide.

For a list of exceptions, see Table 5.2.2.1.ii, "Table of Exceptions," on page 43.

5.2.4 Using Attributes and Resources in Hooks

5.2.4.1 Using Built-in vs. Custom Resources in Hooks

Hooks have more access to built-in resources than they do to custom resources. All hooks can read built-in resources. All event hooks that run at the server can read all custom resources via pbs.event(), as well as via pbs.server(). However, hooks that run at the execution host can read custom resources only via pbs.server(). So for example if a job requests a custom resource, a runjob hook can read the resource, but an execjob_begin hook cannot.

5.2.4.2 Creating and Setting Custom Resources in Hooks

You can create a custom resource only in an exechost_startup hook. You can set a custom resource in a hook that runs at the server or using an exechost_startup hook. To create and set a custom resource in a vnode's resources_available attribute via an exechost_startup hook:

```
# qmgr -c "create hook start event=exechost_startup"
# qmgr -c "import hook start application/x-python default start.py"
# qmgr -c "export hook start application/x-python default"
```

Hook script:

```
import pbs
e=pbs.event()
localnode=pbs.get_local_nodename()

e.vnode_list[localnode].resources_available['foo_i'] = 7
e.vnode_list[localnode].resources_available['foo_f'] = 5.0
e.vnode_list[localnode].resources_available['foo_str'] = "seventyseven"
```

Note that while an exechost_startup hook cannot read an existing custom resource, it can create and set a new one.

When you create a custom job resource in an exechost_startup hook, the *m* flag is set by default. See "Allowing Execution Hooks to Read Custom Job Resources Faster" on page 263 in the PBS Professional Administrator's Guide.

5.2.4.3 Determining Whether to Use Creation Method to Set Attribute or Resource

The way you set an attribute or resource depends on the type of the attribute or resource:

- If the attribute or resource is a string (str), an integer (int), a Boolean (bool), a long (long), or a floating point (float), you can set it directly:

  pbs.event().job.<attribute name> = <attribute value>

  pbs.event().job.Resource_List["<resource name>"]=<resource value>

  For example:

  ```
  jobA = pbs.event().job
  jobA.Account_Name = "AccountA"
  jobA.Priority = 100
  ```

- However, if the attribute or resource is any other type, you must use the corresponding creation method to instantiate an object of the correct type with the desired value as a formatted input string, then assign the object to the job. For example:

  ```
  pbs.event().job.Hold_Types = pbs.hold_types("uo")
  ```

  For creation methods, see section 6.13.3, "PBS Types and Their Methods", on page 143.

5.2.4.3.i Caveat for Objects Requiring Creation Method

You can operate on these objects only as if they are strings. Use `repr()` on the object to get its full string representation. You can then manipulate this representation using the built-in methods for Python 'str'.

5.2.4.3.ii Python Types not Requiring Creation Method

The following Python types do not require you to use an explicit creation method:

bool

float

int

str

5.2.4.4 How to Unset an Attribute or Resource

To unset an attribute or resource, set *<attribute value>* to *None*:

```
pbs.event().job.<attribute name> = None
```

When you unset an attribute or resource, it takes its default value.

5.2.4.4.i How to Unset an Attribute or Resource Requiring Creation Method

You can unset a job attribute or resource that has a creation method by setting it to *None*.

Example:

```
pbs.event().job.Hold_Types = None
```

5.2.4.5 Using Attributes in Hooks: Reading vs. Setting

All hooks can read, but not set, all job, vnode, server, queue, and reservation attributes via pbs.server().job(), pbs.server().vnode(), pbs.server().queue(), etc.

We list which job attributes can be read or set when the job is retrieved through an event in Table 5-6, “Job Attributes Readable & Settable via Events,” on page 55.

We list which vnode attributes can be read or set when the vnode is retrieved through an event in Table 5-7, “Vnode Attributes Readable & Settable via Events,” on page 57.

We list which reservation attributes can be read or set when the reservation is retrieved through an event in Table 5-8, “Reservation Attributes Readable & Settable in resvsub and resv_end Hooks,” on page 59.

No hooks can see or set any scheduler attributes.

The job, vnode, or reservation object's attributes appear to the hook as they would be after the event, not before it, for all hooks except runjob hooks.

5.2.4.6 Setting Time Attributes

For the job attributes Execution_Time, ctime, etime, mtime, qtime, and stime, the pbs.job object expects or shows the number of seconds since Epoch. The only one of these that can be set is Execution_Time.

For the reservation attributes reserve_start, reserve_end, and ctime, the pbs.resv object expects and shows the number of seconds since Epoch. The ctime attribute cannot be set.

If you wish to set the value for Execution_Time, reserve_start, or reserve_end using the *[[CCYY]MMDDhhmm[.ss]* format, or to see the value of any of the time attributes in the ASCII time format, load the Python `time` module and use the functions `time.mktime([CCYY, MM, DD, hh, mm, ss, -1, -1, -1])` and `time.ctime()`.

Example:

```
import time
job.Execution_Time = time.mktime([07, 11, 28, 14, 10, 15, -1, -1, -1])
time.ctime(job.Execution_Time)
'Wed Nov 28 14:10:15 2007'
```

If reserve_duration is unset or set to *None*, the reservation's duration is taken from the walltime resource attribute associated with the reservation request. If reserve_duration and walltime are both specified, meaning not set to *None*, reserve_duration will take precedence.

5.2.4.7 Special Characters in Variable_List Job Attribute

When special characters are used in Variable_List job attributes, they must be escaped. For this attribute, special characters are comma (,), single quote ('), double quote ("), and backslash (\). PBS requires each of these to be escaped with a backslash. However, Python requires that double quotes and backslashes also be escaped with a backslash. If the special character inside a string is a single quote, you must enclose the string in double quotes. If the special character inside the string is a double quote, you must enclose the string in single quotes. The following rules show how to use special characters in a Variable_List attribute when writing a Python script:

Table 5-3: How to Use Special Characters in Python Scripts

Character	Example Value	How to Represent Value in Python Script
, (comma)	*a,b*	"a\\,b" or 'a\\,b'
' (single quote)	*c'd*	"c\\'d"
" (double quote)	*f"g"h*	'f\\\"g\\\"h'
\ (backslash)	*\home\dir\files*	"\\home\\dir\\files" or '\\home\\dir\\files'

For example, if the path is:

```
"\Documents and Settings\pbstest\bin:\windows\system32"
```

This is how the path shows up in a script:

```
job.Variable_List["PATH"] = "\\Documents and Settings\\pbstest\\bin:\\windows\\system32"
```

5.2.4.8 Using string_array Attributes and Resources

5.2.4.8.i Handling Literal Values and Special Characters in string_array Format

In order to capture a literal value or special characters in a string_array attribute or resource, enclose the entire string array in single quotes.

For an attribute or resource whose type is string_array and whose value contains one or more commas (","), the whole string must be enclosed in single quotes, outside of its double quotes. For example:

If our string array has a single element consisting of "glad, elated":

```
job.Resource_List["test_string_array"] = '"glad, elated"'
```

If our string array has two elements, where one is "glad, elated" and the other is "happy":

```
job.Resource_List["test_string_array"] = '"glad, elated","happy"'
```

5.2.4.9 Using Resources in Hooks: Reading vs. Setting

All hooks can read, but not set, all job, vnode, server, queue, and reservation resources via pbs.server().job(), pbs.server().vnode(), pbs.server().queue(), etc.

The resources that can be read or set via pbs.event() vary by hook:

We list the job resources that can be read and set via an event in each kind of hook in Table 5-9, "Built-in Job Resources Readable & Settable by Hooks via Events," on page 60.

We list the vnode resources that can be read and set via an event in each kind of hook in Table 5-10, "Vnode Resources Readable & Settable by Hooks via Events," on page 61.

We give an overview of the resources that can be read and set by each hook in Table 5-4, “Overview of Resources Readable & Settable by Hooks via Pre-execution and Provision Events,” on page 52 and Table 5-5, “Overview of Resources Readable & Settable by Hooks via execjob_ and exechost_ Events,” on page 52. In these tables, if we say that a hook can read or set a group of resources, for example the server's resources_available attribute, that means that the hook can read or set all of the resources for that group.

5.2.4.10 Reading Resources in Hooks

PBS resources are represented as objects of type pbs.pbs_resource, where the resource names are the keys. This type is described in section 6.13.3.19, “Method to Create or Set Resource List”, on page 147. Built-in resources are listed in “List of Built-in Resources” on page 261 of the PBS Professional Reference Guide.

You can read a resource through objects such as the server, the event that triggered the hook, or the vnode to which a resource belongs. For example:

```
pbs.server().resources_available["<resource name>"]
pbs.event().job.Resource_List["<resource name>"]
pbs.event().vnode_list[<vnode name>].resources_available["ncpus"]
```

The resource name must be in quotes.

Example: Get the number of CPUs in a job's Resource_List attribute:

```
ncpus=pbs.event().job.Resource_List["ncpus"]
```

5.2.4.10.i Converting walltime to Seconds

If you want to see a job's walltime in seconds:

```
int(pbs.event().job.Resource_List["walltime"])
```

For example:

```
pbs.logmsg(pbs.LOG_DEBUG, "walltime=%d" % (int(pbs.event().job.Resource_List["walltime"])))
```

If walltime is "00:30:15", this results in the following:

```
walltime=1815
```

5.2.4.11 Setting and Unsetting Vnode Resources and Attributes

You can set and unset vnode resources and attributes using the vnode_list[] object in an exechost_startup or exechost_periodic hook. Any changes made this way are merged with those defined in a Version 2 vnode configuration file.

To set the attributes and resources for a particular vnode:

pbs.event().vnode_list[<vnode name>].<attribute> = <value>

pbs.event().vnode_list["<vnode name>"].resources_available["<resource name>"] = <resource value>

Resource names and string values must be quoted.

Some examples:

```
pbs.event().vnode_list["V2"].pcpus = 5
pbs.event().vnode_list["V2"].resources_available["ncpus"]  = 3
pbs.event().vnode_list["V2"].resources_available.["mem"] = pbs.size("100gb")
pbs.event().vnode_list["V2"].arch = "linux"
pbs.event().vnode_list["V2"].state = pbs.ND_OFFLINE
pbs.event().vnode_list["V2"].sharing = pbs.ND_FORCE_EXCL
```

To unset a resource value, specify "*None*" as its value:

pbs.event().vnode_list[<vnode_name>].resources_available[<res>] = None

pbs.event().vnode_list[<vnode_name>].<attribute> = None

5.2.4.12 Setting Job Resources in Hooks

You can set a job's Resource_List in pre-execution event hooks listed in Table 5-9, "Built-in Job Resources Readable & Settable by Hooks via Events," on page 60.

You can use an execution event hook (execjob_prologue, execjob_epilogue, and exechost_periodic) to set the value of a job's resources_used for host-level resources. The values of these resources are then reported in the job's resources_used attribute. For multi-vnode jobs, numeric values are summed and string resources are aggregated on a per-MoM basis.

5.2.4.12.i Steps for Setting Job Resources in Hooks

You can set values for a job's Resource_List or resources_used attributes as follows:

pbs.event().job.Resource_List["<resource name>"] = <resource value>

pbs.event().job.resources_used["<resource name>"] = <resource value>

For example:

```
pbs.event().job.Resource_List["mem"] = 8gb
```

5.2.4.12.ii String Resource Format for Python

Each string value returned by a MoM is a JSON object (a Python dictionary), which is an unordered set of key-value pairs, where each object begins with a left curly brace ("{"), and ends with a right curly brace ("}"). Each key is followed by a colon (":"), and the key-value pairs are separated using a comma (","). The key is enclosed in double quotes (allowing backslash escapes).

5.2.4.12.iii Setting String Job Resources in Hooks

When all values are in JSON format, the resulting string resource value is a union of all dictionary items, shown in `qstat -f` output and accounting logs as:

```
resources_used.<resource_name> = {<MoMA_JSON_item_value>, <MoMB_JSON_item_value>,
    <MoMC_JSON_item_value>, ..}
```

Example 5-10: If MoMA returns '{ "a":1, "b":2 }', MoMB returns '{ "c":1 }', and MoMC returns '{"d":4}' for resources_used.foo_str. We see the following:

```
resources_used.foo_str='{"a": 1, "b": 2, "c":1,"d": 4}'
```

If two or more values have the same value for the key, only one of them is retained, depending on Python's operation of merging dictionary items. We recommend that hook writers make the keys unique; you can do this by using the value returned by pbs.get_local_nodename() as part of the key.

When at least one of the values obtained from a sister MoM is not of JSON format, the string cannot be accumulated, and resources_used remains unset. PBS writes an error message in the MoM logs as follows:

```
"Job <jobid> resources_used.<string_resource> cannot be accumulated: value <input value> from MoM
    <hostname> not JSON-format: <exception_error_message>."
```

5.2.4.12.iv Example of Setting Resources in Hooks

Example 5-11: Using an epilogue hook that runs on all the MoMs, we set different resources_used values depending on whether the hook executes on the primary execution host or a sister MoM:

```
#: qmgr -c "list hook epi"
Hook epi
type = site
enabled = true
event = execjob_epilogue
user = pbsadmin
alarm = 30
order = 1
debug = false
fail_action = none
# qmgr -c "e h epi application/x-python default"
import pbs
e=pbs.event()
pbs.logmsg(pbs.LOG_DEBUG, "executed epilogue hook")
if e.job.in_ms_mom(): #set in MS mom
    e.job.resources_used["vmem"] = pbs.size("9gb")
    e.job.resources_used["foo_i"] = 9
    e.job.resources_used["foo_f"] = 0.09
    e.job.resources_used["foo_str"] = '{"nine":9}'
    e.job.resources_used["cput"] = 10
    e.job.resources_used["foo_assn2"] = '{"vn1":1,"vn2":2,"vn3":3}'
else: # set in sister mom
    e.job.resources_used["vmem"] = pbs.size("10gb")
    e.job.resources_used["foo_i"] = 10
    e.job.resources_used["foo_f"] = 0.10
    e.job.resources_used["foo_str"] = '{"ten":10}'
    e.job.resources_used["cput"] = 20
    e.job.resources_used["foo_assn2"] = '{"vn4":4,"vn5":5,"vn6":6}'
```

Using two execution hosts, submit the following job:

```
% cat job.scr2
PBS -l select=2:ncpus=1
pbsdsh -n 1 hostname
sleep 300


% qsub job.scr2
102.corretja
```

When the job completes, we can see values for resources_used. With server job_history_enabled=*True*, we can check the values in a finished job. Values in bold show resources accumulated from both MoMs:

```
% qstat -x -f 102
...
resources_used.cpupercent = 0
resources_used.cput = 00:00:30
resources_used.vmem = 19gb
resources_used.foo_f = 0.19
resources_used.foo_i = 19
resources_used.foo_str = '{"nine": 9, "ten": 10}'
resources_used.foo_assn2='{"vn1": 1, "vn2": 2 ,"vn3": 3 ,"vn4": 4, "vn5": 5, "vn6": 6}'
resources_used.mem = 0kb
resources_used.ncpus = 2
resources_used.walltime = 00:00:05
```

The accounting logs show the same values:

```
8/03/2016 18:28:13;E;102.corretja;user=alfie group=users project=_pbs_project_default
    jobname=job.scr2 queue=workq ctime=1470263288 qtime=1470263288 etime=1470263288
    start=1470263288 exec_host=corretja/0+nadal/0 exec_vnode=(corretja:ncpus=1)+(nadal:ncpus=1)
    Resource_List.ncpus=2 Resource_List.nodect=2 Resource_List.place=free
    Resource_List.select=2:ncpus=1 session=16986 end=1470263293 Exit_status=143
    resources_used.cpupercent=0 resources_used.cput=00:00:30 resources_used.vmem=19gb
    resources_used.foo_f=0.19 resources_used.foo_i=19 resources_used.foo_str='{"nine": 9, "ten":
    10}' resources_used.foo_assn2='{"vn1": 1, "vn2": 2 ,"vn3": 3 ,"vn4": 4, "vn5": 5, "vn6": 6}'
    resources_used.mem=0kb resources_used.ncpus=2 resources_used.walltime=00:00:05 run_count=1
```

5.2.4.12.v Setting Built-in Job Resource in Hook Prevents MoM from Updating Resource

If you use a hook to set the value of a built-in host-level resource for a specific job, MoM no longer updates the value of the resource for that job; she leaves that to you. You can get MoM to resume updating the resource for that job only by changing the hook so that it doesn't set the resource, and restarting the job.

Under Linux, job resources_used that MoM does not modify if they've been set in a hook are cput, walltime, mem, vmem, ncpus, and cpupercent.

Under Windows, job resources_used that MoM does not modify if they've been set in a hook are cput, walltime, mem, and ncpus.

5.2.4.13 Overview of Readable & Settable Resources

Here we list an overview of which resources can be read or set in hooks. An "*r*" indicates read, an "*s*" indicates set, and an "*o*" indicates that this resource can be set but the action has no effect. See Table 4-1, "Execution Event Hook Timing," on page 19 for more information about why some operations have no effect. The following table shows which resource categories are readable or settable in pre-execution and provision hooks:

Table 5-4: Overview of Resources Readable & Settable by Hooks via Pre-execution and Provision Events

Resource Category	queuejob	modifyjob (before run)	movejob	runjob	resvsub	resv_end	provision
Job Resource_List (Varies; see Table 5-9)	r, s	r, s	r	Table 5-9	---	---	---
Job resources_used	o	r	r	r	r	---	---
Vnode resources_available	---	---	---	---	---	---	---
Vnode resources_assigned	r	r	r	r	r	---	r
Server resources_available	r	r	r	r	r	r	r
Server resources_assigned	r	r	r	r	r	r	r
Server resources_default	r	r	r	r	r	r	r
Server resources_max	r	r	r	r	r	r	r
Queue resources_available	r	r	r	r	r	r	r
Queue resources_assigned	r	r	r	r	r	r	r
Queue resources_default	r	r	r	r	r	r	r
Queue resources_max	r	r	r	r	r	r	r
Queue resources_min	r	r	r	r	r	r	r
Reservation Resource_List	r	r	r	r	r, s	r	---

The following table lists an overview of which resources can be read or set in execjob and exechost hooks.

Table 5-5: Overview of Resources Readable & Settable by Hooks via execjob_ and exechost_ Events

Resource Category	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic
Job Resource_List (except for custom resources)	r	r	r	r	r	r	r	r	r	r	r
Job resources_used	r, s	r	r, s	r, s	r	r	r	r, s	r, s	r	r, s
Vnode resources_available	r, s	r	r, s	r	r	r	r	r, s	r, s	r, s	r, s
Vnode resources_assigned	r	r	r	r	r	r	r	r	r	r	r
Server resources_available	r	r	r	r	r	r	r	r	r	r	r
Server resources_assigned	r	r	r	r	r	r	r	r	r	r	r
Server resources_default	r	r	r	r	r	r	r	r	r	r	r

Table 5-5: Overview of Resources Readable & Settable by Hooks via execjob_ and exechost_ Events

Resource Category	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic
Server resources_max	r	r	r	r	r	r	r	r	r	r	r
Queue resources_available	r	r	r	r	r	r	r	r	r	r	r
Queue resources_assigned	r	r	r	r	r	r	r	r	r	r	r
Queue resources_default	r	r	r	r	r	r	r	r	r	r	r
Queue resources_max	r	r	r	r	r	r	r	r	r	r	r
Queue resources_min	r	r	r	r	r	r	r	r	r	r	r
Reservation Resource_List	r	r	r	r	r	r	r	r	r	r	r

5.2.4.14 Caveats for Setting and Unsetting Attributes and Resources

5.2.4.14.i When to Change Reservation Attributes

The only time that a reservation's attributes can be altered is during the creation of that reservation in a resvsub hook.

5.2.4.14.ii Caution About Unsetting Reservation walltime Resource

The walltime resource is used to determine the reservation's reserve_duration parameter when the reservation's reserve_duration attribute is not set or is set to *None*. If a resvsub hook attempts to unset the walltime parameter, for example:

```
pbs.event().resv.Resource_List["walltime"] = None
```

This will result in the following error:

```
% pbs_rsub -R 1800 -l ncpus=1
pbs_rsub: Bad time specification(s)
```

5.2.4.14.iii Changing Job Attributes for a Running Job

When a job is running, only the cput and walltime attributes can be modified. Attempting to change any other attributes for a running job will cause the corresponding `qalter` action to be rejected. For example, if the job is running, this line in a hook will cause `qalter` to be rejected:

```
pbs.event().job.Resource_List["mem"] = pbs.size("10mb")
```

To avoid having the `qalter` action rejected, check to see whether the job is running, and follow up accordingly. For example:

```
e = pbs.event()
if e.job.job_state in [ pbs.JOB_STATE_RUNNING, pbs.JOB_STATE_EXITING, pbs.JOB_STATE_TRANSIT ]:
    e.accept()
```

5.2.4.14.iv Do Not Unset Array Job Indices

Do not unset pbs.event().job.array_indices_submitted for an array job in a modifyjob hook. For example:

```
pbs.event().job.array_indices_submitted = None
```

If the hook script is executed for a job array, the qalter request will fail with the message:

```
Cannot modify attribute while job running  <job array ID>
```

5.2.4.14.v Do Not Create Job or Reservation Variable List

Hooks are not allowed to create job or reservation Variable_List attributes. Hooks can modify the existing Variable_List job attribute which is supplied by PBS, by modifying values in the list. The following are disallowed in a hook:

```
pbs.event().job.Variable_List = dict()
pbs.event().resv.Variable_List = dict()
```

These calls will cause the following exception:

```
04/07/2008 11:22:14;0001;Server@host2;Svr;Server@host2;PBS server internal error (15011) in Error
    evaluating Python script, attribute 'Variable_List' cannot be directly set.
```

To modify the Variable_List attribute:

```
pbs.event().job.Variable_List["SIMULATE"] = "HOOK1"
```

5.2.4.14.vi Changing Vnode state Attribute

A vnode's state can be set within a runjob hook only if the runjob hook execution concludes with a pbs.event().reject() call. This means that if a statement that sets a vnode's state appears in a runjob hook script, it takes effect only if the following is the last line to be executed:

```
pbs.event().reject()
```

To set a vnode's state, the syntax is one of the following:

pbs.vnode.state = <vnode state constant>

pbs.vnode.state += <vnode state constant>

pbs.vnode.state -= <vnode state constant>

where *<vnode state constant>* is one of the constant objects listed in section 6.10.4, “Vnode State Constant Objects”, on page 135.

Examples of changing a vnode's state attribute:

- To offline a vnode:
  ```
  pbs.vnode.state = pbs.ND_OFFLINE
  ```
- To add another value to the list of vnode states:
  ```
  pbs.vnode.state += pbs.ND_DOWN
  ```
- To remove a value from the list of vnode states:
  ```
  pbs.vnode.state -= pbs.ND_OFFLINE
  ```

When a vnode's state attribute has no states set, the vnode's state is equivalent to *free*. This means that you can remove all values, and the vnode will become *free*.

When a vnode's state is successfully set, the following message is displayed and logged at event class 0x0004:

```
Node;<vnode-name>;attributes set: state - <vnode state constant> by <hook_name>
```

You can set a vnode's state attribute in any execution hook and in a periodic hook, and changes to vnode attributes take effect whether the execution hook or periodic hook calls accept() or reject().

5.2.4.14.vii Attribute Change Failure is Silent

If you attempt to change the value for an attribute in an unsupported way, PBS does not warn you that your attempt failed.

5.2.4.14.viii Lengthened walltime Can Interfere with Reservations

If a hook lengthens the walltime of a running job, you run the risk that the new walltime will interfere with existing reservations etc.

5.2.4.14.ix Setting Vnode Resources in Hooks Overwrites Previous Value

When you set resources_available for a vnode, inside or outside of a hook, you are overwriting the previous value. There is no way in a hook to know whether a value was set inside or outside a hook (for example, using `qmgr` or a vnode definition file). There is no way to prevent a value set inside a hook from being modified outside of the hook.

5.2.4.14.x Changing Resources in Accounting Logs

If you use a non-execjob_end execution hook to set a value for resources_used, the new value for resources_used appears in the accounting logs.

5.2.4.14.xi When Setting Resources Has No Effect

- If you use an execjob_end execution hook to set a value for resources_used, it has no effect, because MoM has already sent the final values for resources_used to the server.
- You cannot use a hook to set a server-level resource. PBS ignores these actions in a hook.
- You cannot use the `qmgr` command to set resources_used for a job.

5.2.4.15 Table: Reading & Setting Job Attributes in Hooks

The following table lists the job attributes that can be read or set when the job is retrieved via an event. An "*r*" indicates read, an "*s*" indicates set, and an "*o*" indicates that this attribute can be set but the action has no effect. See Table 4-1, "Execution Event Hook Timing," on page 19 for more information about why some operations have no effect.

Table 5-6: Job Attributes Readable & Settable via Events

Job Attribute	queuejob	modifyjob (before run)	movejob	runjob (on reject)	runjob (on accept)	resvsub	resv_end	periodic	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic
accounting_id	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
Account_Name	r, s	r, s	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
accrue_type	---	r	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
alt_id	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
argument_list	---	---	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
array	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
array_id	---	r	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
array_index	---	r	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
array_indices_remaining	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
array_indices_submitted	r, s	---	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
array_state_count	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
block	---	r, s	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
Checkpoint	r, s	r, s	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
comment	---	---	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
create_resv_from_job	r, s	r, s	r, s	r, s	r, s	r, s	r, s	r, s	r, s	r, s	r, s	r, s	r, s	r, s	r, s	r, s	r, s	r, s	r, s

Table 5-6: Job Attributes Readable & Settable via Events

Job Attribute	queuejob	modifyjob (before run)	movejob	runjob (on reject)	runjob (on accept)	resvsub	resv_end	periodic	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic
ctime	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
depend	r, s	r, s	r	r, s	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
egroup	---	r	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
eligible_time	---	r, s	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
Error_Path	r, s	r, s	r	r, s	r, s	---	---	---	r	r	r	r	r	r	r	r	r	---	r
estimated	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
etime	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
euser	---	r	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
Executable	r, s	---	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
Execution_Time	r, s	r, s	r	r, s	r	---	---	---	r, s	r	r, s	r, s	r	r	r	r, s	r, s	---	r, s
exec_host	---	---	r	---	---	---	---	---	r	r	r	r	r	r	r	r	r	---	r
exec_vnode	---	---	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
Exit_status	---	r	r	r	r	---	---	---	---	r	---	---	r	r	r	r	---	---	---
group_list	r, s	r, s	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
hashname	---	r	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
Hold_Types	r, s	r, s	r	r, s	r	---	---	---	r, s	r	r, s	r, s	r	r	r	r, s	r, s	---	r, s
interactive	r, s	r, o	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
jobdir	---	r	r	---	---	---	---	---	---	---	---	---	---	---	---	r	---	---	---
Job_Name	r, s	r, s	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
Job_Owner	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
job_state	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
Join_Path	r, s	r, s	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
Keep_Files	r, s	r, s	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
Mail_Points	r, s	r, s	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
Mail_Users	r, s	r, s	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
max_run_subjobs	r, s	r, s	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
mtime	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
no_stdio_sockets	---	---	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
Output_Path	r, s	r, s	r	r, s	r, s	---	---	---	r	r	r	r	r	r	r	r	r	---	r
Priority	r, s	r, s	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
project	r, s	r, s	r	r, s	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
pset	---	---	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
qtime	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
queue	r, s	r	r, s	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
queue_rank	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
queue_type	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
release_nodes_on_stageout	r, s	r, s	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
Rerunable	r, s	r, s	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
resources_released	r	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
resources_released_list	r	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
resources_used	---	r	r	r	r	---	---	---	r, s	r	r, s	r, s	r	r	r	r, s	r, s	r, s	r, s

Table 5-6: Job Attributes Readable & Settable via Events

Job Attribute	queuejob	modifyjob (before run)	movejob	runjob (on reject)	runjob (on accept)	resvsub	resv_end	periodic	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic
Resource_List (with restrictions; see Table 5-9)	r, s	r, s	r	r, s	r, s	---	---	---	r	r	r	r	r	r	r	r	r	---	r
run_count	r, s	r, s	r	r	r	---	---	---	r, s	r	r, s	r, s	r	r	r	r, s	r, s	---	r, s
run_version	---	r	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
sandbox	r, s	r, s	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
schedselect	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
sched_hint	---	---	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
server	---	r	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
session_id	---	---	---	---	---	---	---	---	---	---	---	---	r	r	r	r	r	---	---
Shell_Path_List	r, s	r, s	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
stagein	r, s	r, s	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
stageout	r, s	r, s	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
Stageout_status	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
stime	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
Submit_arguments	---	---	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
substate	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
sw_index	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
umask	r, s	r, s	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
User_List	r, s	r, s	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
Variable_List	r, s	r, s	r	r, s	r, s	---	---	---	r, s	r	r	r	r	r	r	r	r	---	r, s

5.2.4.16 Table: Reading & Setting Vnode Attributes in Hooks

The following table shows the vnode attributes that can be read or set when the vnode object is retrieved via an event. An "*r*" indicates read, an "*s*" indicates set, and an "*o*" indicates that this attribute can be set but the action has no effect. See Table 4-1, “Execution Event Hook Timing,” on page 19 for more information about why some operations have no effect.

Table 5-7: Vnode Attributes Readable & Settable via Events

Vnode Attribute	queuejob	modifyjob (before run)	movejob	runjob	resvsub	resv_end	periodic	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic	provision
comment	r, s	r, s	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
current_aoe	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
hpcbp_enable	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---

Table 5-7: Vnode Attributes Readable & Settable via Events

Vnode Attribute	queuejob	modifyjob (before run)	movejob	runjob	resvsub	resv_end	periodic	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic	provision
hpcbp_stage_protocol	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
hpcbp_user_name	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
hpcbp_webservice_address	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
in_multivnode_host	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
jobs	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
license	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
license_info	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
Mom	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
name	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
no_multinode_jobs	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
ntype	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
pbs_version	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r	r	---
pcpus	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r	r	---
pnames	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
Port	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
Priority	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
provision_enable	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
queue	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
resources_assigned	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r	r	---
resources_available	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
resv	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
resv_enable	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
sharing	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
state	---	---	---	r, s	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
topology_info	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	r	r	---

5.2.4.17 Table: Reading & Setting Reservation Attributes in resvsub and resv_end Hooks

Reservation attributes can be read and set through an event only in resvsub and resv_end hooks. No other hooks can read or set reservation attributes through an event. All hooks can read, but not set, all reservation attributes by retrieving the reservation object through the server, using pbs.server().resv(). The following table shows the reservation attributes that can be read or set when the reservation object is retrieved via an event, in a resvsub or resv_end hook:

Table 5-8: Reservation Attributes Readable & Settable in resvsub and resv_end Hooks

Reservation Attribute	resvsub	resv_end
Account_Name	r	r
Authorized_Groups	r, s	r
Authorized_Hosts	r, s	r
Authorized_Users	r, s	r
ctime	r	r
group_list	r, s	r
hashname	r	r
interactive	r, s	r
Mail_Points	r, s	r
Mail_Users	r, s	r
mtime	r	r
Priority	r	r
queue	r	r
reserve_count	r	r
reserve_duration	r, s	r
reserve_end	r, s	r
reserve_ID	r	r
reserve_index	r	r
reserve_job	r	r
Reserve_Name	r, s	r
Reserve_Owner	r	r
reserve_retry	r	r
reserve_rrule	r, s	r
reserve_start	r, s	r
reserve_state	r	r
reserve_substate	r	r
reserve_type	r	r

Table 5-8: Reservation Attributes Readable & Settable in resvsub and resv_end Hooks

Reservation Attribute	resvsub	resv_end
Resource_List	r, s	r
resv_nodes	r	r
server	r, s	r
User_List	r	r
Variable_List	r, s	r

5.2.4.18 Table: Reading & Setting Built-in Job Resources in Hooks

The following table shows the built-in members of the job's Resource_List attribute that can be read or set in each type of hook, when retrieving the object through an event.

An "*r*" indicates read, an "*s*" indicates set, and an "*o*" indicates that this resource can be set but the action has no effect. See Table 4-1, “Execution Event Hook Timing,” on page 19 for more information about why some operations have no effect. For more about custom resources, see section 5.2.4.1, “Using Built-in vs. Custom Resources in Hooks”, on page 44.

Table 5-9: Built-in Job Resources Readable & Settable by Hooks via Events

Resource in Resource_List	queuejob	modifyjob (before run)	movejob	runjob (on reject)	runjob (on accept)	periodic	resvsub	resv_end	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic	provision
accelerator	r, s	r, s	r	r, s	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
accelerator_memory	r, s	r, s	r	r, s	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
accelerator_model	r, s	r, s	r	r, s	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
aoe	r, s	r, s	r	r, s	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
arch	r, s	r, s	r	r, s	r, s	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
cput	r, s	r, s	r	r, s	r, s	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
exec_vnode	r, s	r, s	r	r, s	r, s	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
file	r, s	r, s	r	r, s	r, s	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
host	r, s	r, s	r	r, s	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
max_walltime	r, s	r, s	r	r, s	r, s	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
mem	r, s	r, s	r	r, s	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
min_walltime	r, s	r, s	r	r, s	r, s	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
mpiprocs	r, s	r, s	r	r, s	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
naccelerators	r, s	r, s	r	r, s	r, s	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
nchunk	r, s	r, s	r	r, s	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
ncpus	r, s	r, s	r	r, s	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
nice	r, s	r, s	r	r, s	r, s	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
nodect	r, s	r, s	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
nodes	r, s	r, s	r	r, s	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
ompthreads	r, s	r, s	r	r, s	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Table 5-9: Built-in Job Resources Readable & Settable by Hooks via Events

Resource in Resource_List	queuejob	modifyjob (before run)	movejob	runjob (on reject)	runjob (on accept)	periodic	resvsub	resv_end	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic	provision
pcput	r, s	r, s	r	r, s	r, s	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
pmem	r, s	r, s	r	r, s	r, s	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
pvmem	r, s	r, s	r	r, s	r, s	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
site	r, s	r, s	r	r, s	r, s	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
software	r, s	r, s	r	r, s	r, s	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
soft_walltime	r, s	r, s	r	r, s	r, s	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
start_time	r, s	r, s	r	r, s	r, s	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
vmem	r, s	r, s	r	r, s	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
vnode	r, s	r, s	r	r, s	r, s	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
vntype	r, s	r, s	r	r, s	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
walltime	r, s	r, s	r	r, s	r, s	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
PBScrayhost	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
PBScraylabel_<label name>	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
PBScraynid	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
PBScrayorder	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
PBScrayseg	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

5.2.4.19 Table: Reading & Setting Vnode Resources in Hooks

The following table shows the built-in members of the vnode's resources_available attribute that can be read or set in each type of hook, when retrieving the object through an event. An "*r*" indicates read, an "*s*" indicates set, and an "*o*" indicates that this resource can be set but the action has no effect. See Table 4-1, "Execution Event Hook Timing," on page 19 for more information about why some operations have no effect.

Table 5-10: Vnode Resources Readable & Settable by Hooks via Events

Resource in resources_available	queuejob	modifyjob (before run)	movejob	runjob (on reject)	runjob (on accept)	periodic	resvsub	resv_end	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic	provision
accelerator	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
accelerator_memory	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
accelerator_model	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
aoe	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
arch	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
cput	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---

Table 5-10: Vnode Resources Readable & Settable by Hooks via Events

Resource in resources_available	queuejob	modifyjob (before run)	movejob	runjob (on reject)	runjob (on accept)	periodic	resvsub	resv_end	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic	provision
exec_vnode	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
file	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
host	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
max_walltime	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
mem	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
min_walltime	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
mpiprocs	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
naccelerators	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
nchunk	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
ncpus	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
nice	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
nodect	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
nodes	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
ompthreads	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
pcput	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
pmem	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
pvmem	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
site	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
software	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
start_time	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
vmem	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
vnode	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
vntype	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
walltime	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
PBScrayhost	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
PBScraylabel_<label name>	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
PBScraynid	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
PBScrayorder	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
PBScrayseg	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---

5.2.5 Using select and place in Hooks

All hooks can read, but not set, a job's select and place statements via pbs.server().job(), pbs.server().vnode(), pbs.server().queue(), etc. The following table shows the type of hook that can read or set a job's select and place statements, when retrieving the object through an event. An "*r*" indicates *read*, an "*s*" indicates *set*. See Table 4-1, "Execution Event Hook Timing," on page 19 for more information about why some operations have no effect.

Table 5-11: Hooks that Can Read & Set Job select and place Statements via Events

Select or Place	queuejob	modifyjob (before run)	movejob	runjob (on reject)	runjob (on accept)	periodic	resvsub	resv_end	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic	provision
Job place statement	r, s	r, s	r	r, s	r	---	---	---	r	r	r	r	r	r	r	---	---	---
Job select statement	r, s	r, s	r	r, s	r	---	---	---	---	---	---	---	---	---	---	---	---	---

5.2.5.1 How to Set select and place in Hooks

You must use the associated creation method to instantiate an object of the correct type with the desired value, then assign the object to the job. Syntax:

job.Resource_List["place"] = pbs.place("[arrangement]:[sharing]:[group]")

job.Resource_List["select"] = pbs.select("[N:]res=val[:res=val][+[N:]res=val[:res=val] ...]")

Example 5-12: Set a job's select and place directives:

```
jobB = pbs.event().job
jobB.Resource_List["place"] = pbs.place("pack:exclhost")
jobB.Resource_List["select"] = pbs.select("2:mem=2gb:ncpus=1+6:mem=8gb:ncpus=16")
```

See "pbs.select()" on page 148 and "pbs.place()" on page 147.

For modifying a job's select statement when allowing jobs access to extra vnodes, see section 6.13.3.24, "Method to Increment select Object Chunks", on page 148 and section 6.6.4.4, "Job Object Method to Release Vnodes", on page 128. For more about making jobs more reliable, see "Vnode Fault Tolerance for Job Start and Run" on page 437 of the PBS Professional Administrator's Guide.

5.2.5.2 Caveats for Using select and place in Hooks

You may want to check resource requests for a queuejob hook If a user submits a job using old `-lnodes` or `-lncpus` syntax, this is translated to a select statement, but only after a queuejob hook has run.

5.2.6 Restarting Scheduler Cycle After Hook Failure

You can restart the scheduler after an execjob_begin hook fails due to an alarm call or unhandled exception, or when the hook fails due to an internal error such as a full disk or not enough memory on the host, for example, a `malloc()` error. To restart the scheduler after failure of an execjob_begin hook, set the value of an execjob_begin or execjob_prologue hook's fail_action attribute to include "*scheduler_restart_cycle*".

```
# qmgr -c "set hook <hook_name> fail_action += scheduler_restart_cycle"
```

See section 5.1.9.2, "Using the fail_action Hook Attribute", on page 37.

5.2.7 Adding Custom Non-consumable Host-level Resources

You can add new custom host-level, non-consumable resources and set their values in resources_available for a vnode by using vnode_list[]in an exechost_startup hook. Any changes made this way are merged with those defined in a Version 2 vnode configuration file. Upon startup, MoM reads configuration files before executing the exechost_startup hook.

To add a new custom host-level resource, and set its value:

v = pbs.event().vnode_list[<vnode name>]

v.resources_available[<new_resource>] = <value>

The type of the resource is inferred from the value assigned to the resource. Python types map to PBS types as shown in the following table:

Table 5-12: Resource Types when Adding via vnode_list

Python Type	Type
int	*Long*
str	*String*
bool	*Boolean*
pbs.size	*Size*
pbs.duration	*Long*
float	*Float*
Any Python type without an explicit match	*String*

You must also make the resource usable by the scheduler: see section 5.14.2.6, “Allowing Jobs to Use a Resource”, on page 267.

To delete a custom resource created in a hook, use qmgr. See section 5.14.2.4.iii, “Deleting Custom Resources”, on page 265.

Example 5-13: Adding custom resources:

If you have these instructions in a hook:

```
vn.resources_available["fab_int"] = 9
vn.resources_available["fab_str"] = "happy"
vn.resources_available["fab_bool"] = False
vn.resources_available["fab_size"]  = pbs.size("7mb")
vn.resources_available["fab_time"] = pbs.duration("00:30:00")
vn.resources_available["fab_float"] = 7.0
```

This is equivalent to the following qmgr commands:

```
qmgr -c "create resource fab_int type=long,flag=h"
qmgr -c "create resource fab_str type=string,flag=h"
qmgr -c "create resource fab_bool type=boolean,flag=h"
qmgr -c "create resource fab_size type=size,flag=h"
qmgr -c "create resource fab_time type=long,flag=h"
qmgr -c "create resource fab_float type=float,flag=h"
```

5.2.8 Printing And Logging Messages

Hooks can log a custom string in the local daemon's log, at message log event class *pbs.LOG_DEBUG (0x0004)*. This is done using the pbs.logjobmsg(job ID, message) facility. See "pbs.logjobmsg()" on page 151.

Hooks can specify a message for use when the corresponding action is rejected. This message is printed to `stderr` by the command that triggered the event, and is printed in the daemon's log. This is done using the pbs.event().reject(<message>) function. See "pbs.event().reject()" on page 116 for information on how to specify a rejection message.

Hooks cannot directly print to `stdout` or `stderr`, or read from `stdin`. See section 5.3.8.1, "Avoid Hook File I/O", on page 71, and section 8.10.2.8, "Hooks Attempting I/O", on page 225.

5.2.9 Capturing Return Code

To capture an application's return code, you capture the return code in Python and then return it from the hook. You can use the Python subprocess module. Here is an example snippet:

```
import sys
if "<path to subprocess module>" not in sys.path:
    sys.path.append("<path to subprocess module>")
import subprocess

try:
    retcode = subprocess.call("mycommand myarg", shell=True)
except OSError:
    retcode = -1

return retcode
```

5.2.10 When You Need Persistent Data

If you need your data to be persistent, your hook(s) must be able to save and retrieve the information. Hooks are stateless, and each invocation of a hook has no knowledge of prior state of jobs, vnodes, etc. If you want to retain state across invocations of a hook, you can have the hook write what you need to a well-known location such as `PBS_HOME`. When the hook is invoked, it can read in the data, and before the hook exits, it can update the data. You can use whatever format you like for the data.

5.2.11 Setting Up Job Environment on Sisters

If you need to set up the job's environment on sister MoMs, use an execjob_begin hook. This hook can set up the desired environment on sister MoMs so that the job can use the new environment.

If job tasks are spawned on sister MoMs via a tightly-integrated MPI that uses `tm_spawn()`, any execjob_prologue and execjob_launch hooks run on the sister MoMs. However, if job tasks are started using `pbs_attach()`, execjob_attach and execjob_prologue (on the first task attached) hooks run on sister MoMs instead. For a detailed description of the order in which hooks run on the primary and secondary execution hosts, see Table 4-1, "Execution Event Hook Timing," on page 19.

The old-style prologue runs only on the primary execution host; you cannot use it to set up the environment on sister MoMs.

All job tasks running on vnodes managed by the same MoM get the same environment.

5.2.12 Offlining Bad Vnodes

5.2.12.1 General Method for Offlining Bad Vnodes

If you need to offline a bad vnode where a hook is running:

```
this_vnode = pbs.event()vnode_list[pbs.get_local_nodename()]
this_vnode.state = pbs.ND_OFFLINE
this_vnode.comment = "offlining this vnode"
```

5.2.12.2 Offlining Vnodes Associated with an Event

For example, in a job-related event, you can offline the vnodes and reject the job:

```
for v in pbs.event().vnode_list.keys():
    pbs.event().vnode_list[v].state = pbs.ND_OFFLINE
    pbs.event().vnode_list[v].comment = "Offlining this vnode"
pbs.event().reject("Job tried to run on bad vnodes")
```

5.2.12.3 Using List of Failed Vnodes to Offline Vnodes that Have Gone Bad During Start or Run

For each execjob_prologue and execjob_launch event, PBS records the list of vnodes, with their assigned resources, that are marked as bad by MoM. This list can include those vnodes from sister MoMs that failed to join the job, that rejected an execjob_begin hook or execjob_prologue hook request, or that encountered a communication error while the primary MoM was polling the sister MoM host. PBS records this list in the pbs.event().vnode_list_fail[] hook parameter. This parameter is a *dict* (dictionary of pbs.vnode objects keyed by vnode name).

You can use a hook to walk through this list and offline the bad vnodes. Here is a code snippet:

```
for vn in e.vnode_list_fail:
   v = e.vnode_list_fail[vn]
   pbs.logmsg(pbs.LOG_DEBUG, "offlining %s" % (vn,))
   v.state = pbs.ND_OFFLINE
```

5.2.12.4 Offlining and Clearing Vnodes Using the fail_action Hook Attribute

The way this works is that when a vnode fails a health check in an execjob_begin hook, it is offlined via "offline_vnodes". Once the vnode is offlined, no other jobs are sent to the vnode, so no other execjob_begin hooks will run until the vnode is cleared. You then use "clear_vnodes_upon_recovery" in an exechost_startup hook which runs when the MoM starts up or is HUPed.

5.2.12.4.i Offlining Vnodes Using the fail_action Hook Attribute

You can offline vnodes when an execjob_prologue, execjob_begin or exechost_startup hook fails due to an alarm call or unhandled exception, or when the hook fails due to an internal error such as a full disk or not enough memory on the host, for example, a `malloc()` error.

To offline vnodes upon failure, set the value of the hook's fail_action attribute to include "*offline_vnodes*". This marks the vnodes managed by the hook's MoM as *offline*.

```
# qmgr -c "set hook <hook_name> fail_action += offline_vnodes"
```

When a vnode is offlined using the fail_action attribute, the vnode's comment attribute is set to an explanation:

```
"offlined by hook <hook_name> due to hook error"
```

See section 5.1.9.2, “Using the fail_action Hook Attribute”, on page 37.

5.2.12.4.ii Clearing Vnodes Using the fail_action Hook Attribute

When an exechost_startup hook runs successfully and does not encounter any uncaught exception or alarm timeout, you can clear the *offline* state from vnodes that were previously marked *offline* via fail_action.

To clear the *offline* state from vnodes that were previously offlined via the *"offline_vnodes"* fail_action attribute, set the value of the exechost_startup hook's fail_action attribute to include "*clear_vnodes_upon_recovery*". This clears the *offline* state from the vnodes managed by the hook's MoM.

```
# qmgr -c "set hook <hook_name> fail_action += clear_vnodes_upon_recovery"
```

If you have fixed your execjob_begin script, and want to send jobs again to the vnodes managed by the MoM where the script runs, clear the *offline* states and comments from the vnodes managed by that MoM:

- Clear the *offline* state:

  ```
  # pbsnodes -r <MoM host>
  ```

- Clear the comment:

  ```
  qmgr -c "u n <vn1>,<vn2>,... comment"
  ```

  Or for long lists of vnodes:

  ```
  # qmgr -c "unset node `pbsnodes -vl | awk '{if( NR == 1 ) {printf "%s", $1} else {printf ",%s",
     $1}}'` comment"
  ```

You can write an exechost_periodic hook that monitors the states of the vnodes, so that when it finds offlined vnodes with vnode comment messages matching "`offlined by hook…`", the hook clears the comment and *offline* states.

See section 5.1.9.2, “Using the fail_action Hook Attribute”, on page 37.

5.3 Advice and Caveats for Writing Hooks

5.3.1 Rules for Hook Access and Behavior

The following are rules and recommendations for writing hooks:

- When modifying hooks or their configuration files, do not edit the .CF or .PY files directly. You might think this is a shortcut; it's not. Changes to execution hooks will not be propagated to the MoMs.
- Use only the documented interfaces. Hooks which access PBS information or modify PBS in any way except through these interfaces are erroneous and unsupported.
- Do not attempt to manipulate the hook stored by PBS, except as specified in Chapter 7, "Built-in Hooks", on page 155.
- Don't delete attributes.
- Don't change environment variables set by PBS. See "Environment Variables" on page 232 of the PBS Professional Reference Guide for a list of these environment variables.
- Do not try to access the following (a well-written, portable hook will not depend on any of the following information):
  - Server configuration information: `qmgr`, `resourcedef` and `pbs.conf`
  - Scheduling information: `qmgr`, `sched_config, fairshare, dedicated, holidays`
- Do not write hooks that depend on the behavior of other hooks.
- Do not make assumptions about the value of PATH; use `"import sys"` and modify `sys.path`
- Do not make assumptions about the value of the current working directory.
- For information about `umask`, see"qalter" on page 129 of the PBS Professional Reference Guide, "qsub" on page 215 of the PBS Professional Reference Guide, and "Job Attributes" on page 330 of the PBS Professional Reference Guide.
- Do not depend on order of execution of unrelated hooks. For example, do not depend on one job submission's queuejob hooks running entirely before another job submission's queuejob hooks. It is not guaranteed that all of one job's hooks will finish before another job's hooks start.
- The Resource_List attribute, like others, is a pbs.pbs_resource. These objects support a restricted set of operations. They can reference values by index. Other features, such as has_key(), are not available. See section 6.13.3.19, "Method to Create or Set Resource List", on page 147.
- Hooks which execute PBS commands are erroneous and unsupported. The behavior of executing PBS commands inside a hook is undefined (and is likely to cause the hook to hang).

5.3.2 Check for Parameter Validity

To make hook scripts more robust, check first for the validity of the event parameters before using them, by comparing against *None*:

```
if pbs.event().job != None:
If pbs.event().job_o != None:
If pbs.event().src_queue != None:
If pbs.event().resv != None:
If pbs.event().vnode != None:
If pbs.event().aoe != None:
```

5.3.2.1 Resource Requests and queuejob Hooks

You may want to check resource requests for a queuejob hook If a user submits a job using old `-lnodes` or `-lncpus` syntax, this is translated to a select statement, but only after a queuejob hook has run.

5.3.2.2 Example of Checking Validity

```
% cat t2245.ty
import pbs
e = pbs.event()
if e.type == pbs.QUEUEJOB && (e.job == None):
    e.reject("Event Job parameter is unset!")
elif e.type == pbs.MODIFYJOB && ((e.job == None) || (e.job_o == None)):
    e.reject("Event Job or Job_o parameter is unset!")
elif e.type == pbs.RESVSUB && (e.resv == None):
    e.reject("Event Resv parameter is unset!")
elif e.type == pbs.RUNJOB && (e.job == None):
    e.reject("Event Job parameter is unset!")
```

5.3.3 Make Changes Only On Acceptance

We recommend that your hook does not make changes unless the hook accepts its event. You do not want to have to back changes out upon a reject().

5.3.4 Offline Vnodes when exechost_startup Hook Rejects

We recommend that before calling pbs.event().reject() in an exechost_startup hook, you set the vnodes managed by the local MoM offline with an accompanying comment. This stops jobs from being sent to the affected vnodes. For example:

```
vnlist = pbs.event().vnode_list
for v in vnlist.keys():
    vnlist[v].state = pbs.ND_OFFLINE
    vnlist[v].comment = "bad configuration"
pbs.event().reject("not accepting jobs")
```

5.3.5 Minimize Unnecessary Steps

To speed up your hooks, move any steps to where they are used the fewest times possible. For example, if you retrieve several pieces of information about a job, but only use them if one of them fits a certain criterion, put the bulk of the information-retrieval steps in the section where you do the work on the job.

5.3.6 Use Fast Operations

Some of the examples we provide could be faster. Instead of using "==", you can use the bitwise ampersand operator ("&").

5.3.7 Avoiding Interference with Normal Operation

5.3.7.1 Treat SystemExit as a Normal Occurrence

Both pbs.event().accept() and pbs.event().reject() terminate hook execution by throwing a SystemExit exception. A "try...except" clause without arguments will catch all exceptions. If hook content appears in a "try except " clause, add the following to treat SystemExit as a normal occurrence:

```
except SystemExit:
    pass
```

Here is an example of an except clause that will catch SystemExit:

```
try:
    ...
except:
    ...
```

In the above case, we need to add the except SystemExit, so that it will look like this:

```
try:
    ...
except SystemExit:
    pass
except:
    ...
```

If the existing code has a specified exception, we don't need to add "except SystemExit:", since this hook script is only catching one particular exception and will not match SystemExit. For example:

```
try:
    ...
except pbs.BadAttributeValueError:
    ...
```

5.3.7.2 Allow the Server to Modify Jobs

The server uses the `qalter` command during normal operation to modify jobs. Therefore, if you have a modifyjob hook script, make sure you do not interfere with `qalter` commands issued by the server. Catch these cases by starting the hook with an `if` clause that accepts modification of jobs by PBS:

```
e = pbs.event()
if e.requestor in [ "PBS_Server" ]:
    e.accept()
```

While the scheduler also uses the `qalter` command to modify jobs, this does not trigger any modifyjob hooks.

5.3.7.3 Stay Within the Scheduler Alarm Time

Consider setting hook alarm values in runjob hooks so that they do not unduly delay the scheduler. The scheduler will wait for a hook to finish executing. The scheduler's cycle time has a default value of *20 minutes*, and is specified in the scheduler's sched_cycle_length attribute.

5.3.8 Avoiding Problems

5.3.8.1 Avoid Hook File I/O

When the PBS server is running, `stdout`, `stderr`, and `stdin` are closed. A hook script attempting I/O will get an exception. To avoid this, redirect input and output to a file. See section 8.10.2.8, "Hooks Attempting I/O", on page 225.

5.3.8.2 Avoid Contacting Bad Host

Be careful not to specify a bad host in <job ID> in pbs.event().job.depend. If it references a non-existent or heavily loaded PBS server, the current PBS server could hang for a few minutes as it tries to contact the bad host. For example:

```
pbs.event().job.depend = pbs.depend("after:23.bad_host")
```

The PBS server could hang while trying to contact "bad_host".

5.3.8.3 Avoid os._exit() Python Function

Do not use the os._exit() Python function. It will cause the PBS server to exit.

5.3.8.4 Avoid Attempting to Log Message Using Bad Job ID

If the pbs.logjobmsg() method is passed a bad job ID, it raises a Python ValueError.

5.3.8.5 Avoid Taking Up Lots of Memory

Certain function calls in PBS Python hooks are expensive to use in terms of memory. If they are called repeatedly in loops, they can use up a lot of memory, potentially causing the server to hang or crash. For example, the following is expensive since each iterative call to pbs.server().vnodes() causes internal allocation of memory, which won't be freed until after the hook executes.

In order to avoid this, produce the output only once, save it to memory, and iterate using the copy. For example:

```
vnl = []
vni = pbs.server().vnodes()
for vn in vni:
    pbs.logmsg(pbs.LOG_DEBUG, "found vn.name=%s" %(vn.name))
    vnl.append(vn)
```

The following functions in PBS Python hooks return iterators, and should be used carefully:

- Iterate over a list of jobs:

  pbs.server().jobs()

  pbs.queue.jobs()
- Iterate over a list of queues:

  pbs.server().queues()
- Iterate over a list of vnodes:

  pbs.server().vnodes()
- Iterate over a list of reservations:

  pbs.server().resvs()

5.3.8.6 Testing Vnode State

To see whether a vnode has a particular state set:

```
If v.state == pbs.ND_OFFLINE:
    pbs.logmsg(pbs.LOG_DEBUG, "vnode %s is offline!" % (v.name))
```

5.3.9 Restrictions

5.3.9.1 Local Server Only

Hooks cannot access a server other than the local server. Hooks also cannot specify a non-default server. So for example if a job submission specifies a queue at a server other than the default, the hook can allow that submission, or can change it to the default server, but cannot change it to another non-default server.

5.3.9.2 Dictionary Data Type Restriction

The Python types listed as dictionaries, such as pbs.event().env, support a restricted set of operations. They can reference values by index. Other features, such as has_key(), are not available.

5.3.10 Scheduling Impact of Hooks

5.3.10.1 Effect of runjob Hooks on Preemption

With preemption turned on, the scheduler preempts low-priority jobs to run a high-priority job. If the high-priority job is rejected by a runjob hook, then the scheduler undoes the preemption of the low-priority jobs. Suspended jobs are resumed, and checkpointed jobs are restarted.

5.3.10.2 Effect of runjob Hooks with Strict Ordering

When strict_ordering is set to *True* and backfill_depth is set to *0*, a most-deserving job that is repeatedly rejected by a runjob hook will prevent other jobs from being able to run. A well-written hook would put the job on hold or requeue the job with a later execution time to prevent idling the system.

5.3.10.3 Effect of runjob Hooks with round_robin and by_queue

With round_robin and by_queue set to *True*, a job continually rejected by a runjob hook may prevent other jobs from the same queue from being run. A well-written hook would put the job on hold or requeue the job with a later execution time to allow other jobs in the same queue to be run.

A runjob hook's performance directly affects the responsiveness of the PBS scheduler. Consider carefully the trade-off between the work such a hook needs to do and your scheduler's required performance.

5.3.10.4 Peer Scheduling and Hooks

When a job is pulled from one complex to another, the following happens:

- Hooks are applied at the new complex as if the job had been submitted locally
- Any movejob hooks at the furnishing server are run

5.3.10.5 Performance Considerations

5.3.10.5.i Cost of Accessing Data

- Using pbs.server() to get data about server, queues, jobs, vnodes, or reservations can be slow if run in an execution hook. This is because of the overhead involved when the function has to directly connect to the server and pass requests (via TCP). However, you can speed up reading of custom job resources by setting the m flag. See "Allowing Execution Hooks to Read Custom Job Resources Faster" on page 263 in the PBS Professional Administrator's Guide.
- Making queries to pbs.server().resources_available[] can be slow.

5.3.10.5.ii Cost of Different Hooks

- Any queuejob hooks execute once per job submission
- Any runjob hooks execute once per attempt to run a job, after the scheduler has found a place for it

What this means to the hook writer:

- Your queuejob hooks can generally get away with longer run times
- Any hook that needs to listen to queuejob events needs to be quick to decide whether it is needed or not

For a fast hook, avoid these:

- Running external commands
- Network connections
- File I/O and logging
- Storing information in server or vnode settings
- Using pbs.server().resources_available
- Iterating over the entire set of vnodes or jobs using pbs.server().vnodes() or pbs.server().jobs().

In addition, see section 5.3.5, "Minimize Unnecessary Steps", on page 69 and section 5.3.6, "Use Fast Operations", on page 69.

5.3.10.6 Effect of Hooks on Job Eligible Time

When eligible time is enabled and a job is blocked by a queuejob hook, the job accrues initial_time. When a job is accepted or rejected by modifyjob or movejob hooks, the job continues to accrue whatever kind of time it was accruing. When a job is requeued by a runjob hook or an execution event hook, the scheduler evaluates what kind of time the job should accrue based on resources and policy.

5.3.11 Windows Caveats

5.3.11.1 Special Characters in Pathnames

On Windows, where backslashes may appear in pathnames, escape each backslash with another backslash, or use the raw ('r') operator to form the string. Both of the following work:

```
e = pbs.event()
e.progname = "C:\\Program Files\\PBS\\exec\\bin\\pbsnodes.exe"
e.progname = r"C:\Program Files\PBS\exec\bin\pbsnodes.exe"
```

See section 6.3.3, "Event Object Member Caveats", on page 115.

5.3.11.2 Importing and Exporting Hooks

If the name of <*input_file*> contains spaces, <*input file*> must be quoted.

5.3.11.3 Modifying Events

On Windows, in a multi-vnoded job, be careful modifying pbs.event().progname and pbs.event().argv[] parameters; some values are tacked on by `pbs_mom` and are required. See section 6.3.3.1, "Modifying progname or argv[] Under Windows", on page 115.

5.3.11.4 Using Sleep in a Hook Script

Under Windows, the PBS server or MoM cannot interrupt a hook script executing the Python time.sleep(). The server needs to be able to interrupt the script if the script reaches its timeout. In order to be able to interrupt the script, create a sleep that incrementally sleeps for 1 second. The server can then interrupt the hook script in between the sleeps. For example:

```
import time
def mysleep(sec):
for i in range(sec):
    time.sleep(1)
mysleep(30)          <-- pseudo sleep for 30 seconds
```

6
Hook Objects and Methods

Contents

6.1 The pbs Module . . . 76
6.2 PBS Interface Objects . . . 76
6.2.1 Table of PBS Interface Objects . . . 76
6.2.2 Maps of Object Members and Methods . . . 85
6.3 Event Objects . . . 87
6.3.1 Event Types . . . 87
6.3.2 Event Object Members . . . 108
6.3.3 Event Object Member Caveats . . . 115
6.3.4 Event-only Methods . . . 116
6.3.5 Event Object Method Caveats . . . 116
6.3.6 Examples of Using Event Objects . . . 117
6.4 Server Objects . . . 118
6.4.1 Server Object Members . . . 118
6.4.2 Setting Server Object Members . . . 119
6.4.3 Examples of Using Server Object Members . . . 119
6.4.4 Server Object Methods . . . 119
6.5 Queue Objects . . . 121
6.5.1 Queue Object Members . . . 121
6.5.2 Queue Object Methods . . . 122
6.5.3 Queue Type Constant Objects . . . 122
6.6 Job Objects . . . 122
6.6.1 Job Object Members . . . 123
6.6.2 Setting Job Attributes . . . 127
6.6.3 Examples of Using Job Object Members . . . 127
6.6.4 Job Object Methods for Execution Hooks . . . 127
6.7 The exec_vnode Object . . . 129
6.7.1 The exec_vnode Object Members . . . 129
6.7.2 Using pbs.vchunk Objects in exec_vnode . . . 130
6.7.3 Restrictions on exec_vnode Objects . . . 131
6.8 Chunk Objects . . . 131
6.8.1 Chunk Object Members and Methods . . . 131
6.9 Reservation Objects . . . 131
6.9.1 Reservation Object Members . . . 132
6.9.2 Reservation State Constant Objects . . . 133
6.10 Vnode Objects . . . 133
6.10.1 Vnode Object Members . . . 134
6.10.2 Vnode Type Constant Objects . . . 134
6.10.3 Vnode Sharing Constant Objects . . . 135
6.10.4 Vnode State Constant Objects . . . 135
6.11 Configuration File Objects . . . 135
6.11.1 Variable Containing Hook Configuration File Path . . . 135
6.11.2 Dictionary of PBS Configuration File Entries . . . 136
6.12 Constant Objects . . . 140
6.13 Object Members and Methods . . . 140

6.13.1 PBS Objects and Object Members 141
6.13.2 Methods Available in Events 141
6.13.3 PBS Types and Their Methods 143
6.13.4 Global Methods 151

6.1 The pbs Module

The *pbs module* provides an interface to PBS and the hook environment. The interface is made up of Python objects, members, and methods. You can operate on the objects and use the methods in your Python code. In order to use the pbs module, you must begin your Python code by importing the pbs module. For example, in a script that modifies a job:

```
import pbs
pbs.event().job.comment="Modified this job"
```

For the contents of the pbs module, see section 4.5, “Python Modules and PBS”, on page 24.

6.2 PBS Interface Objects

The PBS interface contains different kinds of objects:

- Objects to represent PBS entities, e.g. jobs, server, queues, vnodes, reservations, events, log messages, etc.
- Objects to represent job, server, vnode, queue, and reservation attributes.
- Objects to represent arguments to PBS commands, PBS version information, etc.
- Constant objects to represent event types, states, log event classes, queue types, and exceptions.

6.2.1 Table of PBS Interface Objects

PBS provides a set of interface objects for use in hooks. The following table lists all of the PBS objects in alphabetical order. Each of these objects is described in detail later in the chapter.

Table 6-1: PBS Interface Objects

PBS Interface Object Description
pbs.acl Represents a PBS ACL . See section 6.13.3.1, “Method to Create or Set ACL”, on page 143.
pbs.args Represents a space-separated list of PBS arguments to commands such as `qsub`, `qdel`. See "Method to Create or Set Command Argument List” on page 143.
pbs.argv[] Argument strings to be passed to the program executed for the job. See section 6.3.2.2, “Job Program Arguments Event Member”, on page 109.
pbs.BadAttributeValueError Raised when setting the member value of a pbs.* object and the value given is invalid. See "Table of Exceptions” on page 43

Table 6-1: PBS Interface Objects

PBS Interface Object Description
pbs.BadAttributeValueTypeError Raised when setting the member value of a pbs.* object and the value type is invalid. See "Table of Exceptions" on page 43
pbs.BadResourceValueError Raised when setting the resource value of a pbs.* object and the value given is invalid. See "Table of Exceptions" on page 43
pbs.BadResourceValueTypeError Raised when setting the resource value of a pbs.* object and the value type is invalid. See "Table of Exceptions" on page 43
pbs.checkpoint Represents a job's checkpoint attribute. See "Job Checkpoint Attribute Member" on page 124
pbs.depend Represents a job's dependency attribute. See "Job depend Attribute Member" on page 124.
pbs.duration Represents a time interval. See "Method to Create or Set Duration from Time String or Integer" on page 144.
pbs.email_list Represents the set of users to whom mail may be sent. Example: Job's Mail_Users attribute. See "Method to Create or Set Email List" on page 144
pbs.env[] Dictionary of environment variables. See section 6.3.2.5, "Job Environment Event Member", on page 110.
pbs.event Represents a PBS event. See "Event Objects" on page 87
pbs.EventIncompatibleError Raised when referencing a nonexistent member in pbs.event(). See "Table of Exceptions" on page 43.
pbs.EXECHOST_PERIODIC Type for an exechost_periodic hook event. See section 6.3.1.18, "exechost_periodic: Periodic Events on All Execution Hosts", on page 107.
pbs.EXECHOST_STARTUP Type for an exechost_startup hook event. See section 6.3.1.17, "exechost_startup: Event When Execution Host Starts Up", on page 106.
pbs.EXECJOB_ATTACH Type for an execjob_attach hook event. See section 6.3.1.11, "execjob_attach: Event when pbs_attach() runs", on page 100.
pbs.EXECJOB_BEGIN Type for an execjob_begin hook event. See section 6.3.1.8, "execjob_begin: Event when Execution Host Receives Job", on page 96.
pbs.EXECJOB_END Type for an execjob_end hook event. See section 6.3.1.16, "execjob_end: Event After Job Cleanup", on page 105.

Table 6-1: PBS Interface Objects

PBS Interface Object Description
pbs.EXECJOB_EPILOGUE Type for an execjob_epilogue hook event. See section 6.3.1.15, “execjob_epilogue: Event Just After Killing Job Tasks”, on page 104.
pbs.EXECJOB_LAUNCH Type for an execjob_launch hook event. See section 6.3.1.10, “execjob_launch: Event when Execution Host Receives Job”, on page 98.
pbs.EXECJOB_POSTSUSPEND Type for an execjob_postsuspend hook event. See section 6.3.1.12, “execjob_postsuspend: Event Just After Suspending Job”, on page 101.
pbs.EXECJOB_PRERESUME Type for an execjob_preresume hook event. See section 6.3.1.13, “execjob_preresume: Event Just Before Resuming Job”, on page 102.
pbs.EXECJOB_PRETERM Type for an execjob_preterm hook event. See section 6.3.1.14, “execjob_preterm: Event Just Before Killing Job Tasks”, on page 103.
pbs.EXECJOB_PROLOGUE Type for an execjob_prologue hook event. See section 6.3.1.9, “execjob_prologue: Event Just Before Execution of Top-level Job Process”, on page 97.
pbs.PERIODIC Type for a periodic hook. See section 6.3.1.7, “periodic: Periodic Event at Server Host”, on page 95.
pbs.exec_host Represents a job's exec_host attribute. See "job.exec_host” on page 124 .
pbs.exec_vnode Represents a job's exec_vnode attribute. See "job.exec_vnode” on page 124 .
pbs.group_list Represents a list of group names. See "job.group_list” on page 124.
pbs.hold_types Represents the Hold_Types attribute of a job. See "job.Hold_Types” on page 124.
pbs.job Represents a PBS job. See "Job Objects” on page 122
pbs.job_list[] List of pbs.job objects. See "pbs.event().job_list” on page 111.
pbs.job_sort_formula Represents the job_sort_formula server attribute. See "pbs.job_sort_formula()” on page 145
pbs.JOB_STATE_BEGUN Job arrays only. Job array has started. See "Job job_state Attribute Member” on page 125

Table 6-1: PBS Interface Objects

PBS Interface Object Description
pbs.JOB_STATE_EXITING Job is exiting after having run. See "Job job_state Attribute Member" on page 125
pbs.JOB_STATE_EXPIRED Subjobs only. Subjob is finished (expired). See "Job job_state Attribute Member" on page 125
pbs.JOB_STATE_FINISHED Job is finished: job executed successfully, job was terminated while running, job execution failed, or job was deleted before execution. See "Job job_state Attribute Member" on page 125.
pbs.JOB_STATE_HELD Job is held. See "Job job_state Attribute Member" on page 125
pbs.JOB_STATE_MOVED Job has been moved to another server. See "Job job_state Attribute Member" on page 125.
pbs.JOB_STATE_QUEUED Job is queued, eligible to run or be routed. See "Job job_state Attribute Member" on page 125
pbs.JOB_STATE_RUNNING Job is running. See "Job job_state Attribute Member" on page 125
pbs.JOB_STATE_SUSPEND Job is suspended by server. See "Job job_state Attribute Member" on page 125
pbs.JOB_STATE_SUSPEND_USERACTIVE Job is suspended due to workstation becoming busy. See "Job job_state Attribute Member" on page 125
pbs.JOB_STATE_TRANSIT Job is in transit. See "Job job_state Attribute Member" on page 125
pbs.JOB_STATE_WAITING Job is waiting for its requested execution time to be reached, or the job's stagein request has failed. See "Job job_state Attribute Member" on page 125
pbs.join_path Represents the job's Join_Path attribute. See "Job Join_Path Attribute Member" on page 125.
pbs.keep_files Represents the Keep_Files job attribute. See "Job Keep_Files Attribute Member" on page 125
pbs.license_count Represents a set of licensing-related counters. Server attribute. See section 6.13.3.14, "Method to Create or Set license_count Object", on page 146.
pbs.LOG_DEBUG Log event class. See "Message Log Level Objects" on page 152
pbs.LOG_ERROR Log event class. See "Message Log Level Objects" on page 152

Table 6-1: PBS Interface Objects

PBS Interface Object Description
pbs.LOG_WARNING Log event class. See "Message Log Level Objects" on page 152
pbs.mail_points Represents the Mail_Points attribute of a job. See "Job Mail_Points Attribute Member" on page 126.
pbs.MODIFYJOB The modifyjob hook event type. Triggered by `qalter` or `pbs_alterjob()` API call. Not triggered by scheduler job modification. See "Event Types" on page 87.
pbs.MOVEJOB The movejob hook event type. Triggered by `qmove` or `pbs_movejob()` API call. See "Event Types" on page 87
pbs.ND_BUSY Represents *busy* vnode state. See section 6.10.4, "Vnode State Constant Objects", on page 135.
pbs.ND_DEFAULT_EXCL Represents *default_excl* sharing vnode attribute value. See section 6.10.3, "Vnode Sharing Constant Objects", on page 135.
pbs.ND_DEFAULT_SHARED Represents *default_shared* sharing vnode attribute value. See section 6.10.3, "Vnode Sharing Constant Objects", on page 135.
pbs.ND_DOWN Represents *down* vnode state. See section 6.10.4, "Vnode State Constant Objects", on page 135.
pbs.ND_FORCE_EXCL Represents *force_excl* sharing vnode attribute value. See section 6.10.3, "Vnode Sharing Constant Objects", on page 135.
pbs.ND_FREE Represents *free* vnode state. See section 6.10.4, "Vnode State Constant Objects", on page 135.
pbs.ND_IGNORE_EXCL Represents *ignore_excl* sharing vnode attribute value. See section 6.10.3, "Vnode Sharing Constant Objects", on page 135.
pbs.ND_JOBBUSY Represents *job-busy* vnode state. See section 6.10.4, "Vnode State Constant Objects", on page 135.
pbs.ND_JOB_EXCLUSIVE Represents *job-exclusive* vnode state. See section 6.10.4, "Vnode State Constant Objects", on page 135.
pbs.ND_OFFLINE Represents *offline* vnode state. See section 6.10.4, "Vnode State Constant Objects", on page 135.
pbs.ND_PBS Represents *pbs* value for vnode ntype attribute. See section 6.10.2, "Vnode Type Constant Objects", on page 134
pbs.ND_PROV Represents *provisioning* vnode state. See section 6.10.4, "Vnode State Constant Objects", on page 135.

Table 6-1: PBS Interface Objects

PBS Interface Object Description
pbs.ND_RESV_EXCLUSIVE Represents *resv-exclusive* vnode state. See section 6.10.4, "Vnode State Constant Objects", on page 135.
pbs.ND_STALE Represents *stale* vnode state. See section 6.10.4, "Vnode State Constant Objects", on page 135.
pbs.ND_STATE_UNKNOWN Represents *state-unknown, down* vnode state. See section 6.10.4, "Vnode State Constant Objects", on page 135.
pbs.ND_UNRESOLVABLE Represents *unresolvable* vnode state. See section 6.10.4, "Vnode State Constant Objects", on page 135.
pbs.ND_WAIT_PROV Represents *wait-provisioning* vnode state. See section 6.10.4, "Vnode State Constant Objects", on page 135.
pbs.node_group_key Represents the node_group_key attribute. See "Method to Create or Set node_group_key Object" on page 146.
pbs.path_list Represents a list of pathnames. See "Method to Create or Set path_list Object" on page 146.
pbs.pbs_conf[] Dictionary of entries in `pbs.conf`. See "pbs.pbs_conf[]" on page 136.
pbs.pbs_resource List of resource names and values. See section 6.13.3.19, "Method to Create or Set Resource List", on page 147.
pbs.pid Represents the process ID of a process belonging to a job.
pbs.place Represents the place specification when submitting a job. See section 6.13.3.20, "Method to Create or Set place Object", on page 147.
pbs.progname Path of job shell or executable. See section 6.3.2.15, "Job Executable Event Member", on page 112.
pbs.QTYPE_EXECUTION Represents *execution* value for queue_type queue attribute. See "Queue Type Constant Objects" on page 122
pbs.QTYPE_ROUTE Represents *route* value for queue_type queue attribute. See "Queue Type Constant Objects" on page 122
pbs.queue Represents a PBS queue. See "Queue Objects" on page 121
pbs.QUEUEJOB The queuejob hook event type. Triggered by `qsub` or `pbs_submit()` API call. See section 6.3.1.3, "queuejob: Event when Job is Queued", on page 91.

Table 6-1: PBS Interface Objects

PBS Interface Object Description
pbs.range Represents a range of numbers referring to job array indices. See section 6.13.3.21, "Method to Create or Set range Object", on page 148.
pbs.resv Represents a PBS reservation. See "Reservation Objects" on page 131
pbs.RESVSUB The resvsub hook event type. Triggered by `pbs_rsub` or `pbs_submitresv()` API call. See section 6.3.1.1, "resvsub: Event when Reservation is Created", on page 90.
pbs.RESV_END The resv_end hook event type. Triggered by end of reservation. See section 6.3.1.2, "resv_end: Event when Reservation Ends", on page 90.
pbs.RESV_STATE_BEING_DELETED The reservation state *RESV_BEING_DELETED.* See "Reservation State Constant Objects" on page 133
pbs.RESV_STATE_CONFIRMED The reservation state *RESV_CONFIRMED.* See "Reservation State Constant Objects" on page 133
pbs.RESV_STATE_DEGRADED The reservation state *RESV_DEGRADED.* See "Reservation State Constant Objects" on page 133
pbs.RESV_STATE_DELETED The reservation state *RESV_DELETED.* See "Reservation State Constant Objects" on page 133
pbs.RESV_STATE_DELETING_JOBS The reservation state *RESV_DELETING_JOBS.* See "Reservation State Constant Objects" on page 133
pbs.RESV_STATE_FINISHED The reservation state *RESV_FINISHED.* See "Reservation State Constant Objects" on page 133
pbs.RESV_STATE_NONE The reservation state *RESV_NONE.* See "Reservation State Constant Objects" on page 133
pbs.RESV_STATE_RUNNING The reservation state *RESV_RUNNING.* See "Reservation State Constant Objects" on page 133
pbs.RESV_STATE_TIME_TO_RUN The reservation state *RESV_TIME_TO_RUN.* See "Reservation State Constant Objects" on page 133
pbs.RESV_STATE_UNCONFIRMED The reservation state *RESV_UNCONFIRMED.* See "Reservation State Constant Objects" on page 133
pbs.RESV_STATE_WAIT The reservation state *RESV_WAIT.* See "Reservation State Constant Objects" on page 133
pbs.route_destinations Represents route_destinations queue attribute. See "Method to Create or Set route_destinations Object" on page 148.

Table 6-1: PBS Interface Objects

PBS Interface Object Description
pbs.RUNJOB The runjob hook event type. Triggered by `qrun` or `pbs_runjob()` API call. See section 6.3.1.6, “runjob: Event Before Job is Received by MoM”, on page 94.
pbs.select Represents the select specification when submitting a job. See "Method to Create or Set select Object” on page 148.
pbs.server Represents the local PBS server. See"Server Objects” on page 118
pbs.size Represents a PBS size type. See "Method to Create or Set size Object” on page 150.
pbs.software Represents a site-dependent software specification resource. See "Method to Create or Set Software Resource Object” on page 150.
pbs.staging_list Represents a list of file stagein or stageout parameters. See "Job stagein and stageout Attribute Members” on page 126.
pbs.state_count Represents a set of job-related state counters. See "Method to Create or Set state_count Object” on page 151.
pbs.SV_STATE_ACTIVE Server state is *Scheduling*. See "Server State Member” on page 119
pbs.SV_STATE_HOT Server state is *Hot_Start*. See "Server State Member” on page 119
pbs.SV_STATE_IDLE Server state is *Idle*. See "Server State Member” on page 119
pbs.SV_STATE_SHUTDEL Server state is *Terminating, Delayed*. See "Server State Member” on page 119
pbs.SV_STATE_SHUTIMM Server state is *Terminating*. See "Server State Member” on page 119
pbs.SV_STATE_SHUTSIG Server state is *Terminating*. See "Server State Member” on page 119
pbs.UnsetAttributeNameError Raised when referencing a non-existent member name of a pbs.* object. See "Table of Exceptions” on page 43
pbs.UnsetResourceNameError Raised when referencing a non-existent resource name of a pbs.* object. See "Table of Exceptions” on page 43
pbs.user_list Represents a list of user names. See section 6.13.3.29, “Method to Create or Set user_list Object”, on page 151.
pbs.vchunk Represents a job chunk. See section 6.8, “Chunk Objects”, on page 131.

Table 6-1: PBS Interface Objects

PBS Interface Object Description
pbs.version Represents version information for PBS. See section 6.13.3.30, “Method to Create or Set PBS Version Object”, on page 151.
pbs.vnode Represents a PBS vnode. See section 6.10, “Vnode Objects”, on page 133.
pbs.vnode_list[] Represents a list of pbs.vnode objects. See section 6.3.2.22, “The Vnode List Event Member”, on page 114
pbs.vnode_list_fail[] Represents a list of unhealthy vnodes as pbs.vnode objects. See section 6.3.2.23, “The Failed Vnode List Event Member”, on page 115
SystemExit Raised when accepting or rejecting an action. See "Table of Exceptions” on page 43

6.2.2 Maps of Object Members and Methods

Figure 6-1 shows a map of the PBS Python objects. All hook event objects have the methods listed in "global methods". Each object also has its own members and methods, as shown. We expand hook event objects in Figure 6-2.

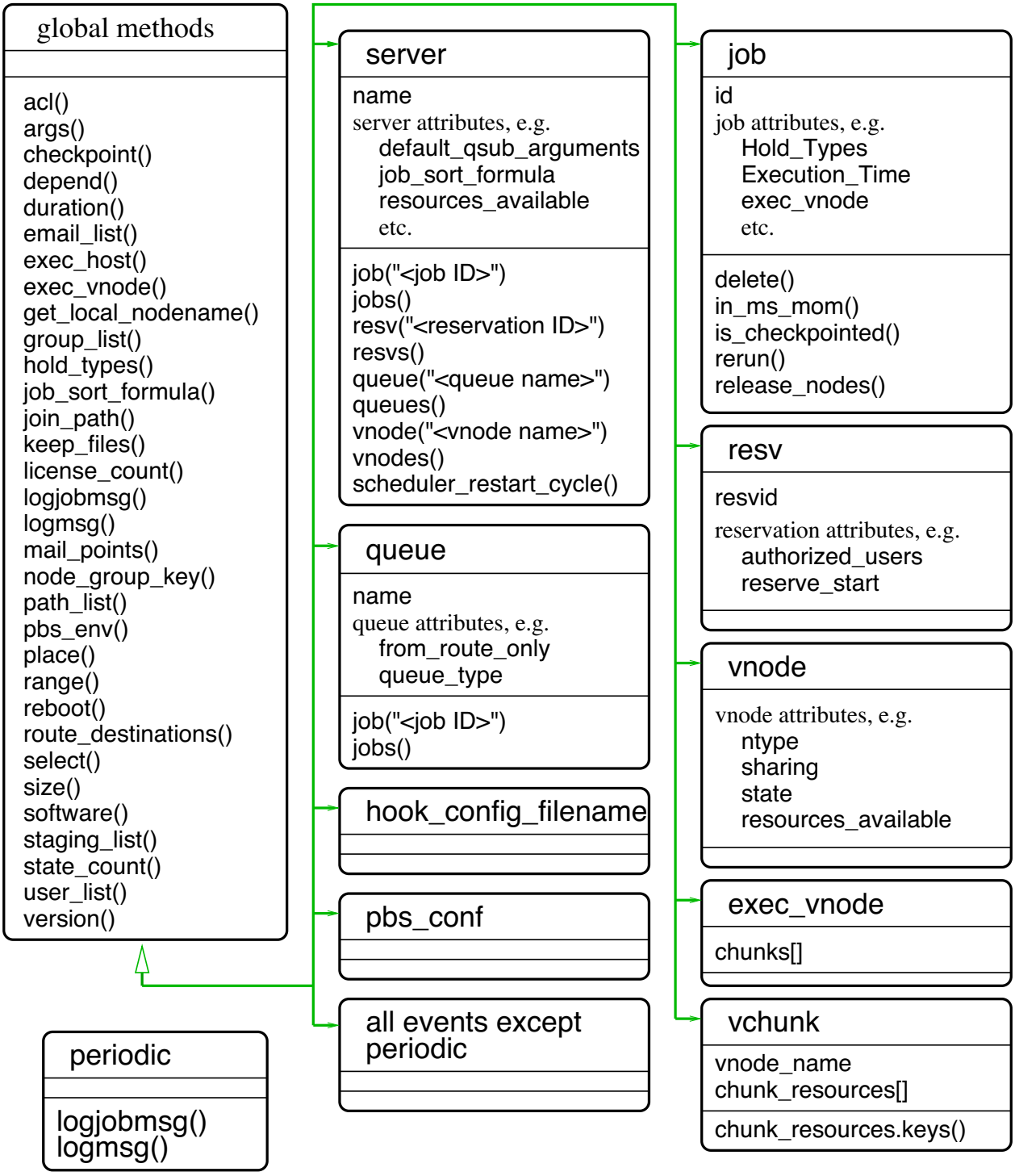

Figure 6-1:Map of members and methods for major PBS objects

Figure 6-2 shows an expanded view of hook event objects. All hook events have the members and methods listed in Figure 6-1, which shows events inheriting global methods. Each type of event also has its own members and/or methods. For example, movejob events have a job member and a src_queue member, in addition to the type, hook_name, requestor, requestor_host, and hook_type members, and the accept(), get_local_nodename(), logjobmsg(), logmsg(), and reject() methods shared by all events. For a description of event objects, see section 6.3, "Event Objects", on page 87.

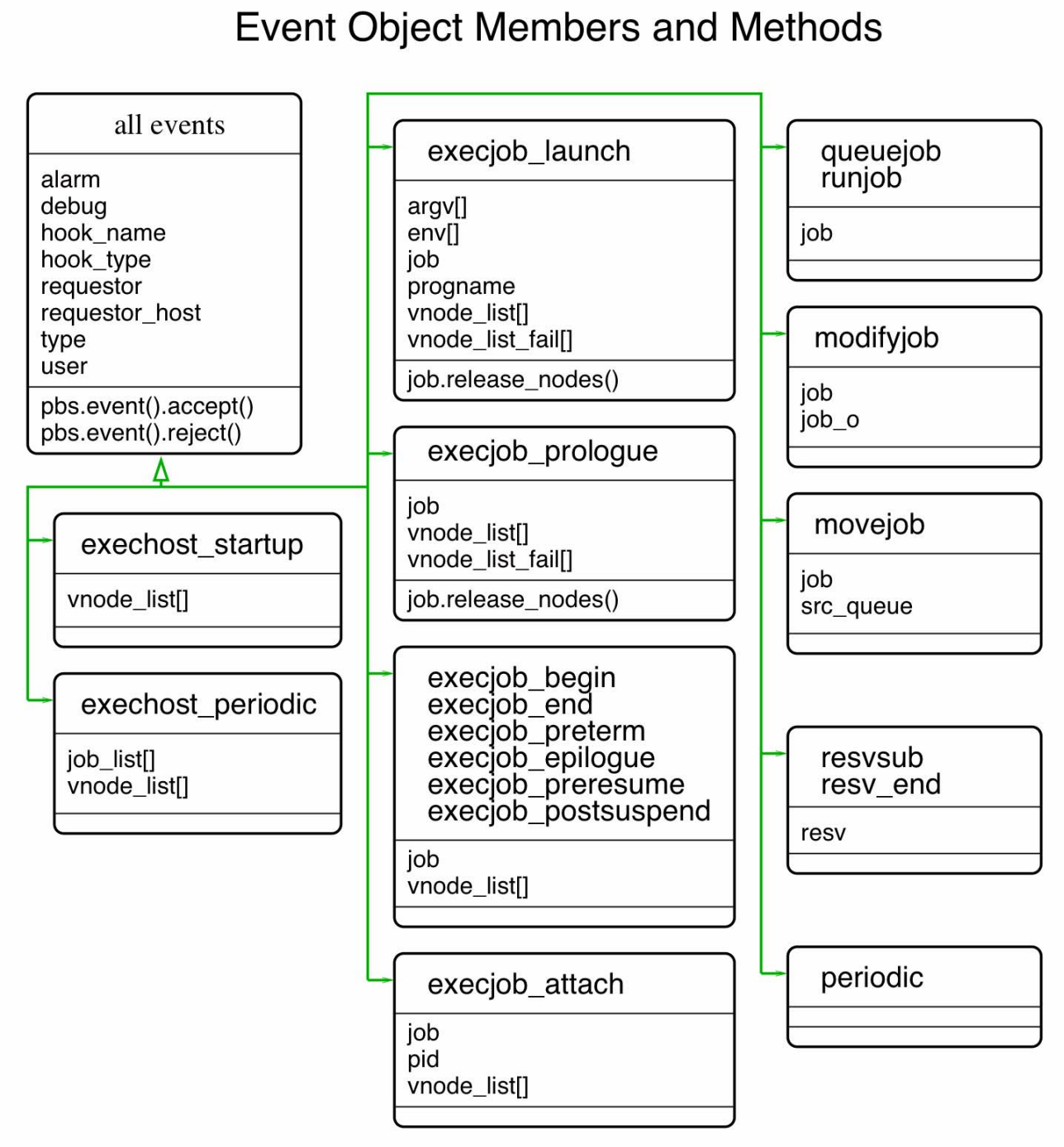

Figure 6-2:Expanded view of event object members and methods

6.3 Event Objects

pbs.event

The event object represents the event that has triggered the hook. You can pass the object to the hook script, and use it in the script. To retrieve objects associated with the event, use this:

pbs.event().<object>

For example, to retrieve the job that triggered an event:

pbs.event().job

There are several types of events. Each type of event is triggered by a different occurrence, and each type has a corresponding hook type. Each type of event has access to different data, and can perform different operations. Some data and operations are common to all events.

Each type of event hook can read and set different job, vnode, and reservation attributes and resources. Each type of event can read different server and queue attributes and resources. We list which attributes and resources can be set for each event in section 5.2.4, "Using Attributes and Resources in Hooks", on page 44.

6.3.1 Event Types

pbs.event().type

The type of the event. Represents the type attribute of the hook. This object can take one or more of the values shown here. The following table summarizes the event types, their constant objects, their triggers, and when and where they run, and gives a pointer to a complete description of the associated hook:

Table 6-2: Event Types and Objects

Event Type & Constant Object	Trigger	Where Run	Description
resvsub pbs.RESVSUB	Triggered by `pbs_rsub` and the `pbs_submitresv()` API call. A resvsub hook is executed after all processing of `pbs_rsub` input, and just before a reservation is created.	At server	See section 6.3.1.1, "resvsub: Event when Reservation is Created", on page 90.
resv_end pbs.RESV_END	Triggered by end of reservation. A resv_end hook is executed when a reservation ends, just before jobs are deleted from the reservation queue.	At server	See section 6.3.1.2, "resv_end: Event when Reservation Ends", on page 90.
queuejob pbs.QUEUEJOB	Triggered by `qsub` and the `pbs_submit()` API call. Not triggered by requeueing a job (`qrerun`) or on node_fail_requeue, when a job is discarded by the MoM because the execution host went down. A queuejob hook is executed after all processing of `qsub` input, and just before the job is queued.	At server	See section 6.3.1.3, "queuejob: Event when Job is Queued", on page 91.

Table 6-2: Event Types and Objects

Event Type & Constant Object	Trigger	Where Run	Description
modifyjob pbs.MODIFYJOB	Triggered by `qalter`, the `pbs_alterjob()` API call, calculating eligible time, and setting the job's comment. A modifyjob hook is executed after all processing of `qalter` input, and just before the job's attributes are modified. Not triggered when the scheduler modifies a job.	At server	See section 6.3.1.4, "modifyjob: Event when Job is Altered", on page 92.
movejob pbs.MOVEJOB	Triggered by `qmove` and the `pbs_movejob()` API call. Not triggered by `pbs_rsub -Wqmove=<job ID>.` A movejob hook is executed after `qmove` arguments are processed, but before a job is moved from one queue to another.	At server	See section 6.3.1.5, "movejob: Event when Job is Moved", on page 92.
runjob pbs.RUNJOB	Triggered by `qrun` and the `pbs_runjob()` API call. A runjob hook is executed just before a job is sent to an execution host.	At server	See section 6.3.1.6, "runjob: Event Before Job is Received by MoM", on page 94.
periodic pbs.PERIODIC	A periodic hook is executed at specified intervals.	At server	See section 6.3.1.7, "periodic: Periodic Event at Server Host", on page 95
execjob_begin pbs.EXECJOB_BEGIN	An execjob_begin hook is executed when MoM receives the job, after any files or directories are staged in.	On primary MoM host, and if successful, on all sister MoM hosts allocated to job	See section 6.3.1.8, "execjob_begin: Event when Execution Host Receives Job", on page 96.
execjob_prologue pbs.EXECJOB_PROLOGUE	An execjob_prologue hook is executed just before the first job process is started.	On primary MoM host, and on all sister MoM hosts where any job task is spawned or attached	See section 6.3.1.9, "execjob_prologue: Event Just Before Execution of Top-level Job Process", on page 97.
execjob_launch pbs.EXECJOB_LAUNCH	An execjob_launch hook is executed just before the user's program is run.	On primary MoM host, and on all sister MoM hosts where MPI tasks are started with `tm_spawn()`	See section 6.3.1.10, "execjob_launch: Event when Execution Host Receives Job", on page 98

Table 6-2: Event Types and Objects

Event Type & Constant Object	Trigger	Where Run	Description
execjob_attach pbs.EXECJOB_ATTACH	An execjob_attach hook is executed before any execjob_prologue hooks run	On each MoM host where `pbs_attach()` runs	See section 6.3.1.11, "execjob_attach: Event when pbs_attach() runs", on page 100.
execjob_postsuspend pbs.EXECJOB_PRETERM	An execjob_postsuspend hook is executed just after successfully suspending the job	On all MoM hosts allocated to the job	See section 6.3.1.12, "execjob_postsuspend: Event Just After Suspending Job", on page 101.
execjob_preresume pbs.EXECJOB_PRETERM	An execjob_preresume hook is executed just before resuming the job	First on the primary MoM host, and if that is successful, on the sister MoM hosts	See section 6.3.1.13, "execjob_preresume: Event Just Before Resuming Job", on page 102.
execjob_preterm pbs.EXECJOB_PRETERM	An execjob_preterm hook is executed when the job receives a termination signal.	On all MoM hosts allocated to the job	See section 6.3.1.14, "execjob_preterm: Event Just Before Killing Job Tasks", on page 103.
execjob_epilogue pbs.EXECJOB_EPILOGUE	An execjob_epilogue hook is executed after all of the job processes have terminated, after executing or killing a job, but before job is cleaned up	On all MoM hosts allocated to the job	See section 6.3.1.15, "execjob_epilogue: Event Just After Killing Job Tasks", on page 104.
execjob_end pbs.EXECJOB_END	An execjob_end hook is executed on all hosts allocated to a job, at the end of all job processing	On all MoM hosts allocated to the job	See section 6.3.1.16, "execjob_end: Event After Job Cleanup", on page 105.
exechost_startup pbs.EXECHOST_STARTUP	An exechost_startup hook is executed when a MoM starts up or receives a HUP (Linux).	On all MoM hosts in the complex.	See section 6.3.1.18, "exechost_periodic: Periodic Events on All Execution Hosts", on page 107.
exechost_periodic pbs.EXECHOST_PERIODIC	An exechost_periodic hook is executed at specified intervals	On all MoM hosts in the complex	See section 6.3.1.18, "exechost_periodic: Periodic Events on All Execution Hosts", on page 107.

6.3.1.1 resvsub: Event when Reservation is Created

6.3.1.1.i Modifying Reservation Creation (pbs_rsub)

- When an advance, standing, or job-specific reservation is created via pbs_rsub, resvsub hooks can modify the reservation's attributes that can be set via pbs_rsub
- When an advance, standing, or job-specific reservation is created, resvsub hooks can specify additional attributes that can be specified via pbs_rsub
- The input reservation attributes on which resvsub hooks operate are those that exist after all pbs_rsub processing of command line arguments is completed
- For resvsub hooks, the input reservation attributes do not include:
  - Server or queue resources_default or default_chunk.
  - Conversions from old syntax (-lnodes & -lncpus) to new select and place syntax

The only time that a reservation can be modified is during its creation. A resvsub event hook can set any settable reservation attribute and any resource that can be specified via pbs_rsub. See Table 5-8, "Reservation Attributes Readable & Settable in resvsub and resv_end Hooks," on page 59 for a complete list of the reservation attributes that this hook can read and set.

6.3.1.1.ii The resvsub Hook Interface

The type for this event is pbs.RESVSUB.

A resvsub hook is executed after all processing of pbs_rsub input, and just before a reservation is created. The hook is triggered by pbs_rsub and the pbs_submitresv() API call.

A reservation object's attributes appear to a resvsub hook as they would be after the event, not before it.

A pbs.RESVSUB event has the following member, in addition to those listed in Table 6-3, "Using Event Object Members in Events," on page 108 and Table 6-14, "Methods Available in Events," on page 142:

pbs.event().resv

A pbs.resv object containing the attributes and resources specified for the reservation being requested. See section 6.9, "Reservation Objects", on page 131.

A pbs.event().accept() terminates hook execution and allows creation of the reservation, and any changes to reservation resources take effect.

A pbs.event().reject() terminates hook execution and causes the reservation not to be created.

6.3.1.2 resv_end: Event when Reservation Ends

A resv_end event hook can read server and reservation attributes. See Table 5-8, "Reservation Attributes Readable & Settable in resvsub and resv_end Hooks," on page 59 for a complete list of the reservation attributes that this hook can read.

6.3.1.2.i The resv_end Hook Interface

The type for this event is pbs.RESV_END.

A resv_end hook is executed when a confirmed reservation ends, and just before jobs are deleted from the reservation queue.

A pbs.RESV_END event has the following member, in addition to those listed in Table 6-3, "Using Event Object Members in Events," on page 108 and Table 6-14, "Methods Available in Events," on page 142:

pbs.event().resv

A pbs.resv object containing the attributes and resources specified for the reservation being requested. See section 6.9, "Reservation Objects", on page 131.

A pbs.event().reject() does not interrupt the execution of the process invoking it.

6.3.1.3 queuejob: Event when Job is Queued

6.3.1.3.i Modifying Job Submission (qsub)

- When a job is submitted via qsub, queuejob hooks can modify the following things explicitly specified in the job submission:
  - Job attributes that can be set via qsub
  - Job comment
  - Resources requested by the job
- When a job is submitted via qsub, queuejob hooks can add resource requests to those specified in the job submission
- The input job attributes on which queuejob hooks operate are those that exist after all qsub processing is completed. These input attributes include:
  - Command line arguments
  - Script directives
  - Server default_qsub_arguments
- When a queuejob hook runs at job submission, the hook can affect only that job.
- For queuejob hooks, the input job attributes do not include:
  - Server or queue resources_default or default_chunk.
  - Conversions from old syntax (-lnodes or -lncpus) to new select and place syntax

See section 5.2.4, "Using Attributes and Resources in Hooks", on page 44, for a complete listing of attributes and resources that this hook can modify.

6.3.1.3.ii The queuejob Hook Interface

The event type for this event is pbs.QUEUEJOB.

A queuejob hook runs after all processing of qsub input, just before the job reaches the server, and before the job is queued, including when a job is peer queued to a server that has a queuejob hook. (See Figure 4-3.) The hook is triggered by qsub or the pbs_submit() API call. A queuejob hook is not triggered by requeueing a job (qrerun) or on node_fail_requeue, when a job is discarded by the MoM because the execution host went down. A queuejob hook runs once per job array.

In a queuejob event, the event's job object members are as they would be if the job were to be successfully submitted.

A pbs.QUEUEJOB event has the following member, in addition to those listed in Table 6-3, "Using Event Object Members in Events," on page 108 and Table 6-14, "Methods Available in Events," on page 142:

pbs.event().job

A pbs.job object with the attributes and resources specified at submission for the job being queued. See section 6.6, "Job Objects", on page 122.

A pbs.event().accept() terminates hook execution and allows the job to be queued, and any changes to job attributes or resources take effect.

A pbs.event().reject() terminates hook execution and causes the job not to be queued. The job is not accepted by the server, and is not assigned a job ID.

6.3.1.3.iii Caveats for queuejob Hook

If a user submits a job using old -lnodes or -lncpus syntax, this is translated to a select statement, but only after a queuejob hook has run. The queuejob hook does have access to the job's resource request.

6.3.1.4 modifyjob: Event when Job is Altered

6.3.1.4.i Modifying Job Change (qalter)

- When a job is changed via qalter, modifyjob hooks can modify the arguments passed to qalter
- When a modifyjob hook runs, it can change the attributes of the job that can be changed via qalter

Before the job runs, this hook can set any job attribute that can be changed via qalter, can set the job's comment, and can set any resource requested by the job.

While the job is running, the only job attributes and resources that the hook can set are those that can be changed via the qalter command: the job's cput and walltime. See section 5.2.4, "Using Attributes and Resources in Hooks", on page 44, for a complete listing of attributes and resources that this hook can modify.

See "qalter" on page 129 of the PBS Professional Reference Guide and "Job Attributes" on page 330 of the PBS Professional Reference Guide.

6.3.1.4.ii The modifyjob Hook Interface

The type for this event is pbs.MODIFYJOB.

A modifyjob hook is executed after all processing of qalter input, and just before the job's attributes are modified. The hook is triggered by the following:

- A qalter command, except when the scheduler calls the command
- The pbs_alterjob() API call
- Calculating eligible time
- Setting the job's comment

A modifyjob hook runs once per job array.

In a modifyjob event hook, the pbs.event().job object's attributes appear to a modifyjob hook as they would be after the job is modified, not before.

A modifyjob event hook shows the original job with all its attributes in pbs.event().job_o.

A pbs.MODIFYJOB event has the following members, in addition to those listed in Table 6-3, "Using Event Object Members in Events," on page 108 and Table 6-14, "Methods Available in Events," on page 142:

pbs.event().job

A pbs.job object representing the job being modified. See section 6.6, "Job Objects", on page 122. In this job object, only attributes and resources that are to be modified by the qalter command are populated. In this job object, attributes or resources that are not slated to be modified are not populated. This job object's attributes appear to a modifyjob hook as they would be after the job is modified, not before.

pbs.event().job_o

A pbs.job object representing the original job, before the job was modified via qalter. All attributes and resources are populated. See section 6.3.2.12, "Original Job Event Member", on page 112.

A pbs.event().accept() terminates hook execution and allows the job to be altered, and any changes to job attributes or resources take effect.

A pbs.event().reject() terminates hook execution and causes the job not to be altered.

6.3.1.5 movejob: Event when Job is Moved

6.3.1.5.i Modifying Job Move (qmove)

- When a job is moved via qmove, movejob hooks can modify the arguments passed to qmove
- When a movejob hook runs, it can change the job's destination queue to any queue on the default server

A movejob hook can specify only local queues as the destination queue. Whether a job is submitted with a local queue or a remote queue as its destination, a movejob hook can change the destination to a local queue.

The only job attribute that a movejob event hook can set is the job's destination queue.

6.3.1.5.ii The movejob Hook Interface

The type for this event is pbs.MOVEJOB.

The server runs its movejob hooks when any of the following happens:

- This server is the furnishing server when peer scheduling a job
- A job is moved from this server to another server via the `qmove` command
- A job is moved between two queues on this server

A movejob hook is executed after `qmove` arguments are processed, but before a job is moved from one queue to another. This hook is triggered by `qmove` and the `pbs_movejob()` API call. movejob hooks are not triggered by `pbs_rsub -Wqmove=<job ID>`. A movejob hook runs once per job array.

A job object's attributes appear to a movejob hook as they would be after the event, not before it.

The hook shows the job's originating queue in the pbs.event().src_queue object member.

A pbs.MOVEJOB event has the following members, in addition to those listed in Table 6-3, “Using Event Object Members in Events,” on page 108 and Table 6-14, “Methods Available in Events,” on page 142:

pbs.event().job

A pbs.job object representing the job being moved. See section 6.6, “Job Objects”, on page 122.

Note that pbs.event().job.queue refers to the destination queue, not the current queue.

pbs.event().src_queue

The pbs.queue object representing the originating queue where pbs.event().job came from.

A pbs.event().accept() terminates hook execution and allows the job to be moved, and any changes to job attributes or resources take effect.

A pbs.event().reject() terminates hook execution and causes the job not to be moved.

6.3.1.6 runjob: Event Before Job is Received by MoM

6.3.1.6.i Changes Before Job is Sent to MoM (`qrun`)

When the scheduler runs a job or the administrator runs a job using the `qrun` command, any runjob hooks are executed.

- On accepting a job, a runjob hook can modify the following:
  - The job's Error_Path attribute
  - The job's Output_Path attribute
  - All of the job's Variable_List attribute members
  - The following Resource_List attribute members:

    cput

    exec_vnode

    file

    max_walltime

    min_walltime

    nice

    pcput

    pmem

    pvmem

    site

    software

    start_time

    walltime
- When a runjob hook rejects a job, it can do the following:
  - Set the job's depend attribute
  - Set any members of the job's Variable_List attribute
  - Place a hold on the job
  - Release a hold on the job
  - Set the job's project attribute
  - Change the time the job is allowed to begin execution
  - Set any of the job's Resource_List attribute members except nodect
  - Change the state of a vnode where the job would have run

See Table 5-6, “Job Attributes Readable & Settable via Events,” on page 55 and Table 5-9, “Built-in Job Resources Readable & Settable by Hooks via Events,” on page 60.

A runjob hook can modify a vnode only if the hook rejects the event, and the vnode is in the job's exec_vnode attribute. For a vnode, the hook can modify only the state attribute. The only pre-execution event hook that can change this attribute is a runjob hook.

6.3.1.6.ii The runjob Hook Interface

The event type is pbs.RUNJOB.

A runjob event occurs when one of the following happens:

- The administrator uses the `qrun` command
- The scheduler chooses to run a job and calls `pbs_runjob()`

A runjob hook is executed just before a job is sent to the execution host. It is triggered by `qrun` and the `pbs_runjob()` API call. A runjob hook runs once per subjob.

For a runjob hook only, job object attributes appear as they would be before the event takes place.

A pbs.RUNJOB event has the following member, in addition to those listed in Table 6-3, "Using Event Object Members in Events," on page 108 and Table 6-14, "Methods Available in Events," on page 142:

pbs.event().job

A pbs.job object representing the job being run. See section 6.6, "Job Objects", on page 122.

A pbs.event().accept() terminates hook execution and allows the job to be sent to the execution host, and any changes to job attributes or resources take effect.

A pbs.event().reject() terminates hook execution and causes the job to be requeued instead of being sent to the execution host. When a job is requeued by this hook, the scheduler considers it for execution in the next scheduling cycle.

6.3.1.7 periodic: Periodic Event at Server Host

6.3.1.7.i Periodic Events at Server Host

Periodically, at the server host, a periodic hook can:

- Run `qstat`, job start time estimator named `pbs_est`, etc.

6.3.1.7.ii The periodic Hook Interface

This event type is pbs.PERIODIC.

The periodic hook runs periodically on the server host, in the background. The hook begins periodic execution, and the interval timer is restarted, when any of the following happens:

- The hook is enabled
- The hook is imported
- The server starts

The periodic hook runs as *pbsadmin*.

The interval between calls to periodic hooks is specified in the freq hook attribute. See section 5.1.13, "Setting Hook Interval (Frequency)", on page 39.

A call to pbs.event().accept() causes any changes made to objects exposed in the hook to take effect.

A call to pbs.event().reject(<message>) prevents any changes from taking effect.

A call to pbs.event().reject(<message>) causes the following messages to appear in the server log:

```
"run_periodic_hook; request rejected by <hook_name>"
<message>
```

The periodic hook continues to be periodically called whether or not there are errors in hook script execution or a call to the pbs.event().reject() action. To stop the hook from being called, either disable it or delete it:

```
#qmgr -c "s h <periodic hook> enabled=f"
#qmgr -c "d h <periodic hook>"
```

If the periodic hook script encounters an unexpected error causing an unhandled exception, or if the script terminates due to a hook alarm, all changes do **not** take effect. In addition, one of the following messages appears in the MoM log at event class PBSEVENT_DEBUG2:

```
"periodic hook <hook_name> encountered an exception, request rejected"
"alarm call while running periodic hook '<hook_name>', request rejected"
```

6.3.1.7.iii Caveats for periodic Event Hooks

The order attribute is ignored for periodic hooks. It does **not** guarantee the execution order of a list of periodic hooks.

6.3.1.8 execjob_begin: Event when Execution Host Receives Job

6.3.1.8.i Changes When Job is Received by MoM

When MoM receives a job, an execjob_begin hook can:

- Modify the job's Execution_Time, Hold_Types, Variable_List, and resources_used attributes
- Flag the job to be rerun
- Kill the job
- Set attributes and resources on the vnode(s) managed by the MoM where this job executes

6.3.1.8.ii The execjob_begin Hook Interface

This event type is pbs.EXECJOB_BEGIN.

An execjob_begin hook executes on the primary MoM host and then, if successful, executes on all the sister MoM hosts allocated to the job. The hook executes when the host first receives the job, after any files or directories are staged in.

A pbs.EXECJOB_BEGIN event has the following members and methods, in addition to those listed in Table 6-3, "Using Event Object Members in Events," on page 108 and Table 6-14, "Methods Available in Events," on page 142:

pbs.event().job

This is a pbs.job object representing the job that is about to run. See section 6.6, "Job Objects", on page 122.

pbs.event().vnode_list[]

This is a dictionary of pbs.vnode objects, keyed by vnode name, listing the vnodes that are assigned to this job. See section 6.3.2.22, "The Vnode List Event Member", on page 114 for information about using pbs.event().vnode_list[].

A call to pbs.event().accept() means the job can proceed with execution, and any changes to job attributes, resources, or the vnode list take effect.

A call to pbs.event().reject(<message>) automatically causes the job to be killed and tells the server to requeue the job. In addition, any changes to job attributes, resources, or vnode list take effect. When a job is requeued by this hook, the scheduler considers it for execution in the next scheduling cycle.

- If the pbs.event().reject(<message>) call is made on a primary execution host, the following message appears in the MoM log at log event class PBSEVENT_DEBUG2:

  "execjob_begin request rejected by <hook_name>"

  <message>

  The rejection message <message> also appears in the STDERR of the program such as qrun invoking pbs_runjob() API:
- If the pbs.event().reject(<message>) call is made on a sister host, the following message appears in the MoM log at log event class PBSEVENT_DEBUG2:

  "execjob_begin request rejected by <hook_name>"

  <message>

  In addition, this message appears in mom_logs on the primary execution host:

  "job_start_error: <hook errno> from node <hostname> could not JOIN_JOB successfully.
- If pbs_runjob() was invoked by the scheduler, the following job comment appears:

  "Not running: PBS Error: <message>"

If the execjob_begin hook script encounters an unexpected error causing an unhandled exception, or if the script terminates due to a hook alarm, the job is automatically killed and the server requeues the job. All job changes, vnode changes, or requests for host reboot or scheduler cycle restarts do **not** take effect. In this case, one of the the following messages appears in the MoM log at event class PBSEVENT_DEBUG2:

```
"execjob_begin hook <hook_name> encountered an exception, request rejected"
"alarm call while running execjob_begin hook '<hook_name>', request rejected"
```

6.3.1.9 execjob_prologue: Event Just Before Execution of Top-level Job Process

6.3.1.9.i Changes Before Job Shell is Executed

Just before a job's top shell is executed, an execjob_prologue hook can:

- Modify the job's Execution_Time, Hold_Types, and resources_used attributes
- Flag the job to be rerun
- Kill the job
- Set attributes and resources on the vnode(s) managed by the MoM where this job executes
- Modify a job's vnode request
- Put a bad vnode in the pbs.event().vnode_list_fail[] list

6.3.1.9.ii The execjob_prologue Hook Interface

This event type is pbs.EXECJOB_PROLOGUE.

An execjob_prologue hook runs on the primary MoM host. If the hook runs successfully on the primary MoM host, an execjob_prologue hook runs on each of the sister MoM hosts allocated to the job. On the primary MoM host, an execjob_prologue hook executes just prior to executing the top-level shell or cmd process of the job. This is where the prologue executes. On a sister MoM host, the hook executes just before the first task of the job on this host is spawned, and before any execjob_launch or execjob_attach hooks. See Table 4-1, “Execution Event Hook Timing,” on page 19.

An execjob_prologue hook overrides a prologue. If an execjob_prologue hook exists and is enabled, MoM executes the hook. Otherwise, she executes the prologue.

A pbs.EXECJOB_PROLOGUE event has the following members and methods, in addition to those listed in Table 6-3, “Using Event Object Members in Events,” on page 108 and Table 6-14, “Methods Available in Events,” on page 142:

pbs.event().job

This is a pbs.job object representing the job that is about to run. See section 6.6, “Job Objects”, on page 122.

pbs.event().job.release_nodes()

This method releases unneeded vnodes from a job's vnode request. See section 6.6.4.4, “Job Object Method to Release Vnodes”, on page 128.

pbs.event().vnode_list[]

This is a dictionary of pbs.vnode objects, keyed by vnode name, listing the vnodes that are assigned to this job. See section 6.3.2.22, “The Vnode List Event Member”, on page 114 for information about using pbs.event().vnode_list[].

This is a pbs.job object representing the job that is about to run. See section 6.6, “Job Objects”, on page 122.

pbs.event().vnode_list_fail[]

This is a dictionary of pbs.vnode objects, keyed by vnode name, listing the vnodes that are assigned to this job but are marked as unhealthy. See section 6.3.2.23, “The Failed Vnode List Event Member”, on page 115 for information about pbs.event().vnode_list_fail[].

A pbs.event().accept() allows the job to continue its normal execution, and any changes to job attributes, resources, or vnode list take effect.

A pbs.event().reject(<message>) causes the job to be killed, and the owning server to requeue the job. Any changes to job attributes, resources, or vnode list take effect. When a job is requeued by this hook, the scheduler considers it for execution in the next scheduling cycle.

- On the primary execution host, the following job-level mom_logs entries appear:
  "execjob_prologue request rejected by <hook_name>"
  <message>
- On a sister vnode, the following job-level mom_logs entries appear:
  "execjob_prologue request rejected by <hook_name>"
  <message>
- In addition, the following message appears in the STDERR of the program invoking the tm_attach() API, such as the pbs_attach() command:
  "a hook has rejected the task manager request"

If the following setting is specified in the hook script, just before issuing a pbs.event().reject(), the job is deleted instead of being requeued:

```
pbs.event().job.delete()
```

If the user attribute of the execjob_prologue hook is set to *pbsuser*, the hook script executes under the context of the job owner (the value of the euser job attribute).

If the execjob_prologue hook script encounters an unexpected error causing an unhandled exception, or if the script terminates due to a hook alarm, the job is killed and the server requeues the job. All job changes, vnode changes, or requests for host reboot or scheduler cycle restarts, do **not** take effect. In addition, one of the following messages appears in the MoM log at event class PBSEVENT_DEBUG2:

"execjob_prologue hook <hook_name> encountered an exception, request rejected"

"alarm call while running execjob_prologue hook '<hook_name>', request rejected"

The standard output and standard error of an execjob_prologue hook script are not connected to the standard output and standard error of the job.

6.3.1.10 execjob_launch: Event when Execution Host Receives Job

6.3.1.10.i Changes Before User Program is Executed

Just before the user's program is executed, an execjob_launch hook can:

- Change the job's top shell or executable
- Change the arguments to the shell or executable
- Change the job's environment variables
- Modify job and vnode attributes
- Modify a job's vnode request
- Put a bad vnode in the pbs.event().vnode_list_fail[] list

An execjob_launch hook cannot modify anything else.

6.3.1.10.ii The execjob_launch Hook Interface

This event type is pbs.EXECJOB_LAUNCH.

An execjob_launch hook runs on the primary MoM host just before executing the user's program. The hook runs on the sister MoM hosts allocated to the job, just before executing the user's program as specified in a tm_spawn() API call, which is called from pbsdsh and pbs_tmrsh.

Any execjob_launch hooks runs after execjob_prologue hooks.

This hook cannot use any of the job's methods.

A pbs.EXECJOB_LAUNCH event hook has access to the following members and methods, in addition to those listed in Table 6-3, “Using Event Object Members in Events,” on page 108 and Table 6-14, “Methods Available in Events,” on page 142:

pbs.event().argv[]

This is a pbs.argv[] object representing the arguments to the shell or executable. See section 6.3.2.2, “Job Program Arguments Event Member”, on page 109.

pbs.event().env

This is a pbs.env[] object representing the job's environment variables. See section 6.3.2.5, “Job Environment Event Member”, on page 110.

pbs.event().job

This is a pbs.job object representing the job that is about to run. See section 6.6, “Job Objects”, on page 122.

pbs.event().job.release_nodes()

This method releases unneeded vnodes from a job's vnode request. See section 6.6.4.4, “Job Object Method to Release Vnodes”, on page 128.

pbs.event().progname

This is a pbs.progname object representing the job shell or executable. See section 6.3.2.15, “Job Executable Event Member”, on page 112.

pbs.event().vnode_list[]

This is a dictionary of pbs.vnode objects, keyed by vnode name, listing the vnodes that are assigned to the job that caused the execjob_launch hook to execute. See section 6.3.2.22, “The Vnode List Event Member”, on page 114 for information about pbs.event().vnode_list[]. This object is read-only for this event. The vnode objects in vnode_list[] cannot be modified. Attempting to modify them result in the following:

- The execjob_launch hook is terminated
- The job ends prematurely with a non-zero Exit_Status value
- The following PBSEVENT_DEBUG2 message appears in mom_logs:

```
"execjob_launch hook 'launch' encountered an exception,  request rejected"
Can only set progname, argv, env event parameters under execjob_launch hook."
```

pbs.event().vnode_list_fail[]

This is a dictionary of pbs.vnode objects, keyed by vnode name, listing the vnodes that are assigned to this job but are marked as unhealthy. Seesection 6.3.2.23, “The Failed Vnode List Event Member”, on page 115 for information about pbs.event().vnode_list_fail[].

A call to pbs.event().accept() means the job can proceed with execution, and any changes to progname, argv[], and env[] take effect. If the hook makes changes to the job's progname, argv[], or env[] parameters, the appropriate PBSEVENT_DEBUG2 message(s) appear in mom_logs for each change in a value:

```
"progname orig: <original_progname>"
"progname new: <updated_progname>"
"argv orig: <original_argv>"
"argv new: <updated_argv>"
"env orig: <original_env>"
"env new: <updated_env>"
```

A call to pbs.event().reject(<message>) causes the job to be terminated with a non-zero Exit_Status value, and the following PBSEVENT_DEBUG2 messages to appear in mom_logs:

```
"execjob_launch" request rejected by '<hook_name>'"
<message>
```

If the execjob_launch hook script encounters an unexpected error causing an unhandled exception, the job is terminated with a non-zero Exit_Status value, and the following PBSEVENT_DEBUG2 messages appear in mom_logs:

```
"execjob_launch hook <hook_name> encountered an exception, request rejected"
```

If the execjob_launch hook script terminates due to a hook alarm, the job is terminated with a non-zero Exit_Status value, and the following PBSEVENT_DEBUG2 messages appear in mom_logs:

```
"alarm call while running execjob_launch hook '<hook_name>', request rejected"
```

6.3.1.11 execjob_attach: Event when pbs_attach() runs

6.3.1.11.i Event when pbs_attach() Runs

When pbs_attach() is called, an execjob_attach hook can accept or reject the procedure where the process ID is attached to the job.

6.3.1.11.ii The execjob_attach Hook Interface

An execjob_attach hook runs on any MoM host where an MPI process is spawned using pbs_attach(). The execjob_attach hook runs for each process ID.

The execjob_attach hook runs before any execjob_prologue hooks run on behalf of the first task. See Table 4-1, "Execution Event Hook Timing," on page 19.

An execjob_attach hook cannot modify any PBS objects.

A pbs.EXECJOB_ATTACH event has the following members and methods, in addition to those listed in Table 6-3, "Using Event Object Members in Events," on page 108 and Table 6-14, "Methods Available in Events," on page 142:

pbs.event().job

This is a pbs.job object representing the job that is about to run. See section 6.6, "Job Objects", on page 122. For this hook, this job object is read-only.

pbs.event().pid

This is a Python int representing the process ID whose session ID is being added to the job tasks list. This hook cannot modify the value of the process ID.

pbs.event().vnode_list[]

This is a dictionary of pbs.vnode objects, keyed by vnode name, listing the vnodes that are assigned to this job. The list of vnodes is read-only for this event. See section 6.3.2.22, "The Vnode List Event Member", on page 114 for information about using pbs.event().vnode_list[].

On a call to pbs.event().accept(), MoM proceeds as usual to add the session ID of the process ID to the job's task list.

On a call to pbs.event().reject(<message>), the following happens:

- Hook execution terminates
- MoM does not get the session ID
- PBS prints the following message in mom_logs at log level PBSEVENT_DEBUG2:

  ```
  "execjob_attach" request rejected by '<hook_name>'"
  <message>
  ```

If the execjob_attach hook script encounters an unhandled exception:

- Hook execution terminates
- MoM does not get the session ID of the process ID
- The following message appears in mom_logs at PBSEVENT_DEBUG2:

  "execjob_attach hook <hook_name> encountered an exception, request rejected"

If the execjob_attach hook script terminates due to a hook alarm, MoM does not get the session ID of the process ID, and the following message appears in mom_logs at PBSEVENT_DEBUG2:

"alarm call while running execjob_attach hook '<hook_name>', request rejected"

6.3.1.11.iii Caveats for execjob_attach Hooks

- Do not attempt to modify pbs.event().pid. If you do:
  - Hook execution is terminated
  - MoM does not get the session ID of the process ID
  - The following messages appear in mom_logs at PBSEVENT_DEBUG2:

    "execjob_attach hook <hook_name> encountered an exception, request rejected"

    "event attribute 'pid' is read-only"
- Do not attempt to modify pbs.event().job or the objects in pbs.event().vnode_list[]. If you do:
  - Hook execution is terminated
  - MoM does not get the session ID of the process ID
  - The following messages appear in mom_logs at PBSEVENT_DEBUG2:

    "execjob_attach hook <hook_name> encountered an exception, request rejected"

    "nothing is settable inside an execjob_attach hook!"

6.3.1.12 execjob_postsuspend: Event Just After Suspending Job

6.3.1.12.i The execjob_postsuspend Hook Interface

This event type is pbs.EXECJOB_POSTSUSPEND.

This hook runs on all of the MoM hosts assigned to a job, after the job has been successfully suspended.

An execjob_postsuspend hook:

- Cannot modify any PBS objects
- Cannot be used to set a fail action
- Must run as *pbsadmin*
- Does not interrupt the flow of suspend/resume

A pbs.EXECJOB_POSTSUSPEND event has the following members and methods, in addition to those listed in Table 6-3, “Using Event Object Members in Events,” on page 108 and Table 6-14, “Methods Available in Events,” on page 142:

pbs.event().job

This is a pbs.job object representing the job that has just been suspended. See section 6.6, “Job Objects”, on page 122. For this hook, this job object is read-only.

pbs.event().vnode_list[]

This is a dictionary of pbs.vnode objects, keyed by vnode name, listing the vnodes that are assigned to this job. The list of vnodes is read-only for this event. See section 6.3.2.22, “The Vnode List Event Member”, on page 114 for information about using pbs.event().vnode_list[].

On a call to pbs.event().accept(), nothing happens.

On a call to pbs.event().reject(<message>), the following happens:

- Hook execution terminates
- PBS prints the following message in mom_logs at log level PBSEVENT_DEBUG2:
  "execjob_postsuspend" request rejected by '<hook_name>'"
  <message>

6.3.1.13 execjob_preresume: Event Just Before Resuming Job

6.3.1.13.i The execjob_preresume Hook Interface

This event type is pbs.EXECJOB_PRERESUME.

This hook runs on the primary MoM host when this MoM receives a request to resume a job, and then if this is successful, the primary MoM sends a request to the sisters to resume the job, at which point this hook runs on the sister MoM hosts. All of the execjob_preresume hooks for a job must succeed in order for the job to resume.

An execjob_preresume hook:

- Cannot modify any PBS objects
- Cannot be used to set a fail action
- Must run as *pbsadmin*

A pbs.EXECJOB_PRERESUME event has the following members and methods, in addition to those listed in Table 6-3, “Using Event Object Members in Events,” on page 108 and Table 6-14, “Methods Available in Events,” on page 142:

pbs.event().job

This is a pbs.job object representing the job that has just been suspended. See section 6.6, “Job Objects”, on page 122. For this hook, this job object is read-only.

pbs.event().vnode_list[]

This is a dictionary of pbs.vnode objects, keyed by vnode name, listing the vnodes that are assigned to this job. The list of vnodes is read-only for this event. See section 6.3.2.22, “The Vnode List Event Member”, on page 114 for information about using pbs.event().vnode_list[].

On a call to pbs.event().accept(), nothing happens

On a call to pbs.event().reject(<message>), the following happens:

- Hook execution terminates
- All MoMs where job processes were running are prevented from resuming the job
- PBS prints the following message in mom_logs at log level PBSEVENT_DEBUG2:
  "execjob_preresume" request rejected by '<hook_name>'"
  <message>

6.3.1.14 execjob_preterm: Event Just Before Killing Job Tasks

6.3.1.14.i Changes Before Job is Killed

Just before a job is killed, an execjob_preterm hook can:

- Modify the job's Execution_Time, Hold_Types, and resources_used attributes
- Flag the job to be rerun
- Kill the job
- Set attributes and resources on the vnode(s) managed by the MoM where this job executes
- Cause the job to keep running

6.3.1.14.ii The execjob_preterm Hook Interface

This event type is pbs.EXECJOB_PRETERM.

An execjob_preterm hook executes on all the MoM hosts allocated to a job. This hook runs only when a `qdel` has been issued. It does not run for any other job termination. For example, it does not run on a `qrerun` or when a job goes over its limit. On the primary MoM host, the hook executes when the job receives a signal from the server for the job to terminate. On a sister MoM host, this hook executes when the sister receives a request from the primary MoM host to terminate the job, just before the sister signals the task on this host to terminate.

A pbs.EXECJOB_PRETERM event has the following members and methods, in addition to those listed in Table 6-3, "Using Event Object Members in Events," on page 108 and Table 6-14, "Methods Available in Events," on page 142:

pbs.event().job

This is a pbs.job object representing the job that is about to run (or be killed). See section 6.6, "Job Objects", on page 122.

pbs.event().vnode_list[]

This is a dictionary of pbs.vnode objects, keyed by vnode name, listing the vnodes that are assigned to this job. See section 6.3.2.22, "The Vnode List Event Member", on page 114 for information about using pbs.event().vnode_list[].

A pbs.event().accept() call allows job cancellation or deletion to happen, and any changes to job attributes, resources, or vnode list take effect.

A pbs.event().reject() call causes the job instance on a vnode to continue running, because the terminate signal is not delivered to the job. Any changes to job attributes, resources, or vnode list take effect.

- On the primary execution host, a pbs.event().reject(<message>) causes the following to appear in the `STDERR` of the program (`qdel`) invoking the `pbs_deljob()` API:

  `"hook rejected request"`
- The following message appears in the MoM log at log event class PBSEVENT_DEBUG2:

  `"execjob_preterm request rejected by <hook_name>"`

  `<message>`
- On a sister host, a pbs.event().reject(<message>) causes the following message to appear in the MoM log at log event class PBSEVENT_DEBUG2:

  `"execjob_preterm request rejected by <hook_name>"`

  `<message>`

If the user attribute of the execjob_preterm hook is set to *pbsuser*, the hook script executes under the context of the job owner (the value of the euser job attribute).

If the execjob_preterm hook script encounters an unexpected error causing an unhandled exception, or if the script terminates due to a hook alarm, the job continues to run, and all job changes, vnode changes, requests for host reboot or scheduler cycle restarts, do **not** take effect. In addition, one of the the following messages appears in the MoM log at event class PBSEVENT_DEBUG2:

```
"execjob_preterm hook <hook_name> encountered an exception, request rejected"
"alarm call while running execjob_preterm hook '<hook_name>', request rejected"
```

6.3.1.15 execjob_epilogue: Event Just After Killing Job Tasks

6.3.1.15.i Changes After Job is Executed

Just after a job is executed, an execjob_epilogue hook can:

- Modify the job's Execution_Time, Hold_Types, and resources_used attributes
- Flag the job to be rerun
- Kill the job
- Set attributes and resources on the vnode(s) managed by the MoM where this job executes
- Use the job's exit status

6.3.1.15.ii The execjob_epilogue Hook Interface

This event type is pbs.EXECJOB_EPILOGUE.

An execjob_epilogue hook executes on all the MoM hosts allocated to the job. On a primary MoM host, the hook executes after all the job tasks/processes on the host have been killed, and basic CPU and memory resource usage information have been logged, but before job processes are cleaned up. This is where the epilogue executes. On a sister MoM host, the hook executes after the sister MoM receives a request to kill the job and has signaled the job tasks to terminate.

When an execjob_epilogue hook modifies the resources_used job attribute, it is modifying only the value counted at the local host. For example, if a job runs on two hosts, using four minutes of CPU time on each host, and the hook changes that to three minutes (and this hook runs at both hosts), the job's final CPU time total is six minutes instead of eight.

An execjob_epilogue hook overrides an epilogue. If an execjob_epilogue hook exists and is enabled, MoM executes the hook. Otherwise, she executes the epilogue.

A pbs.EXECJOB_EPILOGUE event has the following members and methods, in addition to those listed in Table 6-3, “Using Event Object Members in Events,” on page 108 and Table 6-14, “Methods Available in Events,” on page 142:

pbs.event().job

This is a pbs.job object representing the job that just finished. See section 6.6, “Job Objects”, on page 122.

pbs.event().job.Exit_status

A Python int that holds the exit value of the top level shell of the job script. This value is valid only if the hook is executing on a primary execution host.

pbs.event().vnode_list[]

This is a dictionary of pbs.vnode objects, keyed by vnode name, listing the vnodes that are assigned to this job. See section 6.3.2.22, “The Vnode List Event Member”, on page 114 for information about using pbs.event().vnode_list[].

A call to pbs.event().accept() continues the normal end-of-job processing, and any changes to job attributes, resources, or vnode list take effect.

A call to pbs.event().reject() causes the job on the current vnode to exit, and the owning server to completely delete the job. Any changes to job attributes, resources, or vnode list take effect.

- On a primary execution host, a pbs.event().reject(<message>) causes the following message to appear in the MoM log at log event class PBSEVENT_DEBUG2:

  "execjob_epilogue request rejected by <hook_name>"

  <message>
- On a sister host, a pbs.event().reject(<message>) causes the following message to appear in the MoM log at log event class PBSEVENT_DEBUG2:

  "execjob_epilogue request rejected by <hook_name>"

  <message>
- If the following call has been made prior to calling pbs.event().reject(), the owning server requeues the job:

  pbs.event().job.rerun()

If the user attribute of the execjob_epilogue hook is set to *pbsuser*, the hook script executes under the context of the job owner (the value of the euser job attribute).

If the execjob_epilogue hook script encounters an unexpected error causing an unhandled exception, or if the script terminates due to a hook alarm, this causes the job on the current vnode to exit, and the owning server to completely delete the job. All job changes, vnode changes, requests for host reboot or scheduler cycle restarts, do **not** take effect. In addition, one of the following messages appears in the MoM log at event class PBSEVENT_DEBUG2:

"execjob_epilogue hook <hook_name> encountered an exception, request rejected"

"alarm call while running execjob_epilogue hook '<hook_name>', request rejected"

The standard output and standard error of an execjob_epilogue hook script are not connected to the standard output and standard error of the job.

6.3.1.16 execjob_end: Event After Job Cleanup

6.3.1.16.i Changes After Job Finishes or is Killed

Just after a job is cleaned up after it finishes execution or is killed, an execjob_end hook can:

- Set attributes and resources on the vnode(s) managed by the MoM where this job executes
- Use the job's exit status

An execjob_end hook cannot effectively modify the job's Execution_Time and Hold_Types attributes. These changes will not be visible to the server, because the job is already cleaned up and reported.

6.3.1.16.ii The execjob_end Hook Interface

This event type is pbs.EXECJOB_END.

An execjob_end hook executes on all the MoM hosts allocated to a job. The hook is executed after a job is cleaned up.

A pbs.EXECJOB_END event has the following members and methods, in addition to those listed in Table 6-3, “Using Event Object Members in Events,” on page 108 and Table 6-14, “Methods Available in Events,” on page 142:

pbs.event().job

This is a pbs.job object representing the job that just ran. See section 6.6, “Job Objects”, on page 122.

pbs.event().job.Exit_status

A Python int that holds the exit value of the top level shell of the job script. This value is valid only when the hook executes on a primary execution host.

pbs.event().vnode_list[]

This is a dictionary of pbs.vnode objects, keyed by vnode name, listing the vnodes that are assigned to this job. See section 6.3.2.22, "The Vnode List Event Member", on page 114 for information about using pbs.event().vnode_list[].

A call to pbs.event().accept() ends the job, and any changes to job attributes, resources, or vnode list take effect.

A call to pbs.event().reject(<message>) also ends the job, and any changes to job attributes, resources, or vnode list also take effect.

A call to pbs.event().reject(<message>) on a primary execution host causes the following message to appear in the MoM log at log event class PBSEVENT_DEBUG2:

```
"execjob_end request rejected by <hook_name>"
<message>
```

A call to pbs.event().reject(<message>) on a sister host causes the following message to appear in the MoM log at log event class PBSEVENT_DEBUG2:

```
"execjob_end request rejected by <hook_name>"
<message>
```

If the execjob_end hook script encounters an unexpected error causing an unhandled exception, or if the script terminates due to a hook alarm, the job terminates, and all job changes, vnode changes, requests for host reboot or scheduler cycle restarts, do **not** take effect. In addition, one of the following messages appear in the MoM logs at event class PBSEVENT_DEBUG2:

```
"execjob_end hook <hook_name> encountered an exception, request rejected"
"alarm call while running execjob_end hook '<hook_name>', request rejected"
```

6.3.1.17 exechost_startup: Event When Execution Host Starts Up

6.3.1.17.i Event when Execution Host Starts or Receives HUP

When an execution host starts up or receives a HUP, an exechost_startup hook can:

- Create vnodes on local host
- Create custom resources for vnodes
- Offline vnodes that are not ready for use
- Return vnodes to use that have been previously offlined
- Modify the attributes and resources of the vnodes managed by the local MoM

6.3.1.17.ii The exechost_startup Hook Interface

This event type is pbs.EXECHOST_STARTUP.

The exechost_startup hook runs on a MoM host every time its MoM starts up, or when a Linux `pbs_mom` receives a SIGHUP signal. This hook executes after MoM loads `pbs.conf` values, reads `mom_priv/config` values, and runs platform-specific initializations, for example cpuset initialization, including topology data gathering. If there are Version 2 configuration files, this hook sets vnode definitions from those Version 2 configuration files.

The exechost_startup hook runs independently of jobs; it depends only on MoM startup and HUP.

A pbs.EXECHOST_STARTUP event has the following members and methods, in addition to those listed in Table 6-3, "Using Event Object Members in Events," on page 108 and Table 6-14, "Methods Available in Events," on page 142:

pbs.event().vnode_list[]

This is a dictionary of pbs.vnode objects, keyed by vnode name, listing the vnodes that are managed by the MoM where the hook runs. See section 6.3.2.22, "The Vnode List Event Member", on page 114 for information about using pbs.event().vnode_list[].

On a call to pbs.event().accept() or pbs.event().reject(), vnode changes take effect, and MoM continues to run.

A call to pbs.event().reject(<message>) causes the following messages to appear in the MoM log:

```
"exechost_startup" request rejected by hook <hook_name>"
<message>
```

If the exechost_startup hook script encounters an unexpected error causing an unhandled exception:

- Vnode changes do not take effect
- MoM continues to run
- The following message appears at PBSEVENT_DEBUG2 in mom_logs:

```
"exechost_startup hook <hook_name> encountered an exception, request rejected"
```

If the exechost_startup hook script terminates due to a hook alarm, vnode changes do not take effect, MoM continues to run, and the following message appears at PBSEVENT_DEBUG2 in mom_logs:

```
"alarm call while running exechost_startup hook '<hook_name>', request rejected'
```

6.3.1.17.iii Advice on Using exechost_startup Hooks

- We recommend that your hook does not make changes unless the hook accepts its event. You do not want to have to back changes out upon a reject().
- For exceptions, we recommend that you catch them via try... except and accompany them with a call to pbs.event().reject().
- We recommend that before calling pbs.event().reject(), you set the vnodes managed by the local MoM offline with an accompanying comment. This stops jobs from being sent to the affected vnodes. For example:

```
vnlist = pbs.event().vnode_list
for v in vnlist.keys():
    vnlist[v].state = pbs.ND_OFFLINE
    vnlist[v].comment = "bad configuration"
pbs.event().reject("not accepting jobs")
```

6.3.1.18 exechost_periodic: Periodic Events on All Execution Hosts

6.3.1.18.i Periodic Events at Execution Hosts

Periodically, at each execution host, an exechost_periodic hook can:

- Set attributes and resources for any vnode managed by the MoM on the host where the hook runs. This means that an instance of a hook can affect more than one vnode only when the hook is running on a multi-vnode host.
- Set attributes or resources for each job managed by the local MoM.

6.3.1.18.ii The exechost_periodic Hook Interface

This event type is pbs.EXECHOST_PERIODIC.

The exechost_periodic hook runs periodically on all the MoM hosts in the complex.

The interval between calls to exechost_periodic hooks is specified in the freq hook attribute. See section 5.1.13, "Setting Hook Interval (Frequency)", on page 39.

A pbs.EXECHOST_PERIODIC event has the following members and methods, in addition to those listed in Table 6-3, "Using Event Object Members in Events," on page 108 and Table 6-14, "Methods Available in Events," on page 142:

pbs.event().vnode_list[]

This is a dictionary of pbs.vnode objects, keyed by vnode name, listing the vnodes that are managed by the MoM where the hook runs. See section 6.3.2.22, "The Vnode List Event Member", on page 114 for information about using pbs.event().vnode_list[].

pbs.event().job_list[]

List of the pbs.job objects managed by the local MoM. This hook can set the attributes and resources for these jobs. See section 6.3.2.11, "Job List Event Member", on page 111.

A call to pbs.event().accept() or pbs.event().reject(<message>) causes any changes made to vnodes to take effect.

A call to pbs.event().reject(<message>) causes the following messages to appear in the MoM log:

```
"exechost_periodic" request rejected by hook <hook_name>"
<message>
```

The periodic hook continues to be periodically called whether or not there are errors in hook script execution or a call to the pbs.event().reject() action. To stop the hook from being called, either disable it or delete it:

```
#qmgr -c "s h <periodic hook> enabled=f"
#qmgr -c "d h <periodic hook>"
```

If the exechost_periodic hook script encounters an unexpected error causing an unhandled exception, or if the script terminates due to a hook alarm, all vnode changes, requests for host reboot or scheduler cycle restarts, do **not** take effect. In addition, one of the following messages appears in the MoM log at event class PBSEVENT_DEBUG2:

```
"exechost_periodic hook <hook_name> encountered an exception, request rejected"
"alarm call while running exechost_periodic hook '<hook_name>', request rejected"
```

6.3.1.18.iii Caveats for exechost_periodic Event Hooks

The order attribute is ignored for exechost_periodic hooks. It does **not** guarantee the execution order of a list of periodic hooks.

6.3.2 Event Object Members

Some event object members are hook attributes, and some exist as part of the event object but are not hook attributes. The following table summarizes the members for event objects, and shows which event objects have access to each member, and whether the event hook can read and set the member. An "*r*" indicates read, an "*s*" indicates set, and an "*o*" indicates that this member can be set but the action has no effect. See Table 4-1, "Execution Event Hook Timing," on page 19 for more information about why some operations have no effect.

Table 6-3: Using Event Object Members in Events

Event Object Member	queuejob	modifyjob (before run)	movejob	runjob (on reject)	runjob (on accept)	periodic	resvsub	resv_end	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic	provision
pbs.event().alarm	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
pbs.event().argv[]	---	---	---	---	---	---	---	---	---	---	---	r, s	---	---	---	---	---	---	---	---
pbs.event().debug	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
pbs.event().enabled	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
pbs.event().env	---	---	---	---	---	---	---	---	---	---	---	r, s	---	---	---	---	---	---	---	---
pbs.event().fail_action	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
pbs.event().freq	---	---	---	---	r, s	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
pbs.event().hook_name	r	r	r	r	r	---	r	r	r	r	r	r	r	r	r	r	r	r	r	r
pbs.event().hook_type	r	r	r	r	r	---	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Table 6-3: Using Event Object Members in Events

Event Object Member	queuejob	modifyjob (before run)	movejob	runjob (on reject)	runjob (on accept)	periodic	resvsub	resv_end	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic	provision
pbs.event().job	r, s	r, s	r, s	r, s	r, s	---	---	---	r, s	r	r, s	r	r	r	r, s	r, s	r, s	---	---	---
pbs.event().job_list (jobs)	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	r, s	---
pbs.event().job_o	---	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
pbs.event().order	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
pbs.event().pid	---	---	---	---	---	---	---	---	---	r	---	---	---	---	---	---	---	---	---	---
pbs.event().progname	---	---	---	---	---	---	---	---	---	---	---	r, s	---	---	---	---	---	---	---	---
pbs.event().requestor	r	r	r	r	r	---	r	r	r	r	r	r	r	r	r	r	r	r	r	r
pbs.event().requestor_host	r	r	r	r	r	---	r	r	r	r	r	r	r	r	r	r	r	r	r	r
pbs.event().resv	---	---	---	---	---	---	table 5-8	table 5-8	---	---	---	---	---	---	---	---	---	---	---	---
pbs.event().src_queue	---	---	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
pbs.event().type	r	r	r	r	r	---	r	r	r	r	r	r	r	r	r	r	r	r	r	r
pbs.event().user	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
pbs.event().vnode_list[]	---	---	---	---	---	---	---	---	r, s	r	r, s	r	r, s	r, s	r, s	r, s	r, s	r, s	r, s	---
pbs.event().vnode_list_fail[]	---	---	---	---	---	---	---	---	---	---	r	r	---	---	---	---	---	---	---	---

An event object (an object returned by pbs.event()) has one or more of the following members:

6.3.2.1 Hook Alarm Event Member

pbs.event().alarm

Hook attribute. Number of seconds to allow a hook to run before the hook times out. Must be greater than zero. See "alarm" on page 352 of the PBS Professional Reference Guide.

Type: *Integer*

6.3.2.2 Job Program Arguments Event Member

pbs.event().argv[]

The list of arguments to be passed to the job script. The arguments can be modified in an execjob_launch hook.

Type: *Python list of strings*

To add another argument to the argument list, append it:

```
pbs.event().argv.append(<new_argument>)
```

To clear out existing argv[] entries and supply a new set of arguments, use the following:

```
pbs.event().argv = []      (sets argv[] to empty list)
pbs.event().argv.append(<arg0>)
pbs.event().argv.append(<arg1>)
…
pbs.event().argv.append(<argN>)
```

On Windows, where backslashes may appear in pathnames, escape each backslash with another backslash, or use the raw ('r') operator to form the string. Both of the following work:

```
e = pbs.event()
e.progname = "C:\\Program Files\\PBS Pro\\exec\\bin\\pbsnodes.exe"
e.progname = r"C:\Program Files\PBS Pro\exec\bin\pbsnodes.exe"
```

See section 6.3.3, "Event Object Member Caveats", on page 115.

To log the arguments to the program, and update some of them:

```
for a in pbs.event().argv:
    pbs.logmsg(pbs.LOG_DEBUG, "a=%s" % (a,))

argv = pbs.event().argv
argv[1] = "beta"
argv[3] = "gamma"
```

6.3.2.3 Hook Debug Behavior Indicator Event Member

pbs.event().debug

Hook attribute. Specifies whether or not the hook produces debugging files under PBS_HOME/server_priv/hooks/tmp or PBS_HOME/mom_priv/hooks/tmp. Files are named *hook_<hook event>_<hook name>_<unique ID>.in*, *.data*, and *.out*. See "Producing Files for Debugging" on page 159 in the PBS Professional Hooks Guide, and "debug" on page 352 of the PBS Professional Reference Guide.

Type: *Boolean*

6.3.2.4 Hook Enable or Disable Event Member

pbs.event().enabled

Hook attribute. Specifies whether or not the hook is enabled. See "enabled" on page 352 of the PBS Professional Reference Guide.

Type: *Boolean*

6.3.2.5 Job Environment Event Member

pbs.event().env

The job's environment. Can be modified in an execjob_launch hook.

Type: dictionary of *environment "<variable>=<value>"* entries, with <*variable*> serving as the dictionary key.

To modify a particular environment entry:

```
pbs.event().env[<variable>] = <value>
```

To add more entries to the env[] dictionary:

```
pbs.event().env[<new_var>] = <value>
```

To clear out existing env[] entries and specify a new environment:

```
pbs.event().env = pbs.pbs_env()
pbs.event().env[<var1>] = <value1>
pbs.event().env[<var2>] = <value2>
…
pbs.event[<varN>.] = <valueN>
```

To unset an existing environment variable:

```
pbs.event().env[<var>] = None
```

To embed a comma in an environment variable, escape the value with single quotes:

```
pbs.event().env[<var>] = '"<value>"'
```

On Windows, where backslashes appear in pathnames, either escape the backslash with another backslash, or use the raw ('r') operator to form the string. Both of the following examples will work:

```
e = pbs.event()
e.progname = "C:\\Program Files\\PBS Pro\\exec\\bin\\pbsnodes.exe"
e.progname = r"C:\Program Files\PBS Pro\exec\bin\pbsnodes.exe"
```

See section 6.3.3, "Event Object Member Caveats", on page 115.

Example 6-1: To log the contents of a job's environment variables:

```
for v in pbs.event().env.keys():
    e = pbs.event().env[v]
    pbs.logmsg(pbs.LOG_DEBUG, "env[%s]=%s" % (v,e))
```

6.3.2.6 Failure Action Event Member

pbs.event().fail_action

Hook attribute. Action to take on hook failure or on subsequent successful execution. See "fail_action" on page 354 of the PBS Professional Reference Guide.

6.3.2.7 Frequency Event Member

pbs.event().freq

Hook attribute. Frequency at which to run hook. See "freq" on page 354 of the PBS Professional Reference Guide.

6.3.2.8 Hook Name Event Member

pbs.event().hook_name

Name of the hook being executed.

Type: *str*

6.3.2.9 Hook Type Event Member

pbs.event().hook_type

The type of the hook. The only valid value is "*site*". Represents the Type hook attribute.

Type: *str*

6.3.2.10 Job Event Member

pbs.event().job

The job that triggered the event. A pbs.job object. See section 6.6, "Job Objects", on page 122.

6.3.2.11 Job List Event Member

pbs.event().job_list

The list of jobs managed by the local MoM. Each job is a pbs.job, described in section 6.6, "Job Objects", on page 122.

For a list of settable attributes and resources, see Table 5-6, “Job Attributes Readable & Settable via Events,” on page 55 and Table 5-9, “Built-in Job Resources Readable & Settable by Hooks via Events,” on page 60.

Type: dictionary of pbs.job objects

To print the jobs in the list:

```
for k in pbs.event().job_list.keys():
    print pbs.event().job_list[k]
```

To set a job attribute or resource for all jobs in the list:

```
for k in pbs.event().job_list.keys():
    pbs.event().job_list[k].<attribute> = <value>
```

In an exechost_periodic hook, attributes are set after the hook ends in a call to pbs.event().accept() or pbs.event().reject(), but not when the hook encounters an uncaught exception or hits an alarm call.

In an exechost_periodic hook, you can flag a job to be requeued using rerun() or deleted using delete() when the server is notified that the job has terminated.

Example 6-2: Rerun all jobs in this MoM's job list:

```
% cat period.py
import pbs
for k in pbs.event().job_list.keys():
    pbs.event().job_list[k].rerun()
```

6.3.2.12 Original Job Event Member

pbs.event().job_o

This is a pbs.job object representing the original job, before the job was modified via `qalter`. All resources and attributes are populated.

See section 6.6, “Job Objects”, on page 122.

6.3.2.13 Order Event Member

pbs.event().order

Hook attribute. Order in which to run hook. See “order” on page 354 of the PBS Professional Reference Guide.

6.3.2.14 Process ID Event Member

pbs.event().pid

The process ID of a task belonging to a job.

Type: *int*

6.3.2.15 Job Executable Event Member

pbs.event().progname

The path to the job shell or executable. This is settable in an execjob_launch hook as follows:

```
pbs.event().progname = "<path_to_the_script>"
```

When setting the value, specify the full path. Otherwise, the path may not be found, and the shell or executable may not run.

Type: *str*

6.3.2.16 Requestor Event Member

pbs.event().requestor

The requestor of the event.

PBS daemons can request actions. If a daemon requests an action, the requestor member contains one of "*PBS_Server*", "*Scheduler*", or "*pbs_mom*". If the requestor is root, the member contains "*root*".

For Windows systems, if the requestor is the administrator, the member contains the account name of the administrator.

Type: *str*

6.3.2.17 Requestor Host Event Member

pbs.event().requestor_host

The name of the host from which the event was requested.

Type: *str*

6.3.2.18 Reservation Event Member

pbs.event().resv

The reservation being requested in a resvsub event or ending in a resv_end event. See section 6.9, "Reservation Objects", on page 131.

6.3.2.19 Source Queue Event Member

pbs.event().src_queue

The pbs.queue object representing the original queue where pbs.event().job came from.

See section 6.5, "Queue Objects", on page 121.

6.3.2.20 Event Type Event Member

pbs.event().type

Hook attribute. The event type, for example, queuejob or movejob. Valid values: one of the PBS event type constants listed in section 6.3.1, "Event Types", on page 87. See "type" on page 354 of the PBS Professional Reference Guide.

Type: A PBS event type constant, such as pbs.QUEUEJOB, pbs.RESVSUB

6.3.2.21 Event User Event Member

pbs.event().user

Hook attribute. The username under which the hook executes. See "user" on page 354 of the PBS Professional Reference Guide.

Valid values: *pbsadmin*, *pbsuser*.

pbsadmin
: On Linux, this is root. On Windows, this is simply a substitute for the PBS service account; it is not the name of the PBS service account.

pbsuser
: The hook runs under the account of the job owner, which is the value of the euser job attribute. Can be used for execjob_prologue, execjob_epilogue, execjob_preterm events only.

Default value: *pbsadmin*

Type: *String*, *str*

6.3.2.22 The Vnode List Event Member

pbs.event().vnode_list[]

Execution event hooks have access to the list of vnodes assigned to the job. Periodic event hooks have access to the list of vnodes managed by the local MoM. The exechost_startup hook can create and modify the vnodes managed by the local MoM.

When a vnode is in such a list, the hook has access to the attributes and resources of that vnode. Table 6-3, "Using Event Object Members in Events," on page 108 lists which hooks can operate on vnode_list[]. Table 5-7, "Vnode Attributes Readable & Settable via Events," on page 57 and Table 5-10, "Vnode Resources Readable & Settable by Hooks via Events," on page 61 show which hooks can read and/or set each vnode attribute or resource.

When this list is retrieved through an execution event, it is associated with a job, and only vnodes assigned to the job have attributes, resources_available, resources_assigned.ncpus, and resources_assigned.mem filled in; on other vnodes, only pbs.vnode().name is available. See section 6.10, "Vnode Objects", on page 133.

You can use an exechost_startup hook to create vnodes on the host where the hook runs:

```
pbs.event().vnode_list[<new vnode>] = pbs.vnode(<new vnode name>)
```

If you want to use an execjob_ hook to manipulate a vnode that is not assigned to the job, but is still managed by the hook's MoM, you must first instantiate the object for that vnode with the name of the new vnode:

```
pbs.event().vnode_list[<new vnode>] = pbs.vnode(<new vnode name>)
```

Once you have instantiated your new vnode (which must still be managed by your hook's MoM), you can operate on it as shown here:

- To list all vnodes:

```
for v in pbs.event().vnode_list.keys():
    pbs.logmsg(pbs.LOG_DEBUG, "found vnode %s" % (pbs.event().vnode_list[v].name))
```

- To get the vnode managed by the local MoM, use the pbs.get_local_nodename() function to return the local parent vnode name where this hook is executing, and then use pbs.event().vnode_list[<local parent vnode name>].

```
local_node = pbs.get_local_nodename()
```

- To find the other vnodes managed by the hook's MoM:

  a. Query the server for its list of vnodes:

  ```
  pbs.server().vnodes()
  ```

  b. Look in the Mom attribute in the resulting list of vnodes for a match to the output of:

  ```
  pbs.get_local_nodename()
  ```

  On a Cray, the Mom attribute may be a comma-separated list of strings, and you may have to check each one. Furthermore, get_local_nodename() returns the short hostname, and you may need to use instead the canonical hostname. If the name resolution on the server is consistent with the name resolution on the execution nodes, you can use the following, which returns in host_canon_name the name to be matched against Mom attribute of each vnode_list element:

  ```
  import socket
  [...]
  host_short_name = pbs.get_local_nodename()
  host_canon_name, host_aliases, host_addresses = socket.gethostbyname_ex(host_short_name)
  ```

To run your code only for vnodes managed by the specified MoM:

```
vnlist = pbs.event().vnode_list
for v in vnlist.keys():
   if host_canon_name in vnlist[v].Mom.split(","):
      [code]
```

- Setting and unsetting attributes and resources:

  To set the attributes and resources for a particular vnode:

  pbs.event().vnode_list[<vnode name>].<attribute> = <value>

  pbs.event().vnode_list[<vnode name>].<resources_available>["<resource name>"] = <value>

  You can unset a resource value by specifying "*None*" as its value:

  pbs.event().vnode_list[<vnode_name>].resources_available["<resource name>"] = None

  Resource names and string values must be quoted.

  For details and examples, see section 5.2.4.11, "Setting and Unsetting Vnode Resources and Attributes", on page 48.

- You can add new custom host-level, non-consumable resources and their values to resources_available for a vnode:

  vnode_list[<vnode name>].resources_available[<new resource>] = <value>

  For details and examples, see section 5.2.7, "Adding Custom Non-consumable Host-level Resources", on page 64.

You cannot modify a vnode that is managed by a different MoM from where the hook is running. If you try to do this, the following error message appears in the server's log at log event class PBSEVENT_DEBUG2:

```
"<node_host_name>; Not allowed to update <vnode name>, as it is owned by a different mom"
```

A hook that runs as "*pbsuser*" (execjob_prologue, execjob_epilogue, execjob_preterm) is not allowed to manipulate pbs.event().vnode_list[], unless the executing user is a PBS Manager or Operator. If a hook running as an unprivileged user tries to change pbs.event().vnode_list[], the following error message appears in the server's log at log event class PBSEVENT_DEBUG2:

```
"<node_host_name>; Not allowed to update vnodes or to request scheduler restart cycle, if run as
   a non-manager/operator user"
```

6.3.2.23 The Failed Vnode List Event Member

pbs.event().vnode_list_fail[]

For each execjob_prologue and execjob_launch event, PBS records the list of vnodes, with their assigned resources, that are marked as bad by MoM. This list can include those vnodes from sister MoMs that failed to join the job, that rejected an execjob_begin hook or execjob_prologue hook request, or that encountered a communication error while the primary MoM was polling the sister MoM host. PBS records this list in the pbs.event().vnode_list_fail[] hook parameter. For how vnodes are marked as failed, see "Checking Vnodes and Marking Them as Failed" on page 440 of the PBS Professional Administrator's Guide.

Type: *dict* (dictionary of pbs.vnode objects keyed by vnode name)

6.3.3 Event Object Member Caveats

6.3.3.1 Modifying progname or argv[] Under Windows

On Windows, in a multi-vnoded job, be careful modifying pbs.event().progname and pbs.event().argv[] parameters; some values are tacked on by pbs_mom and are required. For example, if a multi-vnode job has in its script:

```
pbsdsh -n 1 cmd.exe /C echo hi
```

This causes an installed execjob_launch hook to execute on the sister MoM specified at node index '*1*'. The execjob_launch hook sees:

```
pbs.event().progname=cmd.exe

pbs.event().argv[0]=cmd.exe
pbs.event().argv[1]=/c
pbs.event().argv[2]=C:/PROGRA~1/PBSPRO~1/exec/sbin/mom_open_demux.exe
pbs.event().argv[3]=174.host1
pbs.event().argv[5]=cmd.exe
pbs.event().argv[6]=/C
pbs.event().argv[7]=echo
pbs.event().argv[8]=hi
```

It is important not to modify pbs.event().progname and pbs.event().argv[0],...,pbs.event().argv[3]. These are automatically added by pbs_mom for execution and collecting output.

You can modify pbs.event().argv[] values starting at index *5*, and you can use pbs.event().argv.extend() to add more arguments. Here we modify values for indices *5* through *8*, and add pbs.event().argv[9], making it "*hello*":

```
pbs.event().argv[5] = "pbsnodes.exe"
pbs.event().argv[6] = "-a"
pbs.event().argv[7] = ""
pbs.event().argv[8] = ""
pbs.event().argv.extend(["hello"])
```

6.3.4 Event-only Methods

6.3.4.1 Event Method for Accepting Event

pbs.event().accept()

Terminates hook execution and causes PBS to perform the associated action.

6.3.4.2 Event Method for Rejecting Event

pbs.event().reject()

pbs.event().reject(["<error message>"][,<error code>])

Terminates hook execution and instructs PBS to not perform the associated action. If the <message> argument is given, it is shown in the appropriate PBS daemon log, and in the stderr of the PBS command that caused this event to take place.

By default, pbs.event().reject() returns *255*. To return an error code other than *255*, specify a value between *2* and *255* in the optional <error code>.

6.3.5 Event Object Method Caveats

pbs.event().accept() terminates hook execution by throwing a SystemExit exception. So if hook content appears in a try…except clause that has no arguments to the except clause, always add the following to treat SystemExit as a normal occurrence:

```
except SystemExit:
    pass
```

See section 5.3.7.1, "Treat SystemExit as a Normal Occurrence", on page 70.

6.3.6 Examples of Using Event Objects

Example 6-3: Inside a hook script, create a PBS event object:

```
e = pbs.event()
```

Example 6-4: Get the event type:

```
type = e.type
```

Example 6-5: Get the user who requested the event action:

```
who = e.requestor
```

Example 6-6: Get the host where the request came from:

```
host = e.requestor_host
```

Example 6-7: The event type is pbs.QUEUEJOB. Get the number of CPUs requested for the job being queued:

```
j = e.job
res = j.Resource_List["ncpus"]
```

Example 6-8: Reset the number of CPUs requested by the job:

```
j.Resource_List["ncpus"] = 1
```

Example 6-9: The event type is pbs.MOVEJOB. Get the request parameters:

```
j = e.job
q = j.queue
```

Example 6-10: Accept an event request:

```
e.accept()
```

Example 6-11: Reject an event request:

```
e.reject("Can't set interactive attribute")
```

Example 6-12: Put a job into a wait state and requeue the job in 3600 seconds (1 hour):

```
import time
...
j.Execution_Time = time.time() + 3600
```

Example 6-13: Put a hold on a job:

```
j = pbs.event().job
j.Hold_Types = pbs.hold_types("u")
j.Hold_Types = pbs.hold_types("uo")
j.Hold_Types += pbs.hold_types("s")
```

or

```
j.Hold_Types = pbs.hold_types("<hold_list>")
```

Example 6-14: Release a hold on a job:

```
j.Hold_Types -= pbs.hold_types("un")
j.Hold_Types -= pbs.hold_types("sp")
j.Hold_Types -= pbs.hold_types("o")
```

or

```
j.Hold_Types -= pbs.hold_types("<hold_list>")
```

6.4 Server Objects

pbs.server

This object represents a PBS server. This object can either represent the local server, or be just a coding construct, not representing an actual server. If it represents the local server, you can read but cannot set its attributes. If it is just a coding construct that does not represent an actual server, you can set its attributes. **You cannot alter the PBS server.** If this server object represents the PBS server, it is the server at which the triggering event is taking place, and at which the hook is executing. The only PBS server available to hooks is the local server.

s = pbs.server(["<name>"])

Creates an instance of a PBS server object. If <*name*> is not specified, the object represents the default server.

You can use pbs.server() to retrieve server, queue, job, vnode, and reservation information, and pass it to a hook script. You **cannot** set attributes or resources for objects that are retrieved through the server via pbs.server().

6.4.1 Server Object Members

Some server object members are server attributes, and some are not. A pbs.server has the following members:

6.4.1.1 Server Name Member

pbs.server().name

The server hostname.

Example: `myhost.mydomain.com`

This member is read-only.

Python type: *str*

6.4.1.2 Server Attribute Members

pbs.server().<attribute name>

The PBS server attribute named <*attribute name*>. The pbs.server object has a member to represent each server attribute, spelled exactly like the attribute. For information about using attributes, see section 5.2.4, “Using Attributes and Resources in Hooks”, on page 44.

Server attributes are listed in “Server Attributes” on page 283 of the PBS Professional Reference Guide. Attribute creation methods are described in section 6.13.3, “PBS Types and Their Methods”, on page 143.

6.4.1.3 Server State Member

pbs.server().server_state

The server_state server attribute. It can take one of the following values, represented by constant objects:

Table 6-4: Server State Constant Objects

Object	State
pbs.SV_STATE_IDLE	*Idle*
pbs.SV_STATE_ACTIVE	*Scheduling*
pbs.SV_STATE_HOT	*Hot_Start*
pbs.SV_STATE_SHUTDEL	*Terminating, Delayed*
pbs.SV_STATE_SHUTIMM	*Terminating*
pbs.SV_STATE_SHUTSIG	*Terminating*

6.4.2 Setting Server Object Members

If the server object does not represent the PBS server, you can set, but not unset, server object members. If a server object does represent the PBS server, you cannot set values for object members. To set the value for the server attribute named *<attribute name>* to *<attribute value>*, where *s* is an instance of pbs.server:

```
s.<attribute name> = <attribute value>
```

6.4.3 Examples of Using Server Object Members

```
s = pbs.server()
```

Example 6-15: Get server name:

```
name = s.name
```

Example 6-16: Get the value of the server attribute pbs_license_min:

```
min = s.pbs_license_min
```

6.4.4 Server Object Methods

6.4.4.1 Method to Return Job

pbs.server().job('<job ID>')

Returns a pbs.job object for the job with ID <*id*>, residing on the local server. Returns *None* if the job with ID <*id*> does not exist at the server. See section 6.6, "Job Objects", on page 122.

6.4.4.2 Method to Return Job Iterator

pbs.server().jobs()

Returns a Python iterator that iterates over a list of pbs.job objects residing on the local server. Returns an empty iterator if no jobs exist on the local server. See section 6.6, "Job Objects", on page 122.

Example:

```
for j in s.jobs():
   pbs.logmsg(pbs.LOG_DEBUG, "found job %s" % (j.id))
```

6.4.4.3 Method to Return Queue

pbs.server().queue("<queue_name>")

Returns a pbs.queue object representing the queue named *<queue name>* that is managed by the local server. See section 6.5, "Queue Objects", on page 121.

A value of *None* is returned if the queue named *<queue_name>* does not exist at the local server.

6.4.4.4 Method to Return Queue Iterator

pbs.server().queues()

Returns a Python iterator that iterates over a list of queue objects managed by the the local server. Returns an empty iterator if no queues exist at the local server. See section 6.5, "Queue Objects", on page 121.

6.4.4.5 Method to Return Reservation

pbs.server().resv("<reservation ID>")

Returns a pbs.resv object for <*reservation ID*> on the local server. Returns *None* if <*reservation ID*> does not exist. See section 6.9, "Reservation Objects", on page 131.

6.4.4.6 Method to Return Reservation Iterator

pbs.server().resvs()

Returns a Python iterator that iterates over a list of pbs.resv objects residing on the local server. Returns an empty iterator if no reservations exist at the local server. See section 6.9, "Reservation Objects", on page 131.

6.4.4.7 Method to Restart Scheduler Cycle

pbs.server().scheduler_restart_cycle()

This directs the current PBS server to tell the scheduler to restart its scheduling cycle.

A hook with its user attribute set to *pbsuser* cannot successfully invoke pbs.scheduler_restart_cycle(), unless the hook's executing user is a PBS Manager or Operator. If this is attempted, the scheduler is not restarted, and the following message appears at log event class PBSEVENT_DEBUG2 in the MoM logs:

```
"<node_host_name>;Not allowed to update vnodes or to request scheduler_restart_cycle, if run as a
   non-manager/operator user"
```

6.4.4.8 Method to Return Named Vnode

pbs.server().vnode("<vnode name>")

Returns a pbs.vnode object representing the vnode with name *<vnode name>* that is managed by the current server.

Returns *None* if *<vnode name>* does not exist.

6.4.4.9 Method to Return Vnode List

pbs.server().vnodes()

Returns a list of pbs.vnode objects managed by current server.

Returns an empty iterator if no vnodes exist at the local server.

Example:

```
for vn in s.vnodes():
    pbs.logmsg(pbs.LOG_DEBUG, "found vn %s" % (vn.name))
```

6.5 Queue Objects

pbs.queue

This object represents a PBS queue. This object can either represent an actual PBS queue, or be just a coding construct, not representing an actual queue. If it is just a coding construct, you can set its attributes. If it represents an actual queue, you can read but cannot set its attributes. **You cannot set the attributes of any actual queue in any hook**.

To get information about a particular queue with name <*name*>, you must go through the associated server. Use:

```
q = pbs.server().queue("<name>")
```

To get a list of queues from the server:

```
pbs.server().queues()
```

6.5.1 Queue Object Members

Some queue object members are queue attributes, and some are not.

6.5.1.1 Queue Object Name Member

queue.name

The queue name.

This member is read-only.

Python type: *str*

6.5.1.2 Queue Object Attribute Members

A pbs.queue has a member representing each of its attributes. Each member that is not a *string*, *int*, *bool*, *long*, or *float* has a corresponding creation method; see section 6.13.3, “PBS Types and Their Methods”, on page 143. See section 5.2.4, “Using Attributes and Resources in Hooks”, on page 44.

queue.<attribute name>

The queue attribute named <*attribute name*>. Queue attributes are listed in “Queue Attributes” on page 313 of the PBS Professional Reference Guide.

Example 6-17: Get the queue object representing the queue *workq*, and its Priority value:

```
q = s.queue("workq")
prio = q.Priority
```

6.5.1.3 Setting Queue Object Attributes

You can set or unset queue object attributes for queue objects that **don't** represent an actual queue. To set the value of a queue object attribute named <*attribute name*>:

```
pbs.queue.<attribute name> = <attribute value>
```

You cannot set or unset attributes for an actual queue.

6.5.2 Queue Object Methods

6.5.2.1 Method to Return Job

queue.job()

pbs.queue.job("<job ID>")

Returns a pbs.job object representing PBS job with ID <*job ID*>. This job must be residing on the queue. Returns *None* if the job with the specified job ID does not exist, or if the job is not in the queue. See section 6.6, "Job Objects", on page 122.

6.5.2.2 Method to Return Job Iterator

queue.jobs()

Returns a Python iterator that iterates over a list of pbs.job objects representing the jobs on the queue. Returns an empty iterator if no jobs exist on the queue. See section 6.6, "Job Objects", on page 122.

Example:

```
for j in pbs.server().queue("workq").jobs():
    pbs.logmsg(pbs.LOG_DEBUG, "found job %s" % (j.id))
```

6.5.3 Queue Type Constant Objects

Queue types are represented by constant objects. The pbs.queue.queue_type member represents the type of the queue. It can take on the following values:

Table 6-5: Queue Type Constant Objects

Object	Queue Type
pbs.QTYPE_EXECUTION	*Execution*
pbs.QTYPE_ROUTE	*Route*

6.6 Job Objects

pbs.job

A job object represents a PBS job.

You can retrieve the job object either through an associated event or through the server. The job object represents one of the following, depending on how it is retrieved:

- The PBS job associated with the event that triggers the hook. To get the job associated with the current event, go through the event that triggered the hook:

  pbs.event().job

  A call to pbs.event().job can return only the job associated with the current event.

When you get a job using pbs.event().job, the hook can read and set the job attributes and resources listed in Table 5-6, "Job Attributes Readable & Settable via Events," on page 55 and Table 5-9, "Built-in Job Resources Readable & Settable by Hooks via Events," on page 60.

- A job at the server at which the hook is executing. To get a particular job with ID <*id*>, go through the server:

  pbs.server().job("<job ID>")

  When you get a job using pbs.server().job("<job ID>"), the hook can read all job attributes and resources, but can set none.

- To get a list of jobs at the server:

  pbs.server().jobs()

For information about a list of jobs visible through events, see section 6.3.2.11, "Job List Event Member", on page 111 and Table 6-3, "Using Event Object Members in Events," on page 108 for the events that can use this list.

All job objects have the same members and methods. Each hook can read or set different attributes and resources. We describe what each type of hook can do in section 6.3, "Event Objects", on page 87.

If you use a hook to make a change to a job, that change is visible to all PBS daemons.

6.6.1 Job Object Members

A pbs.job object has a member to represent each job attribute. Each one is spelled exactly like the corresponding attribute. We list job object members here that require creation methods, require special treatment, or that are not job attributes. All job attribute members that are not listed here are defined this way:

job.<attribute name>

The type of the attribute is given in the attribute description, in "Job Attributes" on page 330 of the PBS Professional Reference Guide.

For information about using job attributes in hooks, see section 5.2.4, "Using Attributes and Resources in Hooks", on page 44.

To see which hooks can set which job attributes and resources, see Table 5-6, "Job Attributes Readable & Settable via Events," on page 55 and Table 5-9, "Built-in Job Resources Readable & Settable by Hooks via Events," on page 60.

A pbs.job object also has the id member, which is not a job attribute.

6.6.1.1 Job ID Member

job.id

The PBS job ID.

Read-only.

Python type: *str*

6.6.1.2 Job array_indices_submitted Attribute Member

job.array_indices_submitted

Job attribute. Python type: *range*

See section 6.13.3.21, "Method to Create or Set range Object", on page 148.

6.6.1.3 Job Checkpoint Attribute Member

job.Checkpoint

Job attribute. Python type: *pbs.checkpoint*

See section 6.13.3.3, “Method to Create or Set Checkpoint String”, on page 144.

6.6.1.4 Job depend Attribute Member

job.depend

Job attribute. Python type: *pbs.depend*

See section 6.13.3.4, “Method to Create or Set Dependency Object”, on page 144.

6.6.1.5 Job Execution_Time Attribute Member

job.Execution_Time

Job attribute. Time when the current job is eligible to run. Syntax:

job.Execution_Time = time.mktime([<YY>, <MM>, <DD>, <HH>, <MM>, <SS>, <WEEKDAY>, <YEARDAY> <ISDST>])

For example, the following sets a job's Execution_Time to: *March 1, 2012 at 09:00 am*:

```
job.Execution_Time = time.mktime([2012, 3, 1, 12, 9, 0, -1, -1, -1])
```

Python type: *int*

6.6.1.6 Job exec_host Attribute Member

job.exec_host

Job attribute. Python type: *pbs.exec_host*

See section 6.13.3.7, “Method to Create or Set exec_host Object”, on page 145.

6.6.1.7 Job exec_vnode Attribute Member

job.exec_vnode

Job attribute. Python type: *pbs.exec_vnode*

This complex object is described in section 6.7, “The exec_vnode Object”, on page 129.

See also section 6.13.3.8, “Method to Create or Set exec_vnode Object”, on page 145.

6.6.1.8 Job group_list Attribute Member

job.group_list

Job attribute. Python type: *pbs.group_list*

See section 6.13.3.9, “Method to Create or Set group_list Object”, on page 145.

6.6.1.9 Job Hold_Types Attribute Member

job.Hold_Types

Job attribute. Python type: *pbs.hold_types*

See section 6.13.3.10, “Method to Create or Set hold_types Object”, on page 145.

6.6.1.10 Job job_state Attribute Member

job.job_state

Job attribute. Represents the job's state. Can be compared to the constants representing job states.

Use job state constant objects to test the state of a job. For example:

```
e = pbs.event()
if e.job.job_state == pbs.JOB_STATE_RUNNING :
    e.accept()
```

The job_state member can take on any of the values listed here:

Table 6-6: Job State Objects

Object	State	Description
pbs.JOB_STATE_BEGUN	*B*	Job arrays only: job array has started
pbs.JOB_STATE_EXITING	*E*	Job is exiting after having run
pbs.JOB_STATE_EXPIRED	*X*	Subjobs only; subjob is finished (expired.)
pbs.JOB_STATE_FINISHED	*F*	Job is finished: job executed successfully, job was terminated while running, job execution failed, or job was deleted before execution
pbs.JOB_STATE_HELD	*H*	Job is held.
pbs.JOB_STATE_MOVED	*M*	Job has been moved to another server
pbs.JOB_STATE_QUEUED	*Q*	Job is queued, eligible to run or be routed
pbs.JOB_STATE_RUNNING	*R*	Job is running
pbs.JOB_STATE_SUSPEND	*S*	Job is suspended by PBS so that a higher-priority job can run.
pbs.JOB_STATE_SUSPEND_USERACTIVE	*U*	Job is suspended due to workstation becoming busy
pbs.JOB_STATE_TRANSIT	*T*	Job is in transition (being moved to a new location)
pbs.JOB_STATE_WAITING	*W*	Job is waiting for its requested execution time to be reached, or the job's specified stagein request has failed for some reason.

6.6.1.11 Job Join_Path Attribute Member

job.Join_Path

Job attribute. Python type: *pbs.join_path*

See section 6.13.3.12, “Method to Create or Set join_path Object”, on page 146.

6.6.1.12 Job Keep_Files Attribute Member

job.Keep_Files

Job attribute. Python type: *pbs.keep_files*

See section 6.13.3.13, “Method to Create or Set keep_files Object”, on page 146.

6.6.1.13 Job Mail_Points Attribute Member

job.Mail_Points

Job attribute. Python type: *pbs.mail_points*

See section 6.13.3.15, "Method to Create or Set mail_points Object", on page 146.

6.6.1.14 Job Mail_Users Attribute Member

job.Mail_Users

Job attribute. Python type: *pbs.email_list*

See section 6.13.3.6, "Method to Create or Set Email List", on page 144.

6.6.1.15 Job Queue Attribute Member

job.queue

Job attribute. Python type: *pbs.queue*

6.6.1.16 Job Resource_List Attribute Member

job.Resource_List[]

job.Resource_List["<resource name>"]

Job attribute. The job's Resource_List attribute.

Python type: dictionary: *Resource_List["<resource name>"]=<value>* where <*resource name*> is any built-in or custom resource

6.6.1.17 Job resources_used Attribute Member

job.resources_used["<resource name>"]

Job attribute. The job's resources_used attribute, which lists the resources used by the job. See section 5.2.4, "Using Attributes and Resources in Hooks", on page 44.

Python type: dictionary: *resources_used["<resource name>"]=<value>* where <*resource name*> is any built-in or custom resource

6.6.1.18 Job run_count Attribute Member

job.run_count

Job attribute. Execution hooks must run with user = *pbsadmin* to reduce the value of this member. Execution hooks running with user = *pbsuser* cannot reduce the value of this member.

Python type: *int*

6.6.1.19 Job stagein and stageout Attribute Members

job.stagein

job.stageout

Job attribute. Python type: *pbs.staging_list*

See section 6.13.3.27, "Method to Create or Set staging_list Object", on page 150.

6.6.1.20 Job User_List Attribute Member

job.User_List

Job attribute. Python type: *pbs.user_list*

See section 6.13.3.29, "Method to Create or Set user_list Object", on page 151.

6.6.1.21 Job Variable_List Attribute Member

job.Variable_List[<variable>]

Job attribute. Holds the job's environment variables. Syntax:

job.Variable_List[<variable>] = <value>

Python type: dictionary: *Variable_List["<variable name>"]=<value>*

6.6.2 Setting Job Attributes

How you set a job attribute depends on the type of the attribute; those of type *str*, *int*, *bool*, *long*, and *float* can be set directly. Job attributes of other types require creation methods. Job attribute creation methods are listed in section 6.13.3, "PBS Types and Their Methods", on page 143.

To set job attributes and resources directly:

pbs.event().job.<attribute> = <value>

pbs.event().job.Resource_List["<resource name>"] = <value>

See section 5.2.4.3, "Determining Whether to Use Creation Method to Set Attribute or Resource", on page 45.

See section 5.2.4, "Using Attributes and Resources in Hooks", on page 44.

6.6.3 Examples of Using Job Object Members

Get the job's Priority value:

```
prio = job.Priority
```

Reset the Priority value of job *j*:

```
job.Priority = 5
```

Get the job's PBS_O_WORKDIR environment variable:

```
workdir = job.Variable_List["PBS_O_WORKDIR"]
```

6.6.4 Job Object Methods for Execution Hooks

Job objects have the following methods. Most methods are available in execjob_ hooks except for the execjob_launch hook, and in the exechost_periodic hook.

6.6.4.1 Job Object Method to Report Checkpoint

job.is_checkpointed()

Returns a Python bool value which is *True* if the job was checkpointed under the control of the PBS MoM.

For example, you could use this in an execjob_epilogue hook, where the hook writer directs the job to be requeued if the job was checkpointed under the control of PBS:

```
# cat epi.py
import pbs
If pbs.event().job.is_checkpointed():
    pbs.event().job.rerun()
    pbs.event().reject("job to be requeued")
# qmgr -c "create hook epi event=execjob_epilogue"
# qmgr - c "import hook epi application/x-python default epi.py"
```

6.6.4.2 Job Object Method to Report Execution Host Role

job.in_ms_mom()

Returns a Python bool value. Returns *True* if this job object is running on the primary execution host.

6.6.4.3 Job Object Method to Delete Job

job.delete()

When this method is used in an execution hook, the job is flagged at the server for deletion after its processes have terminated and any epilogue or execjob_epilogue hook has run.

When this method is used in a non-execution hook script, it raises a Python "NotImplementedError" exception.

If the job.delete() method is used in an execjob_end hook, it has no effect, because in this case the server has already performed end-of-job processing before the execution hook runs.

The job.delete() method overrides the job.rerun() method. If both are used, job.delete() takes precedence.

6.6.4.4 Job Object Method to Release Vnodes

job.release_nodes()

job.release_nodes(keep_select=<select specification>)

Automatically selects vnodes that satisfy the new request and are healthy, keeps them in the job's vnode request, and releases all others. The method automatically trims out any vnodes in the pbs.event().vnode_list_fail[] list.

You can call pbs.event().job.release_nodes(keep_select = <desired vnodes>) in an execjob_launch or execjob_prologue hook. Note that despite the method being named "release_nodes", it **keeps** the specified vnodes and **releases** all other vnodes. You can specify the job's original vnode request as the vnodes to keep.

The pbs.event().job.release_nodes() method returns a PBS job object which has the updated values for the job's exec_vnode and Resource_List attributes.

This method is only effective when it runs at the primary MoM.

This method can be used only when it's used for a job whose tolerate_node_failures attribute is set to *job_start* or *all*.

6.6.4.4.i Advice and Recommendations for Using release_nodes Method

- Put the call to this method in an 'if pbs.event().job.in_ms_mom()' clause
- Request vnodes that are a subset of the existing vnode request
- Because an execjob_launch hook is also called when spawning tasks via pbsdsh and tm_spawn, ensure that any execjob_launch hook invoking release_nodes() has 'PBS_NODEFILE' in the pbs.event().env list. The presence of 'PBS_NODEFILE' in the environment ensures that the primary MoM is executing on behalf of starting the top level job, and not spawning a sister task. You can add the following at the top of the hook:

```
if 'PBS_NODEFILE' not in pbs.event().env:
   pbs.event().accept()
...
pbs.release_nodes(keep_select=...)
```

- On Windows, where PBS_NODEFILE always appears in pbs.event().env, put the following at the top of any execjob_launch hook:

```
if any("mom_open_demux.exe") in s for s in e.argv):
   e.accept()
```

6.6.4.4.ii Side Effects of Using release_nodes() Method

When release_nodes() is successfully executed from execjob_prologue or execjob_launch hooks, the following happen:

- PBS generates the *s* accounting record.
- The primary MoM notifies the sister MoMs to update their internal nodes tables, so that the task manager API (e.g. tm_spawn, pbsdsh) will be aware of the change in the future.
- If the pbs_cgroups hook is enabled, the cgroup already created for the job is updated to match the job's new resources. If the kernel rejects the update to the job's cgroup resources, the job is aborted at the execution host, and requeued/rerun at the server.

6.6.4.5 Job Object Method to Re-run Job

job.rerun()

When this method is used in an execution hook, the job is flagged at the server for requeueing after its processes have terminated and any epilogue or execjob_epilogue hook has run.

When this method is used in a non-execution hook script, it raises a Python "NotImplementedError" exception.

If the job.rerun() method is used in an execjob_end hook, it has no effect, because in this case the server has already performed end-of-job processing before the execution hook runs.

The job.delete() method overrides the job.rerun() method. If both are used, job.delete() takes precedence.

6.7 The exec_vnode Object

pbs.exec_vnode

The exec_vnode object represents the job's exec_vnode attribute.

6.7.1 The exec_vnode Object Members

A pbs.exec_vnode object has the following member:

6.7.1.1 The exec_vnode Chunks Member

pbs.exec_vnode.chunks[]

List of pbs.vchunk objects. These objects represent the chunks assigned to a job. See section 6.8, "Chunk Objects", on page 131.

6.7.2 Using pbs.vchunk Objects in exec_vnode

- To get a list of pbs.vchunks in pbs.event().job.exec_vnode:

  pbs.event().job.exec_vnode.chunks

  For example, to log the name of the vnode containing each vchunk:

```
chunklist = pbs.event().job.exec_vnode.chunks
for chunk in chunklist:
    pbs.logmsg(pbs.LOG_DEBUG, "chunk.vnode_name=%s " % (chunk.vnode_name))
```

- To get a pbs.vchunk with a specific index:

  pbs.event().job.exec_vnode.chunks[<index>]

- For example, to get the vchunk in pbs.event().job.exec_vnode with index number *2*:

  pbs.event().job.exec_vnode.chunks[2]

Example 6-18: List the job ID, vnode name, and resources in exec_vnode:

```
j = pbs.event().job
pbs.logmsg(pbs.LOG_DEBUG, "job %s exec_vnode = %s" % (j.id, j.exec_vnode))

chunklist = j.exec_vnode.chunks
for c in chunklist:
    pbs.logmsg(pbs.LOG_DEBUG, "c.vnode_name=%s " % (c.vnode_name))

    for r in c.chunk_resources.keys():
        pbs.logmsg(pbs.LOG_DEBUG, "c.chunk_resources[%s]=%s" % (r,
            c.chunk_resources[r]))
```

Sample output:

```
10:16:53;0006;Server@jobim;Hook;Server@jobim;job 153.jobim exec_vnode  =
    (jobim[2]:ncpus=2:mem=10240kb)+ (jobim[1]:ncpus=2:mem=10240kb) +
    (jobim[3]:ncpus=2:mem=2048kb)
10:16:53;0006;Server@jobim;Hook;Server@jobim;c.vnode_name= jobim[2]
    10:16:53;0006;Server@jobim;Hook;Server@jobim; c.chunk_resources[ncpus]=2
10:16:53;0006;Server@jobim;Hook;Server@jobim; c.chunk_resources[mem]=10240kb
10:16:53;0006;Server@jobim;Hook;Server@jobim; c.vnode_name=jobim[1]
10:16:53;0006;Server@jobim;Hook;Server@jobim; c.chunk_resources[ncpus]=2
10:16:53;0006;Server@jobim;Hook;Server@jobim; c.chunk_resources[mem]=10240kb
10:16:53;0006;Server@jobim;Hook;Server@jobim; c.vnode_name=jobim[3]
10:16:53;0006;Server@jobim;Hook;Server@jobim; c.chunk_resources[ncpus]=2
    10:16:53;0006;Server@jobim;Hook;Server@jobim; c.chunk_resources[mem]=2048kb
```

6.7.3 Restrictions on exec_vnode Objects

A job's exec_vnode attribute is read-only. You cannot set its value, and you cannot build an exec_vnode object using pbs.vchunk objects.

6.8 Chunk Objects

pbs.vchunk

The pbs.vchunk object represents a chunk specification. It is used in a job's exec_vnode attribute or select statement.

6.8.1 Chunk Object Members and Methods

A pbs.vchunk object has the following members:

6.8.1.1 Chunk Object Vnode Name Member

vchunk.vnode_name

Name of the vnode from which the chunk is taken.

Python type: *str*

6.8.1.2 Chunk Object Chunk Resources Member

vchunk.chunk_resources[]

Resources assigned to the chunk.

Python type: Dictionary containing *<resource name>=<value>* pairs.

Syntax: *chunk_resources['<resource name>'] = <resource value>* where *<resource name>* is any custom or built-in resource.

6.8.1.3 Chunk Object Method to Return chunk_resources Keys

vchunk.chunk_resources.keys()

Returns list of *<resource name>* keys of chunk_resources[]. This list makes it convenient to list all the values of chunk_resources[].

6.9 Reservation Objects

pbs.resv

This represents a PBS reservation. If the reservation is associated with the triggering event, you can read and set reservation attributes and resources in a resvsub hook, and read them in a resv_end hook. See Table 5-8, "Reservation Attributes Readable & Settable in resvsub and resv_end Hooks," on page 59 for a complete list of the reservation attributes and resources that can be set in the resvsub and resv_end hooks. If the reservation is retrieved through the server, and is not associated with the triggering event, you can read all its attributes and resources, but set none.

If you are working with the reservation being created using `pbs_rsub`, you must use *pbs.event().resv*. The server cannot return information about the reservation, because it has not yet been created.

In order to retrieve information about the reservation associated with the triggering action, you must use a reference to the reservation object represented by:

pbs.event().resv

To get a copy of a particular reservation, use:

pbs.server().resv("<reservation name>")

To get a list of the reservations at a server:

pbs.server().resvs()

6.9.1 Reservation Object Members

A pbs.resv object has members that represent reservation attributes, and the resvid member which exists for the job object but is not an attribute of a reservation.

6.9.1.1 Reservation ID Member

resv.resvid

The reservation ID.

Example: "R221.myhost".

This member is read-only.

Python type: *str*

6.9.1.2 Reservation Attribute Members

resv.<attribute name>

The reservation attribute named *<attribute name>*. Each member is spelled exactly like the corresponding attribute.

6.9.1.3 Setting Reservation Object Attribute Values

You can set, but not unset, reservation object attributes.

To see a list of which reservation attributes can be read and set by each hook, see Table 5-8, "Reservation Attributes Readable & Settable in resvsub and resv_end Hooks," on page 59.

Some attributes require creation methods when setting them. See section 5.2.4.3, "Determining Whether to Use Creation Method to Set Attribute or Resource", on page 45. To set a simple reservation object attribute:

```
pbs.resv.<attribute name> = <attribute value>
```

Reservation attribute creation methods are listed in section 6.13.3, "PBS Types and Their Methods", on page 143.

See section 5.2.4, "Using Attributes and Resources in Hooks", on page 44.

6.9.1.4 Examples of Using Reservation Object Attributes

Example 6-19: Get the reservation's owner:

```
owner = pbs.server().resv(<reservation ID>).Reserve_Owner
```

Example 6-20: Reset the reservation's name:

```
pbs.event().resv(<reservation ID>).Reserve_Name = "Resv2008"
```

6.9.2 Reservation State Constant Objects

The pbs.resv.reserve_state member represents the state of the reservation. It can take on the following values, which are represented by constant objects:

Table 6-7: Reservation State Objects

Object	State
pbs.RESV_STATE_NONE	*RESV_NONE*
pbs.RESV_STATE_UNCONFIRMED	*RESV_UNCONFIRMED*
pbs.RESV_STATE_CONFIRMED	*RESV_CONFIRMED*
pbs.RESV_STATE_WAIT	*RESV_WAIT*
pbs.RESV_STATE_TIME_TO_RUN	*RESV_TIME_TO_RUN*
pbs.RESV_STATE_RUNNING	*RESV_RUNNING*
pbs.RESV_STATE_FINISHED	*RESV_FINISHED*
pbs.RESV_STATE_BEING_ALTERED	*RESV_BEING_ALTERED*
pbs.RESV_STATE_BEING_DELETED	*RESV_BEING_DELETED*
pbs.RESV_STATE_DELETED	*RESV_DELETED*
pbs.RESV_STATE_DELETING_JOBS	*RESV_DELETING_JOBS*
pbs.RESV_STATE_DEGRADED	*RESV_DEGRADED*
pbs.RESV_STATE_IN_CONFLICT	*RESV_IN_CONFLICT*

6.10 Vnode Objects

pbs.vnode

Represents a PBS vnode.

The way in which you retrieve a vnode controls what you can do with the vnode. If a vnode is retrieved through an event, using *pbs.event().v node_list[]*, and is managed by the same MoM where the event hook runs, you can set the vnode attributes and resources listed in Table 5-7, “Vnode Attributes Readable & Settable via Events,” on page 57 and Table 5-10, “Vnode Resources Readable & Settable by Hooks via Events,” on page 61. However, if a vnode is not retrieved through an event, or is not managed by the same MoM where the hook runs, you can read all vnode attributes and resources, but set none.

Execution events have access to the list of vnodes associated with the job. Periodic events have access to the list of vnodes managed by the local MoM. See section 6.3.2.22, “The Vnode List Event Member”, on page 114.

- To retrieve the list of vnodes associated with an execution event or a periodic event:

  pbs.event().vnode_list[]

- To retrieve a specific vnode that is associated with an execution or periodic event, use the list of vnodes associated with the event, and specify the vnode name:

  pbs.event().vnode_list["<vnode name>"]

- To retrieve the vnodes associated with a pre-execution event, get the job's exec_vnode attribute:

pbs.event().job.exec_vnode

- To retrieve the server's list of vnodes:

  pbs.server().vnodes()

- To retrieve a named vnode through the server:

  pbs.server().vnode("<vnode name>")

6.10.1 Vnode Object Members

vnode.<attribute name>

A pbs.vnode object has a member representing each attribute, and each member is spelled exactly like the corresponding attribute. Table 5-7, “Vnode Attributes Readable & Settable via Events,” on page 57 lists which vnode attributes can be set by each hook. See section 5.2.4.3, “Determining Whether to Use Creation Method to Set Attribute or Resource”, on page 45. Attribute creation methods are listed in section 6.13.3, “PBS Types and Their Methods”, on page 143. See section 5.2.4, “Using Attributes and Resources in Hooks”, on page 44.

6.10.1.1 The topology_info Attribute Member

vnode.topology_info

Vnode attribute. The topology_info vnode attribute shows topology information. This attribute is visible only in hooks, and can be used only in hooks.

Python type: *str*

6.10.1.2 Vnode Attribute Restrictions

- The only vnode attribute that can be changed by a pre-execution hook is the state attribute
- The only pre-execution hook that can change the vnode state attribute is the runjob hook
- Execution and periodic hooks can change all settable vnode attributes

6.10.2 Vnode Type Constant Objects

The pbs.vnode.ntype member represents the type of the vnode. It can take on the following values:

Table 6-8: Vnode Type Objects

Object	Type
pbs.ND_PBS	Represents *pbs* value for vnode ntype attribute

6.10.3 Vnode Sharing Constant Objects

The pbs.vnode.sharing member represents the vnode's sharing attribute. It can take on the following values:

Table 6-9: Vnode Sharing Objects

Object	Sharing Value
pbs.ND_DEFAULT_EXCL	Represents *default_excl* vnode sharing attribute value
pbs.ND_DEFAULT_EXCLHOST	Represents *default_exclhost* vnode sharing attribute value
pbs.ND_DEFAULT_SHARED	Represents *default_shared* vnode sharing attribute value
pbs.ND_FORCE_EXCL	Represents *force_excl* vnode sharing attribute value
pbs.ND_FORCE_EXCLHOST	Represents *force_exclhost* vnode sharing attribute value
pbs.ND_IGNORE_EXCL	Represents *ignore_excl* vnode sharing attribute value

6.10.4 Vnode State Constant Objects

The pbs.vnode.state member represents the state of the vnode. It can take on the following values:

Table 6-10: Vnode State Constant Objects

Object	State
pbs.ND_BUSY	Represents *busy* vnode state
pbs.ND_DOWN	Represents *down* vnode state
pbs.ND_FREE	Represents *free* vnode state
pbs.ND_JOBBUSY	Represents *job-busy* vnode state
pbs.ND_JOB_EXCLUSIVE	Represents *job-exclusive* vnode state
pbs.ND_OFFLINE	Represents *offline* vnode state
pbs.ND_PROV	Represents *provisioning* vnode state
pbs.ND_RESV_EXCLUSIVE	Represents resv-exclusive vnode state
pbs.ND_STALE	Represents *stale* vnode state
pbs.ND_STATE_UNKNOWN	Represents *state-unknown, down* vnode state
pbs.ND_UNRESOLVABLE	Represents *unresolvable* vnode state
pbs.ND_WAIT_PROV	Represents *wait-provisioning* vnode state

6.11 Configuration File Objects

6.11.1 Variable Containing Hook Configuration File Path

pbs.hook_config_filename

Contains the path to the hook's configuration file, or *None* if there is no configuration file.

6.11.2 Dictionary of PBS Configuration File Entries

pbs.pbs_conf[]

This is a dictionary of values which represent entries in the pbs.conf file.

This reflects the contents of /etc/pbs.conf on the host where a hook runs, so pre-execution event (server) hooks get the entries on the server host, and execution event (MoM) hooks get the entries on the execution host where the hook runs.

Example of using pbs.pbs.conf[]:

```
pbs.logmsg(pbs.LOG_DEBUG, "pbs home is %s" % (pbs.pbs_conf['PBS_HOME']))
```

If you change /etc/pbs.conf, HUP pbs_mom (Linux) and/or restart pbs_server to rebuild the dictionary with the new contents of pbs.conf.

Each parameter in the pbs.conf file is the key to its dictionary entry. The pbs.conf file can contain the following parameters:

Table 6-11: Parameters in pbs.conf

Parameter	Description
PBS_AUTH_METHOD	Specifies default authentication method and library to be used by PBS. Used only at authenticating client. Case-insensitive. Default value: *resvport* To use MUNGE, set to *munge*
PBS_BATCH_SERVICE_PORT	Port on which server listens. Default: 15001
PBS_BATCH_SERVICE_PORT_DIS	DIS port on which server listens.
PBS_COMM_LOG_EVENTS	Communication daemon log mask. Default: *511*
PBS_COMM_ROUTERS	Tells a pbs_comm the location of the other pbs_comms.
PBS_COMM_THREADS	Number of threads for communication daemon.
PBS_CORE_LIMIT	Limit on corefile size for PBS daemons. Can be set to an integer number of bytes or to the string "unlimited". If unset, core file size limit is inherited from the shell environment.
PBS_DATA_SERVICE_PORT	Used to specify non-default port for connecting to data service. Default: 15007
PBS_ENCRYPT_METHOD	Specifies method and library for encrypting and decrypting data in client-server communication. Used only at authentication client side. Case-insensitive. To use TLS encryption in client-server communication, set this parameter to *tls*. No default; if this is not set, PBS does not encrypt or decrypt data.
PBS_ENVIRONMENT	Location of pbs_environment file.
PBS_EXEC	Location of PBS bin and sbin directories.
PBS_HOME	Location of PBS working directories.

Table 6-11: Parameters in `pbs.conf`

Parameter	Description
PBS_LEAF_NAME	Tells endpoint what hostname to use for network. The value does not include a port, since that is usually set by the daemon. By default, the name of the endpoint's host is the hostname of the machine. You can set the name where an endpoint runs. This is useful when you have multiple networks configured, and you want PBS to use a particular network. The server only queries for the canonicalized address of the MoM host, unless you let it know via the Mom attribute; if you have set PBS_LEAF_NAME in `/etc/pbs.conf` to something else, make sure you set the Mom attribute at vnode creation. TPP internally resolves the name to a set of IP addresses, so you do not affect how `pbs_comm` works.
PBS_LEAF_ROUTERS	Location of endpoint's `pbs_comm` daemon(s).
PBS_LOCALLOG=<value>	Enables logging to local PBS log files. Valid values: 0: no local logging 1: local logging enabled Only available when using `syslog`.
PBS_LOG_HIGHRES_TIMESTAMP	Controls whether daemons on this host log timestamps in microseconds. Default timestamp log format is *HH:MM:SS*. With microsecond logging, format is *HH:MM:SS:XXXXXX*. Does not affect accounting log. Not applicable when using `syslog`. Overridden by environment variable of the same name. Valid values: *0*, *1*. Default: *0* (no microsecond logging)
PBS_MAIL_HOST_NAME	Used in addressing mail regarding jobs and reservations that is sent to users specified in a job or reservation's Mail_Users attribute. Optional. If specified, must be a fully qualified domain name. Cannot contain a colon (":"). For how this is used in email address, see section 2.2.2, "Specifying Mail Delivery Domain", on page 22.
PBS_MANAGER_SERVICE_PORT	Port on which MoM listens. Default: 15003
PBS_MOM_HOME	Location of MoM working directories.
PBS_MOM_NODE_NAME	Name that MoM should use for parent vnode, and if they exist, child vnodes. If this is not set, MoM defaults to using the non-canonicalized hostname returned by `gethostname()`. If you use the IP address for a vnode name, set `PBS_MOM_NODE_NAME=<IP address>` in `pbs.conf` on the execution host. Dots are not allowed in this parameter unless they are part of an IP address.
PBS_MOM_SERVICE_PORT	Port on which MoM listens. Default: 15002

Table 6-11: Parameters in `pbs.conf`

Parameter	Description
PBS_OUTPUT_HOST_NAME	Host to which all job standard output and standard error are delivered. If specified in `pbs.conf` on a job submission host, the value of PBS_OUTPUT_HOST_NAME is used in the host portion of the job's Output_Path and Error_Path attributes. If the job submitter does not specify paths for standard output and standard error, the current working directory for the `qsub` command is used, and the value of PBS_OUTPUT_HOST_NAME is appended after an at sign ("@"). If the job submitter specifies only a file path for standard output and standard error, the value of PBS_OUTPUT_HOST_NAME is appended after an at sign ("@"). If the job submitter specifies paths for standard output and standard error that include host names, the specified paths are used. Optional. If specified, must be a fully qualified domain name. Cannot contain a colon (":"). See "Delivering Output and Error Files" on page 60 in the PBS Professional Administrator's Guide.
PBS_PRIMARY	Hostname of primary server. Used only for failover configuration. Overrides PBS_SERVER_HOST_NAME. If you set PBS_LEAF_NAME on the primary server host, make sure that PBS_PRIMARY matches PBS_LEAF_NAME on the corresponding host. If you do not set PBS_LEAF_NAME on the server host, make sure that PBS_PRIMARY matches the hostname of the server host.
PBS_RCP	Location of `rcp` command if `rcp` is used.
PBS_REMOTE_VIEWER	Specifies remote viewer client. If not specified, PBS uses native Remote Desktop client for remote viewer. Set on submission host(s). Supported on Windows only.
PBS_SCHEDULER_SERVICE_PORT	Port on which default scheduler listens. Default value: 15004
PBS_SCHED_THREADS	Maximum number of scheduler threads. Scheduler automatically caps number of threads at the number of cores (or hyperthreads if applicable), regardless of value of this variable. Overridden by `pbs_sched -t` option and PBS_SCHED_THREADS environment variable. Default: 1
PBS_SCP	Location of `scp` command if `scp` is used; setting this parameter causes PBS to first try `scp` rather than `rcp` for file transport.

Table 6-11: Parameters in `pbs.conf`

Parameter	Description
PBS_SECONDARY	Hostname of secondary server. Used only for failover configuration. Overrides PBS_SERVER_HOST_NAME. If you set PBS_LEAF_NAME on the secondary server host, make sure that PBS_SECONDARY matches PBS_LEAF_NAME on the corresponding host. If you do not set PBS_LEAF_NAME on the server host, make sure that PBS_SECONDARY matches the hostname of the server host.
PBS_SERVER	Hostname of host running the server. Cannot be longer than 255 characters. If the short name of the server host resolves to the correct IP address, you can use the short name for the value of the PBS_SERVER entry in `pbs.conf`. If only the FQDN of the server host resolves to the correct IP address, you must use the FQDN for the value of PBS_SERVER. Overridden by PBS_SERVER_HOST_NAME and PBS_PRIMARY.
PBS_SERVER_HOST_NAME	The FQDN of the server host. Used by clients to contact server. Overridden by PBS_PRIMARY and PBS_SECONDARY failover parameters. Overrides PBS_SERVER parameter. Optional. If specified, must be a fully qualified domain name. Cannot contain a colon (":"). See "Contacting the Server" on page 60 in the PBS Professional Administrator's Guide.
PBS_START_COMM	Set this to *1* if a communication daemon is to run on this host.
PBS_START_MOM	Default is *0*. Set this to *1* if a MoM is to run on this host.
PBS_START_SCHED	**Deprecated**. Set this to *1* if default scheduler is to run on this host. Overridden by scheduler's scheduling attribute.
PBS_START_SERVER	Set this to *1* if server is to run on this host.
PBS_SUPPORTED_AUTH_METHODS	Specifies supported authentication methods for client-server communication. Used by authenticating server (PBS server, scheduler, MoM, or comm); ignored at client. Case-insensitive. If this parameter is set, PBS accepts only the methods listed. Format: comma-separated list of authentication methods. Default value: *resvport* Example: *munge,GSS*

Table 6-11: Parameters in `pbs.conf`

Parameter	Description
PBS_SYSLOG=<value>	Controls use of `syslog` facility under which the entries are logged. Valid values: 0: no syslogging 1: logged via LOG_DAEMON facility 2: logged via LOG_LOCAL0 facility 3: logged via LOG_LOCAL1 facility ... 9: logged via LOG_LOCAL7 facility
PBS_SYSLOGSEVR=<value>	Filters `syslog` messages by severity. Valid values: 0: only LOG_EMERG messages are logged 1: messages up to LOG_ALERT are logged ... 7: messages up to LOG_DEBUG are logged
PBS_TMPDIR	Location of temporary files/directories used by PBS components.

6.12 Constant Objects

Constant objects are used to represent PBS elements such as event types, job, server, reservation, and vnode states, log event classes, queue and vnode types, and exceptions. These objects cannot be modified. When the PBS module is imported, the constant objects are imported. PBS uses the following constant objects:

Table 6-12: Constant Objects

Category	Description
Event type objects	The pbs.event().type event member represents the type of the event, for example, pbs.QUEUEJOB or pbs.MOVEJOB. See section 6-2, “Event Types and Objects”, on page 87
Message log event class objects	You can use these objects to indicate log event class when placing messages in the server logs. See section 6-16, “Message Log Level Objects”, on page 152.
Queue type objects	The queue("<queue name>").queue_type member represents the type of the queue. See section 6-5, “Queue Type Constant Objects”, on page 122.
PBS server state objects	The pbs.server().server_state member represents the state of the server. See section 6.4.1.3, “Server State Member”, on page 119.
Job state objects	The job.job_state member represents the job's state. Use these constant objects to test the state of a job. See section 6-6, “Job State Objects”, on page 125.
Reservation state objects	The resv.reserve_state member represents the state of the reservation. See section 6-7, “Reservation State Objects”, on page 133.

Table 6-12: Constant Objects

Category	Description
Vnode state objects	The vnode.state member represents the state of the vnode. See section 6-10, "Vnode State Constant Objects", on page 135.
Vnode sharing objects	The vnode.sharing member represents the vnode's sharing attribute. See section 6-9, "Vnode Sharing Objects", on page 135.
Vnode type objects	The vnode.ntype member represents the type of the vnode. See section 6-8, "Vnode Type Objects", on page 134.

6.13 Object Members and Methods

The relationships between objects and methods are shown in Figure 6-1 and Figure 6-2.

Event object members are listed in Table 6-3, "Using Event Object Members in Events," on page 108.

Non-event objects and object members are listed in Table 6-13, "PBS Objects and Object Members," on page 141.

Table 6-14, "Methods Available in Events," on page 142 shows the methods available for each kind of event.

Each global method is described in section 6.13.3, "PBS Types and Their Methods", on page 143.

Each event-only method is described in section 6.3.4, "Event-only Methods", on page 116.

Each object-only method is described in the section for its object.

6.13.1 PBS Objects and Object Members

The following table lists PBS objects and their members, such as the server or jobs:

Table 6-13: PBS Objects and Object Members

Object	Object Member	Object Sub-member
pbs.hook_config_filename		
pbs.job	job.id	
	job.<attribute name>	
	pbs.exec_vnode (job attribute)	pbs.exec_vnode.chunks[]
pbs.pbs_conf[]		
pbs.queue	queue.<attribute name>	
	queue.name	
pbs.resv	resv.<attribute name>	
	resv.resvid	
pbs.server	pbs.server().<attribute name>	
	pbs.server().name	
pbs.vnode	vnode.<attribute name>	
	pbs.vchunk	vchunk.chunk_resources[]
		vchunk.vnode_name

6.13.2 Methods Available in Events

The following table lists all methods, and shows which event can use each method. A "*y*" means that the hook can use the method, an "*n*" means it cannot, and an "*o*" means that it can but will have no effect:

Table 6-14: Methods Available in Events

Method	queuejob	modifyjob (before run)	movejob	runjob (on reject)	runjob (on accept)	periodic	resvsub	resv_end	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic	provision
job.delete()	n	n	n	n	n	n	o	o	y	o	y	o	o	o	o	y	y	o	y	n
job.in_ms_mom()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
job.is_checkpointed()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
job.release_nodes()	n	n	n	n	n	n	n	o	n	n	y	y	n	n	n	n	n	n	n	n
job.rerun()	n	n	n	n	n	n	o	o	y	o	y	o	o	o	o	y	y	o	y	n
pbs.acl()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.args()	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
pbs.checkpoint()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.depend()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.depend()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.duration()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.email_list()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.event().accept()	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
pbs.event().reject()	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
pbs.exec_host()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.exec_vnode()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.get_local_nodename()	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
pbs.group_list()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.hold_types()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.job_sort_formula()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.join_path()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.keep_files()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.license_count()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.logjobmsg()	y	y	y	y	y	y	y	o	y	y	y	y	y	y	y	y	y	y	y	n
pbs.logmsg()	y	y	y	y	y	y	y	o	y	y	y	y	y	y	y	y	y	y	y	n
pbs.mail_points()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.node_group_key()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.path_list()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.pbs_env()	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
pbs.place()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.range()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.reboot()	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y
pbs.route_destinations()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.select()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.select.increment_chunks()	y	y	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
pbs.server().job('<job ID>')	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
pbs.server().jobs()	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n

Table 6-14: Methods Available in Events

Method	queuejob	modifyjob (before run)	movejob	runjob (on reject)	runjob (on accept)	periodic	resvsub	resv_end	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic	provision
pbs.server().queue("<queue_name>")	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
pbs.server().queues()	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
pbs.server().resv("<reservation ID>")	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
pbs.server().resvs()	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
pbs.server().scheduler_restart_cycle()	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
pbs.server().vnode("<vnode name>")	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
pbs.server().vnodes()	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
pbs.size()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.software()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.staging_list()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.state_count()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.user_list()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.version()	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
queue.job()	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
queue.jobs()	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
vchunk.chunk_resources.keys()	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n

6.13.3 PBS Types and Their Methods

6.13.3.1 Method to Create or Set ACL

pbs.acl()

pbs.acl("[+|-]<entity>][,...]")

Creates an object representing a PBS ACL, from the specified formatted input string.

6.13.3.2 Method to Create or Set Command Argument List

pbs.args()

pbs.args("<args>")

where <*args*> are space-separated arguments to a command.

Creates an object representing the arguments to the command from the specified formatted input string <*args*>.

Example of setting a command argument list:

```
pbs.args("-Wsuppress_email=N -r y")
```

6.13.3.3 Method to Create or Set Checkpoint String

pbs.checkpoint()

pbs.checkpoint("<checkpoint_string>")

where <*checkpoint_string*> must be one of "*n*", "*s*", "*c*", "*c=mmm*", "*w*", or "*w=mmm*"

Creates an object representing the job Checkpoint attribute, using the specified formatted input string <*checkpoint_string*>.

6.13.3.4 Method to Create or Set Dependency Object

pbs.depend()

pbs.depend("<depend_string>")

<*depend_string*> must be of format *"<type>:<jobid>[,<jobid>...]"*, or *"on:<count>"*.

where <*type*> is one of "*after*", "*afterok*", "*afterany*", "*afternotok*", "*before*", "*beforeok*", "*beforeany*", and "*beforenotok*".

Creates a PBS dependency specification object representing the job depend attribute, using the given <*depend_string*>.

Usage:

pbs.event().job.depend = pbs.depend("<depend_string>")

6.13.3.5 Method to Create or Set Duration from Time String or Integer

pbs.duration()

pbs.duration("[[hours:]minutes:]seconds[.milliseconds]")

Creates a time specification duration instance, returning the equivalent number of seconds from the given time string. Represents an interval or elapsed time in number of seconds. Duration objects can be specified using either a time or an integer. See "Method to Create or Set Duration from Time String or Integer".

pbs.duration(<integer>)

Creates an integer duration instance using the specified number of seconds.

A pbs.duration instance can be operated on by any of the Python int functions. When performing arithmetic operations on a pbs.duration type, ensure the resulting value is a pbs.duration() type, before assigning to a job member that expects such a type.

Example:

```
pbs.event().job.Resource_List["cput"] = pbs.duration(300 + d1) # safe
```

The following will **not** work, since Python evaluates from left to right, and returns result as the type at left (int):

```
d1 = pbs.duration(30)
pbs.event().job.Resource_List["cput"] = 300  + d1
```

6.13.3.6 Method to Create or Set Email List

pbs.email_list()

pbs.email_list("<email_address1>[, <email address2>...]")

Creates an object representing a mail list from the specified formatted input string.

6.13.3.7 Method to Create or Set exec_host Object

pbs.exec_host()

*pbs.exec_host("host/N[*C][+...]")*

Create an object representing the exec_host job attribute, using the specified input string containing host and resource specification.

6.13.3.8 Method to Create or Set exec_vnode Object

pbs.exec_vnode()

pbs.exec_vnode("<vchunk>[+<vchunk> ...]")

<vchunk> is *(<vnodename:ncpus=N:mem=M>)*

Creates an object representing the exec_vnode job attribute, using the input string containing the vnode and resource specification. When the `qrun -H` command is used, or when the scheduler runs a job, the job.exec_vnode object contains the vnode specification for the job.

Example:

```
pbs.exec_vnode("(vnodeA:ncpus=N:mem=X)+(nodeB:ncpus=P:mem=Y+nodeC:mem=Z)")
```

This object is managed and accessed via the str() or repr() functions. Example:

```
Python> ev = pbs.server().job("10").exec_vnode
Python> str(ev)                "(vnodeA:ncpus=2:mem=200m)+(vnodeB:ncpus=5:mem=1g)"
```

6.13.3.9 Method to Create or Set group_list Object

pbs.group_list()

pbs.group_list("<group_name>[@<host>][,<group_name>[@<host>]...]")

Creates an object representing a PBS group list from the specified formatted input string.

To use a group list object:

```
job.group_list = pbs.group_list(....)
```

6.13.3.10 Method to Create or Set hold_types Object

pbs.hold_types()

pbs.hold_types("<hold_type_str>")

where *<hold_type_str>* is one of "*u*", "*o*", "*s*", or "*n*".

Creates an object representing the Hold_Types job attribute from the specified formatted input string.

6.13.3.11 Method to Create or Set job_sort_formula Object

pbs.job_sort_formula()

pbs.job_sort_formula("<formula string>")

where *<formula string>* is a string containing a math formula. See section 4.9.21, "Using a Formula for Computing Job Execution Priority", on page 151.

Creates an object representing the job_sort_formula server attribute from the specified formatted input string.

6.13.3.12 Method to Create or Set join_path Object

pbs.join_path()

pbs.join_path({"oe"|"eo"|"n"})

Creates an object representing the Join_Path job attribute from the specified formatted input string.

6.13.3.13 Method to Create or Set keep_files Object

pbs.keep_files()

pbs.keep_files("<Keep_Files option>")

where *<Keep_Files option>* is one of "n", "d", "o", "e", "oe", "eo".

Creates an object representing the Keep_Files job attribute from the specified formatted input string.

6.13.3.14 Method to Create or Set license_count Object

pbs.license_count()

pbs.license_count("Avail_Global:<value> Avail_Local:<value> Used:<value> High_Use:<value>")

Instantiates an object representing a license_count attribute from the specified formatted input string.

6.13.3.15 Method to Create or Set mail_points Object

pbs.mail_points()

pbs.mail_points("<mail points string>")

where *mail points string* is "*a*", "*b*", and/or "*e*", optionally with *"j"*, or "*n*".

Creates a pbs.mail_points object representing a Mail_Points attribute from the specified formatted input string.

6.13.3.16 Method to Create or Set node_group_key Object

pbs.node_group_key()

pbs.node_group_key("<resource(s)>")

Creates a pbs.node_group_key object representing the resource(s) to be used for node grouping, using the specified resource(s). The input string is a comma-separated, quoted list of resources.

6.13.3.17 Method to Create or Set path_list Object

pbs.path_list()

pbs.path_list("<path>[@<host>][,<path>@<host> ...]")

Creates an object representing a PBS pathname list from the specified formatted input string.

To use a path list object:

```
job.Shell_Path_List = pbs.path_list(....)
```

6.13.3.18 Method to Create or Set Job Environment Object

pbs.pbs_env()

Creates an empty environment variable list.

For example, to clear an environment variable list:

```
pbs.event().env = pbs.pbs_env()
```

6.13.3.19 Method to Create or Set Resource List

pbs.pbs_resource()

pbs.pbs_resource(<resource list name>)

Creates a pbs.pbs_resource object with the specified name.

To set values for a pbs.pbs_resource object:

<resource list name>['<resource name>']=<resource value>

For example:

```
Resource_List['ncpus']=8
Resource_List['mem']=pbs.size("10gb")
Resource_List['walltime']=pbs.duration('00:45:00')
```

A pbs.pbs_resource is similar to a dictionary, but you cannot use direct traversal. To loop through entries:

for r in <list name>.keys():

...

For example:

```
for r in <Resource_List>.keys():
    pbs.logmsg(pbs.LOG_DEBUG, "Resource_List[%s]=%s" % (r, Resource_List[r]))
```

which produces the following log message:

```
03/08/2018 18:47:16;0006;pbs_python;Hook;pbs_python;Resource_List[walltime]=00:45:00
03/08/2018 18:47:16;0006;pbs_python;Hook;pbs_python;Resource_List[mem]=10gb
03/08/2018 18:47:16;0006;pbs_python;Hook;pbs_python;Resource_List[ncpus]=7
```

A str(<object of type pbs.pbs_resource>) produces output of the form:

<resource name>=<value>,<resource name>=<value>, ...

To do the equivalent of str():

```
pbs.logmsg(pbs.LOG_DEBUG, "Resource_List is %s (%s)" % (Resource_List, type(Resource_List)))
```

This produces the following log message:

```
03/08/2018 18:47:16;0006;pbs_python;Hook;pbs_python;Resource_List is
   mem=10gb,ncpus=7,walltime=00:45:00 (<class 'pbs.v1._base_types.pbs_resource'>)
```

6.13.3.20 Method to Create or Set place Object

pbs.place()

pbs.place("[arrangement]:[sharing]:[group]")

arrangement can be "*pack*", "*scatter*", "*free*", "*vscatter*"

sharing can be *"shared"*, *"excl", "exclhost"*

group can be of the form "*group=<resource>*"

[arrangement], *[sharing]*, and *[group]* can be given in any order or combination.

Creates a place object representing the job's place specification from the specified formatted input string.

Example:

```
pl = pbs.place("pack:excl")
s = repr(pl)            (or s = `pl`)
letter = pl[0]          (assigns 'p' to letter)
s = s + ":group=host"   (append to string)
pl = pbs.place(s)       (update original pl)
```

6.13.3.21 Method to Create or Set range Object

pbs.range()

pbs.range("<start>-<stop>:<step>")

Creates a PBS object representing a range of values from the specified formatted input string. Can be used to create a job.array_indices_submitted object. See section 6.6.1.2, "Job array_indices_submitted Attribute Member", on page 123.

Example:

```
pbs.range("1-30:3")
```

6.13.3.22 Method to Create or Set route_destinations Object

pbs.route_destinations()

pbs.route_destinations("<queue_spec>[,<queue_spec>,...]")

where *<queue_spec>* is *queue_name[@server_host[:port]]*

Creates an object that represents a route_destinations routing queue attribute from the specified formatted input string.

6.13.3.23 Method to Create or Set select Object

pbs.select()

pbs.select("[N:]res=val[:res=val]...[+[N:]res=val[:res=val] ...]")

Creates a select object representing the job's select specification from the specified formatted input string.

Example:

```
sel = pbs.select("2:ncpus=1:mem=5gb+3:ncpus=2:mem=5gb")
s = repr(sel)               (or s = `sel`)
letter = s[3]               (assigns 'c' to letter)
s = s + "+5:scratch=10gb"   (append to string)
sel = pbs.select(s)         (reset the value of sel)
```

6.13.3.24 Method to Increment select Object Chunks

pbs.select.increment_chunks()

pbs.select.increment_chunks(<increment specification>)

Creates a select object representing the job's new select specification, which has been padded from the original according to the *increment specification.*

You can pad all chunks, but you do not pad the primary vnode request itself; the job can only request one primary vnode. So when a job requests 3:ncpus=8+4:ncpus=1, the non-paddable primary vnode is considered to be a separate request of 1:ncpus=8, and the paddable part is the remaining 2:ncpus=8+4:ncpus=1. :

Table 6-15: Behavior for increment specification

Value of increment specification	Behavior
Integer amount Format: can be with or without quotes (a number or a numeric string), e.g. 5 or "5"	Adds specified number of vnodes to each chunk in the job's vnode request. Examples: Given this initial select statement: my_select=pbs.select("ncpus=3:mem=1gb+1:ncpus=2:mem=2gb+2:ncpus=1:mem=3gb") Calling my_select.increment_chunks(2) returns: "1:ncpus=3:mem=1gb+3:ncpus=2:mem=2gb+4:ncpus=1:mem=3gb" Calling my_select.increment_chunks("3") returns: "1:ncpus=3:mem=1gb+4:ncpus=2:mem=2gb+5:ncpus=1:mem=3gb"
Percentage amount Format: a quoted numeric string ending in a per-cent sign, e.g. "10%"	Adds specified percent of vnodes to each chunk in the job's vnode request. Resulting amounts are rounded up. Example: Given this initial select statement: my_select=pbs.select("ncpus=3:mem=1gb+1:ncpus=2:mem=2gb+2:ncpus=1:mem=3gb") Calling my_select.increment_chunks("23.5%") returns: "1:ncpus=3:mem=1gb+2:ncpus=2:mem=2gb+3:ncpus=1:mem=3gb" The first chunk, which is a single chunk, is left as is, and the second and third chunks are increased by 23.5 %. 1.24 is rounded up to 2, and 2.47 is rounded up to 3.
Per-chunk specification Format: {*<chunk index> : <increment>, ...*} where the increment can be an integer or a percentage	Adds specified amount or percent to specified chunk(s). Chunk index starts at *0*. Examples: Given this initial select statement: my_select=pbs.select("ncpus=3:mem=1gb+1:ncpus=2:mem=2gb+2:ncpus=1:mem=3gb") Calling my_select.increment_chunks({0: 0, 1: 4, 2: "50%"}) returns: "1:ncpus=3:mem=1gb+5:ncpus=2:mem=2gb+3:ncpus=1:mem=3gb" There is no increase (0) for chunk 1, we give 4 additional chunks to chunk 2, and we increase chunk 3 by 50%, resulting in 3. Given this initial select statement: my_select=pbs.select("5:ncpus=3:mem=1gb+4:ncpus=2:mem=2gb+2:ncpus=1:mem=3gb") Calling my_select.increment_chunks("50%") or my_select.increment_chunks({0: "50%", 1: "50%", 2: "50%}) returns: "7:ncpus=3:mem=1gb+6:ncpus=2:mem=2gb+3:ncpus=1:mem=3gb" The primary vnode is broken out as "1:ncpus=3:mem=1gb" and is left as is. The "50%" increase is applied to the remaining portion, "4:ncpus=3:mem=1gb". After the increase is applied, the original first chunk is re-created from the primary vnode and the padded remains of the first chunk to make 7. Chunk 2 gets 6 and chunk 3 gets 3.

6.13.3.24.i Example of Padding Chunks

The following code snippet illustrates padding a job's vnode request by one extra vnode per chunk:

```
import pbs
e=pbs.event()
j = e.job
new_select = e.job.Resource_List["select"].increment_chunks(1)
e.job.Resource_List["select"] = new_select
```

6.13.3.25 Method to Create or Set size Object

pbs.size()

You can create a pbs.size object using either a byte count or a suffix:

pbs.size(<integer>)

Creates a PBS size object using integer byte count, storing the value as the number of bytes. Size objects can be specified using either an integer or a string. See the "pbs.size("<integer><suffix>")" creation method.

pbs.size("<integer><suffix>")

Creates a PBS size object using the specified suffix. The suffix must be a multiplier defined in the table shown in "Size" on page 362 of the PBS Professional Reference Guide. The size of a word is the word size on the execution host. Size objects can be specified using either an integer or a string.

To operate on pbs.size instances, use the "+" and "-" operators.

To compare pbs.size instances, use the "==", "!=", ">", "<", ">=", and "<=" operators.

Example: the sizes are normalized to the smaller of the 2 suffixes. In this case, "10gb" becomes "10240mb" and is added to "10mb":

```
sz = pbs.size("10gb")
sz = sz + 10mb
10250mb
```

Example: the following returns *True* because sz is greater than 100 bytes:

```
if  sz > 100:
    gt100 = True
```

6.13.3.26 Method to Create or Set Software Resource Object

pbs.software()

pbs.software("<software info string>")

Creates an object representing a site-dependent software resource from the specified formatted input string.

6.13.3.27 Method to Create or Set staging_list Object

pbs.staging_list()

pbs.staging_list("<filespec>[,<filespec>,...]")

where *<filespec>* is *<execution_path>@<storage_host>:<storage_path>*

Creates an object representing a job file staging parameters list from the specified formatted input string.

To use a staging list object:

```
job.stagein = pbs.staging_list(....)
```

6.13.3.28 Method to Create or Set state_count Object

pbs.state_count()

pbs.state_count("Transit:<U> Queued:<V> Held:<W> Running:<X> Exiting:<Y> Begun:<Z>")

Instantiates an object representing a state_count attribute from the specified formatted input string.

6.13.3.29 Method to Create or Set user_list Object

pbs.user_list()

pbs.user_list("<user>[@<host>][,<user>@<host>...]")

Creates an object representing a PBS user list from the specified formatted input string.

To use a user list object:

```
job.User_List = pbs.user_list(....)
```

6.13.3.30 Method to Create or Set PBS Version Object

pbs.version()

pbs.version("<pbs version string>")

Creates an object representing the PBS version string from the specified formatted input string.

6.13.4 Global Methods

6.13.4.1 Method to Get Local Vnode Name

pbs.get_local_nodename()

This returns a Python str whose value is the name of the local parent vnode.

If you want to refer to the vnode object representing the current host, you can pass this vnode name as the key to pbs.event().vnode_list[]. For example:

```
Vn = pbs.event().vnode_list[pbs.get_local_nodename()]
```

6.13.4.2 Method to Log Job-related String

pbs.logjobmsg()

pbs.logjobmsg(job ID, message)

where *job ID* must be an existing or previously existing job ID and where *message* is an arbitrary string.

This puts a custom string in the log of the PBS daemon running the hook, so if the method is being run by a server hook such as queuejob, it prints to the server log, but if the method is being run at an execution host hook such as execjob_prologue, it prints to the MoM log.

The `tracejob` command can be used to print out the job-related messages logged by a hook script.

Messages are logged at log event class pbs.LOG_DEBUG. See Table 6.13.4.4, “Message Log Level Objects,” on page 152.

6.13.4.3 Method to Log String

pbs.logmsg()

pbs.logmsg(log event class, message)

where *message* is an arbitrary string, and where *log event class* can be one of the message log event class constants shown in Table 6-16, “Message Log Level Objects,” on page 152.

This puts a custom string in the log of the PBS daemon running the hook, so if the method is being run by a server hook such as queuejob, it prints to the server log, but if the method is being run at an execution host hook such as execjob_prologue, it prints to the MoM log.

Example:

```
for j in pbs.server().jobs():
   pbs.logmsg(pbs.LOG_DEBUG, "found job %s" % (j.id))
```

6.13.4.4 Message Log Level Objects

You can use the following objects to indicate log level when placing messages in the server logs.

Table 6-16: Message Log Level Objects

Object	Decimal	Hex	PBS Log Event Filter	Name and Event Category
pbs. EVENT_ERROR	1	0x0001	error	PBSEVENT_ERROR Internal errors
pbs. EVENT_SYSTEM	2	0x0002	system	PBSEVENT_SYSTEM system errors
pbs. EVENT_ADMIN	4	0x0004	admin	PBSEVENT_ADMIN Administrative events
pbs.LOG_WARNING	4	0x0004	admin	PBSEVENT_ADMIN Administrative events
pbs.LOG_ERROR	4	0x0004	admin	PBSEVENT_ADMIN Administrative events
pbs.LOG_DEBUG	4	0x0004	admin	PBSEVENT_ADMIN Administrative events
pbs. EVENT_JOB	8	0x0008	job	PBSEVENT_JOB Job-related events
pbs. EVENT_JOB_USAGE	16	0x0010	job_usage	PBSEVENT_JOB_USAGE Job accounting info
pbs. EVENT_SECURITY	32	0x0020	Security	PBSEVENT_SECURITY Security violations
pbs. EVENT_SCHED	64	0x0040	sched	PBSEVENT_SCHED Scheduler events

Table 6-16: Message Log Level Objects

Object	Decimal	Hex	PBS Log Event Filter	Name and Event Category
pbs. EVENT_DEBUG	128	0x0080	debug	PBSEVENT_DEBUG Common debug messages
pbs. EVENT_DEBUG2	256	0x0100	debug2	PBSEVENT_DEBUG2 Uncommon debug messages
pbs. EVENT_RESV	512	0x0200	resv	PBSEVENT_RESV Reservation-related events
pbs. EVENT_DEBUG3	1024	0x0400	debug3	PBSEVENT_DEBUG3 Less common than PBSEVENT_DEBUG2
pbs. EVENT_DEBUG4	2048	0x0800	debug4	PBSEVENT_DEBUG4 Less common than debug3
pbs. EVENT_FORCE	4096	0x8000	(No filter applies)	PBSEVENT_FORCE Forces a message to be logged

6.13.4.5 Method to Reboot Host

pbs.reboot()

pbs.reboot([<command>])

This stops hook execution, so that remaining lines in the hook script are not executed, and starts the tasks that would normally begin after the hook is finished, such as flagging the current host to be rebooted. The MoM logs show the following:

```
<hook name> requested for host to be rebooted
```

We recommend that before calling pbs.reboot(), you set any vnodes managed by this MoM offline, and requeue the current job, if this hook is not an exechost_periodic hook. For example:

```
for  v in pbs.event().vnode_list.keys():
    pbs.event().vnode_list[v].state = pbs.ND_OFFLINE
    pbs.event().vnode_list[v].comment = "Rebooting host"
pbs.event().job.rerun()
pbs.reboot()
```

The effect of the call to pbs.reboot() is not instantaneous. The reboot happens after the hook executes, and after any of the other actions such as pbs.event().job.rerun(), pbs.event().delete(), and pbs.event().vnode_list[] take effect.

A hook with its user attribute set to *pbsuser* cannot successfully invoke pbs.reboot(), even if the owner is a PBS Manager or Operator. If this is attempted, the host is not rebooted, and the following message appears at log event class PBSEVENT_DEBUG2 in the MoM logs:

```
<hook_name>; Not allowed to issue reboot if run as user.
```

The <*command*> is an optional argument. It is a Python str which is executed instead of the reboot command that is the default for the system. For example:

```
pbs.reboot("/usr/local/bin/my_reboot -s 10 -c 'going down in 10'")
```

The specified <*command*> is executed in a shell on Linux/UNIX or via cmd on Windows.

7
Built-in Hooks

7.1 Managing Built-in Hooks

PBS comes shipped with built-in hooks that implement features or patch bugs. You can operate on these hooks via `qmgr`. The `qmgr` keyword for built-in hooks is "*pbshook*". These hooks are named with the "*PBS*" prefix.

7.2 Prerequisites

You can operate on built-in hooks only from an account that has root access to the PBS server host.

When operating on a built-in hook, use the keyword "*pbshook*", not "*hook*".

7.3 Allowed Operations

You can perform a limited set of operations on built-in hooks. You can do the following:

- View attributes
- Set all attributes except for type
- Edit configuration files
- Replace with your own hook

7.4 Viewing Built-in Hooks

You can view attributes of built-in hooks:

```
# qmgr -c "list pbshook"
Hook PBS_example_hook
    type = pbs
    enabled = false
    event = queuejob,resvsub
    user = pbsadmin
    alarm = 90
    order = 1000
```

7.5 Setting Attributes of Built-in Hooks

You can set all attributes except for the type attribute for a built-in hook. For example, you can enable and disable built-in hooks:

```
# qmgr -c "set pbshook <built-in hook name> enabled=true"
# qmgr -c "set pbshook <built-in hook name> enabled=false"
```

If you disable a built-in hook, the following message is printed to qmgr's STDERR:

```
"WARNING: Disabling a PBS hook results in an unsupported configuration!"
```

7.6 Editing and Importing Configuration Files for Built-in Hooks

You can edit and re-import a configuration file for a built-in hook. Get the contents of the configuration file by exporting the file:

```
#qmgr -c "export pbshook <hook name> application/x-config default" > config_file_save
```

Edit the file (here, config_file.save), then re-import it:

```
# qmgr -c "import pbshook <hook name> application/x-config <content-encoding> default
    config_file.save"
```

7.7 Restrictions

- You cannot create or delete a built-in hook. Attempting to do so results in the following error being printed to qmgr's STDERR:
  `Invalid request`
- You cannot import or export content of a built-in hook. Attempting to do so results in the following error being printed to qmgr's STDERR:
  `<content-type> must be application/x-config`
- You cannot display the commands to re-create a built-in hook: using `qmgr -c "print pbshook"` won't work.

7.8 Replacing a Built-in Hook with Your Own Hook

You can replace a built-in hook with your own hook. For example, to replace a built-in exechost_startup hook:

1. Disable the built-in hook:
   ```
   # qmgr -c "set pbshook <built-in startup hook> enabled=false"
   ```
2. Create your own site-defined hook instead:
   ```
   # qmgr -c "create hook <your startup hook> event=exechost_startup"
   # qmgr -c "import hook <your startup hook> application/x-python default <your startup script>
   ```

7.9 Errors and Logging when Operating on Built-in Hooks

- If you try to operate on a built-in hook from an account that does not have root or Admin access, the following error message is issued to STDERR:

  "unable to generate a hook_tempfile from <filepath> - Permission denied"

  <user>@<host> is unauthorized to access hooks data from server <hostname>"

- If you try to import or export a built-in hook, you will see one of the following messages on STDERR:

  # qmgr -c "import pbshook <hook name> application/x-python default my_hook.py"

  <content-type> must be application/x-config

  or

  #qmgr -c "export pbshook <hook name> application/x-python default"

  <content-type> must be application/x-config

8 Debugging Hooks

8.1 The pbs_python Hook Debugging Tool

You can use the pbs_python wrapper that is shipped with PBS to debug hooks. Either:

- Use the --hook option to pbs_python to run pbs_python as a wrapper to Python, employing the pbs_python options. With the --hook option, you cannot use the standard Python options. The rest of this section covers how to use pbs_python with the --hook option.
- Do not use the --hook option, so pbs_python runs the Python interpreter, with the standard Python options, and without access to the pbs_python options.

Usage for pbs_python:

pbs_python --hook [-e <log event mask>] [-i <event input_file>] [-L <log dir>] [-l <log file>] [-o <hook execution record>] [-r <resourcedef file>] [-s site data file] [<python script>]

For a complete description of pbs_python, see "pbs_python" on page 82 of the PBS Professional Reference Guide.

8.2 Files for Debugging

You can get each hook to write out debugging files, and then modify the files and use them as debugging input to pbs_python. Alternatively, you can write the files yourself.

Debugging files can contain information about the event, about the site, and about what the hook changed. You can use these as inputs to a hook when debugging.

8.2.1 Producing Files for Debugging

To get a hook to write out event and site debugging files, and a hook execution record, set its debug attribute to *True* (the default is *False*). The files are named *hook_<event type>_<hook name>_<random integer>.in*, *.data*, and *.out*. The <*random integer*> is the same for all output files for one run of a hook. The <*random integer*> is different for each run.

The hook writes these files:

- *Event file*, containing the values that populate the pbs.event() objects in the hook and any other top level pbs objects like pbs.get_local_nodename(), and job and job list information. This file always contains the event type. The file is named *hook_<event type>_<hook name>_<random integer>.in*. Can be passed to pbs_python using -i <event file> option. See section 8.2.5, "Event File", on page 160.
- *Site data file*, containing the values that populate the pbs.server() objects: server, queue, vnode, etc. information. The site data file is named *hook_<event type>_<hook name>_<random integer>.data*. Can be passed to pbs_python using -s <site data file> option. This file is populated only when the hook calls pbs.server(). See section 8.2.6, "Site Data File", on page 164.
- *Hook execution record*, listing whether the hook accepted or rejected the event, and whatever was changed by the hook, named *hook_<event type>_<hook name>_<random integer>.out*. See section 8.2.7, "Hook Execution Record File", on page 164.

So for example an execjob_begin hook named *BeginHook* will produce files, if PBS chooses "*15223*" as its random integer, named *hook_execjob_begin_BeginHook_15223.in*, *hook_execjob_begin_BeginHook_15223.data*, and *hook_execjob_begin_BeginHook_15223.out*.

8.2.2 Locations for Debugging Files

These files are written to these locations:

- Pre-execution hooks: PBS_HOME/server_priv/hooks/tmp
- All exechost_* and execjob_* hooks: PBS_HOME/mom_priv/hooks/tmp

8.2.3 Format for Debugging Files

File format for debugging files is text. For example:

```
pbs.event().job.Hold_Types=u
pbs.event().job.Job_Name=STDIN
pbs.event().job.Checkpoint=u
pbs.event().job.Join_Path=n
pbs.event().job.Keep_Files=n
pbs.event().job.Mail_Points=a
pbs.event().job.Priority=0
pbs.event().job.Rerunable=TRUE
pbs.event().job.Resource_List[ncpus]=5
pbs.event().job.Resource_List[mem]=2gb
pbs.get_local_nodename()=mars.example.com
pbs.event().type=queuejob
pbs.event().hook_name=qjob
pbs.event().hook_type=site
pbs.event().requestor=TestUser
pbs.event().requestor_host=mars.example.com
pbs.event().user=pbsadmin
pbs.event().alarm=30
```

8.2.4 Time Limit for Debugging Files

PBS deletes hook .in, .data, and .out files in PBS_HOME/*/hooks/tmp that are older than 20 minutes. If you need to keep any of these files, copy them to another location.

8.2.5 Event File

The event file must contain the event type, and can contain any relevant information about the triggering event, the current job, or list of jobs.

When the hook writes it, this file contains the values that populate the pbs.event() objects in the hook and any other top level pbs objects such as the local vnode, the job, and the list of jobs. When a hook writes this file, it includes pbs.event().type and the result of get_local_nodename(). Each kind of hook writes different additional pbs.event() information.

The file is named *hook_<event type>_<hook name>_<random integer>.in*. It can be passed to `pbs_python` using the `-i <event file>` option.

The following table shows which information is written to the event file by each kind of hook:

Table 8-1: Event File by Hook

Event Information Written by Hooks: pbs.event.<list item>	queuejob	modifyjob (before run)	movejob	runjob (on reject)	runjob (on accept)	periodic	resvsub	resv_end	execjob_begin	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic	provision
pbs.get_local_nodename()	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	
alarm	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	
argv[]											y								
env											y								
freq																		y	
hook_name	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	
hook_type	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	
job.Checkpoint									y	y	y	y	y	y	y	y			
job.egroup									y	y	y	y	y	y	y	y			
job.Error_Path									y	y	y	y	y	y	y	y			
job.euser									y	y	y	y	y	y	y	y			
job.exec_vnode									y	y	y	y	y	y	y	y			
job.Exit_Status												y	y	y					
job.id									y	y	y	y	y	y	y	y			
job.jobdir												y	y	y	y	y			
job.job_kill_delay									y	y	y	y	y	y	y	y			
job.Job_Name									y	y	y	y	y	y	y	y			
job.Job_Owner									y	y	y	y	y	y	y	y			
job.job_state										y	y	y	y	y	y	y			
job.Join_Path									y	y	y	y	y	y	y	y			
job.Keep_Files									y	y	y	y	y	y	y	y			
job.mtime									y	y	y	y	y	y	y	y			
job.Output_Path									y	y	y	y	y	y	y	y			
job.Priority										y	y	y	y	y	y	y			
job.project									y	y	y	y	y	y	y	y			
job.queue									y	y	y	y	y	y	y	y			
job.resources_used[cpupercent]												y	y	y	y	y			
job.resources_used[cput]												y	y	y	y	y			
job.resources_used[mem]												y	y	y	y	y			
job.resources_used[ncpus]												y	y	y	y	y			
job.resources_used[vmem]												y	y	y		y			
job.resources_used[walltime]												y	y	y	y	y			
job.Resource_List[file]									y	y	y	y	y	y	y	y			
job.Resource_List[ncpus]									y	y	y	y	y	y	y	y			
job.Resource_List[place]									y	y	y	y	y	y	y	y			
job.run_count									y	y	y	y	y	y	y	y			
job.run_version									y	y	y	y	y	y	y	y			
job.schedselect									y	y	y	y	y	y	y	y			

Table 8-1: Event File by Hook

Event Information Written by Hooks: pbs.event.<list item>	queuejob	modifyjob (before run)	movejob	runjob (on reject)	runjob (on accept)	periodic	resvsub	resv_end	execjob_begin	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic	provision
job.server									y	y	y	y	y	y	y	y			
job.session_id												y	y	y	y	y			
job.substate										y	y	y	y	y	y	y			
job.Variable_List									y	y	y	y	y	y	y	y			
job_list["<job ID>"].Checkpoint																		y	
job_list["<job ID>"].egroup																		y	
job_list["<job ID>"].Error_Path																		y	
job_list["<job ID>"].euser																		y	
job_list["<job ID>"].exec_vnode																		y	
job_list["<job ID>"].hashname																		y	
job_list["<job ID>"].jobdir																		y	
job_list["<job ID>"].job_kill_delay																		y	
job_list["<job ID>"].Job_Name																		y	
job_list["<job ID>"].Job_Owner																		y	
job_list["<job ID>"].job_state																		y	
job_list["<job ID>"].Join_Path																		y	
job_list["<job ID>"].Keep_Files																		y	
job_list["<job ID>"].mtime																		y	
job_list["<job ID>"].Output_Path																		y	
job_list["<job ID>"].project																		y	
job_list["<job ID>"].queue																		y	
job_list["<job ID>"].resources_used[cpupercent]																		y	
job_list["<job ID>"].resources_used[cput]																		y	
job_list["<job ID>"].resources_used[mem]																		y	
job_list["<job ID>"].resources_used[ncpus]																		y	
job_list["<job ID>"].resources_used[walltime]																		y	
job_list["<job ID>"].Resource_List[file]																		y	
job_list["<job ID>"].Resource_List[ncpus]																		y	
job_list["<job ID>"].Resource_List[place]																		y	
job_list["<job ID>"].run_count																		y	
job_list["<job ID>"].run_version																		y	
job_list["<job ID>"].schedselect																		y	
job_list["<job ID>"].server																		y	
job_list["<job ID>"].session_id																		y	
job_list["<job ID>"].substate																		y	
job_list["<job ID>"].Variable_List																		y	
job_list["<job ID>"]._msmom																		y	
job_list["<job ID>"]._stderr_file																		y	
job_list["<job ID>"]._stdout_file																		y	
progname											y								

Table 8-1: Event File by Hook

Event Information Written by Hooks: pbs.event.<list item>	queuejob	modifyjob (before run)	movejob	runjob (on reject)	runjob (on accept)	periodic	resvsub	resv_end	execjob_begin	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic	provision
requestor	y	y	y	y	y		y	y	y	y	y	y	y	y	y	y	y	y	
requestor_host	y	y	y	y	y		y	y	y	y	y	y	y	y	y	y	y	y	
resv.reserve_end							y	y											
resv.reserve_start							y	y											
resv.Variable_List							y	y											
type	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	
user	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	
vnode_list["<local vnode name>"] .pbs_version									y	y	y	y	y	y	y	y		y	
vnode_list["<local vnode name>"] .pcpus									y	y	y	y	y	y	y	y		y	
vnode_list["<local vnode name>"] .resources_assigned[mem]									y	y	y	y	y	y	y	y			
vnode_list["<local vnode name>"] .resources_assigned[ncpus]									y	y	y	y	y	y	y	y			
vnode_list["<local vnode name>"] .resources_available[arch]									y	y	y	y	y	y	y	y		y	
vnode_list["<local vnode name>"] .resources_available[file]																		y	
vnode_list["<local vnode name>"] .resources_available[mem]									y	y	y	y	y	y	y	y	y	y	
vnode_list["<local vnode name>"] .resources_available[ncpus]									y		y	y	y	y	y	y	y	y	

For example, an event file created by a queuejob hook contains this data:

```
pbs.get_local_nodename()=jupiter.example.com
pbs.event().type=queuejob
pbs.event().hook_name=qjob
pbs.event().hook_type=site
pbs.event().requestor=TestUser
pbs.event().requestor_host=jupiter.example.com
pbs.event().user=pbsadmin
pbs.event().alarm=30
```

The equivalent command is:

```
Qmgr: list hook
```

8.2.5.1 Caveats

When the execjob_epilogue or execjob_end hook writes resources such as resources_used to the event file, it is writing about only the resources on the local host.

8.2.6 Site Data File

The site data file can contain any relevant information about the server, queues, vnodes, and jobs at the server.

This file is populated only when the hook calls pbs.server().

When the hook writes it, this file contains the values that populate the server, queues, vnodes, reservations, and jobs, with all attributes and resources for which there are values.

The site data file is named *hook_<event type>_<hook name>_<random integer>.data*. It can be passed to `pbs_python` using the `-s <site data file>` option.

The following commands give equivalent information:

```
qstat -Bf
qstat -Qf
qstat -f
pbsnodes -av
```

For example, here are some representative parts of a site data file:

```
pbs.server().scheduling=True
pbs.server().total_jobs=2
pbs.server().state_count=Transit:0 Queued:0 Held:2 Waiting:0 Running:0 Exiting:0 Begun:0
...
ppbs.server().default_chunk[ncpus]=1
pbs.server().resources_assigned[mem]=0mb
pbs.server().resources_assigned[ncpus]=0
...
pbs.server().job(501.jupiter.example.com).Job_Owner=TestUser@jupiter.example.com
pbs.server().job(501.jupiter.example.com).job_state=H
pbs.server().job(501.jupiter.example.com).queue=workq
...
pbs.server().job(501.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().job(501.jupiter.example.com).Resource_List[place]=pack
...
pbs.server().queue(workq).queue_type=Execution
pbs.server().queue(workq).total_jobs=2
pbs.server().queue(workq).resources_assigned[mem]=0mb
pbs.server().queue(workq).resources_assigned[ncpus]=0
```

8.2.7 Hook Execution Record File

The hook execution record file is produced when the hook runs. This file lists the following:

- Whether the event was accepted or rejected
- Any job values that were changed by the hook, showing the new values

This file is named *hook_<event type>_<hook name>_<random integer>.out*.

8.3 Steps to Debug a Hook Using pbs_python

When you debug a hook using pbs_python, give it the following information:

- Use the --hook option to pbs_python so that you can use the other pbs_python options
- Specify event information by using -i <event file>. At a minimum, include the type of the event, but you can also include job and job list information. Information about the event can be one of these:
  - An event information file (.in) written by the hook
  - A file written by you
  - Interactive input

  See section 8.2.5, "Event File", on page 160.
- Optionally, provide site data. Site data includes data about the server, queues, vnodes, etc. You specify site data in a file by using -s <site data file name>. If you do not specify the -s option, pbs_python connects to the server and obtains live data about the site. See section 8.2.6, "Site Data File", on page 164. Site data can come from one of these sources:
  - A site data file (.data) written by the hook
  - A file written by you
  - Interactive input
  - Live data from the server
- If you have added any custom resources, specify the PBS_HOME/server_priv/resourcedef file with the -r option to pbs_python. Make sure you specify the whole path to the file. For example:

  ```
  pbs_python --hook -r $PBS_HOME/server_priv/resourcedef -i <input_file> <hook.py>
  ```
- If your hook uses a configuration file, set the environment variable PBS_HOOK_CONFIG_FILE to the file's pathname before calling pbs_python. See section 5.1.6, "Using Hook Configuration Files", on page 32.
- Run pbs_python on the hook:

  ```
  pbs_python --hook -s <site data> -i <event file> <hook script>
  ```

8.4 Caveats and Restrictions for pbs_python

- When you run a hook inside pbs_python, it has access to the extended set of PBS_EXEC/python modules listed in section 4.5, "Python Modules and PBS", on page 24. When you run pbs_python at the command line (without --hook), the hook does not have access to the PBS_EXEC/lib set of modules.
- If PBS has attempted to run a job multiple times in the 20 minute window, you may need to check the timestamp of hook debugging files (e.g. ls -lt) to figure out which files were produced during a particular hook run.
- The site data file is populated only when the hook calls pbs.server().

8.5 Examples of Using `pbs_python` to Debug Hooks

Example 8-1: Basic periodic hook, with updates to vnodes:

- Input file:

```
% cat hook.input
pbs.event().type=exechost_periodic
pbs.event().vnode_list["host1"].state=free
pbs.get_local_nodename()=host1
```

- Hook file:

```
$ cat test.py
import pbs

e = pbs.event()

pbs.event().vnode_list[pbs.get_local_nodename()].resources_available["ncpus"]=7
pbs.event().vnode_list[pbs.get_local_nodename()].resources_available["mem"]=pbs.size("7gb")
```

- Run:

```
$ pbs_python --hook -i hook.input test.py
pbs.event().accept=True
pbs.event().reject=False
pbs.event().vnode_list["host1"].resources_available[ncpus]=7
pbs.event().vnode_list["host1"].resources_available[mem]=7gb
```

Example 8-2: A queuejob hook:

- Input file:

```
$ cat qjob.input
pbs.event().hook_name=qjob
pbs.event().hook_type=site
pbs.event().type=queuejob
pbs.event().requestor=user1
pbs.event().requestor_host=host1
pbs.event().alarm=40
pbs.event().job.Job_Name=pact
pbs.event().job.Resource_List[ncpus]=1
pbs.event().job.Resource_List[mem]=1mb
```

- Hook file:

```
$ cat qjob.py
import pbs

e = pbs.event()

e.job.Priority = 7
e.job.Account_Name = "mammoth"
e.job.Resource_List["ncpus"] = 5
```

```
e.job.Resource_List["mem"] = pbs.size("5gb")
```

- Run:

```
% pbs_python --hook -i qjob.input qjob.py
pbs.event().accept=True
pbs.event().reject=False
pbs.event().job.Priority=7
pbs.event().job.Resource_List[ncpus]=5
pbs.event().job.Resource_List[mem]=5gb
pbs.event().job.Account_Name=mammoth
```

Example 8-3: Reservation hook:

- Input file:

```
% cat rsub.input
pbs.event().hook_name=qjob
pbs.event().hook_type=site
pbs.event().type=resvsub
pbs.event().requestor=user1
pbs.event().requestor_host=host1
pbs.event().alarm=40
pbs.event().resv.Reserve_Name=my_resv
pbs.event().resv.Resource_List[ncpus]=1
pbs.event().resv.Resource_List[mem]=1mb
```

- Hook file:

```
% cat rsub.py
import pbs

def print_attribs(pbs_obj):
    for a in pbs_obj.attributes:
        v = getattr(pbs_obj, a)
        if v and str(v) != "":
            pbs.logmsg(pbs.LOG_DEBUG, "%s = %s" % (a,v))

e = pbs.event()
r = e.resv

print_attribs(r)

r.Resource_List["select"] = pbs.select("1:ncpus=1:mem=5mb")
r.Resource_List["place"] = pbs.place("pack:shared")

# group_list = pbs.group_list
r.group_list = pbs.group_list("Everyone,Everyone@host2,group1@jobim")

# Mail_Points= pbs.mail_points
r.Mail_Points = pbs.mail_points("a")
```

```
# User_List = pbs.user_list
r.User_List =  pbs.user_list("pbstest,pbstest@host2")

# Authorized_Users = pbs.acl
r.Authorized_Users =  pbs.acl("pbstest,user1,Administrator")

# Authorized_Groups = pbs.acl
r.Authorized_Groups =  pbs.acl("Everyone,group1,group2")
```

- Run:

```
% pbs_python  --hook -i rsub.input rsub.py
pbs.event().accept=True
pbs.event().reject=False
pbs.event().resv.group_list=Everyone,Everyone@host2,group1@jobim
pbs.event().resv.User_List=pbstest,pbstest@host2
pbs.event().resv.Resource_List[select]=1:ncpus=1:mem=5mb
pbs.event().resv.Resource_List[place]=pack:shared
pbs.event().resv.Mail_Points=a
pbs.event().resv.Authorized_Users=pbstest,user1,Administrator
pbs.event().resv.Authorized_Groups=Everyone,group1,group2
```

Example 8-4: A modifyjob hook:

- Hook script:

```
$ cat modifyjob.py

import pbs

def print_attribs(pbs_obj):
    for a in pbs_obj.attributes:
        v = getattr(pbs_obj, a)
        if v and str(v) != "":
            pbs.logmsg(pbs.LOG_DEBUG, "%s = %s" % (a,v))

e = pbs.event()

pbs.logmsg(pbs.LOG_DEBUG, "------> printing job %s" % (e.job_o.id))
print_attribs(e.job_o)
e.job.Priority = 5
e.job.Resource_List["file"] = pbs.size("7gb")
e.job.Variable_List["FILE"] = "7gb"
```

- Use the `pbs_python` debugging tool.

Ensure you have the following input file:

```
% cat hook.input
pbs.event().type=modifyjob
pbs.event().job.id=0.host1
pbs.event().job.Variable_List=A=b
```

- Run the hook:

```
% pbs_python --hook -i hook.input modifyjob.py
```

- The following are printed:

```
pbs.event().accept=True
pbs.event().reject=False

pbs.event().job.Variable_List=A=b,FILE=7gb
pbs.event().job.Priority=5
pbs.event().job.Resource_List[file]=7gb
```

- The `pbs_python` log file shows this:

```
% cat <yyyymmdd>

01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;----------------> printing job 0.host1
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;qtime = 1357387083
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Error_Path =
host1.example.com:/home/user1/STDIN.e0
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;job_state = 1
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;schedselect =1:ncpus=1
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;ctime = 1357387083
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Rerunable = 1
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;server = host1
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;egroup = pbs
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Variable_List =A=b,FILE=7gb
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Checkpoint = u
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;etime = 1357387083
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;queue = workq
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Job_Name = STDIN
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;comment = Not Running:
Could not run job - nodes are not licensed or unable to obtain 1 cpu licenses. avail_licenses=0
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;substate = 10
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;queue_rank = 1
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;euser = user1
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Mail_Points = a
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Priority = 0
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;project = _pbs_project_default
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;queue_type = 1
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Output_Path =
host1.example.com:/home/user1/STDIN.o0
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Hold_Types = n
```

```
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Join_Path = n
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;mtime = 1357387083
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;id = 0.host1
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Resource_List =
    select=1:ncpus=1,nodect=1,ncpus=1,place=free
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Keep_Files = n
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;_connect_server = host1
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Job_Owner = user1@host1.example.com
```

Example 8-5: A movejob hook which prints job ID, src_queue, and movejob event parameters, and sets src_queue:

- Hook script:

```
$ cat movejob.py

import pbs

def print_attribs(pbs_obj):
    for a in pbs_obj.attributes:
        v = getattr(pbs_obj, a)
        if v and str(v) != "":
            pbs.logmsg(pbs.LOG_DEBUG, "%s = %s" % (a,v))

e = pbs.event()

pbs.logmsg(pbs.LOG_DEBUG, "----------------> printing src_queue %s" % (e.src_queue.name))
print_attribs(e.src_queue)
pbs.logmsg(pbs.LOG_DEBUG, "----------------> printing job %s" % (e.job.id))
print_attribs(e.job)
e.job.queue = pbs.server().queue("workq2")
```

- Use the pbs_python debugging tool:

```
Use the following input file:
% cat hook.input2
pbs.event().type=movejob
pbs.event().job.id=<existing-job-id>
```

where <existing-job-id> must be some arbitrary job currently existing in the queue *workq*. Submit one (`qsub -h`) if it doesn't exist.

- Run the hook:

  `% pbs_python --hook -i hook.input2 movejob.py`

- The following is printed:

  ```
  pbs.event().accept=True
  pbs.event().reject=False
  pbs.event().src_queue=workq2
  ```

- The `pbs_python` log file shows this:

  `% cat <yyyymmdd>`

  ```
  01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;----------------> printing
  src_queue workq
  01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;name = workq
  01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;----------------> printing
  job 0.host1
  01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;qtime = 1357387083
  01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;Error_Path =
      host1.example.com:/home/user1/STDIN.e0
  01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;job_state = 1
  01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;ctime = 1357387083
  01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;Rerunable = 1
  01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;server = host1
  01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;Variable_List =
  PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash,PBS_O_HOME=/home/user1,PBS_O_HOST=host1.example.com,PBS
      _O_LOGNAME=user1,PBS_O_WORKDIR=/home/user1,PBS_O_LANG=en_US.UTF-8,PBS_O_PATH=/opt/pbs/bin:/o
      pt/pbs/python/bin:/opt/pbs/tcltk/bin:/home/user1/bin:/opt/pbs/bin:/opt/pbs/python/bin:/opt/p
      bs/tcltk/bin:/home/user1/bin:/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/home/user1/bin:/us
      r/local/rational/releases/purify.i386_linux2.2003a.06.15.FixPack.0194:/usr/local/purify/base
      /cots/flexlm.10.8.0.1/i386_linux2:/home/user1/PbsTestLab/bin:/home/user1/bin:/home/user1/bin
      :/usr/local/rational/releases/purify.i386_linux2.2003a.06.15.FixPack.0194:/usr/local/purify/
      base/cots/flexlm.10.8.0.1/i386_linux2:/home/user1/PbsTestLab/bin,PBS_O_QUEUE=workq,PBS_O_MAI
      L=/var/spool/mail/user1
  01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;Checkpoint = u
  01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;etime = 1357387083
  01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;Job_Name = STDIN
  01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;comment = Not Running:
  Could not run job - nodes are not licensed or unable to obtain 1 cpu licenses. avail_licenses=0
  01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;substate = 10
  01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;Mail_Points = a
  01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;Priority = 0
  01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;project = _pbs_project_default
  01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;Output_Path =
  host1.example.com:/home/user1/STDIN.o0
  01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;Hold_Types = n
  01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;Join_Path = n
  ```

```
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;mtime = 1357387083
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;id = 0.host1
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;Resource_List =
select=1:ncpus=1,nodect=1,ncpus=1,place=free
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;Keep_Files = n
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;_connect_server = host1
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;Job_Owner =
user1@host1.example.com
```

Example 8-6: A runjob hook to print attributes:

- Hook script:

```
$ cat runjob.py

import pbs
import time

def print_attribs(pbs_obj):
    for a in pbs_obj.attributes:
        v = getattr(pbs_obj, a)
        if v and str(v) != "":
            pbs.logmsg(pbs.LOG_DEBUG, "%s = %s" % (a,v))

e = pbs.event()

pbs.logmsg(pbs.LOG_DEBUG, "----------------> printing job %s" % (e.job.id))
print_attribs(e.job)
e.job.Hold_Types = pbs.hold_types("us")
e.job.Execution_Time = time.mktime([15, 11, 28, 14, 10, 15, -1, -1, 01])
e.job.project="looper"

pbs.event().reject("not allowed to run at this time!")
```

- Use the following input file:

```
% cat hook.input3
pbs.event().type=runjob
pbs.event().job.id=<existing-job-id>
```

where <*existing-job-id*> must be some arbitrary job currently existing in the server. Submit one (`qsub -h`) if it doesn't exist.

- Run the hook:

  `# pbs_python --hook -i hook.input3 runjob.py`

- The execution record contains the following:

```
pbs.event().reject=True
pbs.event().accept=False
pbs.event().reject_msg=not allowed to run at this time!
pbs.event().job.Execution_Time=1448745015
pbs.event().job.Hold_Types=us
pbs.event().job.project=looper
```

- The `pbs_python` log file shows:

```
% cat <yyyymmdd>

01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;----------------> printing
job 5.host1
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;qtime = 1357424154
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Error_Path =
host1.example.com:/home/user1/bugs/sp260361/STDIN.e5
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;job_state = 2
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;ctime = 1357424154
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Rerunable = 1
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;server = host1
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Variable_List =
PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash,PBS_O_HOME=/home/user1,PBS_O_HOST=host1.example.com,PBS
    _O_LOGNAME=user1,PBS_O_WORKDIR=/home/user1/bugs/sp260361,PBS_O_LANG=en_US.UTF-8,PBS_O_PATH=/
    opt/pbs/bin:/opt/pbs/python/bin:/opt/pbs/tcltk/bin:/home/user1/bin:/opt/pbs/bin:/opt/pbs/pyt
    hon/bin:/opt/pbs/tcltk/bin:/home/user1/bin:/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/home
    /user1/bin:/usr/local/rational/releases/purify.i386_linux2.2003a.06.15.FixPack.0194:/usr/loc
    al/purify/base/cots/flexlm.10.8.0.1/i386_linux2:/home/user1/PbsTestLab/bin:/home/user1/bin:/
    home/user1/bin:/usr/local/rational/releases/purify.i386_linux2.2003a.06.15.FixPack.0194:/usr
    /local/purify/base/cots/flexlm.10.8.0.1/i386_linux2:/home/user1/PbsTestLab/bin,PBS_O_QUEUE=w
    orkq,PBS_O_MAIL=/var/spool/mail/user1
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Checkpoint = u
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Submit_arguments =
<jsdl-hpcpa:Argument>-h</jsdl-hpcpa:Argument>
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;queue = workq
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Job_Name = STDIN
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;substate = 20
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Mail_Points = a
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Priority = 0
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;project = _pbs_project_default
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Output_Path =
host1.example.com:/home/user1/bugs/sp260361/STDIN.o5
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Hold_Types = u
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Join_Path = n
```

```
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;mtime = 1357424154
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;id = 5.host1
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Resource_List =
select=1:ncpus=1,nodect=1,ncpus=1,place=pack
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Keep_Files = n
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;_connect_server = host1
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Job_Owner =
user1@host1.example.com
```

8.6 Using Log Messages to Debug Hook Scripts

The following steps may help you avoid errors in hook scripts:

1. Create a hook, and import its content.
2. Temporarily set the server's log_events to a higher value such as *2047* to see plenty of logging.
3. Do a test run of the hook script, by causing events (e.g. `qsub`, `qalter`, `qmove`, `pbs_rsub`) that invoke the hook script. Check for error messages in the server logs.
4. Correct the hook script, re-import the fixed code, and rerun the test.
5. Once the hook script is running fine, then set the server's log_events back to the default (i.e. *511*).

8.7 Checking Hook Syntax using Python

You can check hook syntax using Python. If you run Python on the hook, the hook cannot import the pbs module. If the first error you see is a failure to import the pbs module, Python did not find any syntax errors.

8.8 Examples of Debugging Files

Example 8-7: We show several hooks and their debugging files. Our example hooks are queuejob, exechost_startup, exechost_periodic, execjob_begin, and execjob_launch.

Given the following two jobs in the system:

```
TestUser@jupiter:~/jobs> qstat
Job id            Name             User              Time Use S Queue
----------------  ---------------- ----------------  -------- - -----
501.jupiter          STDIN            TestUser                 0 H workq
502.jupiter          STDIN            TestUser                 0 H workq
```

Given the following reservations:

```
TestUser@jupiter:~/jobs> pbs_rstat
Resv ID    Queue    User     State             Start / Duration / End
--------------------------------------------------------------------
R503.jupiter. R503     TestUser@ CO          Today 08:00 / 1800 / Today 08:30
R504.jupiter. R504     TestUser@ CO          Today 09:00 / 1800 / Today 09:30
```

Given the following set of vnodes:

```
TestUser@jupiter:~/jobs> pbsnodes -av
jupiter
    Mom = jupiter.example.com
    Port = 15002
    pbs_version = PBSPro_10.0
    ntype = PBS
    state = free
    pcpus = 1
    resv = R504.jupiter.example.com, R503.jupiter.example.com
    resources_available.arch = linux
    resources_available.host = jupiter
    resources_available.mem = 8gb
    resources_available.ncpus = 8
    resources_available.vnode = jupiter
    resources_assigned.accelerator_memory = 0kb
    resources_assigned.mem = 0kb
    resources_assigned.naccelerators = 0
    resources_assigned.ncpus = 0
    resources_assigned.vmem = 0kb
    resv_enable = True
    sharing = default_shared
mars
    Mom = mars.example.com
    Port = 15002
    pbs_version = PBSPro_10.0
    ntype = PBS
    state = free
    pcpus = 1
    resources_available.arch = linux
    resources_available.host = mars
    resources_available.mem = 8gb
    resources_available.ncpus = 8
    resources_available.vnode = mars
    resources_assigned.accelerator_memory = 0kb
    resources_assigned.mem = 0kb
    resources_assigned.naccelerators = 0
    resources_assigned.ncpus = 0
    resources_assigned.vmem = 0kb
    resv_enable = True
    sharing = default_shared
```

queuejob hook attributes:

```
Hook qjob
    type = site
    enabled = true
    event = queuejob
    user = pbsadmin
    alarm = 30
    order = 1
    debug = true
    fail_action = none
```

queuejob hook contents:

```
import pbs
e=pbs.event()

e.job.Priority=7
e.job.Resource_List["file"] = pbs.size("7gb")

s=pbs.server()
for j in s.jobs():
    pbs.logmsg(pbs.LOG_DEBUG, "got j %s" % (j.id,))

for q in s.queues():
    pbs.logmsg(pbs.LOG_DEBUG, "got q %s" % (q.name,))

for v in s.vnodes():
    pbs.logmsg(pbs.LOG_DEBUG, "got vnode %s" % (v.name,))

for r in s.resvs():
    pbs.logmsg(pbs.LOG_DEBUG, "got resv %s" % (r.resvid))
```

Submit the job:

```
% qsub job.scr
```

Here are the resulting *.in, *.data, and *.out files:

```
jupiter:/var/spool/PBS/mom_priv/hooks/tmp # ls -ltr /var/spool/PBS/server_priv/hooks/tmp
-rw-r--r-- 1 root root   241 Sep 17 03:54 hook_queuejob_qjob_1410940476.in
-rw-r--r-- 1 root root 18619 Sep 17 03:54 hook_queuejob_qjob_1410940476.data
-rw-r--r-- 1 root root   805 Sep 17 03:54 hook_queuejob_qjob_1410940476.out
```

List the queuejob hook event file:

```
jupiter:/var/spool/PBS/server_priv/hooks/tmp # cat hook_queuejob_qjob_1410940476.in
pbs.get_local_nodename()=jupiter.example.com
pbs.event().type=queuejob
pbs.event().hook_name=qjob
pbs.event().hook_type=site
pbs.event().requestor=TestUser
pbs.event().requestor_host=jupiter.example.com
pbs.event().user=pbsadmin
pbs.event().alarm=30
```

List the queuejob hook site data file:

```
jupiter:/var/spool/PBS/server_priv/hooks/tmp # cat hook_queuejob_qjob_1410940476.data
pbs.server().server_state=Active
pbs.server().server_host=jupiter.example.com
pbs.server().scheduling=True
pbs.server().total_jobs=2
pbs.server().state_count=Transit:0 Queued:0 Held:2 Waiting:0 Running:0 Exiting:0 Begun:0
pbs.server().managers=TestUser@*
pbs.server().default_queue=workq
pbs.server().log_events=511
pbs.server().mail_from=adm
pbs.server().query_other_jobs=True
pbs.server().resources_default[ncpus]=1
pbs.server().default_chunk[ncpus]=1
pbs.server().resources_assigned[mem]=0mb
pbs.server().resources_assigned[ncpus]=0
pbs.server().resources_assigned[nodect]=0
pbs.server().scheduler_iteration=600
pbs.server().flatuid=True
pbs.server().resv_enable=True
pbs.server().node_fail_requeue=310
pbs.server().max_array_size=10000
pbs.server().pbs_license_min=1
pbs.server().pbs_license_max=2147483647
pbs.server().pbs_license_linger_time=3600
pbs.server().license_count=Avail_Global:0 Avail_Local:32 Used:0 High_Use:2
pbs.server().pbs_version=PBSPro_10.0
pbs.server().eligible_time_enable=False
pbs.server().max_concurrent_provision=5
pbs.server().job(501.jupiter.example.com).Job_Name=STDIN
pbs.server().job(501.jupiter.example.com).Job_Owner=TestUser@jupiter.example.com
pbs.server().job(501.jupiter.example.com).job_state=H
pbs.server().job(501.jupiter.example.com).queue=workq
pbs.server().job(501.jupiter.example.com).server=jupiter.example.com
pbs.server().job(501.jupiter.example.com).Checkpoint=u
pbs.server().job(501.jupiter.example.com).ctime=1410940219
pbs.server().job(501.jupiter.example.com).Error_Path=jupiter.example.com:/home/TestUser/jobs/STD
    IN.e501
pbs.server().job(501.jupiter.example.com).Hold_Types=u
pbs.server().job(501.jupiter.example.com).Join_Path=n
pbs.server().job(501.jupiter.example.com).Keep_Files=n
pbs.server().job(501.jupiter.example.com).Mail_Points=a
pbs.server().job(501.jupiter.example.com).mtime=1410940219
pbs.server().job(501.jupiter.example.com).Output_Path=jupiter.example.com:/home/TestUser/jobs/ST
    DIN.o501
pbs.server().job(501.jupiter.example.com).Priority=7
```

```
pbs.server().job(501.jupiter.example.com).qtime=1410940219
pbs.server().job(501.jupiter.example.com).Rerunable=True
pbs.server().job(501.jupiter.example.com).Resource_List[file]=7gb
pbs.server().job(501.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().job(501.jupiter.example.com).Resource_List[nodect]=1
pbs.server().job(501.jupiter.example.com).Resource_List[place]=pack
pbs.server().job(501.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().job(501.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().job(501.jupiter.example.com).substate=20
pbs.server().job(501.jupiter.example.com).Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash
    ,PBS_O_HOME=/home/TestUser,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LA
    NG=en_US.UTF-8,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/op
    enmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/g
    ames:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_QUEUE=
    workq,PBS_O_HOST=jupiter.example.com
pbs.server().job(501.jupiter.example.com).euser=TestUser
pbs.server().job(501.jupiter.example.com).egroup=users
pbs.server().job(501.jupiter.example.com).hop_count=1
pbs.server().job(501.jupiter.example.com).queue_rank=185
pbs.server().job(501.jupiter.example.com).queue_type=E
pbs.server().job(501.jupiter.example.com).Submit_arguments=<jsdl-hpcpa:Argument>-h</jsdl-hpcpa:A
    rgument>
pbs.server().job(501.jupiter.example.com).project=_pbs_project_default
pbs.server().queue(workq).queue_type=Execution
pbs.server().queue(workq).total_jobs=2
pbs.server().queue(workq).state_count=Transit:0 Queued:0 Held:2 Waiting:0 Running:0 Exiting:0
    Begun:0
pbs.server().queue(workq).resources_assigned[mem]=0mb
pbs.server().queue(workq).resources_assigned[ncpus]=0
pbs.server().queue(workq).resources_assigned[nodect]=0
pbs.server().queue(workq).enabled=True
pbs.server().queue(workq).started=True
pbs.server().job(502.jupiter.example.com).Job_Name=STDIN
pbs.server().job(502.jupiter.example.com).Job_Owner=TestUser@jupiter.example.com
pbs.server().job(502.jupiter.example.com).job_state=H
pbs.server().job(502.jupiter.example.com).queue=workq
pbs.server().job(502.jupiter.example.com).server=jupiter.example.com
pbs.server().job(502.jupiter.example.com).Checkpoint=u
pbs.server().job(502.jupiter.example.com).ctime=1410940221
pbs.server().job(502.jupiter.example.com).Error_Path=jupiter.example.com:/home/TestUser/jobs/STD
    IN.e502
pbs.server().job(502.jupiter.example.com).Hold_Types=u
pbs.server().job(502.jupiter.example.com).Join_Path=n
pbs.server().job(502.jupiter.example.com).Keep_Files=n
pbs.server().job(502.jupiter.example.com).Mail_Points=a
pbs.server().job(502.jupiter.example.com).mtime=1410940221
pbs.server().job(502.jupiter.example.com).Output_Path=jupiter.example.com:/home/TestUser/jobs/ST
```

```
    DIN.o502
pbs.server().job(502.jupiter.example.com).Priority=7
pbs.server().job(502.jupiter.example.com).qtime=1410940223
pbs.server().job(502.jupiter.example.com).Rerunable=True
pbs.server().job(502.jupiter.example.com).Resource_List[file]=7gb
pbs.server().job(502.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().job(502.jupiter.example.com).Resource_List[nodect]=1
pbs.server().job(502.jupiter.example.com).Resource_List[place]=pack
pbs.server().job(502.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().job(502.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().job(502.jupiter.example.com).substate=20
pbs.server().job(502.jupiter.example.com).Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash
    ,PBS_O_HOME=/home/TestUser,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LA
    NG=en_US.UTF-8,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/op
    enmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/g
    ames:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_QUEUE=
    workq,PBS_O_HOST=jupiter.example.com
pbs.server().job(502.jupiter.example.com).euser=TestUser
pbs.server().job(502.jupiter.example.com).egroup=users
pbs.server().job(502.jupiter.example.com).hop_count=1
pbs.server().job(502.jupiter.example.com).queue_rank=186
pbs.server().job(502.jupiter.example.com).queue_type=E
pbs.server().job(502.jupiter.example.com).Submit_arguments=<jsdl-hpcpa:Argument>-h</jsdl-hpcpa:A
    rgument>
pbs.server().job(502.jupiter.example.com).project=_pbs_project_default
pbs.server().queue(R503).queue_type=Execution
pbs.server().queue(R503).total_jobs=0
pbs.server().queue(R503).state_count=Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0
    Begun:0
pbs.server().queue(R503).acl_user_enable=True
pbs.server().queue(R503).acl_users=TestUser@jupiter.example.com
pbs.server().queue(R503).resources_max[ncpus]=1
pbs.server().queue(R503).resources_max[walltime]=00:30:00
pbs.server().queue(R503).resources_available[ncpus]=1
pbs.server().queue(R503).resources_available[walltime]=00:30:00
pbs.server().queue(R503).enabled=True
pbs.server().queue(R503).started=False
pbs.server().queue(R504).queue_type=Execution
pbs.server().queue(R504).total_jobs=0
pbs.server().queue(R504).state_count=Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0
    Begun:0
pbs.server().queue(R504).acl_user_enable=True
pbs.server().queue(R504).acl_users=TestUser@jupiter.example.com
pbs.server().queue(R504).resources_max[ncpus]=1
pbs.server().queue(R504).resources_max[walltime]=00:30:00
pbs.server().queue(R504).resources_available[ncpus]=1
pbs.server().queue(R504).resources_available[walltime]=00:30:00
```

```
pbs.server().queue(R504).enabled=True
pbs.server().queue(R504).started=False
pbs.server().vnode(jupiter).Mom=jupiter.example.com
pbs.server().vnode(jupiter).Port=15002
pbs.server().vnode(jupiter).pbs_version=PBSPro_10.0
pbs.server().vnode(jupiter).pcpus=1
pbs.server().vnode(jupiter).resv=R504.jupiter.example.com, R503.jupiter.example.com
pbs.server().vnode(jupiter).resources_available[arch]=linux
pbs.server().vnode(jupiter).resources_available[host]=jupiter
pbs.server().vnode(jupiter).resources_available[mem]=8gb
pbs.server().vnode(jupiter).resources_available[ncpus]=8
pbs.server().vnode(jupiter).resources_available[vnode]=jupiter
pbs.server().vnode(jupiter).resources_assigned[accelerator_memory]=0kb
pbs.server().vnode(jupiter).resources_assigned[mem]=0kb
pbs.server().vnode(jupiter).resources_assigned[naccelerators]=0
pbs.server().vnode(jupiter).resources_assigned[ncpus]=0
pbs.server().vnode(jupiter).resources_assigned[vmem]=0kb
pbs.server().vnode(jupiter).resv_enable=True
pbs.server().vnode(jupiter).sharing=1
pbs.server().resv(R503.jupiter.example.com).Reserve_Name=NULL
pbs.server().resv(R503.jupiter.example.com).Reserve_Owner=TestUser@jupiter.example.com
pbs.server().resv(R503.jupiter.example.com).reserve_type=2
pbs.server().resv(R503.jupiter.example.com).reserve_state=2
pbs.server().resv(R503.jupiter.example.com).reserve_substate=2
pbs.server().resv(R503.jupiter.example.com).reserve_start=1410955200
pbs.server().resv(R503.jupiter.example.com).reserve_end=1410957000
pbs.server().resv(R503.jupiter.example.com).reserve_duration=1800
pbs.server().resv(R503.jupiter.example.com).queue=R503
pbs.server().resv(R503.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().resv(R503.jupiter.example.com).Resource_List[walltime]=00:30:00
pbs.server().resv(R503.jupiter.example.com).Resource_List[nodect]=1
pbs.server().resv(R503.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().resv(R503.jupiter.example.com).Resource_List[place]=free
pbs.server().resv(R503.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().resv(R503.jupiter.example.com).resv_nodes=(jupiter:ncpus=1)
pbs.server().resv(R503.jupiter.example.com).Authorized_Users=TestUser@jupiter.example.com
pbs.server().resv(R503.jupiter.example.com).server=jupiter.example.com
pbs.server().resv(R503.jupiter.example.com).ctime=1410940237
pbs.server().resv(R503.jupiter.example.com).mtime=1410940237
pbs.server().resv(R503.jupiter.example.com).hop_count=1
pbs.server().resv(R503.jupiter.example.com).Variable_List=PBS_O_LOGNAME=TestUser,PBS_O_HOST=jupi
    ter.example.com,PBS_O_MAIL=/var/spool/mail/TestUser
pbs.server().resv(R503.jupiter.example.com).euser=TestUser
pbs.server().resv(R503.jupiter.example.com).egroup=users
pbs.server().resv(R504.jupiter.example.com).Reserve_Name=NULL
pbs.server().resv(R504.jupiter.example.com).Reserve_Owner=TestUser@jupiter.example.com
```

```
pbs.server().resv(R504.jupiter.example.com).reserve_type=2
pbs.server().resv(R504.jupiter.example.com).reserve_state=2
pbs.server().resv(R504.jupiter.example.com).reserve_substate=2
pbs.server().resv(R504.jupiter.example.com).reserve_start=1410958800
pbs.server().resv(R504.jupiter.example.com).reserve_end=1410960600
pbs.server().resv(R504.jupiter.example.com).reserve_duration=1800
pbs.server().resv(R504.jupiter.example.com).queue=R504
pbs.server().resv(R504.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().resv(R504.jupiter.example.com).Resource_List[walltime]=00:30:00
pbs.server().resv(R504.jupiter.example.com).Resource_List[nodect]=1
pbs.server().resv(R504.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().resv(R504.jupiter.example.com).Resource_List[place]=free
pbs.server().resv(R504.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().resv(R504.jupiter.example.com).resv_nodes=(jupiter:ncpus=1)
pbs.server().resv(R504.jupiter.example.com).Authorized_Users=TestUser@jupiter.example.com
pbs.server().resv(R504.jupiter.example.com).server=jupiter.example.com
pbs.server().resv(R504.jupiter.example.com).ctime=1410940250
pbs.server().resv(R504.jupiter.example.com).mtime=1410940250
pbs.server().resv(R504.jupiter.example.com).hop_count=1
pbs.server().resv(R504.jupiter.example.com).Variable_List=PBS_O_LOGNAME=TestUser,PBS_O_HOST=jupi
    ter.example.com,PBS_O_MAIL=/var/spool/mail/TestUser
pbs.server().resv(R504.jupiter.example.com).euser=TestUser
pbs.server().resv(R504.jupiter.example.com).egroup=users
```

List the queuejob hook execution record file:

```
jupiter:/var/spool/PBS/server_priv/hooks/tmp # cat hook_queuejob_qjob_1410940476.out
pbs.event().job.Rerunable=1
pbs.event().job.Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash,PBS_O_HOME=/home/TestUser
    ,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LANG=en_US.UTF-8,PBS_O_PATH=
    /usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/openmpi/bin:/home/TestUser/b
    in:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/games:/opt/pbs/bin:/opt/pbs
    /bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser
pbs.event().job.Checkpoint=u
pbs.event().job.Submit_arguments=<jsdl-hpcpa:Argument>job.scr</jsdl-hpcpa:Argument>
pbs.event().job.Job_Name=job.scr
pbs.event().job.Mail_Points=a
pbs.event().job.Priority=7
pbs.event().job.Hold_Types=n
pbs.event().job.Join_Path=n
pbs.event().job.Resource_List[file]=7gb
pbs.event().job.Keep_Files=n
```

The exechost_startup hook attributes:

```
Hook start
    type = site
    enabled = true
    event = exechost_startup
    user = pbsadmin
    alarm = 30
    order = 1
    debug = true
    fail_action = none
```

The exechost_startup hook contents:

```
import pbs
e=pbs.event()


e.vnode_list[pbs.get_local_nodename()].resources_available["file"] = pbs.size("7gb")

s=pbs.server()
for j in s.jobs():
    pbs.logmsg(pbs.LOG_DEBUG, "got j %s" % (j.id,))

for q in s.queues():
    pbs.logmsg(pbs.LOG_DEBUG, "got q %s" % (q.name,))

for v in s.vnodes():
    pbs.logmsg(pbs.LOG_DEBUG, "got vnode %s" % (v.name,))

for r in s.resvs():
    pbs.logmsg(pbs.LOG_DEBUG, "got resv %s" % (r.resvid))
```

Restart pbs_mom. Upon startup, the exechost_startup hook writes the following files:

```
jupiter:/home/TestUser/jobs # cd /var/spool/PBS/mom_priv/hooks/tmp
jupiter:/var/spool/PBS/mom_priv/hooks/tmp # ls -ltr
total 24
-rw-r--r-- 1 root root   455 Sep 17 04:02 hook_exechost_startup_start_11607.in
-rw-r--r-- 1 root root   115 Sep 17 04:02 hook_exechost_startup_start_11607.out
-rw-r--r-- 1 root root 12389 Sep 17 04:02 hook_exechost_startup_start_11607.data
```

List the exechost_startup hook event file:

```
jupiter:/var/spool/PBS/mom_priv/hooks/tmp # cat hook_exechost_startup_start_11607.in
pbs.event().vnode_list["jupiter"].resources_available[mem]=757388kb
pbs.event().vnode_list["jupiter"].resources_available[ncpus]=1
pbs.get_local_nodename()=jupiter
pbs.event().type=exechost_startup
pbs.event().hook_name=start
pbs.event().hook_type=site
pbs.event().requestor=pbs_mom
pbs.event().requestor_host=jupiter.example.com
pbs.event().user=pbsadmin
pbs.event().alarm=30
```

List the exechost_startup hook site data file:

```
jupiter:/var/spool/PBS/mom_priv/hooks/tmp # cat hook_exechost_startup_start_11607.data
pbs.server().server_state=Active
pbs.server().server_host=jupiter.example.com
pbs.server().scheduling=True
pbs.server().total_jobs=2
pbs.server().state_count=Transit:0 Queued:0 Held:2 Waiting:0 Running:0 Exiting:0 Begun:0
pbs.server().managers=TestUser@*
pbs.server().default_queue=workq
pbs.server().log_events=511
pbs.server().mail_from=adm
pbs.server().query_other_jobs=True
pbs.server().resources_default[ncpus]=1
pbs.server().default_chunk[ncpus]=1
pbs.server().resources_assigned[mem]=0mb
pbs.server().resources_assigned[ncpus]=0
pbs.server().resources_assigned[nodect]=0
pbs.server().scheduler_iteration=600
pbs.server().flatuid=True
pbs.server().resv_enable=True
pbs.server().node_fail_requeue=310
pbs.server().max_array_size=10000
pbs.server().pbs_license_min=1
pbs.server().pbs_license_max=2147483647
pbs.server().pbs_license_linger_time=3600
pbs.server().license_count=Avail_Global:0 Avail_Local:32 Used:0 High_Use:2
pbs.server().pbs_version=PBSPro_10.0
pbs.server().eligible_time_enable=False
pbs.server().max_concurrent_provision=5
pbs.server().job(501.jupiter.example.com).Job_Name=STDIN
pbs.server().job(501.jupiter.example.com).Job_Owner=TestUser@jupiter.example.com
pbs.server().job(501.jupiter.example.com).job_state=H
pbs.server().job(501.jupiter.example.com).queue=workq
pbs.server().job(501.jupiter.example.com).server=jupiter.example.com
pbs.server().job(501.jupiter.example.com).Checkpoint=u
pbs.server().job(501.jupiter.example.com).ctime=1410940219
pbs.server().job(501.jupiter.example.com).Error_Path=jupiter.example.com:/home/TestUser/jobs/STD
    IN.e501
pbs.server().job(501.jupiter.example.com).Hold_Types=u
pbs.server().job(501.jupiter.example.com).Join_Path=n
pbs.server().job(501.jupiter.example.com).Keep_Files=n
pbs.server().job(501.jupiter.example.com).Mail_Points=a
pbs.server().job(501.jupiter.example.com).mtime=1410940219
pbs.server().job(501.jupiter.example.com).Output_Path=jupiter.example.com:/home/TestUser/jobs/ST
    DIN.o501
pbs.server().job(501.jupiter.example.com).Priority=7
```

```
pbs.server().job(501.jupiter.example.com).qtime=1410940219
pbs.server().job(501.jupiter.example.com).Rerunable=True
pbs.server().job(501.jupiter.example.com).Resource_List[file]=7gb
pbs.server().job(501.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().job(501.jupiter.example.com).Resource_List[nodect]=1
pbs.server().job(501.jupiter.example.com).Resource_List[place]=pack
pbs.server().job(501.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().job(501.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().job(501.jupiter.example.com).substate=20
pbs.server().job(501.jupiter.example.com).Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash
    ,PBS_O_HOME=/home/TestUser,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LA
    NG=en_US.UTF-8,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/op
    enmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/g
    ames:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_QUEUE=
    workq,PBS_O_HOST=jupiter.example.com
pbs.server().job(501.jupiter.example.com).euser=TestUser
pbs.server().job(501.jupiter.example.com).egroup=users
pbs.server().job(501.jupiter.example.com).queue_rank=185
pbs.server().job(501.jupiter.example.com).queue_type=E
pbs.server().job(501.jupiter.example.com).Submit_arguments=<jsdl-hpcpa:Argument>-h</jsdl-hpcpa:A
    rgument>
pbs.server().job(501.jupiter.example.com).project=_pbs_project_default
pbs.server().job(502.jupiter.example.com).Job_Name=STDIN
pbs.server().job(502.jupiter.example.com).Job_Owner=TestUser@jupiter.example.com
pbs.server().job(502.jupiter.example.com).job_state=H
pbs.server().job(502.jupiter.example.com).queue=workq
pbs.server().job(502.jupiter.example.com).server=jupiter.example.com
pbs.server().job(502.jupiter.example.com).Checkpoint=u
pbs.server().job(502.jupiter.example.com).ctime=1410940221
pbs.server().job(502.jupiter.example.com).Error_Path=jupiter.example.com:/home/TestUser/jobs/STD
    IN.e502
pbs.server().job(502.jupiter.example.com).Hold_Types=u
pbs.server().job(502.jupiter.example.com).Join_Path=n
pbs.server().job(502.jupiter.example.com).Keep_Files=n
pbs.server().job(502.jupiter.example.com).Mail_Points=a
pbs.server().job(502.jupiter.example.com).mtime=1410940221
pbs.server().job(502.jupiter.example.com).Output_Path=jupiter.example.com:/home/TestUser/jobs/ST
    DIN.o502
pbs.server().job(502.jupiter.example.com).Priority=7
pbs.server().job(502.jupiter.example.com).qtime=1410940223
pbs.server().job(502.jupiter.example.com).Rerunable=True
pbs.server().job(502.jupiter.example.com).Resource_List[file]=7gb
pbs.server().job(502.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().job(502.jupiter.example.com).Resource_List[nodect]=1
pbs.server().job(502.jupiter.example.com).Resource_List[place]=pack
pbs.server().job(502.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().job(502.jupiter.example.com).schedselect=1:ncpus=1
```

```
pbs.server().job(502.jupiter.example.com).substate=20
pbs.server().job(502.jupiter.example.com).Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash
    ,PBS_O_HOME=/home/TestUser,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LA
    NG=en_US.UTF-8,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/op
    enmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/g
    ames:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_QUEUE=
    workq,PBS_O_HOST=jupiter.example.com
pbs.server().job(502.jupiter.example.com).euser=TestUser
pbs.server().job(502.jupiter.example.com).egroup=users
pbs.server().job(502.jupiter.example.com).queue_rank=186
pbs.server().job(502.jupiter.example.com).queue_type=E
pbs.server().job(502.jupiter.example.com).Submit_arguments=<jsdl-hpcpa:Argument>-h</jsdl-hpcpa:A
    rgument>
pbs.server().job(502.jupiter.example.com).project=_pbs_project_default
pbs.server().queue(workq).queue_type=Execution
pbs.server().queue(workq).total_jobs=2
pbs.server().queue(workq).state_count=Transit:0 Queued:0 Held:2 Waiting:0 Running:0 Exiting:0
    Begun:0
pbs.server().queue(workq).resources_assigned[mem]=0mb
pbs.server().queue(workq).resources_assigned[ncpus]=0
pbs.server().queue(workq).resources_assigned[nodect]=0
pbs.server().queue(workq).enabled=True
pbs.server().queue(workq).started=True
pbs.server().queue(R503).queue_type=Execution
pbs.server().queue(R503).total_jobs=0
pbs.server().queue(R503).state_count=Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0
    Begun:0
pbs.server().queue(R503).acl_user_enable=True
pbs.server().queue(R503).acl_users=TestUser@jupiter.example.com
pbs.server().queue(R503).resources_max[ncpus]=1
pbs.server().queue(R503).resources_max[walltime]=00:30:00
pbs.server().queue(R503).resources_available[ncpus]=1
pbs.server().queue(R503).resources_available[walltime]=00:30:00
pbs.server().queue(R503).enabled=True
pbs.server().queue(R503).started=False
pbs.server().queue(R504).queue_type=Execution
pbs.server().queue(R504).total_jobs=0
pbs.server().queue(R504).state_count=Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0
    Begun:0
pbs.server().queue(R504).acl_user_enable=True
pbs.server().queue(R504).acl_users=TestUser@jupiter.example.com
pbs.server().queue(R504).resources_max[ncpus]=1
pbs.server().queue(R504).resources_max[walltime]=00:30:00
pbs.server().queue(R504).resources_available[ncpus]=1
pbs.server().queue(R504).resources_available[walltime]=00:30:00
pbs.server().queue(R504).enabled=True
pbs.server().queue(R504).started=False
```

```
pbs.server().vnode(jupiter).Mom=jupiter.example.com
pbs.server().vnode(jupiter).Port=15002
pbs.server().vnode(jupiter).pbs_version=PBSPro_10.0
pbs.server().vnode(jupiter).ntype=0
pbs.server().vnode(jupiter).state=0
pbs.server().vnode(jupiter).pcpus=1
pbs.server().vnode(jupiter).resv=R504.jupiter.example.com, R503.jupiter.example.com
pbs.server().vnode(jupiter).resources_available[arch]=linux
pbs.server().vnode(jupiter).resources_available[host]=jupiter
pbs.server().vnode(jupiter).resources_available[mem]=8gb
pbs.server().vnode(jupiter).resources_available[ncpus]=8
pbs.server().vnode(jupiter).resources_available[vnode]=jupiter
pbs.server().vnode(jupiter).resources_assigned[accelerator_memory]=0kb
pbs.server().vnode(jupiter).resources_assigned[mem]=0kb
pbs.server().vnode(jupiter).resources_assigned[naccelerators]=0
pbs.server().vnode(jupiter).resources_assigned[ncpus]=0
pbs.server().vnode(jupiter).resources_assigned[vmem]=0kb
pbs.server().vnode(jupiter).resv_enable=True
pbs.server().vnode(jupiter).sharing=1
pbs.server().vnode(mars).Mom=mars.example.com
pbs.server().vnode(mars).Port=15002
pbs.server().vnode(mars).pbs_version=PBSPro_10.0
pbs.server().vnode(mars).ntype=0
pbs.server().vnode(mars).state=0
pbs.server().vnode(mars).pcpus=1
pbs.server().vnode(mars).resources_available[arch]=linux
pbs.server().vnode(mars).resources_available[host]=mars
pbs.server().vnode(mars).resources_available[mem]=8gb
pbs.server().vnode(mars).resources_available[ncpus]=8
pbs.server().vnode(mars).resources_available[vnode]=mars
pbs.server().vnode(mars).resources_assigned[accelerator_memory]=0kb
pbs.server().vnode(mars).resources_assigned[mem]=0kb
pbs.server().vnode(mars).resources_assigned[naccelerators]=0
pbs.server().vnode(mars).resources_assigned[ncpus]=0
pbs.server().vnode(mars).resources_assigned[vmem]=0kb
pbs.server().vnode(mars).resv_enable=True
pbs.server().vnode(mars).sharing=1
pbs.server().resv(R503.jupiter.example.com).Reserve_Name=NULL
pbs.server().resv(R503.jupiter.example.com).Reserve_Owner=TestUser@jupiter.example.com
pbs.server().resv(R503.jupiter.example.com).reserve_type=2
pbs.server().resv(R503.jupiter.example.com).reserve_state=2
pbs.server().resv(R503.jupiter.example.com).reserve_substate=2
pbs.server().resv(R503.jupiter.example.com).reserve_start=1410955200
pbs.server().resv(R503.jupiter.example.com).reserve_end=1410957000
pbs.server().resv(R503.jupiter.example.com).reserve_duration=1800
pbs.server().resv(R503.jupiter.example.com).queue=R503
```

```
pbs.server().resv(R503.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().resv(R503.jupiter.example.com).Resource_List[walltime]=00:30:00
pbs.server().resv(R503.jupiter.example.com).Resource_List[nodect]=1
pbs.server().resv(R503.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().resv(R503.jupiter.example.com).Resource_List[place]=free
pbs.server().resv(R503.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().resv(R503.jupiter.example.com).resv_nodes=(jupiter:ncpus=1)
pbs.server().resv(R503.jupiter.example.com).Authorized_Users=TestUser@jupiter.example.com
pbs.server().resv(R503.jupiter.example.com).server=jupiter.example.com
pbs.server().resv(R503.jupiter.example.com).ctime=1410940237
pbs.server().resv(R503.jupiter.example.com).mtime=1410940237
pbs.server().resv(R503.jupiter.example.com).Variable_List=PBS_O_LOGNAME=TestUser,PBS_O_HOST=jupi
    ter.example.com,PBS_O_MAIL=/var/spool/mail/TestUser
pbs.server().resv(R503.jupiter.example.com).euser=TestUser
pbs.server().resv(R503.jupiter.example.com).egroup=users
pbs.server().resv(R504.jupiter.example.com).Reserve_Name=NULL
pbs.server().resv(R504.jupiter.example.com).Reserve_Owner=TestUser@jupiter.example.com
pbs.server().resv(R504.jupiter.example.com).reserve_type=2
pbs.server().resv(R504.jupiter.example.com).reserve_state=2
pbs.server().resv(R504.jupiter.example.com).reserve_substate=2
pbs.server().resv(R504.jupiter.example.com).reserve_start=1410958800
pbs.server().resv(R504.jupiter.example.com).reserve_end=1410960600
pbs.server().resv(R504.jupiter.example.com).reserve_duration=1800
pbs.server().resv(R504.jupiter.example.com).queue=R504
pbs.server().resv(R504.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().resv(R504.jupiter.example.com).Resource_List[walltime]=00:30:00
pbs.server().resv(R504.jupiter.example.com).Resource_List[nodect]=1
pbs.server().resv(R504.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().resv(R504.jupiter.example.com).Resource_List[place]=free
pbs.server().resv(R504.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().resv(R504.jupiter.example.com).resv_nodes=(jupiter:ncpus=1)
pbs.server().resv(R504.jupiter.example.com).Authorized_Users=TestUser@jupiter.example.com
pbs.server().resv(R504.jupiter.example.com).server=jupiter.example.com
pbs.server().resv(R504.jupiter.example.com).ctime=1410940250
pbs.server().resv(R504.jupiter.example.com).mtime=1410940250
pbs.server().resv(R504.jupiter.example.com).Variable_List=PBS_O_LOGNAME=TestUser,PBS_O_HOST=jupi
    ter.example.com,PBS_O_MAIL=/var/spool/mail/TestUser
pbs.server().resv(R504.jupiter.example.com).euser=TestUser
pbs.server().resv(R504.jupiter.example.com).egroup=users
```

List the exechost_startup hook execution record file:

```
jupiter:/var/spool/PBS/mom_priv/hooks/tmp # cat hook_exechost_startup_start_11607.out
pbs.event().accept=True
pbs.event().reject=False
pbs.event().vnode_list["jupiter"].resources_available[file,size]=7gb
```

The exechost_periodic hook attributes:

```
Hook period
    type = site
    enabled = true
    event = exechost_periodic
    user = pbsadmin
    alarm = 30
    freq = 30
    order = 1
    debug = true
    fail_action = none
```

The contents of the exechost_periodic hook:

```
jupiter:/home/TestUser/jobs # qmgr -c "e h period application/x-python default"
import pbs
e=pbs.event()

e.vnode_list[pbs.get_local_nodename()].resources_available["file"] = pbs.size("7gb")

s=pbs.server()
for j in s.jobs():
    pbs.logmsg(pbs.LOG_DEBUG, "got j %s" % (j.id,))

for q in s.queues():
    pbs.logmsg(pbs.LOG_DEBUG, "got q %s" % (q.name,))

for v in s.vnodes():
    pbs.logmsg(pbs.LOG_DEBUG, "got vnode %s" % (v.name,))

for r in s.resvs():
    pbs.logmsg(pbs.LOG_DEBUG, "got resv %s" % (r.resvid))
```

In our example, we have two jobs running on the execution host:

```
jupiter:/var/spool/PBS/mom_priv/hooks/tmp # qstat
Job id         Name        User        Time Use  S Queue
------------   ----------- ----------- --------  - -----
501.jupiter       STDIN       TestUser      0        H workq
502.jupiter       STDIN       TestUser      0        H workq
506.jupiter       STDIN       TestUser      00:00:00 R workq
507.jupiter       STDIN       TestUser      00:00:00 R workq
```

The *.in, *.out, and *.data files end up here:

```
jupiter:/var/spool/PBS/mom_priv/hooks/tmp # ls -ltr
-rw-r--r-- 1 root root  6885 Sep 17 04:09 hook_exechost_periodic_period_11753.in
-rw-r--r-- 1 root root  1387 Sep 17 04:09 hook_exechost_periodic_period_11753.out
-rw-r--r-- 1 root root 19039 Sep 17 04:09 hook_exechost_periodic_period_11753.data
```

List the exechost_periodic event file:

```
jupiter:/var/spool/PBS/mom_priv/hooks/tmp # cat hook_exechost_periodic_period_11753.in
pbs.event().freq=30
pbs.event().vnode_list["jupiter"].pcpus=1
pbs.event().vnode_list["jupiter"].resources_available[ncpus]=1
pbs.event().vnode_list["jupiter"].resources_available[mem]=757388kb
pbs.event().vnode_list["jupiter"].resources_available[arch]=linux
pbs.event().vnode_list["jupiter"].pbs_version=PBSPro_10.0
pbs.event().vnode_list["jupiter"].resources_available[file]=7gb
pbs.event().job_list["506.jupiter.example.com"].Job_Name=STDIN
pbs.event().job_list["506.jupiter.example.com"].Job_Owner=TestUser@jupiter.example.com
pbs.event().job_list["506.jupiter.example.com"].resources_used[cpupercent]=0
pbs.event().job_list["506.jupiter.example.com"].resources_used[cput]=00:00:00
pbs.event().job_list["506.jupiter.example.com"].resources_used[mem]=3880kb
pbs.event().job_list["506.jupiter.example.com"].resources_used[ncpus]=1
pbs.event().job_list["506.jupiter.example.com"].resources_used[vmem]=32192kb
pbs.event().job_list["506.jupiter.example.com"].resources_used[walltime]=00:00:13
pbs.event().job_list["506.jupiter.example.com"].job_state=T
pbs.event().job_list["506.jupiter.example.com"].queue=workq
pbs.event().job_list["506.jupiter.example.com"].server=jupiter.example.com
pbs.event().job_list["506.jupiter.example.com"].Checkpoint=u
pbs.event().job_list["506.jupiter.example.com"].Error_Path=jupiter.example.com:/home/TestUser/jo
    bs/STDIN.e506
pbs.event().job_list["506.jupiter.example.com"].exec_host2=jupiter.example.com:15002/0
pbs.event().job_list["506.jupiter.example.com"].exec_vnode=(jupiter:ncpus=1)
pbs.event().job_list["506.jupiter.example.com"].Join_Path=n
pbs.event().job_list["506.jupiter.example.com"].Keep_Files=n
pbs.event().job_list["506.jupiter.example.com"].mtime=1410941347
pbs.event().job_list["506.jupiter.example.com"].Output_Path=jupiter.example.com:/home/TestUser/j
    obs/STDIN.o506
pbs.event().job_list["506.jupiter.example.com"].Resource_List[file]=7gb
pbs.event().job_list["506.jupiter.example.com"].Resource_List[ncpus]=1
pbs.event().job_list["506.jupiter.example.com"].Resource_List[place]=pack
pbs.event().job_list["506.jupiter.example.com"].schedselect=1:ncpus=1
pbs.event().job_list["506.jupiter.example.com"].session_id=11683
pbs.event().job_list["506.jupiter.example.com"].jobdir=/home/TestUser
pbs.event().job_list["506.jupiter.example.com"].substate=0
pbs.event().job_list["506.jupiter.example.com"].Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bi
    n/bash,PBS_O_HOME=/home/TestUser,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PB
    S_O_LANG=en_US.UTF-8,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/
    gcc/openmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:
    /usr/games:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_
    QUEUE=workq,PBS_O_HOST=jupiter.example.com
pbs.event().job_list["506.jupiter.example.com"].euser=TestUser
pbs.event().job_list["506.jupiter.example.com"].egroup=users
pbs.event().job_list["506.jupiter.example.com"].hashname=506.jupiter.example.com
pbs.event().job_list["506.jupiter.example.com"].cookie=000000002CEAFC4E0000000043354104
```

```
pbs.event().job_list["506.jupiter.example.com"].run_count=1
pbs.event().job_list["506.jupiter.example.com"].job_kill_delay=10
pbs.event().job_list["506.jupiter.example.com"].project=_pbs_project_default
pbs.event().job_list["506.jupiter.example.com"].run_version=1
pbs.event().job_list["506.jupiter.example.com"]._msmom=True
pbs.event().job_list["506.jupiter.example.com"]._stdout_file=/var/spool/PBS/spool/506.jupiter.example.com.OU
pbs.event().job_list["506.jupiter.example.com"]._stderr_file=/var/spool/PBS/spool/506.jupiter.example.com.ER
pbs.event().job_list["507.jupiter.example.com"].Job_Name=STDIN
pbs.event().job_list["507.jupiter.example.com"].Job_Owner=TestUser@jupiter.example.com
pbs.event().job_list["507.jupiter.example.com"].resources_used[cpupercent]=0
pbs.event().job_list["507.jupiter.example.com"].resources_used[cput]=00:00:00
pbs.event().job_list["507.jupiter.example.com"].resources_used[mem]=3892kb
pbs.event().job_list["507.jupiter.example.com"].resources_used[ncpus]=1
pbs.event().job_list["507.jupiter.example.com"].resources_used[vmem]=32192kb
pbs.event().job_list["507.jupiter.example.com"].resources_used[walltime]=00:00:10
pbs.event().job_list["507.jupiter.example.com"].job_state=T
pbs.event().job_list["507.jupiter.example.com"].queue=workq
pbs.event().job_list["507.jupiter.example.com"].server=jupiter.example.com
pbs.event().job_list["507.jupiter.example.com"].Checkpoint=u
pbs.event().job_list["507.jupiter.example.com"].Error_Path=jupiter.example.com:/home/TestUser/jobs/STDIN.e507
pbs.event().job_list["507.jupiter.example.com"].exec_host2=jupiter.example.com:15002/1
pbs.event().job_list["507.jupiter.example.com"].exec_vnode=(jupiter:ncpus=1)
pbs.event().job_list["507.jupiter.example.com"].Join_Path=n
pbs.event().job_list["507.jupiter.example.com"].Keep_Files=n
pbs.event().job_list["507.jupiter.example.com"].mtime=1410941350
pbs.event().job_list["507.jupiter.example.com"].Output_Path=jupiter.example.com:/home/TestUser/jobs/STDIN.o507
pbs.event().job_list["507.jupiter.example.com"].Resource_List[file]=7gb
pbs.event().job_list["507.jupiter.example.com"].Resource_List[ncpus]=1
pbs.event().job_list["507.jupiter.example.com"].Resource_List[place]=pack
pbs.event().job_list["507.jupiter.example.com"].schedselect=1:ncpus=1
pbs.event().job_list["507.jupiter.example.com"].session_id=11716
pbs.event().job_list["507.jupiter.example.com"].jobdir=/home/TestUser
pbs.event().job_list["507.jupiter.example.com"].substate=0
pbs.event().job_list["507.jupiter.example.com"].Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash,PBS_O_HOME=/home/TestUser,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LANG=en_US.UTF-8,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/openmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/games:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_QUEUE=workq,PBS_O_HOST=jupiter.example.com
pbs.event().job_list["507.jupiter.example.com"].euser=TestUser
pbs.event().job_list["507.jupiter.example.com"].egroup=users
pbs.event().job_list["507.jupiter.example.com"].hashname=507.jupiter.example.com
pbs.event().job_list["507.jupiter.example.com"].cookie=000000003C3AB5AC000000007A31CFD4
```

```
pbs.event().job_list["507.jupiter.example.com"].run_count=1
pbs.event().job_list["507.jupiter.example.com"].job_kill_delay=10
pbs.event().job_list["507.jupiter.example.com"].project=_pbs_project_default
pbs.event().job_list["507.jupiter.example.com"].run_version=1
pbs.event().job_list["507.jupiter.example.com"]._msmom=True
pbs.event().job_list["507.jupiter.example.com"]._stdout_file=/var/spool/PBS/spool/507.jupiter.ex
    ample.com.OU
pbs.event().job_list["507.jupiter.example.com"]._stderr_file=/var/spool/PBS/spool/507.jupiter.ex
    ample.com.ER
pbs.get_local_nodename()=jupiter
pbs.event().type=exechost_periodic
pbs.event().hook_name=period
pbs.event().hook_type=site
pbs.event().requestor=pbs_mom
pbs.event().requestor_host=jupiter.example.com
pbs.event().user=pbsadmin
pbs.event().alarm=30
```

List the exechost_periodic site data file:

```
jupiter:/var/spool/PBS/mom_priv/hooks/tmp # cat hook_exechost_periodic_period_11753.data
pbs.server().server_state=Active
pbs.server().server_host=jupiter.example.com
pbs.server().scheduling=True
pbs.server().total_jobs=4
pbs.server().state_count=Transit:0 Queued:0 Held:2 Waiting:0 Running:2 Exiting:0 Begun:0
pbs.server().managers=TestUser@*
pbs.server().default_queue=workq
pbs.server().log_events=511
pbs.server().mail_from=adm
pbs.server().query_other_jobs=True
pbs.server().resources_default[ncpus]=1
pbs.server().default_chunk[ncpus]=1
pbs.server().resources_assigned[mem]=0mb
pbs.server().resources_assigned[ncpus]=2
pbs.server().resources_assigned[nodect]=2
pbs.server().scheduler_iteration=600
pbs.server().flatuid=True
pbs.server().resv_enable=True
pbs.server().node_fail_requeue=310
pbs.server().max_array_size=10000
pbs.server().pbs_license_min=1
pbs.server().pbs_license_max=2147483647
pbs.server().pbs_license_linger_time=3600
pbs.server().license_count=Avail_Global:0 Avail_Local:30 Used:2 High_Use:2
pbs.server().pbs_version=PBSPro_10.0
pbs.server().eligible_time_enable=False
pbs.server().max_concurrent_provision=5
pbs.server().job(501.jupiter.example.com).Job_Name=STDIN
pbs.server().job(501.jupiter.example.com).Job_Owner=TestUser@jupiter.example.com
pbs.server().job(501.jupiter.example.com).job_state=H
pbs.server().job(501.jupiter.example.com).queue=workq
pbs.server().job(501.jupiter.example.com).server=jupiter.example.com
pbs.server().job(501.jupiter.example.com).Checkpoint=u
pbs.server().job(501.jupiter.example.com).ctime=1410940219
pbs.server().job(501.jupiter.example.com).Error_Path=jupiter.example.com:/home/TestUser/jobs/STD
    IN.e501
pbs.server().job(501.jupiter.example.com).Hold_Types=u
pbs.server().job(501.jupiter.example.com).Join_Path=n
pbs.server().job(501.jupiter.example.com).Keep_Files=n
pbs.server().job(501.jupiter.example.com).Mail_Points=a
pbs.server().job(501.jupiter.example.com).mtime=1410940219
pbs.server().job(501.jupiter.example.com).Output_Path=jupiter.example.com:/home/TestUser/jobs/ST
    DIN.o501
pbs.server().job(501.jupiter.example.com).Priority=7
```

```
pbs.server().job(501.jupiter.example.com).qtime=1410940219
pbs.server().job(501.jupiter.example.com).Rerunable=True
pbs.server().job(501.jupiter.example.com).Resource_List[file]=7gb
pbs.server().job(501.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().job(501.jupiter.example.com).Resource_List[nodect]=1
pbs.server().job(501.jupiter.example.com).Resource_List[place]=pack
pbs.server().job(501.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().job(501.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().job(501.jupiter.example.com).substate=20
pbs.server().job(501.jupiter.example.com).Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash
    ,PBS_O_HOME=/home/TestUser,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LA
    NG=en_US.UTF-8,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/op
    enmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/g
    ames:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_QUEUE=
    workq,PBS_O_HOST=jupiter.example.com
pbs.server().job(501.jupiter.example.com).euser=TestUser
pbs.server().job(501.jupiter.example.com).egroup=users
pbs.server().job(501.jupiter.example.com).queue_rank=185
pbs.server().job(501.jupiter.example.com).queue_type=E
pbs.server().job(501.jupiter.example.com).Submit_arguments=<jsdl-hpcpa:Argument>-h</jsdl-hpcpa:A
    rgument>
pbs.server().job(501.jupiter.example.com).project=_pbs_project_default
pbs.server().job(502.jupiter.example.com).Job_Name=STDIN
pbs.server().job(502.jupiter.example.com).Job_Owner=TestUser@jupiter.example.com
pbs.server().job(502.jupiter.example.com).job_state=H
pbs.server().job(502.jupiter.example.com).queue=workq
pbs.server().job(502.jupiter.example.com).server=jupiter.example.com
pbs.server().job(502.jupiter.example.com).Checkpoint=u
pbs.server().job(502.jupiter.example.com).ctime=1410940221
pbs.server().job(502.jupiter.example.com).Error_Path=jupiter.example.com:/home/TestUser/jobs/STD
    IN.e502
pbs.server().job(502.jupiter.example.com).Hold_Types=u
pbs.server().job(502.jupiter.example.com).Join_Path=n
pbs.server().job(502.jupiter.example.com).Keep_Files=n
pbs.server().job(502.jupiter.example.com).Mail_Points=a
pbs.server().job(502.jupiter.example.com).mtime=1410940221
pbs.server().job(502.jupiter.example.com).Output_Path=jupiter.example.com:/home/TestUser/jobs/ST
    DIN.o502
pbs.server().job(502.jupiter.example.com).Priority=7
pbs.server().job(502.jupiter.example.com).qtime=1410940223
pbs.server().job(502.jupiter.example.com).Rerunable=True
pbs.server().job(502.jupiter.example.com).Resource_List[file]=7gb
pbs.server().job(502.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().job(502.jupiter.example.com).Resource_List[nodect]=1
pbs.server().job(502.jupiter.example.com).Resource_List[place]=pack
pbs.server().job(502.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().job(502.jupiter.example.com).schedselect=1:ncpus=1
```

```
pbs.server().job(502.jupiter.example.com).substate=20
pbs.server().job(502.jupiter.example.com).Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash
    ,PBS_O_HOME=/home/TestUser,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LA
    NG=en_US.UTF-8,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/op
    enmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/g
    ames:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_QUEUE=
    workq,PBS_O_HOST=jupiter.example.com
pbs.server().job(502.jupiter.example.com).euser=TestUser
pbs.server().job(502.jupiter.example.com).egroup=users
pbs.server().job(502.jupiter.example.com).queue_rank=186
pbs.server().job(502.jupiter.example.com).queue_type=E
pbs.server().job(502.jupiter.example.com).Submit_arguments=<jsdl-hpcpa:Argument>-h</jsdl-hpcpa:A
    rgument>
pbs.server().job(502.jupiter.example.com).project=_pbs_project_default
pbs.server().job(506.jupiter.example.com).Job_Name=STDIN
pbs.server().job(506.jupiter.example.com).Job_Owner=TestUser@jupiter.example.com
pbs.server().job(506.jupiter.example.com).resources_used[cpupercent]=0
pbs.server().job(506.jupiter.example.com).resources_used[cput]=00:00:00
pbs.server().job(506.jupiter.example.com).resources_used[mem]=3880kb
pbs.server().job(506.jupiter.example.com).resources_used[ncpus]=1
pbs.server().job(506.jupiter.example.com).resources_used[vmem]=32192kb
pbs.server().job(506.jupiter.example.com).resources_used[walltime]=00:00:13
pbs.server().job(506.jupiter.example.com).job_state=R
pbs.server().job(506.jupiter.example.com).queue=workq
pbs.server().job(506.jupiter.example.com).server=jupiter.example.com
pbs.server().job(506.jupiter.example.com).Checkpoint=u
pbs.server().job(506.jupiter.example.com).ctime=1410941347
pbs.server().job(506.jupiter.example.com).Error_Path=jupiter.example.com:/home/TestUser/jobs/STD
    IN.e506
pbs.server().job(506.jupiter.example.com).exec_host=jupiter/0
pbs.server().job(506.jupiter.example.com).exec_vnode=(jupiter:ncpus=1)
pbs.server().job(506.jupiter.example.com).Hold_Types=n
pbs.server().job(506.jupiter.example.com).Join_Path=n
pbs.server().job(506.jupiter.example.com).Keep_Files=n
pbs.server().job(506.jupiter.example.com).Mail_Points=a
pbs.server().job(506.jupiter.example.com).mtime=1410941347
pbs.server().job(506.jupiter.example.com).Output_Path=jupiter.example.com:/home/TestUser/jobs/ST
    DIN.o506
pbs.server().job(506.jupiter.example.com).Priority=7
pbs.server().job(506.jupiter.example.com).qtime=1410941347
pbs.server().job(506.jupiter.example.com).Rerunable=True
pbs.server().job(506.jupiter.example.com).Resource_List[file]=7gb
pbs.server().job(506.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().job(506.jupiter.example.com).Resource_List[nodect]=1
pbs.server().job(506.jupiter.example.com).Resource_List[place]=pack
pbs.server().job(506.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().job(506.jupiter.example.com).schedselect=1:ncpus=1
```

```
pbs.server().job(506.jupiter.example.com).stime=1410941347
pbs.server().job(506.jupiter.example.com).session_id=11683
pbs.server().job(506.jupiter.example.com).jobdir=/home/TestUser
pbs.server().job(506.jupiter.example.com).substate=42
pbs.server().job(506.jupiter.example.com).Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash
    ,PBS_O_HOME=/home/TestUser,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LA
    NG=en_US.UTF-8,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/op
    enmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/g
    ames:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_QUEUE=
    workq,PBS_O_HOST=jupiter.example.com
pbs.server().job(506.jupiter.example.com).euser=TestUser
pbs.server().job(506.jupiter.example.com).egroup=users
pbs.server().job(506.jupiter.example.com).hashname=506.jupiter.example.com
pbs.server().job(506.jupiter.example.com).queue_rank=188
pbs.server().job(506.jupiter.example.com).queue_type=E
pbs.server().job(506.jupiter.example.com).comment=Job run at Wed Sep 17 at 04:09 on
    (jupiter:ncpus=1)
pbs.server().job(506.jupiter.example.com).etime=1410941347
pbs.server().job(506.jupiter.example.com).run_count=1
pbs.server().job(506.jupiter.example.com).project=_pbs_project_default
pbs.server().job(506.jupiter.example.com).run_version=1
pbs.server().job(507.jupiter.example.com).Job_Name=STDIN
pbs.server().job(507.jupiter.example.com).Job_Owner=TestUser@jupiter.example.com
pbs.server().job(507.jupiter.example.com).resources_used[cpupercent]=0
pbs.server().job(507.jupiter.example.com).resources_used[cput]=00:00:00
pbs.server().job(507.jupiter.example.com).resources_used[mem]=3892kb
pbs.server().job(507.jupiter.example.com).resources_used[ncpus]=1
pbs.server().job(507.jupiter.example.com).resources_used[vmem]=32192kb
pbs.server().job(507.jupiter.example.com).resources_used[walltime]=00:00:10
pbs.server().job(507.jupiter.example.com).job_state=R
pbs.server().job(507.jupiter.example.com).queue=workq
pbs.server().job(507.jupiter.example.com).server=jupiter.example.com
pbs.server().job(507.jupiter.example.com).Checkpoint=u
pbs.server().job(507.jupiter.example.com).ctime=1410941350
pbs.server().job(507.jupiter.example.com).Error_Path=jupiter.example.com:/home/TestUser/jobs/STD
    IN.e507
pbs.server().job(507.jupiter.example.com).exec_host=jupiter/1
pbs.server().job(507.jupiter.example.com).exec_vnode=(jupiter:ncpus=1)
pbs.server().job(507.jupiter.example.com).Hold_Types=n
pbs.server().job(507.jupiter.example.com).Join_Path=n
pbs.server().job(507.jupiter.example.com).Keep_Files=n
pbs.server().job(507.jupiter.example.com).Mail_Points=a
pbs.server().job(507.jupiter.example.com).mtime=1410941350
pbs.server().job(507.jupiter.example.com).Output_Path=jupiter.example.com:/home/TestUser/jobs/ST
    DIN.o507
pbs.server().job(507.jupiter.example.com).Priority=7
pbs.server().job(507.jupiter.example.com).qtime=1410941350
```

```
pbs.server().job(507.jupiter.example.com).Rerunable=True
pbs.server().job(507.jupiter.example.com).Resource_List[file]=7gb
pbs.server().job(507.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().job(507.jupiter.example.com).Resource_List[nodect]=1
pbs.server().job(507.jupiter.example.com).Resource_List[place]=pack
pbs.server().job(507.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().job(507.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().job(507.jupiter.example.com).stime=1410941350
pbs.server().job(507.jupiter.example.com).session_id=11716
pbs.server().job(507.jupiter.example.com).jobdir=/home/TestUser
pbs.server().job(507.jupiter.example.com).substate=42
pbs.server().job(507.jupiter.example.com).Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash
    ,PBS_O_HOME=/home/TestUser,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LA
    NG=en_US.UTF-8,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/op
    enmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/g
    ames:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_QUEUE=
    workq,PBS_O_HOST=jupiter.example.com
pbs.server().job(507.jupiter.example.com).euser=TestUser
pbs.server().job(507.jupiter.example.com).egroup=users
pbs.server().job(507.jupiter.example.com).hashname=507.jupiter.example.com
pbs.server().job(507.jupiter.example.com).queue_rank=189
pbs.server().job(507.jupiter.example.com).queue_type=E
pbs.server().job(507.jupiter.example.com).comment=Job run at Wed Sep 17 at 04:09 on
    (jupiter:ncpus=1)
pbs.server().job(507.jupiter.example.com).etime=1410941350
pbs.server().job(507.jupiter.example.com).run_count=1
pbs.server().job(507.jupiter.example.com).project=_pbs_project_default
pbs.server().job(507.jupiter.example.com).run_version=1
pbs.server().queue(workq).queue_type=Execution
pbs.server().queue(workq).total_jobs=4
pbs.server().queue(workq).state_count=Transit:0 Queued:0 Held:2 Waiting:0 Running:2 Exiting:0
    Begun:0
pbs.server().queue(workq).resources_assigned[mem]=0mb
pbs.server().queue(workq).resources_assigned[ncpus]=2
pbs.server().queue(workq).resources_assigned[nodect]=2
pbs.server().queue(workq).enabled=True
pbs.server().queue(workq).started=True
pbs.server().queue(R503).queue_type=Execution
pbs.server().queue(R503).total_jobs=0
pbs.server().queue(R503).state_count=Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0
    Begun:0
pbs.server().queue(R503).acl_user_enable=True
pbs.server().queue(R503).acl_users=TestUser@jupiter.example.com
pbs.server().queue(R503).resources_max[ncpus]=1
pbs.server().queue(R503).resources_max[walltime]=00:30:00
pbs.server().queue(R503).resources_available[ncpus]=1
pbs.server().queue(R503).resources_available[walltime]=00:30:00
```

```
pbs.server().queue(R503).enabled=True
pbs.server().queue(R503).started=False
pbs.server().queue(R504).queue_type=Execution
pbs.server().queue(R504).total_jobs=0
pbs.server().queue(R504).state_count=Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0
    Begun:0
pbs.server().queue(R504).acl_user_enable=True
pbs.server().queue(R504).acl_users=TestUser@jupiter.example.com
pbs.server().queue(R504).resources_max[ncpus]=1
pbs.server().queue(R504).resources_max[walltime]=00:30:00
pbs.server().queue(R504).resources_available[ncpus]=1
pbs.server().queue(R504).resources_available[walltime]=00:30:00
pbs.server().queue(R504).enabled=True
pbs.server().queue(R504).started=False
pbs.server().vnode(jupiter).Mom=jupiter.example.com
pbs.server().vnode(jupiter).Port=15002
pbs.server().vnode(jupiter).pbs_version=PBSPro_10.0
pbs.server().vnode(jupiter).ntype=0
pbs.server().vnode(jupiter).state=0
pbs.server().vnode(jupiter).pcpus=1
pbs.server().vnode(jupiter).jobs=506.jupiter.example.com/0, 507.jupiter.example.com/1
pbs.server().vnode(jupiter).resv=R504.jupiter.example.com, R503.jupiter.example.com
pbs.server().vnode(jupiter).resources_available[arch]=linux
pbs.server().vnode(jupiter).resources_available[file]=7gb
pbs.server().vnode(jupiter).resources_available[host]=jupiter
pbs.server().vnode(jupiter).resources_available[mem]=8gb
pbs.server().vnode(jupiter).resources_available[ncpus]=8
pbs.server().vnode(jupiter).resources_available[vnode]=jupiter
pbs.server().vnode(jupiter).resources_assigned[accelerator_memory]=0kb
pbs.server().vnode(jupiter).resources_assigned[mem]=0kb
pbs.server().vnode(jupiter).resources_assigned[naccelerators]=0
pbs.server().vnode(jupiter).resources_assigned[ncpus]=2
pbs.server().vnode(jupiter).resources_assigned[vmem]=0kb
pbs.server().vnode(jupiter).resv_enable=True
pbs.server().vnode(jupiter).sharing=1
pbs.server().vnode(mars).Mom=mars.example.com
pbs.server().vnode(mars).Port=15002
pbs.server().vnode(mars).pbs_version=PBSPro_10.0
pbs.server().vnode(mars).ntype=0
pbs.server().vnode(mars).state=0
pbs.server().vnode(mars).pcpus=1
pbs.server().vnode(mars).resources_available[arch]=linux
pbs.server().vnode(mars).resources_available[file]=7gb
pbs.server().vnode(mars).resources_available[host]=mars
pbs.server().vnode(mars).resources_available[mem]=8gb
pbs.server().vnode(mars).resources_available[ncpus]=8
```

```
pbs.server().vnode(mars).resources_available[vnode]=mars
pbs.server().vnode(mars).resources_assigned[accelerator_memory]=0kb
pbs.server().vnode(mars).resources_assigned[mem]=0kb
pbs.server().vnode(mars).resources_assigned[naccelerators]=0
pbs.server().vnode(mars).resources_assigned[ncpus]=0
pbs.server().vnode(mars).resources_assigned[vmem]=0kb
pbs.server().vnode(mars).resv_enable=True
pbs.server().vnode(mars).sharing=1
pbs.server().resv(R503.jupiter.example.com).Reserve_Name=NULL
pbs.server().resv(R503.jupiter.example.com).Reserve_Owner=TestUser@jupiter.example.com
pbs.server().resv(R503.jupiter.example.com).reserve_type=2
pbs.server().resv(R503.jupiter.example.com).reserve_state=2
pbs.server().resv(R503.jupiter.example.com).reserve_substate=2
pbs.server().resv(R503.jupiter.example.com).reserve_start=1410955200
pbs.server().resv(R503.jupiter.example.com).reserve_end=1410957000
pbs.server().resv(R503.jupiter.example.com).reserve_duration=1800
pbs.server().resv(R503.jupiter.example.com).queue=R503
pbs.server().resv(R503.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().resv(R503.jupiter.example.com).Resource_List[walltime]=00:30:00
pbs.server().resv(R503.jupiter.example.com).Resource_List[nodect]=1
pbs.server().resv(R503.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().resv(R503.jupiter.example.com).Resource_List[place]=free
pbs.server().resv(R503.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().resv(R503.jupiter.example.com).resv_nodes=(jupiter:ncpus=1)
pbs.server().resv(R503.jupiter.example.com).Authorized_Users=TestUser@jupiter.example.com
pbs.server().resv(R503.jupiter.example.com).server=jupiter.example.com
pbs.server().resv(R503.jupiter.example.com).ctime=1410940237
pbs.server().resv(R503.jupiter.example.com).mtime=1410940237
pbs.server().resv(R503.jupiter.example.com).Variable_List=PBS_O_LOGNAME=TestUser,PBS_O_HOST=jupi
    ter.example.com,PBS_O_MAIL=/var/spool/mail/TestUser
pbs.server().resv(R503.jupiter.example.com).euser=TestUser
pbs.server().resv(R503.jupiter.example.com).egroup=users
pbs.server().resv(R504.jupiter.example.com).Reserve_Name=NULL
pbs.server().resv(R504.jupiter.example.com).Reserve_Owner=TestUser@jupiter.example.com
pbs.server().resv(R504.jupiter.example.com).reserve_type=2
pbs.server().resv(R504.jupiter.example.com).reserve_state=2
pbs.server().resv(R504.jupiter.example.com).reserve_substate=2
pbs.server().resv(R504.jupiter.example.com).reserve_start=1410958800
pbs.server().resv(R504.jupiter.example.com).reserve_end=1410960600
pbs.server().resv(R504.jupiter.example.com).reserve_duration=1800
pbs.server().resv(R504.jupiter.example.com).queue=R504
pbs.server().resv(R504.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().resv(R504.jupiter.example.com).Resource_List[walltime]=00:30:00
pbs.server().resv(R504.jupiter.example.com).Resource_List[nodect]=1
pbs.server().resv(R504.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().resv(R504.jupiter.example.com).Resource_List[place]=free
```

```
pbs.server().resv(R504.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().resv(R504.jupiter.example.com).resv_nodes=(jupiter:ncpus=1)
pbs.server().resv(R504.jupiter.example.com).Authorized_Users=TestUser@jupiter.example.com
pbs.server().resv(R504.jupiter.example.com).server=jupiter.example.com
pbs.server().resv(R504.jupiter.example.com).ctime=1410940250
pbs.server().resv(R504.jupiter.example.com).mtime=1410940250
pbs.server().resv(R504.jupiter.example.com).Variable_List=PBS_O_LOGNAME=TestUser,PBS_O_HOST=jupi
    ter.example.com,PBS_O_MAIL=/var/spool/mail/TestUser
pbs.server().resv(R504.jupiter.example.com).euser=TestUser
pbs.server().resv(R504.jupiter.example.com).egroup=users
```

List the exechost_periodic hook execution record file:

```
jupiter:/var/spool/PBS/mom_priv/hooks/tmp # cat hook_exechost_periodic_period_11753.out
pbs.event().accept=True
pbs.event().reject=False
pbs.event().vnode_list["jupiter"].resources_available[file,size]=7gb
pbs.event().job_list["506.jupiter.example.com"].Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bi
    n/bash,PBS_O_HOME=/home/TestUser,PBS_O_HOST=jupiter.example.com,PBS_O_LOGNAME=TestUser,PBS_O
    _WORKDIR=/home/TestUser/jobs,PBS_O_LANG=en_US.UTF-8,PBS_O_QUEUE=workq,PBS_O_MAIL=/var/spool/
    mail/TestUser,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/ope
    nmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/ga
    mes:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin
pbs.event().job_list["506.jupiter.example.com"]._delete=False
pbs.event().job_list["506.jupiter.example.com"]._rerun=False
pbs.event().job_list["507.jupiter.example.com"].Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bi
    n/bash,PBS_O_HOME=/home/TestUser,PBS_O_HOST=jupiter.example.com,PBS_O_LOGNAME=TestUser,PBS_O
    _WORKDIR=/home/TestUser/jobs,PBS_O_LANG=en_US.UTF-8,PBS_O_QUEUE=workq,PBS_O_MAIL=/var/spool/
    mail/TestUser,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/ope
    nmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/ga
    mes:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin
pbs.event().job_list["507.jupiter.example.com"]._delete=False
pbs.event().job_list["507.jupiter.example.com"]._rerun=False
```

Attributes of the execjob_begin hook:

```
Hook begin
    type = site
    enabled = true
    event = execjob_begin
    user = pbsadmin
    alarm = 30
    order = 1
    debug = true
    fail_action = none
```

Contents of the execjob_begin hook:

```
import pbs
e=pbs.event()

e.job.Priority=7
e.job.Variable_List["Monsieur"] = "Shlomi"

s=pbs.server()
for j in s.jobs():
    pbs.logmsg(pbs.LOG_DEBUG, "got j %s" % (j.id,))

for q in s.queues():
    pbs.logmsg(pbs.LOG_DEBUG, "got q %s" % (q.name,))

for v in s.vnodes():
    pbs.logmsg(pbs.LOG_DEBUG, "got vnode %s" % (v.name,))

for r in s.resvs():
    pbs.logmsg(pbs.LOG_DEBUG, "got resv %s" % (r.resvid))
```

We submit a job:

```
% qsub job.scr
```

The resulting execjob_begin debug files are here:

```
jupiter:/var/spool/PBS/mom_priv/hooks/tmp # ls -ltr
-rw-r--r-- 1 root root  2263 Sep 17 04:15 hook_execjob_begin_begin_11883.in
-rw-r--r-- 1 root root   585 Sep 17 04:15 hook_execjob_begin_begin_11883.out
-rw-r--r-- 1 root root 15327 Sep 17 04:15 hook_execjob_begin_begin_11883.data
```

List the execjob_begin event file:

```
jupiter:/var/spool/PBS/mom_priv/hooks/tmp # cat hook_execjob_begin_begin_11883.in
pbs.event().job.id=509.jupiter.example.com
pbs.event().job.Job_Name=job.scr
pbs.event().job.Job_Owner=TestUser@jupiter.example.com
pbs.event().job.queue=workq
pbs.event().job.server=jupiter.example.com
pbs.event().job.Checkpoint=u
pbs.event().job.Error_Path=jupiter.example.com:/home/TestUser/jobs/job.scr.e509
pbs.event().job.exec_host2=jupiter.example.com:15002/0
pbs.event().job.exec_vnode=(jupiter:ncpus=1)
pbs.event().job.Join_Path=n
pbs.event().job.Keep_Files=n
pbs.event().job.mtime=1410941704
pbs.event().job.Output_Path=jupiter.example.com:/home/TestUser/jobs/job.scr.o509
pbs.event().job.Resource_List[file]=7gb
pbs.event().job.Resource_List[ncpus]=1
pbs.event().job.Resource_List[place]=pack
pbs.event().job.schedselect=1:ncpus=1
pbs.event().job.Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash,PBS_O_HOME=/home/TestUser
    ,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LANG=en_US.UTF-8,PBS_O_PATH=
    /usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/openmpi/bin:/home/TestUser/b
    in:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/games:/opt/pbs/bin:/opt/pbs
    /bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_QUEUE=workq,PBS_O_HOST=jupiter.e
    xample.com
pbs.event().job.euser=TestUser
pbs.event().job.egroup=users
pbs.event().job.hashname=509.jupiter.example.com
pbs.event().job.run_count=1
pbs.event().job.job_kill_delay=10
pbs.event().job.project=_pbs_project_default
pbs.event().job.run_version=1
pbs.event().job._msmom=True
pbs.event().job._stdout_file=
pbs.event().job._stderr_file=
pbs.event().vnode_list["jupiter"].resources_assigned[ncpus]=1
pbs.event().vnode_list["jupiter"].resources_assigned[mem]=0kb
pbs.event().vnode_list["jupiter"].pcpus=1
pbs.event().vnode_list["jupiter"].resources_available[ncpus]=1
pbs.event().vnode_list["jupiter"].resources_available[mem]=757388kb
pbs.event().vnode_list["jupiter"].resources_available[arch]=linux
pbs.event().vnode_list["jupiter"].pbs_version=PBSPro_10.0
pbs.get_local_nodename()=jupiter
pbs.event().type=execjob_begin
pbs.event().hook_name=begin
pbs.event().hook_type=site
pbs.event().requestor=pbs_mom
```

```
pbs.event().requestor_host=jupiter.example.com
pbs.event().user=pbsadmin
pbs.event().alarm=30
```

List the execjob_begin site data file:

```
jupiter:/var/spool/PBS/mom_priv/hooks/tmp # cat  hook_execjob_begin_begin_11883.data
pbs.server().server_state=Active
pbs.server().server_host=jupiter.example.com
pbs.server().scheduling=True
pbs.server().total_jobs=3
pbs.server().state_count=Transit:0 Queued:0 Held:2 Waiting:0 Running:1 Exiting:0 Begun:0
pbs.server().managers=TestUser@*
pbs.server().default_queue=workq
pbs.server().log_events=511
pbs.server().mail_from=adm
pbs.server().query_other_jobs=True
pbs.server().resources_default[ncpus]=1
pbs.server().default_chunk[ncpus]=1
pbs.server().resources_assigned[mem]=0mb
pbs.server().resources_assigned[ncpus]=1
pbs.server().resources_assigned[nodect]=1
pbs.server().scheduler_iteration=600
pbs.server().flatuid=True
pbs.server().resv_enable=True
pbs.server().node_fail_requeue=310
pbs.server().max_array_size=10000
pbs.server().pbs_license_min=1
pbs.server().pbs_license_max=2147483647
pbs.server().pbs_license_linger_time=3600
pbs.server().license_count=Avail_Global:0 Avail_Local:31 Used:1 High_Use:2
pbs.server().pbs_version=PBSPro_10.0
pbs.server().eligible_time_enable=False
pbs.server().max_concurrent_provision=5
pbs.server().job(501.jupiter.example.com).Job_Name=STDIN
pbs.server().job(501.jupiter.example.com).Job_Owner=TestUser@jupiter.example.com
pbs.server().job(501.jupiter.example.com).job_state=H
pbs.server().job(501.jupiter.example.com).queue=workq
pbs.server().job(501.jupiter.example.com).server=jupiter.example.com
pbs.server().job(501.jupiter.example.com).Checkpoint=u
pbs.server().job(501.jupiter.example.com).ctime=1410940219
pbs.server().job(501.jupiter.example.com).Error_Path=jupiter.example.com:/home/TestUser/jobs/STD
    IN.e501
pbs.server().job(501.jupiter.example.com).Hold_Types=u
pbs.server().job(501.jupiter.example.com).Join_Path=n
pbs.server().job(501.jupiter.example.com).Keep_Files=n
pbs.server().job(501.jupiter.example.com).Mail_Points=a
pbs.server().job(501.jupiter.example.com).mtime=1410940219
pbs.server().job(501.jupiter.example.com).Output_Path=jupiter.example.com:/home/TestUser/jobs/ST
    DIN.o501
pbs.server().job(501.jupiter.example.com).Priority=7
```

```
pbs.server().job(501.jupiter.example.com).qtime=1410940219
pbs.server().job(501.jupiter.example.com).Rerunable=True
pbs.server().job(501.jupiter.example.com).Resource_List[file]=7gb
pbs.server().job(501.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().job(501.jupiter.example.com).Resource_List[nodect]=1
pbs.server().job(501.jupiter.example.com).Resource_List[place]=pack
pbs.server().job(501.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().job(501.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().job(501.jupiter.example.com).substate=20
pbs.server().job(501.jupiter.example.com).Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash
    ,PBS_O_HOME=/home/TestUser,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LA
    NG=en_US.UTF-8,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/op
    enmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/g
    ames:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_QUEUE=
    workq,PBS_O_HOST=jupiter.example.com
pbs.server().job(501.jupiter.example.com).euser=TestUser
pbs.server().job(501.jupiter.example.com).egroup=users
pbs.server().job(501.jupiter.example.com).queue_rank=185
pbs.server().job(501.jupiter.example.com).queue_type=E
pbs.server().job(501.jupiter.example.com).Submit_arguments=<jsdl-hpcpa:Argument>-h</jsdl-hpcpa:A
    rgument>
pbs.server().job(501.jupiter.example.com).project=_pbs_project_default
pbs.server().job(502.jupiter.example.com).Job_Name=STDIN
pbs.server().job(502.jupiter.example.com).Job_Owner=TestUser@jupiter.example.com
pbs.server().job(502.jupiter.example.com).job_state=H
pbs.server().job(502.jupiter.example.com).queue=workq
pbs.server().job(502.jupiter.example.com).server=jupiter.example.com
pbs.server().job(502.jupiter.example.com).Checkpoint=u
pbs.server().job(502.jupiter.example.com).ctime=1410940221
pbs.server().job(502.jupiter.example.com).Error_Path=jupiter.example.com:/home/TestUser/jobs/STD
    IN.e502
pbs.server().job(502.jupiter.example.com).Hold_Types=u
pbs.server().job(502.jupiter.example.com).Join_Path=n
pbs.server().job(502.jupiter.example.com).Keep_Files=n
pbs.server().job(502.jupiter.example.com).Mail_Points=a
pbs.server().job(502.jupiter.example.com).mtime=1410940221
pbs.server().job(502.jupiter.example.com).Output_Path=jupiter.example.com:/home/TestUser/jobs/ST
    DIN.o502
pbs.server().job(502.jupiter.example.com).Priority=7
pbs.server().job(502.jupiter.example.com).qtime=1410940223
pbs.server().job(502.jupiter.example.com).Rerunable=True
pbs.server().job(502.jupiter.example.com).Resource_List[file]=7gb
pbs.server().job(502.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().job(502.jupiter.example.com).Resource_List[nodect]=1
pbs.server().job(502.jupiter.example.com).Resource_List[place]=pack
pbs.server().job(502.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().job(502.jupiter.example.com).schedselect=1:ncpus=1
```

```
pbs.server().job(502.jupiter.example.com).substate=20
pbs.server().job(502.jupiter.example.com).Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash
    ,PBS_O_HOME=/home/TestUser,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LA
    NG=en_US.UTF-8,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/op
    enmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/g
    ames:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_QUEUE=
    workq,PBS_O_HOST=jupiter.example.com
pbs.server().job(502.jupiter.example.com).euser=TestUser
pbs.server().job(502.jupiter.example.com).egroup=users
pbs.server().job(502.jupiter.example.com).queue_rank=186
pbs.server().job(502.jupiter.example.com).queue_type=E
pbs.server().job(502.jupiter.example.com).Submit_arguments=<jsdl-hpcpa:Argument>-h</jsdl-hpcpa:A
    rgument>
pbs.server().job(502.jupiter.example.com).project=_pbs_project_default
pbs.server().job(509.jupiter.example.com).Job_Name=job.scr
pbs.server().job(509.jupiter.example.com).Job_Owner=TestUser@jupiter.example.com
pbs.server().job(509.jupiter.example.com).job_state=R
pbs.server().job(509.jupiter.example.com).queue=workq
pbs.server().job(509.jupiter.example.com).server=jupiter.example.com
pbs.server().job(509.jupiter.example.com).Checkpoint=u
pbs.server().job(509.jupiter.example.com).ctime=1410941704
pbs.server().job(509.jupiter.example.com).Error_Path=jupiter.example.com:/home/TestUser/jobs/job
    .scr.e509
pbs.server().job(509.jupiter.example.com).exec_host=jupiter/0
pbs.server().job(509.jupiter.example.com).exec_vnode=(jupiter:ncpus=1)
pbs.server().job(509.jupiter.example.com).Hold_Types=n
pbs.server().job(509.jupiter.example.com).Join_Path=n
pbs.server().job(509.jupiter.example.com).Keep_Files=n
pbs.server().job(509.jupiter.example.com).Mail_Points=a
pbs.server().job(509.jupiter.example.com).mtime=1410941704
pbs.server().job(509.jupiter.example.com).Output_Path=jupiter.example.com:/home/TestUser/jobs/jo
    b.scr.o509
pbs.server().job(509.jupiter.example.com).Priority=7
pbs.server().job(509.jupiter.example.com).qtime=1410941704
pbs.server().job(509.jupiter.example.com).Rerunable=True
pbs.server().job(509.jupiter.example.com).Resource_List[file]=7gb
pbs.server().job(509.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().job(509.jupiter.example.com).Resource_List[nodect]=1
pbs.server().job(509.jupiter.example.com).Resource_List[place]=pack
pbs.server().job(509.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().job(509.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().job(509.jupiter.example.com).substate=41
pbs.server().job(509.jupiter.example.com).Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash
    ,PBS_O_HOME=/home/TestUser,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LA
    NG=en_US.UTF-8,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/op
    enmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/g
    ames:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_QUEUE=
    workq,PBS_O_HOST=jupiter.example.com
```

```
pbs.server().job(509.jupiter.example.com).euser=TestUser
pbs.server().job(509.jupiter.example.com).egroup=users
pbs.server().job(509.jupiter.example.com).hashname=509.jupiter.example.com
pbs.server().job(509.jupiter.example.com).queue_rank=190
pbs.server().job(509.jupiter.example.com).queue_type=E
pbs.server().job(509.jupiter.example.com).comment=Job run at Wed Sep 17 at 04:15 on
    (jupiter:ncpus=1)
pbs.server().job(509.jupiter.example.com).etime=1410941704
pbs.server().job(509.jupiter.example.com).run_count=1
pbs.server().job(509.jupiter.example.com).Submit_arguments=<jsdl-hpcpa:Argument>job.scr</jsdl-hp
    cpa:Argument>
pbs.server().job(509.jupiter.example.com).project=_pbs_project_default
pbs.server().job(509.jupiter.example.com).run_version=1
pbs.server().queue(workq).queue_type=Execution
pbs.server().queue(workq).total_jobs=3
pbs.server().queue(workq).state_count=Transit:0 Queued:0 Held:2 Waiting:0 Running:1 Exiting:0
    Begun:0
pbs.server().queue(workq).resources_assigned[mem]=0mb
pbs.server().queue(workq).resources_assigned[ncpus]=1
pbs.server().queue(workq).resources_assigned[nodect]=1
pbs.server().queue(workq).enabled=True
pbs.server().queue(workq).started=True
pbs.server().queue(R503).queue_type=Execution
pbs.server().queue(R503).total_jobs=0
pbs.server().queue(R503).state_count=Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0
    Begun:0
pbs.server().queue(R503).acl_user_enable=True
pbs.server().queue(R503).acl_users=TestUser@jupiter.example.com
pbs.server().queue(R503).resources_max[ncpus]=1
pbs.server().queue(R503).resources_max[walltime]=00:30:00
pbs.server().queue(R503).resources_available[ncpus]=1
pbs.server().queue(R503).resources_available[walltime]=00:30:00
pbs.server().queue(R503).enabled=True
pbs.server().queue(R503).started=False
pbs.server().queue(R504).queue_type=Execution
pbs.server().queue(R504).total_jobs=0
pbs.server().queue(R504).state_count=Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0
    Begun:0
pbs.server().queue(R504).acl_user_enable=True
pbs.server().queue(R504).acl_users=TestUser@jupiter.example.com
pbs.server().queue(R504).resources_max[ncpus]=1
pbs.server().queue(R504).resources_max[walltime]=00:30:00
pbs.server().queue(R504).resources_available[ncpus]=1
pbs.server().queue(R504).resources_available[walltime]=00:30:00
pbs.server().queue(R504).enabled=True
pbs.server().queue(R504).started=False
pbs.server().vnode(jupiter).Mom=jupiter.example.com
```

```
pbs.server().vnode(jupiter).Port=15002
pbs.server().vnode(jupiter).pbs_version=PBSPro_10.0
pbs.server().vnode(jupiter).ntype=0
pbs.server().vnode(jupiter).state=0
pbs.server().vnode(jupiter).pcpus=1
pbs.server().vnode(jupiter).jobs=509.jupiter.example.com/0
pbs.server().vnode(jupiter).resv=R504.jupiter.example.com, R503.jupiter.example.com
pbs.server().vnode(jupiter).resources_available[arch]=linux
pbs.server().vnode(jupiter).resources_available[file]=7gb
pbs.server().vnode(jupiter).resources_available[host]=jupiter
pbs.server().vnode(jupiter).resources_available[mem]=8gb
pbs.server().vnode(jupiter).resources_available[ncpus]=8
pbs.server().vnode(jupiter).resources_available[vnode]=jupiter
pbs.server().vnode(jupiter).resources_assigned[accelerator_memory]=0kb
pbs.server().vnode(jupiter).resources_assigned[mem]=0kb
pbs.server().vnode(jupiter).resources_assigned[naccelerators]=0
pbs.server().vnode(jupiter).resources_assigned[ncpus]=1
pbs.server().vnode(jupiter).resources_assigned[vmem]=0kb
pbs.server().vnode(jupiter).resv_enable=True
pbs.server().vnode(jupiter).sharing=1
pbs.server().vnode(mars).Mom=mars.example.com
pbs.server().vnode(mars).Port=15002
pbs.server().vnode(mars).pbs_version=PBSPro_10.0
pbs.server().vnode(mars).ntype=0
pbs.server().vnode(mars).state=0
pbs.server().vnode(mars).pcpus=1
pbs.server().vnode(mars).resources_available[arch]=linux
pbs.server().vnode(mars).resources_available[file]=7gb
pbs.server().vnode(mars).resources_available[host]=mars
pbs.server().vnode(mars).resources_available[mem]=8gb
pbs.server().vnode(mars).resources_available[ncpus]=8
pbs.server().vnode(mars).resources_available[vnode]=mars
pbs.server().vnode(mars).resources_assigned[accelerator_memory]=0kb
pbs.server().vnode(mars).resources_assigned[mem]=0kb
pbs.server().vnode(mars).resources_assigned[naccelerators]=0
pbs.server().vnode(mars).resources_assigned[ncpus]=0
pbs.server().vnode(mars).resources_assigned[vmem]=0kb
pbs.server().vnode(mars).resv_enable=True
pbs.server().vnode(mars).sharing=1
pbs.server().resv(R503.jupiter.example.com).Reserve_Name=NULL
pbs.server().resv(R503.jupiter.example.com).Reserve_Owner=TestUser@jupiter.example.com
pbs.server().resv(R503.jupiter.example.com).reserve_type=2
pbs.server().resv(R503.jupiter.example.com).reserve_state=2
pbs.server().resv(R503.jupiter.example.com).reserve_substate=2
pbs.server().resv(R503.jupiter.example.com).reserve_start=1410955200
pbs.server().resv(R503.jupiter.example.com).reserve_end=1410957000
```

```
pbs.server().resv(R503.jupiter.example.com).reserve_duration=1800
pbs.server().resv(R503.jupiter.example.com).queue=R503
pbs.server().resv(R503.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().resv(R503.jupiter.example.com).Resource_List[walltime]=00:30:00
pbs.server().resv(R503.jupiter.example.com).Resource_List[nodect]=1
pbs.server().resv(R503.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().resv(R503.jupiter.example.com).Resource_List[place]=free
pbs.server().resv(R503.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().resv(R503.jupiter.example.com).resv_nodes=(jupiter:ncpus=1)
pbs.server().resv(R503.jupiter.example.com).Authorized_Users=TestUser@jupiter.example.com
pbs.server().resv(R503.jupiter.example.com).server=jupiter.example.com
pbs.server().resv(R503.jupiter.example.com).ctime=1410940237
pbs.server().resv(R503.jupiter.example.com).mtime=1410940237
pbs.server().resv(R503.jupiter.example.com).Variable_List=PBS_O_LOGNAME=TestUser,PBS_O_HOST=jupi
    ter.example.com,PBS_O_MAIL=/var/spool/mail/TestUser
pbs.server().resv(R503.jupiter.example.com).euser=TestUser
pbs.server().resv(R503.jupiter.example.com).egroup=users
pbs.server().resv(R504.jupiter.example.com).Reserve_Name=NULL
pbs.server().resv(R504.jupiter.example.com).Reserve_Owner=TestUser@jupiter.example.com
pbs.server().resv(R504.jupiter.example.com).reserve_type=2
pbs.server().resv(R504.jupiter.example.com).reserve_state=2
pbs.server().resv(R504.jupiter.example.com).reserve_substate=2
pbs.server().resv(R504.jupiter.example.com).reserve_start=1410958800
pbs.server().resv(R504.jupiter.example.com).reserve_end=1410960600
pbs.server().resv(R504.jupiter.example.com).reserve_duration=1800
pbs.server().resv(R504.jupiter.example.com).queue=R504
pbs.server().resv(R504.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().resv(R504.jupiter.example.com).Resource_List[walltime]=00:30:00
pbs.server().resv(R504.jupiter.example.com).Resource_List[nodect]=1
pbs.server().resv(R504.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().resv(R504.jupiter.example.com).Resource_List[place]=free
pbs.server().resv(R504.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().resv(R504.jupiter.example.com).resv_nodes=(jupiter:ncpus=1)
pbs.server().resv(R504.jupiter.example.com).Authorized_Users=TestUser@jupiter.example.com
pbs.server().resv(R504.jupiter.example.com).server=jupiter.example.com
pbs.server().resv(R504.jupiter.example.com).ctime=1410940250
pbs.server().resv(R504.jupiter.example.com).mtime=1410940250
pbs.server().resv(R504.jupiter.example.com).Variable_List=PBS_O_LOGNAME=TestUser,PBS_O_HOST=jupi
    ter.example.com,PBS_O_MAIL=/var/spool/mail/TestUser
pbs.server().resv(R504.jupiter.example.com).euser=TestUser
pbs.server().resv(R504.jupiter.example.com).egroup=users
```

List the execjob_begin hook execution record file:

```
jupiter:/var/spool/PBS/mom_priv/hooks/tmp # cat hook_execjob_begin_begin_11883.out
pbs.event().accept=True
pbs.event().reject=False
pbs.event().job.Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash,Monsieur=Shlomi,PBS_O_HOM
    E=/home/TestUser,PBS_O_HOST=jupiter.example.com,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/T
    estUser/jobs,PBS_O_LANG=en_US.UTF-8,PBS_O_QUEUE=workq,PBS_O_MAIL=/var/spool/mail/TestUser,PB
    S_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/openmpi/bin:/home/T
    estUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/games:/opt/pbs/bin
    :/opt/pbs/bin:/opt/pbs/bin
pbs.event().job.Priority=7
```

Attributes of the execjob_launch hook:

```
Hook launch
    type = site
    enabled = true
    event = execjob_launch
    user = pbsadmin
    alarm = 30
    order = 1
    debug = true
    fail_action = none
```

Contents of the execjob_launch hook:

```
import pbs
e=pbs.event()

e.progname = "/bin/sleep"
e.argv[1] = "30"

s=pbs.server()
for j in s.jobs():
    pbs.logmsg(pbs.LOG_DEBUG, "got j %s" % (j.id,))

for q in s.queues():
    pbs.logmsg(pbs.LOG_DEBUG, "got q %s" % (q.name,))

for v in s.vnodes():
    pbs.logmsg(pbs.LOG_DEBUG, "got vnode %s" % (v.name,))

for r in s.resvs():
    pbs.logmsg(pbs.LOG_DEBUG, "got resv %s" % (r.resvid))
```

Submit a job:

```
% qsub job.scr
```

The execjob_launch hook writes the *.in, *.data, and *.out files in /var/spool/PBS/spool:

```
jupiter:/var/spool/PBS/spool # ls -ltr /var/spool/PBS/spool
-rw------- 1 TestUser users  3489 Sep 17 04:24 hook_execjob_launch_launch_12135.in
-rw------- 1 TestUser users  1045 Sep 17 04:24 hook_execjob_launch_launch_12135.out
-rw------- 1 TestUser users 15906 Sep 17 04:24 hook_execjob_launch_launch_12135.data
```

List the execjob_launch hook event file:

```
cat hook_execjob_launch_launch_12135.in
pbs.event().progname=/bin/bash
pbs.event().argv[0]=-bash
pbs.event().env=TZ=US/Eastern,PATH=/bin:/usr/bin,PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash,Monsieur=Shlomi,PBS_O_HOME=/home/TestUser,PBS_O_HOST=jupiter.example.com,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LANG=en_US.UTF-8,PBS_O_QUEUE=workq,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/openmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/games:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,HOME=/home/TestUser,LOGNAME=TestUser,PBS_JOBNAME=job.scr,PBS_JOBID=511.jupiter.example.com,PBS_QUEUE=workq,SHELL=/bin/bash,USER=TestUser,PBS_JOBCOOKIE=00000000434AB4BA000000000BDC62D3,PBS_NODENUM=0,PBS_TASKNUM=1,PBS_MOMPORT=15003,OMP_NUM_THREADS=1,NCPUS=1,PBS_NODEFILE=/var/spool/PBS/aux/511.jupiter.example.com,PBS_TMPDIR=/var/tmp/pbs.511.jupiter.example.com,PBS_JOBDIR=/home/TestUser,PBS_ENVIRONMENT=PBS_BATCH,ENVIRONMENT=BATCH
pbs.event().job.id=511.jupiter.example.com
pbs.event().job.Job_Name=job.scr
pbs.event().job.Job_Owner=TestUser@jupiter.example.com
pbs.event().job.job_state=T
pbs.event().job.queue=workq
pbs.event().job.server=jupiter.example.com
pbs.event().job.Checkpoint=u
pbs.event().job.Error_Path=jupiter.example.com:/home/TestUser/jobs/job.scr.e511
pbs.event().job.exec_host2=jupiter.example.com:15002/0
pbs.event().job.exec_vnode=(jupiter:ncpus=1)
pbs.event().job.Join_Path=n
pbs.event().job.Keep_Files=n
pbs.event().job.mtime=1410942248
pbs.event().job.Output_Path=jupiter.example.com:/home/TestUser/jobs/job.scr.o511
pbs.event().job.Priority=7
pbs.event().job.Resource_List[file]=7gb
pbs.event().job.Resource_List[ncpus]=1
pbs.event().job.Resource_List[place]=pack
pbs.event().job.schedselect=1:ncpus=1
pbs.event().job.substate=0
pbs.event().job.Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash,Monsieur=Shlomi,PBS_O_HOME=/home/TestUser,PBS_O_HOST=jupiter.example.com,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LANG=en_US.UTF-8,PBS_O_QUEUE=workq,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/openmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/games:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin
pbs.event().job.euser=TestUser
pbs.event().job.egroup=users
pbs.event().job.hashname=511.jupiter.example.com
pbs.event().job.cookie=00000000434AB4BA000000000BDC62D3
pbs.event().job.run_count=1
pbs.event().job.job_kill_delay=10
pbs.event().job.project=_pbs_project_default
```

```
pbs.event().job.run_version=1
pbs.event().job._msmom=True
pbs.event().job._stdout_file=/var/spool/PBS/spool/511.jupiter.example.com.OU
pbs.event().job._stderr_file=/var/spool/PBS/spool/511.jupiter.example.com.ER
pbs.event().vnode_list["jupiter"].resources_assigned[ncpus]=1
pbs.event().vnode_list["jupiter"].resources_assigned[mem]=0kb
pbs.event().vnode_list["jupiter"].pcpus=1
pbs.event().vnode_list["jupiter"].resources_available[ncpus]=1
pbs.event().vnode_list["jupiter"].resources_available[mem]=757388kb
pbs.event().vnode_list["jupiter"].resources_available[arch]=linux
pbs.event().vnode_list["jupiter"].pbs_version=PBSPro_10.0
pbs.get_local_nodename()=jupiter
pbs.event().type=execjob_launch
pbs.event().hook_name=launch
pbs.event().hook_type=site
pbs.event().requestor=pbs_mom
pbs.event().requestor_host=jupiter.example.com
pbs.event().user=pbsadmin
pbs.event().alarm=30
```

List the execjob_launch hook site data file:

```
jupiter:/var/spool/PBS/spool # cat hook_execjob_launch_launch_12135.data
pbs.server().server_state=Active
pbs.server().server_host=jupiter.example.com
pbs.server().scheduling=True
pbs.server().total_jobs=3
pbs.server().state_count=Transit:0 Queued:0 Held:2 Waiting:0 Running:1 Exiting:0 Begun:0
pbs.server().managers=TestUser@*
pbs.server().default_queue=workq
pbs.server().log_events=511
pbs.server().mail_from=adm
pbs.server().query_other_jobs=True
pbs.server().resources_default[ncpus]=1
pbs.server().default_chunk[ncpus]=1
pbs.server().resources_assigned[mem]=0mb
pbs.server().resources_assigned[ncpus]=1
pbs.server().resources_assigned[nodect]=1
pbs.server().scheduler_iteration=600
pbs.server().flatuid=True
pbs.server().resv_enable=True
pbs.server().node_fail_requeue=310
pbs.server().max_array_size=10000
pbs.server().pbs_license_min=1
pbs.server().pbs_license_max=2147483647
pbs.server().pbs_license_linger_time=3600
pbs.server().license_count=Avail_Global:0 Avail_Local:31 Used:1 High_Use:2
pbs.server().pbs_version=PBSPro_10.0
pbs.server().eligible_time_enable=False
pbs.server().max_concurrent_provision=5
pbs.server().job(501.jupiter.example.com).Job_Name=STDIN
pbs.server().job(501.jupiter.example.com).Job_Owner=TestUser@jupiter.example.com
pbs.server().job(501.jupiter.example.com).job_state=H
pbs.server().job(501.jupiter.example.com).queue=workq
pbs.server().job(501.jupiter.example.com).server=jupiter.example.com
pbs.server().job(501.jupiter.example.com).Checkpoint=u
pbs.server().job(501.jupiter.example.com).ctime=1410940219
pbs.server().job(501.jupiter.example.com).Error_Path=jupiter.example.com:/home/TestUser/jobs/STD
    IN.e501
pbs.server().job(501.jupiter.example.com).Hold_Types=u
pbs.server().job(501.jupiter.example.com).Join_Path=n
pbs.server().job(501.jupiter.example.com).Keep_Files=n
pbs.server().job(501.jupiter.example.com).Mail_Points=a
pbs.server().job(501.jupiter.example.com).mtime=1410940219
pbs.server().job(501.jupiter.example.com).Output_Path=jupiter.example.com:/home/TestUser/jobs/ST
    DIN.o501
pbs.server().job(501.jupiter.example.com).Priority=7
```

```
pbs.server().job(501.jupiter.example.com).qtime=1410940219
pbs.server().job(501.jupiter.example.com).Rerunable=True
pbs.server().job(501.jupiter.example.com).Resource_List[file]=7gb
pbs.server().job(501.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().job(501.jupiter.example.com).Resource_List[nodect]=1
pbs.server().job(501.jupiter.example.com).Resource_List[place]=pack
pbs.server().job(501.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().job(501.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().job(501.jupiter.example.com).substate=20
pbs.server().job(501.jupiter.example.com).Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash
    ,PBS_O_HOME=/home/TestUser,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LA
    NG=en_US.UTF-8,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/op
    enmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/g
    ames:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_QUEUE=
    workq,PBS_O_HOST=jupiter.example.com
pbs.server().job(501.jupiter.example.com).euser=TestUser
pbs.server().job(501.jupiter.example.com).egroup=users
pbs.server().job(501.jupiter.example.com).queue_rank=185
pbs.server().job(501.jupiter.example.com).queue_type=E
pbs.server().job(501.jupiter.example.com).Submit_arguments=<jsdl-hpcpa:Argument>-h</jsdl-hpcpa:A
    rgument>
pbs.server().job(501.jupiter.example.com).project=_pbs_project_default
pbs.server().job(502.jupiter.example.com).Job_Name=STDIN
pbs.server().job(502.jupiter.example.com).Job_Owner=TestUser@jupiter.example.com
pbs.server().job(502.jupiter.example.com).job_state=H
pbs.server().job(502.jupiter.example.com).queue=workq
pbs.server().job(502.jupiter.example.com).server=jupiter.example.com
pbs.server().job(502.jupiter.example.com).Checkpoint=u
pbs.server().job(502.jupiter.example.com).ctime=1410940221
pbs.server().job(502.jupiter.example.com).Error_Path=jupiter.example.com:/home/TestUser/jobs/STD
    IN.e502
pbs.server().job(502.jupiter.example.com).Hold_Types=u
pbs.server().job(502.jupiter.example.com).Join_Path=n
pbs.server().job(502.jupiter.example.com).Keep_Files=n
pbs.server().job(502.jupiter.example.com).Mail_Points=a
pbs.server().job(502.jupiter.example.com).mtime=1410940221
pbs.server().job(502.jupiter.example.com).Output_Path=jupiter.example.com:/home/TestUser/jobs/ST
    DIN.o502
pbs.server().job(502.jupiter.example.com).Priority=7
pbs.server().job(502.jupiter.example.com).qtime=1410940223
pbs.server().job(502.jupiter.example.com).Rerunable=True
pbs.server().job(502.jupiter.example.com).Resource_List[file]=7gb
pbs.server().job(502.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().job(502.jupiter.example.com).Resource_List[nodect]=1
pbs.server().job(502.jupiter.example.com).Resource_List[place]=pack
pbs.server().job(502.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().job(502.jupiter.example.com).schedselect=1:ncpus=1
```

```
pbs.server().job(502.jupiter.example.com).substate=20
pbs.server().job(502.jupiter.example.com).Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash
    ,PBS_O_HOME=/home/TestUser,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LA
    NG=en_US.UTF-8,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/op
    enmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/g
    ames:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_QUEUE=
    workq,PBS_O_HOST=jupiter.example.com
pbs.server().job(502.jupiter.example.com).euser=TestUser
pbs.server().job(502.jupiter.example.com).egroup=users
pbs.server().job(502.jupiter.example.com).queue_rank=186
pbs.server().job(502.jupiter.example.com).queue_type=E
pbs.server().job(502.jupiter.example.com).Submit_arguments=<jsdl-hpcpa:Argument>-h</jsdl-hpcpa:A
    rgument>
pbs.server().job(502.jupiter.example.com).project=_pbs_project_default
pbs.server().job(511.jupiter.example.com).Job_Name=job.scr
pbs.server().job(511.jupiter.example.com).Job_Owner=TestUser@jupiter.example.com
pbs.server().job(511.jupiter.example.com).resources_used[cpupercent]=0
pbs.server().job(511.jupiter.example.com).resources_used[cput]=00:00:00
pbs.server().job(511.jupiter.example.com).resources_used[mem]=0kb
pbs.server().job(511.jupiter.example.com).resources_used[ncpus]=1
pbs.server().job(511.jupiter.example.com).resources_used[vmem]=0kb
pbs.server().job(511.jupiter.example.com).resources_used[walltime]=00:00:00
pbs.server().job(511.jupiter.example.com).job_state=R
pbs.server().job(511.jupiter.example.com).queue=workq
pbs.server().job(511.jupiter.example.com).server=jupiter.example.com
pbs.server().job(511.jupiter.example.com).Checkpoint=u
pbs.server().job(511.jupiter.example.com).ctime=1410942249
pbs.server().job(511.jupiter.example.com).Error_Path=jupiter.example.com:/home/TestUser/jobs/job
    .scr.e511
pbs.server().job(511.jupiter.example.com).exec_host=jupiter/0
pbs.server().job(511.jupiter.example.com).exec_vnode=(jupiter:ncpus=1)
pbs.server().job(511.jupiter.example.com).Hold_Types=n
pbs.server().job(511.jupiter.example.com).Join_Path=n
pbs.server().job(511.jupiter.example.com).Keep_Files=n
pbs.server().job(511.jupiter.example.com).Mail_Points=a
pbs.server().job(511.jupiter.example.com).mtime=1410942250
pbs.server().job(511.jupiter.example.com).Output_Path=jupiter.example.com:/home/TestUser/jobs/jo
    b.scr.o511
pbs.server().job(511.jupiter.example.com).Priority=7
pbs.server().job(511.jupiter.example.com).qtime=1410942249
pbs.server().job(511.jupiter.example.com).Rerunable=True
pbs.server().job(511.jupiter.example.com).Resource_List[file]=7gb
pbs.server().job(511.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().job(511.jupiter.example.com).Resource_List[nodect]=1
pbs.server().job(511.jupiter.example.com).Resource_List[place]=pack
pbs.server().job(511.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().job(511.jupiter.example.com).schedselect=1:ncpus=1
```

```
pbs.server().job(511.jupiter.example.com).stime=1410942250
pbs.server().job(511.jupiter.example.com).session_id=12134
pbs.server().job(511.jupiter.example.com).jobdir=/home/TestUser
pbs.server().job(511.jupiter.example.com).substate=42
pbs.server().job(511.jupiter.example.com).Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash
    ,Monsieur=Shlomi,PBS_O_HOME=/home/TestUser,PBS_O_HOST=jupiter.example.com,PBS_O_LOGNAME=Test
    User,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LANG=en_US.UTF-8,PBS_O_QUEUE=workq,PBS_O_MAIL=/
    var/spool/mail/TestUser,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/m
    pi/gcc/openmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/b
    in:/usr/games:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin
pbs.server().job(511.jupiter.example.com).euser=TestUser
pbs.server().job(511.jupiter.example.com).egroup=users
pbs.server().job(511.jupiter.example.com).hashname=511.jupiter.example.com
pbs.server().job(511.jupiter.example.com).queue_rank=192
pbs.server().job(511.jupiter.example.com).queue_type=E
pbs.server().job(511.jupiter.example.com).comment=Job run at Wed Sep 17 at 04:24 on
    (jupiter:ncpus=1)
pbs.server().job(511.jupiter.example.com).etime=1410942249
pbs.server().job(511.jupiter.example.com).run_count=1
pbs.server().job(511.jupiter.example.com).Submit_arguments=<jsdl-hpcpa:Argument>job.scr</jsdl-hp
    cpa:Argument>
pbs.server().job(511.jupiter.example.com).project=_pbs_project_default
pbs.server().job(511.jupiter.example.com).run_version=1
pbs.server().queue(workq).queue_type=Execution
pbs.server().queue(workq).total_jobs=3
pbs.server().queue(workq).state_count=Transit:0 Queued:0 Held:2 Waiting:0 Running:1 Exiting:0
    Begun:0
pbs.server().queue(workq).resources_assigned[mem]=0mb
pbs.server().queue(workq).resources_assigned[ncpus]=1
pbs.server().queue(workq).resources_assigned[nodect]=1
pbs.server().queue(workq).enabled=True
pbs.server().queue(workq).started=True
pbs.server().queue(R503).queue_type=Execution
pbs.server().queue(R503).total_jobs=0
pbs.server().queue(R503).state_count=Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0
    Begun:0
pbs.server().queue(R503).acl_user_enable=True
pbs.server().queue(R503).acl_users=TestUser@jupiter.example.com
pbs.server().queue(R503).resources_max[ncpus]=1
pbs.server().queue(R503).resources_max[walltime]=00:30:00
pbs.server().queue(R503).resources_available[ncpus]=1
pbs.server().queue(R503).resources_available[walltime]=00:30:00
pbs.server().queue(R503).enabled=True
pbs.server().queue(R503).started=False
pbs.server().queue(R504).queue_type=Execution
pbs.server().queue(R504).total_jobs=0
pbs.server().queue(R504).state_count=Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0
    Begun:0
```

```
pbs.server().queue(R504).acl_user_enable=True
pbs.server().queue(R504).acl_users=TestUser@jupiter.example.com
pbs.server().queue(R504).resources_max[ncpus]=1
pbs.server().queue(R504).resources_max[walltime]=00:30:00
pbs.server().queue(R504).resources_available[ncpus]=1
pbs.server().queue(R504).resources_available[walltime]=00:30:00
pbs.server().queue(R504).enabled=True
pbs.server().queue(R504).started=False
pbs.server().vnode(jupiter).Mom=jupiter.example.com
pbs.server().vnode(jupiter).Port=15002
pbs.server().vnode(jupiter).pbs_version=PBSPro_10.0
pbs.server().vnode(jupiter).ntype=0
pbs.server().vnode(jupiter).state=0
pbs.server().vnode(jupiter).pcpus=1
pbs.server().vnode(jupiter).jobs=511.jupiter.example.com/0
pbs.server().vnode(jupiter).resv=R504.jupiter.example.com, R503.jupiter.example.com
pbs.server().vnode(jupiter).resources_available[arch]=linux
pbs.server().vnode(jupiter).resources_available[file]=7gb
pbs.server().vnode(jupiter).resources_available[host]=jupiter
pbs.server().vnode(jupiter).resources_available[mem]=8gb
pbs.server().vnode(jupiter).resources_available[ncpus]=8
pbs.server().vnode(jupiter).resources_available[vnode]=jupiter
pbs.server().vnode(jupiter).resources_assigned[accelerator_memory]=0kb
pbs.server().vnode(jupiter).resources_assigned[mem]=0kb
pbs.server().vnode(jupiter).resources_assigned[naccelerators]=0
pbs.server().vnode(jupiter).resources_assigned[ncpus]=1
pbs.server().vnode(jupiter).resources_assigned[vmem]=0kb
pbs.server().vnode(jupiter).resv_enable=True
pbs.server().vnode(jupiter).sharing=1
pbs.server().vnode(mars).Mom=mars.example.com
pbs.server().vnode(mars).Port=15002
pbs.server().vnode(mars).pbs_version=PBSPro_10.0
pbs.server().vnode(mars).ntype=0
pbs.server().vnode(mars).state=0
pbs.server().vnode(mars).pcpus=1
pbs.server().vnode(mars).resources_available[arch]=linux
pbs.server().vnode(mars).resources_available[file]=7gb
pbs.server().vnode(mars).resources_available[host]=mars
pbs.server().vnode(mars).resources_available[mem]=8gb
pbs.server().vnode(mars).resources_available[ncpus]=8
pbs.server().vnode(mars).resources_available[vnode]=mars
pbs.server().vnode(mars).resources_assigned[accelerator_memory]=0kb
pbs.server().vnode(mars).resources_assigned[mem]=0kb
pbs.server().vnode(mars).resources_assigned[naccelerators]=0
pbs.server().vnode(mars).resources_assigned[ncpus]=0
pbs.server().vnode(mars).resources_assigned[vmem]=0kb
```

```
pbs.server().vnode(mars).resv_enable=True
pbs.server().vnode(mars).sharing=1
pbs.server().resv(R503.jupiter.example.com).Reserve_Name=NULL
pbs.server().resv(R503.jupiter.example.com).Reserve_Owner=TestUser@jupiter.example.com
pbs.server().resv(R503.jupiter.example.com).reserve_type=2
pbs.server().resv(R503.jupiter.example.com).reserve_state=2
pbs.server().resv(R503.jupiter.example.com).reserve_substate=2
pbs.server().resv(R503.jupiter.example.com).reserve_start=1410955200
pbs.server().resv(R503.jupiter.example.com).reserve_end=1410957000
pbs.server().resv(R503.jupiter.example.com).reserve_duration=1800
pbs.server().resv(R503.jupiter.example.com).queue=R503
pbs.server().resv(R503.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().resv(R503.jupiter.example.com).Resource_List[walltime]=00:30:00
pbs.server().resv(R503.jupiter.example.com).Resource_List[nodect]=1
pbs.server().resv(R503.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().resv(R503.jupiter.example.com).Resource_List[place]=free
pbs.server().resv(R503.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().resv(R503.jupiter.example.com).resv_nodes=(jupiter:ncpus=1)
pbs.server().resv(R503.jupiter.example.com).Authorized_Users=TestUser@jupiter.example.com
pbs.server().resv(R503.jupiter.example.com).server=jupiter.example.com
pbs.server().resv(R503.jupiter.example.com).ctime=1410940237
pbs.server().resv(R503.jupiter.example.com).mtime=1410940237
pbs.server().resv(R503.jupiter.example.com).Variable_List=PBS_O_LOGNAME=TestUser,PBS_O_HOST=jupi
    ter.example.com,PBS_O_MAIL=/var/spool/mail/TestUser
pbs.server().resv(R503.jupiter.example.com).euser=TestUser
pbs.server().resv(R503.jupiter.example.com).egroup=users
pbs.server().resv(R504.jupiter.example.com).Reserve_Name=NULL
pbs.server().resv(R504.jupiter.example.com).Reserve_Owner=TestUser@jupiter.example.com
pbs.server().resv(R504.jupiter.example.com).reserve_type=2
pbs.server().resv(R504.jupiter.example.com).reserve_state=2
pbs.server().resv(R504.jupiter.example.com).reserve_substate=2
pbs.server().resv(R504.jupiter.example.com).reserve_start=1410958800
pbs.server().resv(R504.jupiter.example.com).reserve_end=1410960600
pbs.server().resv(R504.jupiter.example.com).reserve_duration=1800
pbs.server().resv(R504.jupiter.example.com).queue=R504
pbs.server().resv(R504.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().resv(R504.jupiter.example.com).Resource_List[walltime]=00:30:00
pbs.server().resv(R504.jupiter.example.com).Resource_List[nodect]=1
pbs.server().resv(R504.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().resv(R504.jupiter.example.com).Resource_List[place]=free
pbs.server().resv(R504.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().resv(R504.jupiter.example.com).resv_nodes=(jupiter:ncpus=1)
pbs.server().resv(R504.jupiter.example.com).Authorized_Users=TestUser@jupiter.example.com
pbs.server().resv(R504.jupiter.example.com).server=jupiter.example.com
pbs.server().resv(R504.jupiter.example.com).ctime=1410940250
pbs.server().resv(R504.jupiter.example.com).mtime=1410940250
```

```
pbs.server().resv(R504.jupiter.example.com).Variable_List=PBS_O_LOGNAME=TestUser,PBS_O_HOST=jupi
    ter.example.com,PBS_O_MAIL=/var/spool/mail/TestUser
pbs.server().resv(R504.jupiter.example.com).euser=TestUser
pbs.server().resv(R504.jupiter.example.com).egroup=users
```

List the execjob_launch hook execution record file:

```
jupiter:/var/spool/PBS/spool # cat hook_execjob_launch_launch_12135.out
pbs.event().accept=True
pbs.event().reject=False
pbs.event().progname=/bin/sleep
pbs.event().argv[0]=sleep
pbs.event().env=PBS_O_SYSTEM=Linux,PBS_JOBCOOKIE=00000000434AB4BA000000000BDC62D3,PBS_O_SHELL=/b
    in/bash,PBS_O_HOME=/home/TestUser,PBS_O_HOST=jupiter.example.com,PBS_NODENUM=0,PBS_O_LOGNAME
    =TestUser,PBS_JOBID=511.jupiter.example.com,PBS_JOBNAME=job.scr,PBS_O_LANG=en_US.UTF-8,USER=
    TestUser,PATH=/bin:/usr/bin,HOME=/home/TestUser,PBS_QUEUE=workq,PBS_O_MAIL=/var/spool/mail/T
    estUser,PBS_TMPDIR=/var/tmp/pbs.511.jupiter.example.com,ENVIRONMENT=BATCH,PBS_NODEFILE=/var/
    spool/PBS/aux/511.jupiter.example.com,SHELL=/bin/bash,PBS_ENVIRONMENT=PBS_BATCH,Monsieur=Shl
    omi,OMP_NUM_THREADS=1,NCPUS=1,PBS_JOBDIR=/home/TestUser,PBS_O_QUEUE=workq,PBS_MOMPORT=15003,
    PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/u
    sr/lib64/mpi/gcc/openmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/u
    sr/X11R6/bin:/usr/games:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,LOGNAME=TestUser,PBS_TASKNUM=
    1,TZ=US/Eastern
```

8.9 Interactive Debugging using pbs_python

You can perform interactive debugging by leaving out the hook name and supplying event input information and/or site data information. For example, to interactively debug with event input and site data information:

```
pbs_python --hook -i MyEventInputFile -s MySiteData
```

You get a pbs_python prompt, and in order to end the session, issue a pbs.event().accept() or pbs.event().reject():

```
>>import pbs
>>print pbs.event().job.id
1234.examplehost
>>pbs.event().accept()
```

8.10 Error Reporting and Logging

Hook errors are printed to stderr for the command (qsub, qalter, pbs_rsub, or qmove) that triggered the hook. If the hook provides a custom error message, that message is treated the same way.

Hooks can log custom strings to the log file of the daemon from which the hook is executing. When logging a message, a hook uses message logging methods to specify the message, and constant objects to specify the log event class. See "pbs.logmsg()" on page 152, and section 6.13.4.4, "Message Log Level Objects", on page 152.

When the PBS server starts, it prints to the server logs both the Python version integrated with the server, and a list of all the hook names registered with the server.

To see only hook-related 0x0400 messages in the MoM logs, such as "<hook name>;started", "<hook_name>;finished", set the $logevent MoM parameter to *0x400* in the MoM configuration file.

To see all the different types of MoM log messages, set $logevent to *0xffff*.

The default value for the $logevent MoM parameter is *975*, so that the following log events are captured. See "Log Levels" on page 377 of the PBS Professional Reference Guide for more about log levels.

PBSEVENT_ERROR
PBSEVENT_SYSTEM
PBSEVENT_ADMIN
PBSEVENT_JOB
PBSEVENT_JOB_USAGE
PBSEVENT_SECURITY
PBSEVENT_DEBUG
PBSEVENT_DEBUG2
PBSEVENT_RESV

8.10.1 Errors During Creation and Deployment

8.10.1.1 Hook Name Matches Existing Hook

Creating a hook whose name matches that of an existing hook: the following error message is printed in stderr and in the server logs:

```
"hook error: hook name <hook_name> already registered, try another name"
```

8.10.1.2 Using a Hook Name that Starts with "*PBS*"

Using a hook name that starts with "*PBS*": the hook name is rejected with the following error in qmgr's stderr, as well as in the server logs:

```
"hook error: cannot use PBS as a prefix - it is reserved for PBS hooks"
```

8.10.1.3 Deleting a Non-Existent Hook

Deleting a non-existent hook: the following is returned in qmgr's stderr and server logs:

```
"qmgr: hook error: <non-existent hook name> does not exist"
```

8.10.1.4 Specifying a Non-Existent Event Type

Specifying a non-existent event type: an error message is printed to qmgr's stderr and also to the server logs:

Example:

```
Qmgr: set hook hook1 event="mom_checkpoint"
"hook error: invalid argument to event. Should be one of: queuejob, modifyjob, resvsub, movejob,
    runjob, provision, execjob_begin, execjob_prologue, execjob_epilogue, execjob_preterm,
    execjob_end, exechost_periodic, execjob_launch, exechost_startup, execjob_attach or "" for no
    event."
"qmgr: hook error returned from server"
```

8.10.1.5 Using a Bad Hook Value

Putting in a bad hook value: an error is printed to qmgr's stderr and also to the server logs:

Example:

```
Qmgr: set hook hook2 order=1025
"qmgr obj=hookA svr=default: order given (1025) is outside the acceptable range of [1, 1000] for
   type 'site'."
"qmgr: hook error returned from server"
```

8.10.1.6 Unauthorized User

If qmgr is invoked, and the object being operated on is "*hook*", and the executing user at some host does not have access to the target server's private location for hooks data, then the following error is issued to stderr and server logs:

```
"<user>@<host> is unauthorized to access hooks data from server <hostname>"
```

8.10.1.7 Setting a Bad Hook Type

Setting a bad type to a hook produces the following error message in qmgr's stderr and also in the server logs:

```
"hook error: invalid argument to type. Must be site"
```

8.10.1.8 Setting a Bad Alarm Value

Setting a bad alarm value to a hook produces the following error message in qmgr's stderr and also in the server logs:

```
"hook error: alarm value of a hook must be > 0"
```

8.10.1.9 Exporting To Non-Writable File

Exporting a hook's content to a file that is not writable due to ownership or permission problems results in the following error message being printed to stderr:

```
"qmgr: hook error: <output_file> permission denied"
```

8.10.1.10 Setting Bad Hook user Attribute

Setting a value for the user attribute of a hook to something other than "*pbsadmin*" produces the following error message in qmgr's stderr and also in the server logs:

```
"hook error: user value of a hook must be pbsadmin, pbsuser"
```

This attribute does not need to be set to the actual name of the PBS service account.

8.10.1.11 Importing From Non-Readable File

Importing a hook where the PBS server is unable to open the input file because the file is non-existent, has a permission problem, or any other system-related error causes the following error message to be printed in stderr and in the server logs:

```
"qmgr: hook error: unable to open <filename> by server run by <user>@<host>: <error message>"
```

Examples:

```
"qmgr: hook error: unable to open hook1.py by server run by pbsadmin@hostX: permission denied"
"qmgr: hook error: unable to open hook1.py by server run by pbsadmin@hostY: No such file or
   directory"
```

8.10.1.12 Importing or Exporting with Wrong Content Type

Importing or exporting a hook where the <*content-type*> is something other than "*application/x-python*" causes the following error message to be printed in stderr and in the server logs:

"qmgr: hook error: <content_type> must be 'application/x-python'"

Importing/exporting a hook where the <*content-encoding*> is something other than "*default*" or "*base64*" causes the following error message to be printed in stderr and on the server logs:

"qmgr: hook error: <content_encoding> must be 'default' or 'base64'"

An import call on a hook that already has a content script results in the following informational message being printed in stdout and server logs:

"qmgr: hook <hook_name> contents overwritten by file <hook input file>

8.10.1.13 Setting Vnode State to Invalid Value

Setting a vnode's state attribute to an invalid value causes the pbs.BadAttributeValueError exception to be raised.

8.10.1.14 Creating a Hook with Same Name as Existing Hook

You may find that when you remove a hook, it may take some time for the hook to be completely purged. If you run "qmgr -c 'create hook <hook_name>'" where a previous hook of the same <*hook_name*> still exists, you will see the following message:

"hook name <hook_name> is pending delete, try another name"

Either specify another name for the hook, or retry the qmgr request again later, after the previous hook is completely purged.

8.10.2 Errors And Messages During Hook Execution

8.10.2.1 Successful Operation of runjob Hook

When a hook successfully sets an attribute, one of the following is written to the server's log:

<job ID>; '<hook name>' hook set job's <attribute name> = <value>

or

Job held by '<hook name>' hook on <timedate>

8.10.2.2 Unsuccessful Operation for runjob Hook

When a hook fails to set an attribute, the following is written to the server's log:

<job ID>; '<hook name>' hook failed to set job's <attribute name> = <value>

8.10.2.3 Rejecting an Action

If a hook rejects an action by calling the pbs.event().reject() function:

- The following messages are printed to stderr of the command that triggered the hook:

  "<command_name>: Request rejected by filter hook <hook_name>" "<command_name>:<'msg' value passed to pbs.event().reject()>"

where 'msg' is the message passed (if any) as input to pbs.event().reject().

- The following messages are printed in the appropriate PBS daemon log, logged at event class 0x0400:

  "<user>@<host>…<request type> request rejected by <hook name> "<user>@<host> …<request type> <'msg' value passed to pbs.event().reject()>"

8.10.2.4 Triggering an Alarm

If the alarm was triggered while executing a hook:

- The command that initiated the request gets the following messages in its stderr:

  "<command_name>: Request rejected by filter hook <hook_name>" "<command_name>: alarm call while running hook <hook_name>"

- The following entry appears in the appropriate PBS daemon log, logged under event class PBSEVENT_DEBUG2:

  "<user>@<host>…<request type> alarm call while running hook <hook_name>, request rejected"

8.10.2.5 Encountering an Unhandled Exception

If a hook encounters an unhandled exception:

- PBS rejects the corresponding action. The command that triggered the hook gets the following message in stderr:

  "<command_name>: request rejected as filter hook <hook_name> encountered an exception. Inform admin."

- The following message appears on the appropriate PBS daemon log, logged under PBSEVENT_DEBUG2 event class:

  "<request type> hook <hook_name> encountered an exception, request rejected"

See section 5.2.3, “Hook Alarm Calls and Unhandled Exceptions”, on page 43.

8.10.2.6 Starting and Finishing Hook Execution

Whenever hook execution starts or finishes, timestamped 0x0400 event class log messages appear in the appropriate PBS daemon log:

"11/13/2007 00:00:42 …<user>@<host>…<request type> running hook named <hook name>"

"11/13/2007 00:01:42<user@><host>…<request type> <hook_name> finished"

8.10.2.7 Hook Timeout

When a hook timeout is triggered, the hook script gets a Python KeyboardInterrupt from the PBS server. The server logs show the following:

06/17/2008 17:57:16;0001;Server@host2;Svr;Server@host2;PBS server internal error (15011) in Python script received a KeyboardInterrupt, <type 'exceptions.KeyboardInterrupt'>

8.10.2.8 Hooks Attempting I/O

When the PBS server is running, stdout, stderr, and stdin are closed, so that a hook script containing calls to print to standard output or standard error, or to read input from standard input, gets the following exception:

02/24/2008 08:03:34;0086;Server@a-centauri;Svr;Server@a-centauri;Compiling script file: </var/spool/pbs/server_priv/hooks/hook_test.PY>

02/24/2008 08:03:34;0001;Server@a-centauri;Svr;Server@a-centauri;PBS server internal error (15011) in Error evaluating Python script, <type 'exceptions.IOError'>

8.10.2.9 Bad Value for debug Attribute

If you specify an invalid value for a hook's debug attribute, the following error message appears in qmgr's STDERR:

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

8.10.2.10 Commands Fail Inside Hooks

When a command fails inside a hook, but succeeds outside the hook, the problem may be a difference in the environments.

8.10.2.11 runjob Hook Errors

8.10.2.11.i Modifying Hold, Execution Time, Dependency, or Project of Accepted Job

If a runjob hook accepts an event request, using pbs.event().accept(), but attempts to set a disallowed attribute, the hook request is rejected.

If the hook is triggered by a qrun command, the following message is sent to stderr where the qrun command was run. If the hook is triggered when the scheduler tries to run the job, the following message is written to the job's comment attribute:

```
request rejected by filter hook <hook_name>: cannot modify job after runjob request has been
    accepted.
```

The following message is written to the PBS server log, at log event class PBSEVENT_DEBUG2:

```
<hook name>; Found job <attribute name> attribute flagged to be set
runjob request rejected by <hook name>: cannot modify job after runjob request has been accepted.
```

8.10.2.11.ii Modifying Disallowed Attributes of Rejected Job

If a runjob hook rejects an event request, using pbs.event().reject(), but attempts to do any of the above, the following message is written to the PBS server log, at log event class 0x0100:

```
runjob request rejected by <hook name>: cannot modify job attribute
<attribute name> after runjob request has been rejected.
```

8.10.2.11.iii Modifying Vnode

If a runjob hook event is accepted via a pbs.event().accept() call, and yet an attempt is made to modify a vnode's state, then the hook request is rejected. The following message is sent to the stderr of qrun, and becomes the job's comment:

```
request rejected by filter hook <hook_name>: cannot modify vnode after runjob request has been
    accepted.
```

The following message appears in the PBS server log, logged at event class PBSEVENT_DEBUG2:

```
runjob request rejected by <hook name>: cannot modify a vnode after runjob request has been
    accepted.
```

8.10.2.11.iv runjob Hook Referencing Wrong Parameter

If a runjob hook attempts to reference a pbs.event() parameter other than pbs.event().job, the exception pbs.EventIncompatibleError is raised.

8.10.2.11.v Attempting to Set Restricted Resource

A runjob hook cannot set the value of a Resource_List member other than those listed in Table 5-9, “Built-in Job Resources Readable & Settable by Hooks via Events,” on page 60.

Setting any of the wrong resources results in the following:

- The hook request is rejected
- The following message is sent to the STDERR of qrun, or 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)"
- The scheduler updates the affected job's comment attribute with the above message.
- The following message appears in the server's log, logged at level PBSEVENT_DEBUG2:
  "runjob request rejected: '<hook name>' hook failed to set job's Resource_list.<resc_name> = <resc_value> (not allowed)"

8.10.2.12 Special Errors Requiring Support

If you encounter any of the following log messages, an internal failure has occurred during hook setup. Please contact PBS Professional support:

```
04/15/2011 17:55:23;0100;Server@jobim;Hook;<hook_name>t3;Encountered an error while setting event
04/15/2011 17:55:23;0001;Server@jobim;Svr;Server@jobim;PBS server internal error (15011) in
    _get_job, partially populated python job object
04/15/2011 17:55:23;0001;Server@jobim;Svr;Server@jobim;PBS server internal error (15011) in
    _get_server, partially populated python server object
04/15/2011 17:55:26;0001;Server@jobim;Svr;Server@jobim;PBS server internal error (15011) in
    _get_queue, partially populated python queue object
04/15/2011 17:55:26;0001;Server@jobim;Svr;Server@jobim;PBS server internal error (15011) in
    _get_vnode, partially populated python vnode object
04/15/2011 17:55:26;0001;Server@jobim;Svr;Server@jobim;PBS server internal error (15011) in
    _get_resv, warning: partially populated python resv object
```

8.10.3 Errors During Startup

If the server starts up and encounters a hook that has no content (no script was imported into the hook), PBS displays the following warning:

```
"failed to stat <path_server_priv_hooks>/<hook_name>.PY"
"failed to allocate storage for python script
<path_server_priv_hooks>/<hook_name>.PY"
```

8.10.4 Errors in Hook Updates

Updates to hooks are asynchronous with respect to jobs. During an update, some jobs may run on updated MoMs while others run on MoMs that are not yet updated. A multi-host job that started running before the update may find itself running on some MoMs that are updated and some that are not. In addition, a multi-host job that starts during the update may start on updated and non-updated MoMs. When a job triggers a hook, the hook that runs is the current hook, not the hook that was there when the job started. If you change, delete, or add a hook while a job is running, and the job subsequently triggers the hook, that job will encounter whatever changes have propagated to the MoM.

- If a job runs where a hook update is incomplete, PBS prints the following to the server's log file:
  "vnode <node_name>'s parent mom <mom_host>:<mom_port> has a pending copy hook or delete hook request"

Bear in mind that hooks are updated asynchronously with respect to jobs, so a multi-host job that started before the update may encounter an incompletely updated hook.

- As PBS copies or deletes execution or periodic hooks to the MoMs, the following messages are printed in the server's log file at 2047:

  "successfully sent hook file <filename> to <mom_hostname>"

  "successfully sent rescdef file <filename> to <mom host name>"

  "successfully deleted hook file <filename> from <mom host name>"

  "successfully deleted rescdef file <filename> from <mom host name>"

  "failed to copy hook file <filename> to <mom host name>"

  "failed to copy rescdef file <filename> to <mom host name>"

  "failed to delete hook file <filename> from <mom host name>"

  "failed to delete rescdef file <filename> from <mom host name>"

- You may find that when you remove a hook, it takes some time for the hook to be completely purged. If you run "`qmgr -c 'create hook <hook_name>'`" where a previous hook of the same <*hook_name*> still exists, you will see the following message:

  "hook name <hook_name> is pending delete, try another name"

  Either specify another name for the hook, or retry the `qmgr` request again later, after the previous hook is completely purged.

- If a hook tries to use a resource that is not yet propagated, this will cause an exception, which if unhandled, may delete the job. Write your hooks so that they trap exceptions and deal gracefully with the job. For example, you can use pbs.event().job.rerun(). Custom resources are propagated to MoMs under the following circumstances:
  - When you install PBS on a multi-vnoded machine
  - When you add MoMs, resources are propagated to those MoMs
  - When you create a custom resource inside a hook

8.10.5 Hook-related Error Codes

The following are hook-related error codes:

Table 8-2: Hook-related Error Codes

Error Name	Code	Description
PBSE_MOM_INCOMPLETE_HOOK	*15167*	Execution hook not fully transferred to a particular MoM
PBSE_MOM_REJECT_ROOT_SCRIPTS	*15168*	A MoM has rejected a request to copy a hook-related file, or a job script to be executed by root
PBSE_HOOK_REJECT	*15169*	A MoM received a reject result from an execution or periodic hook
PBSE_HOOK_REJECT_RERUNJOB	*15170*	Hook rejection requiring a job to be rerun
PBSE_HOOK_REJECT_DELETEJOB	*15171*	Hook rejection requiring a job to be deleted

8.10.6 Troubleshooting

8.10.6.1 Bad Interpreter Path

If you see the following error:

```
/opt/pbs/bin/pbs_python: bad interpreter: No such file or directory
```

You should check to see whether this is a valid path on this host. Try to `cd` to the job execution directory and execute any command using this interpreter path.

8.10.6.2 Viewing Hook Propagation

You don't need to restart `pbs_mom` for a MoM hook to take effect. If you use `qmgr`, PBS takes care of copying the new hook over to the MoM, in the background. It's possible a job may have seen the old MoM hook before the new hook arrives. After the new hook arrives, you'll see a message in the `server_logs` with the following:

```
vnode <name>'s parent mom <mom_name> has a pending copy hook or delete hook request
```

9
Hook Examples

Contents

9.1 resvsub Hook Examples . . . 232
9-1 Restrict ability to submit reservations to PBS administrators . . . 232
9.2 queuejob Hook Examples . . . 234
9-2 Reject jobs which do not specify walltime. . . . 234
9-3 Reject jobs with CPU requests that are not multiples of 8 . . . 235
9-4 If a user asks for -l ncpus=8:ppn=24, change ncpus to 24 . . . 237
9-5 Calculate and set custom resource . . . 238
9-6 Put interactive jobs in a particular queue . . . 239
9-7 Set job project based on queue where job is submitted . . . 240
9-8 Speed up throughput of interactive jobs . . . 241
9-9 Validate job account . . . 242
9-10 Check job resource request and verify that job can run in this complex . . . 244
9.3 modifyjob Hook Examples . . . 262
9-11 Prevent users from using qalter to change their jobs . . . 262
9-12 Reject jobs requesting a specific queue that do not request mem . . . 263
9.4 periodic Hook Examples . . . 264
9-13 Run job start time estimator . . . 264
9.5 execjob_launch Hook Examples . . . 265
9-14 Modify arguments to job program . . . 265
9.6 execjob_prologue and execjob_epilogue Hook Examples . . . 266
9-15 Run shell script prologue or epilogue. . . . 266
9.7 exechost_startup Hook Examples . . . 279
9-16 Create vnode and set vnode resources . . . 279
9.8 exechost_periodic Hook Examples . . . 281
9-17 Monitor load; offline or free vnode depending on CPU load. . . . 281
9-18 Periodically update resources on vnodes . . . 282
9-19 Log loads on vnodes . . . 284
9-20 Set job attributes and resources . . . 285
9.9 Multi-event Hooks. . . . 286
9-21 Helper function for logging exceptions more completely and flexibly:. . . . 286

9.1 resvsub Hook Examples

Example 9-1: Restrict ability to submit reservations to PBS administrators

Hook type: resvsub

Script NoSub.py on Windows:

```
import pbs
import os
e = pbs.event()
r = e.resv
who = e.requestor
pbs.logmsg(pbs.LOG_DEBUG, "requestor=%s" % (who,))
isadmin=0

admin_ulist = ["PBS_Server", "Scheduler", "pbs_mom", "Administrator"]
if who in admin_ulist:
    isadmin=1
else:
    cmd = "net user "  + who + "/domain"
    admin_glist = ['Administrators', 'Domain Admins', 'Enterprise
        Admins']
    for line in os.popen(cmd).readlines():
        if line.find("Group") >= 0:
            for li in line.split("*"):
                if li.strip() in admin_glist:
                    isadmin=1
                    break
if e.type == pbs.RESVSUB and not isadmin:
    e.reject("Only admins allowed to create reservations!")
```

Script NoSub.py on Linux:

```
import pbs
import os
e = pbs.event()
r = e.resv
who = e.requestor
pbs.logmsg(pbs.LOG_DEBUG, "requestor=%s" % (who,))

admin_ulist = ["PBS_Server", "Scheduler", "pbs_mom", "root"]

if e.type == pbs.RESVSUB and who not in admin_ulist:
    e.reject("Only admins allowed to create reservations!")
```

Create hook and import script:

```
qmgr -c 'create hook NoSub event="resvsub"'
qmgr -c 'import hook NoSub application/x-python default NoSub.py'
```

9.2 queuejob Hook Examples

Example 9-2: Reject jobs which do not specify walltime

Hook type: queuejob

Script RequireWalltime.py:

```
import pbs
import sys

try:
    e = pbs.event()
    j = e.job
    if j.Resource_List["walltime"] == None :
        e.reject("Job has no walltime requested")

except SystemExit:
    pass

except pbs.UnsetResourceNameError:
    e.reject("Job has no walltime requested")
```

Create hook and import script:

```
qmgr -c 'create hook RequireWalltime event="queuejob"'
qmgr -c 'import hook RequireWalltime application/x-python default RequireWalltime.py'
```

Example 9-3: *Reject jobs with CPU requests that are not multiples of 8*

Hook type: queuejob

Script Multiple8.py:

```
import pbs
import sys

e = pbs.event()
j = e.job

mult_limit = 8

if j.Resource_List["ncpus"] != None:
    try:
        e = pbs.event()
        j = e.job
        R = j.Resource_List["ncpus"] % mult_limit
        if R != 0:
            e.reject("Ncpus resource is not a multiple of %s." % (mult_limit,))
    except SystemExit:
        pass
    except (pbs.UnsetResourceNameError, TypeError):
        e.reject("Bad ncpus resource value.")

else:
    R = pbs.event().job.Resource_List
    sel = repr(R["select"])
    tot_ncpus = 0
    for chunk in sel.split("+"):
        nchunks = 1
        for c in chunk.split(":"):
            kv = c.split("=")
            if len(kv) == 1:
                nchunks = kv[0]
            elif len(kv) == 2:
                if kv[0] == "ncpus":
                    tot_ncpus += (int(nchunks) * int(kv[1]))

    try:
        mod = tot_ncpus % mult_limit
        if mod != 0:
            e.reject("Ncpus resource is not a multiple of %s." % \
            (mult_limit,))
    except SystemExit:
        pass
    except (pbs.UnsetResourceNameError, TypeError):
```

```
        e.reject("Bad Ncpus resource value.")
```

Create hook and import script:

```
qmgr -c 'create hook Multiple8 event="queuejob"'
qmgr -c 'import hook Multiple8 application/x-python default Multiple8.py'
```

Example 9-4: *If a user asks for* `-l ncpus=8:ppn=24`*, change* ncpus *to 24*

Hook type: queuejob

Script `ChangeNcpus.py`:

```
import pbs
import sys

try:
    e = pbs.event()
    j = e.job
    j.Resource_List["ncpus"] = max(j.Resource_List["ncpus"],j.Resource_List["ppn"])

except SystemExit:
    pass

except (pbs.UnsetResourceNameError, pbs.BadResourceValError):
    e.reject("Failed to reset ncpus value")
```

Create hook and import script:

```
qmgr -c 'create hook ChangeNcpus event="queuejob"'
qmgr -c 'import hook ChangeNcpus application/x-python default ChangeNcpus.py'
```

Example 9-5: *Calculate and set custom resource*

Hook type: queuejob

Custom resource cph == total ncpus * walltime (in hours). Calculate it and set its value.

You must create the cph resource before using it.

Script CustCPH.py:

```
import pbs

R = pbs.event().job.Resource_List
sel = repr(R["select"])
tot_ncpus = 0
for chunk in sel.split("+"):
    nchunks = 1
    for c in chunk.split(":"):
        kv = c.split("=")
        if len(kv) == 1:
            nchunks = kv[0]
        elif len(kv) == 2:
            if kv[0] == "ncpus":
                tot_ncpus += (int(nchunks) * int(kv[1]))
R["cph"] = tot_ncpus * R["walltime"]
```

Create hook and import script:

```
qmgr -c 'create hook CustCPH event="queuejob"'
qmgr -c 'import hook CustCPH application/x-python default CustCPH.py'
```

Example 9-6: *Put interactive jobs in a particular queue*

Hook type: queuejob

Put job into "interQ" if the job was submitted interactively (using *qsub -I*).

Script IQueue.py:

```
# get the pbs module
import pbs
import sys
try:
    # Get the hook event information and parameters
    # This will be for the 'queuejob' event type.
    e = pbs.event()

    # Get the information for the job being queued
    j = e.job
    if j.interactive:
        # Get the "interQ" queue object
        q = pbs.server().queue("interQ")
        # Reset the job's destination queue
        # parameter for this event
        j.queue = q
        # accept the event
        e.accept()
except SystemExit:
    pass
except:
    e.reject("Failed to route job to queue interQ")
```

Create hook and import script:

```
qmgr -c 'create hook IQueue event="queuejob"'
qmgr -c 'import hook IQueue application/x-python default IQueue.py'
```

Example 9-7: *Set job project based on queue where job is submitted*

Hook type: queuejob

The following is a snippet of a queuejob hook:

```
import pbs
e = pbs.event()
If e.job.queue == None:
    # user did not specify a queue to submit to, so use default
    target_qname = pbs.server().default_queue
else:
    target_qname = e.job.queue.name
If (target_qname == "large") or (target_qname == "medium"):
    e.job.project = "some_large_medium_project"
```

Example 9-8: Speed up throughput of interactive jobs

Hook type: queuejob

Use a queuejob hook that determines whether a job entering the system is an interactive job. If so, it directs the job to the high priority queue specified in 'high_priority_queue', and tells the server to restart the scheduling cycle. You must first define a "high" queue as follows:

```
qmgr -c "create queue high queue_type=e,Priority=150"
qenable high
qstart high
```

The default priority for an express queue is *150*. If you do not want interactive jobs to go into an express queue, set the priority of the queue named "high" to a value greater than ordinary queues but lower than the value for an express queue. See section 4.9.18, “Express Queues”, on page 139.

Instantiate the hook as follows:

```
qmgr -c "create hook rapid_inter event=queuejob"
qmgr -c "import hook rapid_inter application/x-python default rapid_inter.py"
```

Hook script:

```
import pbs

high_priority_queue="high"

e = pbs.event()
if e.job.interactive:
    high = pbs.server().queue(high_priority_queue)
    if high != None:
        e.job.queue = high
        pbs.logmsg(pbs.LOG_DEBUG, "quick start interactive job")
        pbs.server().scheduler_restart_cycle()
```

Example 9-9: *Validate job account*

This hook reads valid accounts from a JSON file.

```
import os
import simplejson


try:
    import pbs

    pbs_conf = pbs.pbs_conf

except ImportError:
    pass

# Read in the configurations file
pbs_hook_cfg = pbs.hook_config_filename
if pbs_hook_cfg == None:
    pbs.logmsg(pbs.EVENT_DEBUG3,"%s"%os.environ)
    pbs_hook_cfg = os.environ["PBS_HOOK_CONFIG_FILE"]
pbs.logmsg(pbs.EVENT_DEBUG3,"read config file: %s"%pbs.hook_config_filename)
config_file = open(pbs.hook_config_filename).read()

va_cfg = simplejson.loads(config_file)

#pbs.logmsg(pbs.EVENT_DEBUG2,"config file: %s"%va_cfg)

je = pbs.event()
j = pbs.event().job

user=je.requestor

account=j.Account_Name

#pbs.logmsg(pbs.EVENT_DEBUG2,"my Account_Name is: %s"%account)
#pbs.logmsg(pbs.EVENT_DEBUG2,"allowed users for this account are:
    %s"%va_cfg["accounts"][account])

if user in va_cfg["accounts"][account]:
    pbs.logmsg(pbs.EVENT_DEBUG2,"user is allowed to submit to account")
else:
    pbs.logmsg(pbs.EVENT_DEBUG2,"user is NOT allowed to submit to account")
    je.reject("user is unauthorized to submit to this account")
```

Here is an example JSON file with valid accounts:

```
{
   "accounts": {
      "account1":["user1"],
      "account11":["user11"],
      "account2":["user2"],
      "accountall":["user1","user2"]
   }
}
```

Example 9-10: *Check job resource request and verify that job can run in this complex*

```
#!/usr/bin/env python

# -*- coding: utf-8 -*-

############################################################################
# Purpose: To check the request of the job and verify that it will be able to
#          run on this cluster
#
# Setup: Modify the config file. You will need to provide the correct
#    information for your complex
############################################################################

import pbs
import sys
import os
from string import join
import simplejson as json
import traceback
import re
import string

pbs.logmsg(pbs.EVENT_DEBUG, "Entering the check limits hook")
e = pbs.event()

py_base_dir = pbs.pbs_conf['PBS_EXEC'] + "/python/lib/python2.5"
try:
    sys.path.index(py_base_dir + 'site-packages')
except ValueError:
    sys.path = [py_base_dir,
                py_base_dir + 'plat-linux2',
                py_base_dir + 'lib-tk',
                py_base_dir + 'lib-dynload',
                py_base_dir + 'site-packages'] \
        + sys.path

def caller_name():
    return str(sys._getframe(1).f_code.co_name)

# Define error codes

class AdminError(Exception):
    pass

class ConfigError(AdminError):
    pass
```

```
def e_reject(msg):
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Message: %s" % (caller_name(), msg))
    pbs.event().reject(msg)

#
# FUNCTION decode_dict
#

def decode_dict(data):
    rv = {}
    for key, value in data.iteritems():
        if isinstance(key, unicode):
            key = key.encode('utf-8')
        if isinstance(value, unicode):
            value = value.encode('utf-8')
        elif isinstance(value, list):
            value = decode_list(value)
        elif isinstance(value, dict):
            value = decode_dict(value)
        rv[key] = value
    return rv

def decode_list(data):
    rv = []
    for item in data:
        if isinstance(item, unicode):
            item = item.encode('utf-8')
        elif isinstance(item, list):
            item = decode_list(item)
        elif isinstance(item, dict):
            item = decode_dict(item)
        rv.append(item)
    return rv

#
# FUNCTION convert_size
#
# Convert a string containing a size specification (e.g. "1m") to a
# string using different units (e.g. "1024k").
#
# This function only interprets a decimal number at the start of the string,
# stopping at any unrecognized character and ignoring the rest of the string.
#
# When down-converting (e.g. MB to KB), all calculations involve integers and
# the result returned is exact. When up-converting (e.g. KB to MB) floating
```

```
# point numbers are involved. The result is rounded up. For example:
#
# 1023MB -> GB yields 1g
# 1024MB -> GB yields 1g
# 1025MB -> GB yields 2g  <-- This value was rounded up
#
# Pattern matching or conversion may result in exceptions.
#

def convert_size(value, units='b'):
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Method called" % (caller_name()))
    pbs.logmsg(pbs.EVENT_DEBUG3, "value: %s, units: %s" % (value, units))
    logs = {'b': 0, 'k': 10, 'm': 20, 'g': 30,
            't': 40, 'p': 50, 'e': 60, 'z': 70, 'y': 80}
    try:
        new = units[0].lower()
        if new not in logs:
            new = 'b'
        val, old = re.match('([-+]?\d+)([bkmgtpezy]?)',
                            str(value).lower()).groups()

        val = int(val)
        if val < 0:
            raise ValueError('Value may not be negative')
        if old not in logs.keys():
            old = 'b'
        factor = logs[old] - logs[new]
        val *= 2 ** factor
        slop = val - int(val)
        val = int(val)
        if slop > 0:
            val += 1
        # pbs.size() does not like units following zero
        if val <= 0:
            pbs.logmsg(pbs.EVENT_DEBUG3, "Return value: %s" % str(0))
            return '0'
        else:
            pbs.logmsg(pbs.EVENT_DEBUG3, "Return value: %s" % str(val) + new)
            return str(val) + new
    except:
        pbs.logmsg(pbs.EVENT_DEBUG3, "Return value: None")
        return None

#
# FUNCTION size_as_int
#
```

```
# Convert a size string to an integer representation of size in bytes
#


def size_as_int(value):
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Method called" % (caller_name()))
    return int(convert_size(value).rstrip(string.ascii_lowercase))

#
# FUNCTION caller_name
#
# Return the name of the calling function or method.
#


# Read the config file in json format
def parse_config_file(e, s):
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Method called" % (caller_name()))
    config = {}

    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Server Name: %s" %
               (caller_name(), s.name))

    # Identify the config file and read in the data
    if pbs.hook_config_filename is not None:
        config_file = pbs.hook_config_filename
        pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Config file is %s" %
                   (caller_name(), config_file))
        try:
            config = json.load(open(config_file, 'r'),
                               object_hook=decode_dict)
        except IOError:
            pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Encountered IOError:\n %s" %
                       (caller_name(), sys.exec_info()[0]))
            raise ConfigError("I/O error reading config file")
        except json.JSONDecodeError:
            pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Encountered DecodeError:\n %s" %
                       (caller_name(), sys.exec_info()[0]))
            raise ConfigError(
                "JSON parsing error reading config file")
        except Exception:
            pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Encountered error:\n %s" %
                       (caller_name(), sys.exec_info()[0]))
            raise
    else:
        raise ConfigError("No configuration file present")
```

```
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Config file:\n %s" %
               (caller_name(), config))
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: I am here 0 " % (caller_name()))
    # Set some defaults if they are not present
    if 'clusters' not in config:
        pbs.logmsg(pbs.EVENT_DEBUG3, "%s: I am here 1 " % (caller_name()))
        e_reject("Please define the cluster inputs ")
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: I am here 1.5 " % (caller_name()))
    if s.name not in config['clusters']:
        pbs.logmsg(pbs.EVENT_DEBUG3, "%s: I am here 2 " % (caller_name()))
        e_reject("Cluster: %s needs to be " % s.name +
                 "defined in the config file: %s" % config['clusters'].keys())
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: I am here 2.5 " % (caller_name()))
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: %s " % (caller_name(), s.name))
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: %s " %
               (caller_name(), config["clusters"][s.name]))
    if 'default_queue' not in config["clusters"][s.name]:
        pbs.logmsg(pbs.EVENT_DEBUG3, "%s: I am here 3 " % (caller_name()))
        # Find the default queue
        config["clusters"][s.name]['default_queue'] = s.default_queue
        # e_reject("Please define the default queue for the job")
    else:
        pbs.logmsg(pbs.EVENT_DEBUG3, "%s: I am here 4 " % (caller_name()))
        try:
            s.queues(config["clusters"][s.name]['default_queue'])
            pbs.logmsg(pbs.EVENT_DEBUG3, "%s: I am here 5 " % (caller_name()))
        except:
            pbs.logmsg(pbs.EVENT_DEBUG3, "%s not found in pbs complex " %
                       config["clusters"][s.name]['default_queue'])
            config["clusters"][s.name]['default_queue'] = s.default_queue
            pbs.logmsg(pbs.EVENT_DEBUG3, "Changed default queue to %s" %
                       config["clusters"][s.name]['default_queue'])
    if 'site_info' not in config["clusters"][s.name]:
        config["clusters"][s.name]['site_info'] = "not undefined"

    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Return Config file:\n %s" %
               (caller_name(), config))
    return config


def chunk_resource_check(e, request, cluster):
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Method called" % (caller_name()))
    pbs.logmsg(pbs.EVENT_DEBUG3, "Request: %s" % (request))
    pbs.logmsg(pbs.EVENT_DEBUG3, "type: %s" % (type(request[0])))
    pbs.logmsg(pbs.EVENT_DEBUG3, "type: %s" % (type(request[1])))
```

```
resource = request[0]
if resource not in cluster:
    pbs.logmsg(pbs.EVENT_DEBUG3,
               "Resource %s is not defined in the cfg" %
               resource)
    try:
        pbs.logmsg(pbs.EVENT_DEBUG3,
                   "Trying to return an int for %s" % (request))
        return int(request[1])
    except:
        pbs.logmsg(pbs.EVENT_DEBUG3,
                   "Returning a string")
        return request[1]
elif resource == 'mem':
    value = pbs.size(request[1])
    clust_res = pbs.size(str(cluster[resource]))
elif isinstance(cluster[resource], int):
    value = int(request[1])
    clust_res = cluster[resource]
elif isinstance(cluster[resource], float):
    value = int(request[1])
    clust_res = cluster[resource]
else:
    pbs.logmsg(pbs.EVENT_DEBUG3, "Not checking resource: %s" % (resource))
    return True

pbs.logmsg(pbs.EVENT_DEBUG3, "Resource: %s" % (resource))
pbs.logmsg(pbs.EVENT_DEBUG3, "R:%s A:%s" %
           (str(request[1]), str(clust_res)))
if value > clust_res:
    pbs.logmsg(pbs.EVENT_DEBUG3, "I am here")
    line = "\nError: You requested %s=%s per " % (resource, value) + \
           "node. This exceeds the available %s " % resource + \
           "on a %s node (%s)\n" % (cluster['name'], clust_res)
    if resource == 'ncpus':
        clust_mem = pbs.size(str(cluster['mem']))
        line += "For example on %s use -l " % cluster['name'] + \
                "select=2:%s=%s:mem=%s" % \
                (resource, cluster[resource], clust_mem)
    else:
        line += "For example on %s use -l " % cluster['name'] + \
                "select=2:ncpus=%s:%s=%s" % \
                (cluster['ncpus'], resource, clust_res)
    line += "\nIf you still have questions, " + \
            "please refer to %s" % \
```

```
                cluster['site_info']
            pbs.logmsg(pbs.EVENT_DEBUG3, "line: %s" % (line))
            e_reject(line)
            return False
        pbs.logmsg(pbs.EVENT_DEBUG3, "Return %s value: %s" % (resource, value))
        return value


def job_ncpus_check(e, request, cluster):
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Method called" % (caller_name()))
    pbs.logmsg(pbs.EVENT_DEBUG3, "Request: %s" % (request))
    ncpus = int(request[1])
    if ncpus > int(cluster['ncpus']):
        line = "\nError: You requested ncpus=%d in a chunk. This " % ncpus + \
               "is more than is available on a %s " % cluster['name'] + \
               "compute node (%d).\n" % (int(cluster['ncpus']))
        line += "For example, on %s, use -l " % cluster['name'] + \
                "select=2:ncpus=%s:mpiprocs=%s " % \
                (cluster['ncpus'], cluster['ncpus']) + \
                "to use %d cores\n" % \
                (2 * int(cluster['ncpus']))
        line += "If you still have questions, refer to %s.\n" % \
                cluster['site_info']
        pbs.logmsg(pbs.EVENT_DEBUG3, "line: %s" % (line))
        e_reject(line)
    return ncpus


def job_mem_check(e, request, s, cluster):
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Method called" % (caller_name()))
    pbs.logmsg(pbs.EVENT_DEBUG3, "Request: %s" % (request))
    mem = pbs.size(request[1])

    clust_mem = pbs.size(str(cluster['mem']))
    pbs.logmsg(pbs.EVENT_DEBUG3, "Cluster mem: %s" % (clust_mem))
    if mem > clust_mem:
        pbs.logmsg(pbs.EVENT_DEBUG3, "I am here")
        line = "\nError: You requested %s of memory per " % mem + \
               "node. This exceeds the available memory " + \
               "on a %s node (%s)\n" % (cluster['name'], clust_mem)
        pbs.logmsg(pbs.EVENT_DEBUG3, "line: %s" % (line))
        line += "For example on %s use -l " % cluster['name'] + \
                "select=2:ncpus=%s:mpiprocs=%s:mem=%s" % \
                (cluster['ncpus'], cluster['ncpus'], clust_mem)
        pbs.logmsg(pbs.EVENT_DEBUG3, "line: %s" % (line))
        line += "\nIf you still have questions, " + \
```

```
                "please refer to %s" % \
                cluster['site_info']
            e_reject(line)
        pbs.logmsg(pbs.EVENT_DEBUG3, "Return mem value: %s" % (mem))
        return mem


    def job_size_mem(mem, ncpus, R, cluster):
        pbs.logmsg(pbs.EVENT_DEBUG3, "Check placement: %s" %
                   (repr(R['place'])))
        if repr(R['place']).find('excl') == -1:
            pbs.logmsg(pbs.EVENT_DEBUG3, "Set mem for non excl job")
            mem_line = 'mem=%s' % pbs.size(convert_size(ncpus * size_as_int(
                cluster['default_mem_per_core']), "mb"))
        else:
            pbs.logmsg(pbs.EVENT_DEBUG3, "Set mem for excl job")
            mem_line = 'mem=%s' % pbs.size(convert_size(
                cluster['ncpus'] * size_as_int(
                    cluster['default_mem_per_core']), "mb"))
        pbs.logmsg(pbs.EVENT_DEBUG3, "mem_line: %s" % mem_line)

        return mem_line


    def job_requested_queue(j, s, cluster):
        pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Method called" % (caller_name()))
        # Check to see if the queue has been specified, if not specify the default
        # as defined in the config file.
        if hasattr(j.queue, 'name'):
            pbs.logmsg(pbs.EVENT_DEBUG3, "job queue: %s" %
                       j.queue.name)
        else:
            pbs.logmsg(pbs.EVENT_DEBUG3, "Set job queue to : %s" %
                       cluster['default_queue'])
            j.queue = s.queue("%s" % cluster['default_queue'])
            pbs.logmsg(pbs.EVENT_DEBUG3, "job queue: %s" %
                       j.queue.name)
        pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Leaving" % (caller_name()))


    def job_select_cores_only(sel, s, cluster):

        # Initialize local variables
        R = pbs.event().job.Resource_List
        tmp_select = list()
```

```
try:
    tot_ncpus = int(sel)
    tot_mpiprocs = int(sel)
    pbs.logmsg(pbs.EVENT_DEBUG3, "tot_ncpus: %s" % tot_ncpus)
    pbs.logmsg(pbs.EVENT_DEBUG3, "Mem/Core: %s" %
               cluster['default_mem_per_core'])
    tot_mem = pbs.size(convert_size(tot_ncpus *
                                    size_as_int(cluster['default_mem_per_core']), "mb"))

    pbs.logmsg(pbs.EVENT_DEBUG3, "tot_ncpus: %d\ttot_mem: %s" %
               (tot_ncpus, tot_mem))

    # Check to see if tot_ncpus > total ncpus on cluster
    if tot_ncpus > cluster['total_cpus']:
        reject_job("total", "ncpus", tot_ncpus, cluster['total_cpus'],
                   s.name, cluster)

    cores_per_node = int(cluster['ncpus'])
    full_nodes = int(tot_ncpus / cores_per_node)
    remaining_cores = tot_ncpus % cores_per_node

    pbs.logmsg(pbs.EVENT_DEBUG3, "tot_ncpus: %d\tfull_nodes: %d\t" %
               (tot_ncpus, full_nodes) +
               "remaining_cores: %d\ttot_mem: %s" %
               (remaining_cores, tot_mem))

    if 'resize_select' in cluster and cluster['resize_select']:
        pbs.logmsg(pbs.EVENT_DEBUG3, "Resizing select statement")
        if full_nodes == 0:
            tmp_select.append("1:ncpus=%d:mpiprocs=%d:mem=%s" %
                              (remaining_cores, remaining_cores, pbs.size(
                               convert_size(remaining_cores * size_as_int(
                                   cluster['default_mem_per_core']), "mb"))))
        elif remaining_cores == 0:
            pbs.logmsg(pbs.EVENT_DEBUG3, "Remaining cores:  0")
            pbs.logmsg(pbs.EVENT_DEBUG3, "Cluster:  %s" % cluster)
            tmp_select.append("%d:ncpus=%d:mpiprocs=%d:mem=%s" %
                              (full_nodes, cores_per_node, cores_per_node,
                               pbs.size(convert_size(
                                   int(cores_per_node) * size_as_int(
                                       cluster['default_mem_per_core']), "mb"))))
            pbs.logmsg(pbs.EVENT_DEBUG3, "tmp_select:  %s" % tmp_select)
        else:
            tmp_select.append("%d:ncpus=%d:mpiprocs=%d:mem=%s" %
                              (full_nodes, cores_per_node, cores_per_node,
                               pbs.size(convert_size(
```

```
                                    cores_per_node * size_as_int(
                                        cluster['default_mem_per_core']), "mb"))))
                tmp_select.append("1:ncpus=%d:mpiprocs=%d:mem=%s" %
                                  (remaining_cores, remaining_cores,
                                   pbs.size(convert_size(
                                    remaining_cores * size_as_int(
                                        cluster['default_mem_per_core']), "mb"))))

            # Replace the old select statement with the new select statement
            R['select'] = pbs.select(join(tmp_select, '+'))
            pbs.logmsg(pbs.EVENT_DEBUG3, "New select Line: %s" % R['select'])
            pbs.logmsg(pbs.EVENT_DEBUG3, "Server: %s" % s.name)
            pbs.logmsg(pbs.EVENT_DEBUG3, "tot_ncpus: %d,\tcluster cores: %s" %
                       (tot_ncpus, int(cluster['ncpus'])))
            return True
    except ValueError:
        return False


def job_requested_resources(e, j, s, cluster, cfg):
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Method called" % (caller_name()))

    R = j.Resource_List
    sel = repr(R["select"])

    if R["select"] == None:
        sel = '1'
        pbs.logmsg(pbs.LOG_WARNING, "%s: Requested Resources: %s" %
                   (caller_name(), R))
        if 'accept_empty_select' in cfg['clusters'][s.name]:
            if cfg['clusters'][s.name]['accept_empty_select']:
                return False

    # Calculate the ncpus and memory requested by this job
    tot_ncpus = 0
    tot_mpiprocs = 0
    tot_mem = pbs.size("0kb")
    tot_ngpus = 0
    tot_nmics = 0

    # Initialize a tmp select list
    tmp_select = list()

    pbs.logmsg(pbs.EVENT_DEBUG3, "Select Line: %s" % sel)

    # Check to see if the users just selected ncpus verses a chunk
```

```
# i.e select=32 vs select=1:ncpus=32:mpiprocs=32:mem=32gb

status = job_select_cores_only(sel, s, cluster)
pbs.logmsg(pbs.EVENT_DEBUG3,
           "job_select_cores_only Status: %s" % status)

if not status:
    pbs.logmsg(pbs.EVENT_DEBUG3, "Eval select Line: %s" % R['select'])
    for chunk in sel.split("+"):
        nchunks = 1

        tmp_chunk = chunk.split(":")
        mpiprocs = -1
        ncpus = -1
        mem = 1900

        for c in tmp_chunk:
            pbs.logmsg(pbs.EVENT_DEBUG3, "Chunk: %s" % c)
            kv = c.split("=")
            if len(kv) == 1:
                nchunks = kv[0]
            elif kv[0] == "ncpus":
                pbs.logmsg(pbs.EVENT_DEBUG3, "ncpus: %s" % kv)
                ncpus = chunk_resource_check(e, kv, cluster)
                tot_ncpus += int(nchunks) * ncpus
            elif kv[0] == "ngpus":
                pbs.logmsg(pbs.EVENT_DEBUG3, "ngpus: %s" % kv)
                ngpus = chunk_resource_check(e, kv, cluster)
                tot_ngpus += int(nchunks) * ngpus

            elif kv[0] == "nmics":
                pbs.logmsg(pbs.EVENT_DEBUG3, "nmics: %s" % kv)
                nmics = chunk_resource_check(e, kv, cluster)
                tot_nmics += int(nchunks) * nmics

            elif kv[0] == "mpiprocs":
                pbs.logmsg(pbs.EVENT_DEBUG3, "mpiprocs: %s" % kv)
                mpiprocs = chunk_resource_check(e, kv, cluster)
                tot_mpiprocs += int(nchunks) * mpiprocs

            elif kv[0] == "mem":
                pbs.logmsg(pbs.EVENT_DEBUG3, "mem: %s" % kv)
                mem = chunk_resource_check(e, kv, cluster)
                pbs.logmsg(pbs.EVENT_DEBUG3, "mem: %s" % mem)
                pbs.logmsg(pbs.EVENT_DEBUG3, "nchunks: %d" % int(nchunks))
                pbs.logmsg(pbs.EVENT_DEBUG3, "mem (b): %d" %
```

```
                                size_as_int(str(mem)))
                        mem = pbs.size(convert_size(
                                    int(nchunks) * size_as_int(mem), "mb"))
                        pbs.logmsg(pbs.EVENT_DEBUG3, "mem: %s" % mem)
                        tot_mem = tot_mem + mem
                        pbs.logmsg(pbs.EVENT_DEBUG3, "Total mem: %s" % tot_mem)

                # Set up the ncpus, mpiprocs, and mem if not set by the user
                if ncpus == -1:
                    ncpus = 1
                    tmp_chunk.append('ncpus=%d' % ncpus)
                if mpiprocs == -1:
                    tmp_chunk.append('mpiprocs=%d' % ncpus)
                if mpiprocs > ncpus:
                    e_reject("You cannot specify more mpiprocs than ncpus\n" +
                             "You specified: %s" % sel)
                if mem == 1900 and cluster['assign_mem_per_core']:
                    tmp_chunk.append(job_size_mem(mem, ncpus, R, cluster))

                tmp_select.append(join(tmp_chunk, ':'))
                R['select'] = pbs.select(join(tmp_select, '+'))

            pbs.logmsg(pbs.EVENT_DEBUG3, "Check tot_ncpus: %d" %
                       (tot_ncpus))

            # Check to see if tot_ncpus > total ncpus on cluster
            if tot_ncpus > cluster['total_cpus']:
                reject_job("total", "ncpus", tot_ncpus, cluster['total_cpus'],
                           s.name, cluster)
            if pbs.size(tot_mem) > pbs.size(cluster['total_mem']):
                tot_mem = pbs.size(convert_size(tot_mem, "gb"))
                reject_job("total", "mem", tot_mem, cluster['total_mem'],
                           s.name, cluster)

        pbs.logmsg(pbs.EVENT_DEBUG3, "Return resource totals")
        return {'tncpus': tot_ncpus, 'tmem': tot_mem, 'tmpiprocs': tot_mpiprocs,
                'tnmics': tot_nmics, 'tngpus': tot_ngpus}


    def reject_job(ltype, lres, lrequest, rlimit, lname, cluster):
        pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Method called" % (caller_name()))

        line = "\nInvalid job request.\n"
        if ltype == "total":
            line += "Job requested %s=%s and the total available for %s " % \
                    (lres, lrequest, lname) + \
```

```
            "is %s=%s\n" % (lres, rlimit)

    elif (lres == "ncpus" or lres == "mem" or lres == "nmics" or
          lres == "ngpus"):
        if lname is not "":
            line += "Job requested %s=%s and the %s limit for the %s %s " % \
                    (lres, lrequest, ltype, lname, ltype) + \
                    "is %s=%s\n" % (lres, rlimit)
        else:
            line += "Job requested %s=%s and the limit for the %s is " % \
                    (lres, lrequest, ltype) + "%s=%s\n" % (lres, rlimit)
    elif lres == "walltime":
        pbs.logmsg(pbs.EVENT_DEBUG3, "%s: resource limit %s" %
                   (caller_name(), lres))
        if ltype == "None" or ltype is None:
            line += "Job has not requested a walltime \nPlease add a " + \
                    "walltime and resubmit.\n"
            line += "For example: To request 24 hours add this to the " + \
                    "submission line -lwalltime=24:00:00\n"
        if ltype == "max":
            if lname is not "":
                line += "Job has requested %s walltime which " % lrequest + \
                        "exceeds the %s %s walltime limit of %s\n" % \
                        (lname, ltype, rlimit)
            else:
                line += "Job has requested %s walltime which " % lrequest + \
                        "exceeds the %s walltime limit of %s\n" % \
                        (ltype, rlimit)
            line += "Please change the walltime or queue (depending on " + \
                    "the violated walltime limits) and resubmit.\n"
        if ltype == "min":
            if lname is not "":
                line += "Job requested %s walltime which is " % lrequest + \
                        "less than the %s %s walltime limit of %s\n" % \
                        (lname, ltype, rlimit)
            else:
                line += "Job requested %s walltime which is " % lrequest + \
                        "less than the %s walltime limit of %s\n" % \
                        (ltype, rlimit)
            line += "Please change the walltime or queue (depending on " + \
                    "the violated walltime limits) and resubmit.\n"
    else:
        pbs.logmsg(pbs.EVENT_DEBUG3, "Unknown resource: %s" % (lres))

    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: line %s" %
               (caller_name(), line))
```

```
        line += "If you believe that this is a valid " + \
                "job request, please contact the HPC staff\n"
        line += "For more information, please refer to %s\n" % \
                cluster['site_info']
        pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Ready to reject job" % (caller_name()))
        e_reject(line)


    def check_ncpus_limits(job_res, j, s, cluster):
        pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Method called" % (caller_name()))
        ncpus_max_qlim = s.queue(j.queue.name).resources_max['ncpus']
        ncpus_min_qlim = s.queue(j.queue.name).resources_min['ncpus']
        ncpus_max_slim = s.resources_max['ncpus']

        pbs.logmsg(pbs.EVENT_DEBUG3, "ncpus_max_qlim: %s" %
                   (ncpus_max_qlim))
        pbs.logmsg(pbs.EVENT_DEBUG3, "ncpus_max_slim: %s" %
                   (ncpus_max_slim))

        R = j.Resource_List
        pbs.logmsg(pbs.EVENT_DEBUG3, "Job Resource List: %s" % (R))
        if R["ncpus"] != None:
            ncpus_req = R["ncpus"]
        else:
            ncpus_req = job_res['tncpus']
        pbs.logmsg(pbs.EVENT_DEBUG3, "Required ncpus: %s" % (ncpus_req))

        pbs.logmsg(pbs.EVENT_DEBUG3,
                   "Above Find the PBS_GENERIC ncpus limit")

        # Find the PBS_GENERIC ncpus limit
        pbs.logmsg(pbs.EVENT_DEBUG3, "Above ncpus checks:")
        # Check to see if requested ncpus does not violate the limits
        if ((ncpus_max_qlim is not None) and
                (int(ncpus_req) > int(ncpus_max_qlim))):
            pbs.logmsg(pbs.EVENT_DEBUG3, "ncpus_max_qlim:")
            reject_job("queue", "ncpus", ncpus_req, ncpus_max_qlim,
                       j.queue.name, cluster)
        elif ((ncpus_min_qlim is not None) and
              (int(ncpus_req) < int(ncpus_min_qlim))):
            pbs.logmsg(pbs.EVENT_DEBUG3, "ncpus_min_qlim:")
            reject_job("queue", "ncpus", ncpus_req, ncpus_min_qlim,
                       j.queue.name, cluster)
        elif ((ncpus_max_slim is not None) and
              (int(ncpus_req) > int(ncpus_max_slim))):
            pbs.logmsg(pbs.EVENT_DEBUG3, "ncpus_max_slim:")
```

```
            reject_job("server", "ncpus", ncpus_req, ncpus_max_slim,
                       "", cluster)

        pbs.logmsg(pbs.EVENT_DEBUG3, "Done with ncpus checks")


    def check_mem_limits(job_res, j, s, cluster):
        pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Method called" % (caller_name()))

        # Mem limit checking section
        pbs.logmsg(pbs.EVENT_DEBUG3, "Looking at mem limits")
        mem_lim = s.queue(j.queue.name).max_run_res['mem']
        pbs.logmsg(pbs.EVENT_DEBUG3,
                   "PBS_GENERIC mem limit: %s" % (mem_lim))
        mem_max_qlim = s.queue(j.queue.name).resources_max['mem']
        pbs.logmsg(pbs.EVENT_DEBUG3, "mem_max_qlim: %s" % (mem_max_qlim))

        # Get the requested memory
        R = j.Resource_List
        if R["mem"] != None:
            mem_req = R["mem"]
        else:
            mem_req = job_res['tmem']

        pbs.logmsg(pbs.EVENT_DEBUG3, "Required mem: %s" % (mem_req))

        # Find the PBS_GENERIC mem limit
        if mem_lim is not None:
            tmp_mem_lim = mem_lim.split(',')
            pbs.logmsg(pbs.EVENT_DEBUG3, "Above tmp_mem_lim: %s" %
                       (tmp_mem_lim))
            mem_lim = -1
            for limit in tmp_mem_lim:
                if "PBS_GENERIC=" in limit:
                    mem_lim = pbs.size(limit.split('=')[1].replace(']',
                                                                   ''))
            else:
                mem_lim = -1

        # Check to see if requested mem does not violate the limits
        pbs.logmsg(pbs.EVENT_DEBUG3, "mem_lim: %s" % (mem_lim))
        pbs.logmsg(pbs.EVENT_DEBUG3, "mem_req: %s" % (mem_req))
        if mem_lim != -1 and pbs.size(mem_req) > mem_lim:
            pbs.logmsg(pbs.EVENT_DEBUG3, "mem_req > mem_lim")
            reject_job("user", "mem", mem_req, mem_lim,
                       j.queue.name, cfg['clusters'][s.name])
```

```
    elif mem_max_qlim is not None and pbs.size(mem_req) > mem_max_qlim:
        pbs.logmsg(pbs.EVENT_DEBUG3, "mem_req > mem_max_lim")
        reject_job("queue", "mem", mem_req, mem_max_qlim,
                   j.queue.name, cfg['clusters'][s.name])


def check_walltime(j, s, cluster):
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Method called" % (caller_name()))
    R = j.Resource_List
    if R["walltime"] != None:
        wt_req = R["walltime"]
    else:
        wt_req = None
    pbs.logmsg(pbs.EVENT_DEBUG3, "Requested walltime: %s" %
               (wt_req))

    # Get the walltime limits
    limit_name = j.queue.name
    wt_max = s.queue(limit_name).resources_max['walltime']
    wt_min = s.queue(limit_name).resources_min['walltime']
    pbs.logmsg(pbs.EVENT_DEBUG3, "Wall Limit: %s" % (wt_max))
    pbs.logmsg(pbs.EVENT_DEBUG3, "Wall Requested: %s" % (wt_req))
    if wt_max is not None and wt_req > wt_max:
        pbs.logmsg(pbs.EVENT_DEBUG3, "This job should exit: %s" %
                   (wt_req))
    pbs.logmsg(pbs.EVENT_DEBUG3, "Check the walltimes")

    if wt_max is None:
        # Check to see if it is set at the server level
        wt_max = s.resources_max['walltime']
        if wt_max is not None:
            limit_name = "server"

    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Method called" % (caller_name()))
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: req: %s, max: %s, min: %s, name: %s" %
               (caller_name(), wt_req, wt_max, wt_min, limit_name))
    if wt_req is None:
        if 'require_walltime' in cluster:
            if cluster['require_walltime']:
                reject_job("None", "walltime", "", "", limit_name, cluster)
    elif wt_max is not None and wt_req > wt_max:
        reject_job("max", "walltime", wt_req, wt_max,
                   limit_name, cluster)
    elif wt_min is not None and wt_req < wt_min:
        reject_job("min", "walltime", wt_req, wt_min,
                   limit_name, cluster)
```

```
        else:
            return True

        return False


    def main():
        pbs.logmsg(pbs.EVENT_DEBUG3, "Entering check limits hook")

        e = pbs.event()
        j = e.job
        s = pbs.server()
        who = e.requestor

        # Read in the config file
        cfg = parse_config_file(e, s)
        cluster = cfg['clusters'][s.name]

        # Check to see if we are running in test mode
        if 'test_mode' in cfg['clusters'][s.name]:
            if cfg['clusters'][s.name]['test_mode']:
                pbs.logmsg(pbs.EVENT_DEBUG3, "Entering check user")
                pbs.logmsg(pbs.EVENT_DEBUG3, "cfg: %s" % cfg['clusters'][s.name])
                if 'test_users' in cfg['clusters'][s.name]:
                    pbs.logmsg(pbs.EVENT_DEBUG3, "check user: %s" % who)
                    if who not in cfg['clusters'][s.name]['test_users']:
                        pbs.logmsg(pbs.EVENT_DEBUG3,
                                   "User %s not in test users %s" %
                                   (who, cfg['clusters'][s.name]['test_users']))
                        e.accept()
                    else:
                        pbs.logmsg(pbs.EVENT_DEBUG3,
                                   "Running hook for User %s" % who)

        # Collect the job requested resources
        pbs.logmsg(pbs.EVENT_DEBUG3, "Ready to look at job requested resources")
        job_res = job_requested_resources(e, j, s, cluster, cfg)
        pbs.logmsg(pbs.EVENT_DEBUG3, "Returned totals: %s" % job_res)

        try:
            pbs.logmsg(pbs.EVENT_DEBUG3, "Default Queue: %s" %
                       cluster['default_queue'])
            job_requested_queue(j, s, cluster)
            pbs.logmsg(pbs.EVENT_DEBUG3, "job queue %s" % j.queue.name)

            # Find the server limits
```

```
            q = s.queue(j.queue.name)
            pbs.logmsg(pbs.EVENT_DEBUG3, "queue %s" % q.name)
            for key in q.attributes.keys():
                exec "a=q.%s" % key

            # check server/queue limits section
            pbs.logmsg(pbs.EVENT_DEBUG3, "Ready to check queue/server limits")
            if job_res is not False:
                if cfg['clusters'][s.name]['check_ncpus']:
                    check_ncpus_limits(job_res, j, s, cluster)

                if cfg['clusters'][s.name]['check_mem']:
                    check_mem_limits(job_res, j, s, cluster)

            if cfg['clusters'][s.name]['check_walltime']:
                check_walltime(j, s, cluster)

        except:
            err = sys.exc_info()[0]
            pbs.logmsg(pbs.EVENT_DEBUG3, "This job had an exception: %s" % err)
            pass

    if __name__ == 'builtins':
        try:
            pbs.logmsg(pbs.EVENT_DEBUG, "Entering the main loop")
            main()

        except SystemExit:
            pass
        except AdminError, exc:
            pbs.logmsg(pbs.EVENT_DEBUG3, "Encountered Admin Error")
            # Something on the system is misconfigured
            pbs.logmsg(pbs.EVENT_DEBUG3,
                       str(traceback.format_exc().strip().splitlines()))
            msg = ("Admin error in %s handling %s event" %
                   (e.hook_name, "queuejob"))
            pbs.logmsg(pbs.EVENT_ERROR, msg)
            e_reject(msg)
        except:
            e_reject("%s hook failed with %s.\nPlease contact your sys admin " +
                     "if this problem persists for more than 10 minutes" %
                     (e.hook_name, sys.exc_info()[:2]))
```

9.3 modifyjob Hook Examples

Example 9-11: Prevent users from using *qalter* to change their jobs

Hook type: modifyjob

Allow only administrators to change jobs.

Script NoAlter.py, on Windows, in a domain:

```
import os
import pbs

e = pbs.event()
j = e.job
who = e.requestor
pbs.logmsg(pbs.LOG_DEBUG, "requestor=%s" % (who,))
isadmin=0
admin_ulist = ["PBS_Server", "Scheduler", "pbs_mom", "Administrator"]

if who in admin_ulist:
    isadmin=1
else:
    cmd = "net user "  + who + " /domain"
    admin_glist = ['Administrators', 'Domain Admins', 'Enterprise
        Admins']
    for line in os.popen(cmd).readlines():
        if line.find("Group") >= 0:
            for li in line.split("*"):
                if li.strip() in admin_glist:
                    isadmin=1
                    break
if e.type == pbs.MODIFYJOB and not isadmin:
    e.reject("Normal users are not allowed to modify their jobs")
```

Script NoAlter.py, on Linux:

```
import pbs
e = pbs.event()
j = e.job
who = e.requestor
pbs.logmsg(pbs.LOG_DEBUG, "requestor=%s" % (who,))
admin_ulist = ["PBS_Server", "Scheduler", "pbs_mom", "root"]
if who not in admin_ulist:
    e.reject("Normal users are not allowed to modify their jobs")
```

Create hook and import script:

```
qmgr -c 'create hook NoAlter event="modifyjob"'
qmgr -c 'import hook NoAlter application/x-python default NoAlter.py'
```

Example 9-12: *Reject jobs requesting a specific queue that do not request mem*

Hook type: modifyjob

Reject jobs requesting workq2 if they don't also request memory.

Script queuespec.py:

```
import pbs
import sys
try:
    e = pbs.event()
    j = e.job
    if j.queue.name == "workq2" and not j.Resource_List["mem"]:
        e.reject("workq2 requires job to have mem specification")

except SystemExit:
    pass

except:
    e.reject("%s hook failed with %s. Please contact
        Admin" % (e.hook_name, sys.exc_info()[:2]))
```

Create hook, import script:

```
qmgr -c 'create hook queuespec event="modifyjob"'
qmgr -c 'import hook queuespec application/x-python default queuespec.py'
```

9.4 periodic Hook Examples

Example 9-13: Run job start time estimator

Hook type: periodic

Run job start time estimator named pbs_est.

Script run_pbs_est.py:

```
import pbs
import time
import os
import subprocess

pbs_est_cmd = os.path.join(pbs.pbs_conf['PBS_EXEC'], 'sbin', 'pbs_est')

e = pbs.event()

pbs.logmsg(pbs.LOG_DEBUG, "Starting job start time estimation task")

exit_stat = subprocess.call([pbs_est_cmd], shell=True)
if exit_stat != 0:
    e.reject("%s exited abnormally with return code %d" % (pbs_est_cmd, exit_stat))
else:
    e.accept()
```

9.5 execjob_launch Hook Examples

Example 9-14: Modify arguments to job program

Hook type: execjob_launch

The argv[] entries can be modified to change the existing arguments to progname.

Given the following hook:

```
# cat launch.py
import pbs
e = pbs.event()
e.argv[1] = "cool"

# qmgr -c "create hook launch event=execjob_launch"
# qmgr -c "import hook launch application/x-python default launch.py"
```

So if a job is submitted as follows:

```
% qsub -- /bin/echo uncool
```

When the job is submitted, progname = "/*bin*/*echo*", argv[0] = "/*bin*/*echo*", argv[1]="*uncool*". However, when the job executes, the execjob_launch hook runs, causing "/bin/echo cool" to execute instead of "/bin/echo uncool".

9.6 execjob_prologue and execjob_epilogue Hook Examples

Example 9-15: Run shell script prologue or epilogue.

You can use this hook when the execjob_prologue and execjob_epilogue events are used in other hooks, such as the cgroups hook, and you still want to run the classic prologue and epilogue scripts we describe in section "Using Shell Scripts for Prologue and Epilogue", on page 508 in the PBS Professional Administrator's Guide. Additionally, the hook introduces parallel prologue and epilogue shell scripts.

See "Using Hooks for Prologue and Epilogue", on page 512 in the PBS Professional Administrator's Guide, for configuration and installation instructions.

This hook is included in $PBS_EXEC/unsupported. as run_pelog_shell.py, along with its configuration file, run_pelog_shell.ini.

Configuration File

Here is the contents of run_pelog_shell.ini:

```
[run_pelog_shell]
# Enable parallel prologues/epilogues that run on sister moms. Note that all
# the normal requirements apply, except the scripts should be named pprologue
# and pepilogue.
ENABLE_PARALLEL=False

# Provide verbose hook output to the user's .o/.e file
VERBOSE_USER_OUTPUT=False

# DEFAULT_ACTION can be one of DELETE or RERUN
DEFAULT_ACTION=RERUN

# Enable Torque argument compatibility
TORQUE_COMPAT=False
```

Hook Script

Here is the hook script (the contents of run_pelog_shell.py):

```
import pbs
import os, sys
import time

# Set up a few variables
start_time=time.time()
pbs_event=pbs.event()
hook_name=pbs_event.hook_name
hook_alarm=30 # default, we'll read it from the .HK later
DEBUG=False # default, we'll read it from the .HK later
job=pbs_event.job

# The trace_hook function has been written to be portable between hooks.
def trace_hook(**kwargs):
    """Simple exception trace logger for PBS hooks
    loglevel=<int> (pbs.LOG_DEBUG): log level to pass to pbs.logmsg()
    reject=True: reject the job upon completion of logging trace
    trace_in_reject=<bool> (False): pass trace to pbs.event().reject()
    trace_in_reject=<str>: message to pass to pbs.event().reject() with trace
    Usage:
    try:
        your=code(here)
    except:
        trace_hook()
    """
    import sys

    if 'loglevel' in kwargs:
        loglevel=kwargs['loglevel']
    else:
        loglevel=pbs.LOG_ERROR
    if 'reject' in kwargs:
        reject=kwargs['reject']
    else:
        reject=True
    if 'trace_in_reject' in kwargs:
        trace_in_reject=kwargs['trace_in_reject']
    else:
        trace_in_reject=False

    # Associate hook events with the appropriate PBS constant. This is a list
    # of all hook events as of PBS Pro 13.0. If the event does not exist, it is
    # removed from the list.
    hook_events=['queuejob', 'modifyjob', 'movejob', 'runjob', 'execjob_begin',
```

```
            'execjob_prologue', 'execjob_launch', 'execjob_attach',
            'execjob_preterm', 'execjob_epilogue', 'execjob_end',
            'resvsub', 'resv_end', 'provision', 'exechost_periodic',
            'exechost_startup', 'periodic']

hook_event={}
for he in hook_events:
    # Only set available hooks for the current version of PBS.
    if hasattr(pbs, he.upper()):
        event_code=eval('pbs.'+he.upper())
        hook_event[event_code]=he
        hook_event[he]=event_code
        hook_event[he.upper()]=event_code
        del event_code
    else:
        del hook_events[hook_events.index(he)]

trace={
    'line':      sys.exc_info()[2].tb_lineno,
    'module':    sys.exc_info()[2].tb_frame.f_code.co_name,
    'exception': sys.exc_info()[0].__name__,
    'message':   sys.exc_info()[1].message,
}
tracemsg='%s hook %s encountered an exception: Line %s in %s %s: %s' %(
    hook_event[pbs.event().type], pbs.event().hook_name,
    trace['line'], trace['module'], trace['exception'], trace['message']
)
rejectmsg="Hook Error: request rejected as filter hook '%s' encountered " \
    "an exception. Please inform Admin" % pbs.event().hook_name
if not isinstance(loglevel, int):
    pbs.logmsg(pbs.LOG_ERROR, 'trace_hook() called with invalid argument' \
        ' (loglevel=%s), setting to pbs.LOG_ERROR. ' % repr(loglevel))
    loglevel=pbs.LOG_ERROR

pbs.logmsg(loglevel, tracemsg)

if reject:
    tracemsg+=', request rejected'
    if isinstance(trace_in_reject, bool):
        if trace_in_reject:
            pbs.event().reject(tracemsg)
        else:
            pbs.event().reject(rejectmsg)
    else:
        pbs.event().reject(str(trace_in_reject)+'Line %s in %s %s:\n%s' % (
            trace['line'],trace['module'],trace['exception'],
```

```
                trace['message'] ))

class JobLog:
    """ Class for managing output to job stdout and stderr."""

    def __init__(self):
        PBS_SPOOL=os.path.join(pbs_conf()['PBS_MOM_HOME'], 'spool')
        self.stdout_log=os.path.join(PBS_SPOOL,
                                     '%s.OU' % str(pbs.event().job.id))
        self.stderr_log=os.path.join(PBS_SPOOL,
                                     '%s.ER' % str(pbs.event().job.id))

        if str(pbs.event().job.Join_Path) == 'oe':
            self.stderr_log=self.stdout_log
        elif str(pbs.event().job.Join_Path) == 'eo':
            self.stdout_log=self.stderr_log

    def stdout(self, msg):
        """Write msg to appropriate file handle for stdout"""
        import sys

        try:
            if not pbs.event().job.interactive and pbs.event().job.in_ms_mom:
                logfile=open(self.stdout_log, 'ab+')
            else:
                logfile=sys.stdout

            if DEBUG:
                pbs.logmsg(pbs.EVENT_DEBUG3,
                    '%s;%s;[DEBUG3]: writing %s to %s' %
                        (pbs.event().hook_name,
                         pbs.event().job.id,
                         repr(msg),
                         logfile.name))

            logfile.write(msg)
            logfile.flush()
            logfile.close()
        except IOError:
            trace_hook()

    def stderr(self, msg):
        """Write msg to appropriate file handle for stdout"""
        import sys

        try:
```

```
            if not pbs.event().job.interactive and pbs.event().job.in_ms_mom():
                logfile=open(self.stderr_log, 'ab+')
            else:
                logfile=sys.stderr

            if DEBUG:
                pbs.logmsg(pbs.EVENT_DEBUG3,
                    '%s;%s;[DEBUG3]: writing %s to %s' %
                        (pbs.event().hook_name,
                         pbs.event().job.id,
                         repr(msg),
                         logfile.name))

            logfile.write(msg)
            logfile.flush()
            logfile.close()
        except IOError:
            trace_hook()


# Read in pbs.conf
def pbs_conf(pbs_key=None):
    """Function to return the values from /etc/pbs.conf
    If the PBS python interpreter hasn't been recycled, it is not necessary
    to re-read and re-parse /etc/pbs.conf. This function will simply return
    the variable that exists from the first time this function ran.
    Creates a dict containing the key/value pairs in pbs.conf, accounting for
    comments in lines and empty lines.
    Returns a string representing the pbs.conf setting for pbs_key if set, or
    the dict of all pbs.conf settings if pbs_key is not set.
    """
    import os

    if hasattr(pbs_conf, 'pbs_keys'):
        return pbs_conf.pbs_keys[pbs_key] if pbs_key else pbs_conf.pbs_keys

    if 'PBS_CONF_FILE' in os.environ.keys():
        pbs_conf_file=os.environ['PBS_CONF_FILE']
    elif sys.platform == 'win32':
        if 'ProgramFiles(x86)' in os.environ.keys():
            program_files=os.environ['ProgramFiles(x86)']
        else:
            program_files=os.environ['ProgramFiles']
        pbs_conf_file='%s\\PBS Pro\\pbs.conf' % program_files
    else:
        pbs_conf_file='/etc/pbs.conf'
```

```
    pbs_conf.pbs_keys=dict([line.split('#')[0].strip().split('=') \
        for line in open(pbs_conf_file) \
        if not line.startswith('#') and '=' in line])

    if 'PBS_MOM_HOME' not in pbs_conf.pbs_keys.keys():
        pbs_conf.pbs_keys['PBS_MOM_HOME'] = \
            pbs_conf.pbs_keys['PBS_HOME']

    return pbs_conf.pbs_keys[pbs_key] if pbs_key else pbs_conf.pbs_keys

# Primary hook execution begins here
try:

    def rejectjob(reason, action=DEFAULT_ACTION):
        """Log job rejection and then call pbs.event().reject()"""

        # Arguments to pbs.event().reject() do nothing in execjob events. Log a
        # warning instead, update the job comment, then reject the job.
        if action == RERUN:
            job.rerun()
            reason='Requeued - %s' % reason
        elif action == DELETE:
            job.delete()
            reason='Deleted - %s' % reason
        else:
            reason='Rejected - %s' % reason

        job.comment='%s: %s' % (hook_name, reason)
        pbs.logmsg(pbs.LOG_WARNING, ';'.join([hook_name, job.id, reason]))
        pbs.logjobmsg(job.id, reason) # Add a message that can be tracejob'd
        if VERBOSE_USER_OUTPUT:
            print reason
        pbs_event.reject()

    # For the path to mom_priv, we use PBS_MOM_HOME in case that is set,
    # pbs_conf() will return PBS_HOME if it is not.
    mom_priv=os.path.abspath(os.path.join(
        pbs_conf()['PBS_MOM_HOME'],'mom_priv'))

    # Get the hook alarm time from the .HK file if it exists.
    hk_file=os.path.join(mom_priv,'hooks','%s.HK' % hook_name)
    if os.path.exists(hk_file):
        hook_settings=dict([l.strip().split('=') for l in
                            open(hk_file,'r').readlines()])
        if 'alarm' in hook_settings.keys():
```

```
        hook_alarm=int(hook_settings['alarm'])
    if 'debug' in hook_settings.keys():
        DEBUG=True if hook_settings['debug']=='true' else False

if DEBUG:
    pbs.logmsg(pbs.LOG_DEBUG, '%s;%s;[DEBUG] starting.' %
               (hook_name, job.id))

if 'PBS_HOOK_CONFIG_FILE' in os.environ:
    config_file = os.environ["PBS_HOOK_CONFIG_FILE"]
    config=dict([l.split('#')[0].strip().split('=')
                 for l in open(config_file,'r').readlines() if '=' in l])

    # Set the true/false configurations
    if 'ENABLE_PARALLEL' in config.keys():
        ENABLE_PARALLEL=config['ENABLE_PARALLEL'].lower()[0] in ['t', '1']
    if 'VERBOSE_USER_OUTPUT' in config.keys():
        VEROSE_USER_OUTPUT=config['VERBOSE_USER_OUTPUT'].lower()[0] in ['t', '1']
    if 'DEFAULT_ACTION' in config.keys():
        if config['DEFAULT_ACTION'].upper() == 'DELETE':
            DEFAULT_ACTION=DELETE
        elif config['DEFAULT_ACTION'].upper() == 'RERUN':
            DEFAULT_ACTION=RERUN
        else:
            pbs.logmsg(pbs.LOG_WARN,
                '%s;%s;[ERROR] ' % (hook_name, job.id) + \
                'DEFAULT_ACTION in %s.ini must be one ' % (hook_name) + \
                'of DELETE or RERUN.')
    if 'TORQUE_COMPAT' in config.keys():
        TORQUE_COMPAT=config['TORQUE_COMPAT'].lower()[0] in ['t', '1']

# Skip sister mom if parallel pelogs aren't enabled.
if not ENABLE_PARALLEL and not job.in_ms_mom():
    pbs_event.accept()

# Prologues and epilogues have different arguments
if pbs_event.type == pbs.EXECJOB_PROLOGUE:
    event='prologue'
    args=[
        job.id,                         # argv[1]
        job.euser,                      # argv[2]
        job.egroup                      # argv[3]
    ]
    if TORQUE_COMPAT:
        args.extend([
            job.Job_Name,               # argv[4]
```

```
            job.Resource_List,          # argv[5]
            job.queue.name,             # argv[6]
            job.Account_Name or ''      # argv[7]
        ])
elif pbs_event.type == pbs.EXECJOB_EPILOGUE:
    null='null' if not TORQUE_COMPAT else ''
    event='epilogue'
    args=[
        job.id,                         # argv[1]
        job.euser,                      # argv[2]
        job.egroup,                     # argv[3]
        job.Job_Name,                   # argv[4]
        job.session_id,                 # argv[5]
        job.Resource_List,              # argv[6]
        job.resources_used,             # argv[7]
        job.queue.name,                 # argv[8]
        job.Account_Name or null,       # argv[9]
        job.Exit_status                 # argv[10]
    ]
else: # hook has wrong events added
    pbs.logmsg(pbs.LOG_WARNING,
     '%s;%s;[ERROR] PBS event type %s not supported in this hook.' %
      (hook_name, job.id, pbs_event.type))
    pbs_event.accept()

# Handle empty arguments
args=[str(a) if ( a or a == 0 ) else '' for a in args]

if DEBUG: pbs.logmsg(pbs.LOG_DEBUG,
    '%s;%s;[DEBUG] %s event triggered.' % \
    (hook_name, job.id, event))

if DEBUG:
    pbs.logmsg(pbs.LOG_DEBUG, '%s;%s;[DEBUG3] args=%s' % \
    (hook_name, job.id, repr(args)))

# execjob_prologue and execjob_epilogue hooks can run on all nodes, so use
# pprologue/pepilogue if available and not on primary execution node.
p='' if job.in_ms_mom() else 'p'

if DEBUG:
    pbs.logmsg(pbs.LOG_DEBUG, '%s;%s;[DEBUG] %s.' %
        (pbs_event.hook_name,
         job.id,
         'in sister mom' if p else 'in the primary execution host'))
```

```
script=os.path.join(mom_priv, p+event)

if sys.platform == 'win32':
    script=script + '.bat'

if DEBUG:
    pbs.logmsg(pbs.EVENT_DEBUG3, '%s;%s;[DEBUG3] script set to %s.' % (
        pbs_event.hook_name, job.id, script))

correct_permissions = False
if not script:
    pbs_event.accept()

if not os.path.exists(script):
    pbs_event.accept()

if sys.platform == 'win32':
    # Windows support is currently not implemented.
    pbs.logmsg(pbs.LOG_WARNING,
               '%s;%s;[ERROR] ' % (hook_name, job.id) + \
               'Classic prologues and epilogues on Windows are not ' + \
               'currently implemented in this hook.')
    pbs_event.accept()

else:
    try:
        struct_stat = os.stat(script)
    except OSError:
        rejectjob('Could not stat the %s script (%s).' %
                  (event, script), RERUN)

    # We mask for read and execute on owner make sure no one else can write
    # with 0522 (?r?x?w??w?). With this, permissions such as 0777 masked by
    # 522 will return 522. Acceptable permissions will return 500.
    correct_permissions = bool(struct_stat.st_mode & 0522 == 0500 and
                               struct_stat.st_uid == 0)

if correct_permissions:
    import signal
    import subprocess
    import shlex

    # Correction for subprocess SIGPIPE handling courtesy of Colin Watson:
    # http://www.chiark.greenend.org.uk/~cjwatson/blog/python-sigpipe.html
    def subprocess_setup():
        """subprocess_setup corrects a known bug where python installs a
```

```
    SIGPIPE handler by default. This is usually not what non-Python
    subprocesses expect"""
    signal.signal(signal.SIGPIPE, signal.SIG_DFL)

if DEBUG:
    pbs.logmsg(pbs.EVENT_DEBUG2,
        '%s;%s;[DEBUG2] script %s has appropriate permissions.' %
        (hook_name, job.id, script))

# change to the correct working directory (PBS_HOME):
os.chdir(pbs_conf()['PBS_MOM_HOME'])

# add PBS_JOBDIR environment variable, accounting for empty job.jobdir
os.environ['PBS_JOBDIR'] = job.jobdir or ''

shell=""
if sys.platform == 'win32': #win32 is _always_ cmd
    shell="cmd /c"
else:
    # check the script for the interpreter line
    shebang=open(script, 'r').readline().strip().split('#!')
    if len(shebang)==2:
        shell=shebang[1].split()[0]
        if not os.path.exists(shell):
            rejectjob(
                'Interpreter specified in %s (%s) does not exist.' %
                (p+event, shell),
                RERUN)
    else:
        rejectjob('No interpreter specified in %s.' % (p+event), RERUN)

if DEBUG:
    pbs.logmsg(pbs.EVENT_DEBUG2,
               '%s;%s;[DEBUG2] interpreter set to "%s".' %
               (hook_name, job.id, shell))

pbs.logmsg(pbs.LOG_DEBUG, '%s;%s;running %s.' %
           (hook_name, job.id, p+event))

# We perform a shlex.split to make sure we capture any #! arguments
cmd=shlex.split('%s %s' % (shell, script))
cmd.extend(args)

if DEBUG:
    pbs.logmsg(pbs.EVENT_DEBUG3,
        '%s;%s;[DEBUG3] cmd=%s' % (hook_name, job.id, repr(cmd)))
```

```
if str(job.Join_Path) in ['oe','eo']:
    proc=subprocess.Popen(
        cmd,
        stdout=subprocess.PIPE,
        stderr=subprocess.STDOUT,
        preexec_fn=subprocess_setup)
else:
    proc=subprocess.Popen(
        cmd,
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE,
        preexec_fn=subprocess_setup)

# Wait for the script to gracefully exit.
while time.time() < start_time + hook_alarm - 5:
    if proc.poll() is not None:
        break
    time.sleep(1)

# If we reach the alarm time - 5 seconds, send a SIGTERM
if proc.poll() is None:
    pbs.logmsg(pbs.LOG_WARNING,
               '%s;%s;[WARNING] Terminating %s after %s seconds' % \
               (hook_name, job.id, event, int(time.time() - start_time)))
    os.kill(proc.pid, signal.SIGTERM)
    while time.time() < start_time + hook_alarm - 3:
        if proc.poll() is not None:
            break
        time.sleep(0.5)

# If we reach an alarm time - 3 seconds, send a SIGKILL
if proc.poll() is None:
    pbs.logmsg(pbs.LOG_WARNING,
               '%s;%s;[WARNING] Killing %s after %s seconds' % \
               (hook_name, job.id, event, int(time.time() - start_time)))
    os.kill(proc.pid, signal.SIGKILL)
    while time.time() < start_time + hook_alarm - 1:
        if proc.poll() is not None:
            break
        time.sleep(0.5)

# If we still can't kill the script, log a warning and let pbs kill it
if proc.poll() is None:
    pbs.logmsg(pbs.LOG_WARNING,
               '%s;%s;[WARNING] Unable to kill %s after %s seconds' % \
```

```
                (hook_name, job.id, event, start_time - time.time()))

    # Get the stdout and stderr from the pelog
    (o, e)=proc.communicate()

    if DEBUG:
        pbs.logmsg(
            pbs.EVENT_DEBUG2,
            '%s;%s;[DEBUG2]: stdout=%s, stderr=%s.' %
            (hook_name, job.id, repr(o), repr(e)))

    joblog=JobLog()
    if o:
        joblog.stdout(o)
    if e:
        joblog.stderr(e)

    if proc.returncode:
        return_action=RERUN
        if event == 'prologue':
            return_action=RERUN
            if proc.returncode == 1:
                return_action=DELETE
        elif event == 'epilogue':
            return_action=DELETE
            if proc.returncode == 2:
                return_action=RERUN

        rejectjob(
            '%s exited with a status of %s.' % (p+event, proc.returncode),
            return_action)
    else:
        if DEBUG:
            pbs.logmsg(pbs.LOG_DEBUG,
                '%s;%s;[DEBUG] %s exited with a status of 0.' %
                (hook_name, job.id, p+event))

        if pbs_event.type == pbs.EXECJOB_PROLOGUE and VERBOSE_USER_OUTPUT:
            print '%s: attached as primary execution host.' % \
                pbs.get_local_nodename()

        pbs_event.accept()
else:
    rejectjob("The %s does not have the correct " % (p+event) + \
              'permissions. See the section entitled, ' + \
              '"Prologue and Epilogue Requirements" in the PBS Pro ' + \
```

```
                    "Administrator's Guide.", RERUN)

    except SystemExit:
        pass
    except:
        trace_hook()
```

9.7 exechost_startup Hook Examples

Example 9-16: Create vnode and set vnode resources

Hook type: exechost_startup

```
% cat startup.py
import pbs
e=pbs.event()
for v in e.vnode_list.keys():
    vn = e.vnode_list[v]
    vn.resources_available["file"] = pbs.size("7gb")
    vn.resources_available["fab_int"] = 9
    vn.resources_available["fab_str"] = "happy"
    vn.resources_available["fab_bool"] = False
    vn.resources_available["fab_size"] = pbs.size("7mb")
    vn.resources_available["fab_time"] = pbs.duration("00:30:00")
    vn.resources_available["fab_float"] = 7.0

e.vnode_list["mars[1]"] = pbs.vnode("mars[1]")
e.vnode_list["mars[1]"].resources_available["ncpus"] = 7
```

Create hook

```
# qmgr -c "create hook start event=exechost_startup"
# qmgr -c "import hook start application/x-python default startup.py"
```

Restart MoM

```
# kill <pbs_mom PID>
```

then

```
# systemctl start pbs
```

or

```
# /etc/init.d/pbs start  (start MoM)
```

Output

```
# pbsnodes -av
mars
    Mom = mars.example.com
    Port = 15002
    pbs_version = PBSPro_12.3.0.140813
    ntype = PBS
    state = free
    pcpus = 4
    resources_available.arch = linux
    resources_available.fab_bool = False
    resources_available.fab_float = 7
    resources_available.fab_int = 9
    resources_available.fab_size = 7mb
    resources_available.fab_str = happy
    resources_available.fab_time = 1800
    resources_available.file = 7gb
    resources_available.host = mars
    resources_available.mem = 8gb
    resources_available.ncpus = 5
    resources_available.vmem = 16gb
    resources_available.vnode = mars
    …


mars[1]
    Mom = mars.example.com
    Port = 15002
    pbs_version = PBSPro_12.3.0.140813
    ntype = PBS
    state = free
    resources_available.arch = linux
    resources_available.file = 7gb
    resources_available.host = mars
    resources_available.ncpus = 7      (set in hook)
    resources_available.vnode = mars[1]
```

9.8 exechost_periodic Hook Examples

Example 9-17: Monitor load; offline or free vnode depending on CPU load

Hook type: exechost_periodic

Monitor load average on the local host. Offline or free the vnode representing the host depending on the CPU load.

You can modify values for ideal_load and max_load. Your hook does the following:

If the system's CPU load average rises above max_load, the state of the vnode corresponding to the current host is set to *offline*. This prevents the scheduler from scheduling jobs on this vnode.

If the system's CPU load average falls below ideal_load, the state of the vnode representing the current host is set to *free*. This allows the scheduler to schedule jobs on this vnode.

To instantiate this hook, specify the following:

```
qmgr -c "create hook load_balance event=exechost_periodic,freq=10"
qmgr -c "import hook load_balance application/x-python default load_balance.py"
```

Hook script:

```
import pbs
import os
import re

ideal_load=1.5
max_load=2.0

# get_la: returns a list of load averages within the past 1-minute, 5-minute, 15-minutes range.
def get_la():
    line=os.popen("uptime").read()
    r = re.search(r'load average: (\S+), (\S+), (\S+)$', line).groups()
    return map(float, r)

local_node = pbs.get_local_nodename()

vnl = pbs.event().vnode_list
current_state = pbs.server().vnode(local_node).state
mla = get_la()[0]
if (mla >= max_load) and ((current_state == pbs.ND_OFFLINE) == 0):
    vnl[local_node].state = pbs.ND_OFFLINE
    vnl[local_node].comment = "offlined node as it is heavily loaded"
elif (mla < ideal_load) and ((current_state == pbs.ND_OFFLINE) != 0):
    vnl[local_node].state = pbs.ND_FREE
    vnl[local_node].comment = None
```

Example 9-18: Periodically update resources on vnodes

Hook type: exechost_periodic

Periodically update the values of a set of custom resources for the vnode where the current MoM runs.

The current set includes two size types, which are scratch and home

Prerequisites:

1. Create the following custom resources:

```
qmgr -c "create resource scratch type=size, flag=nh"
qmgr -c "create resource home type=size, flag=nh"
```

2. Add the new resources to the "resources:" line in the sched_config file and restart pbs_sched:

```
% cat PBS_HOME/sched_priv/sched_config resources
ncpus, mem, arch, [...], scratch, home
```

3. Install this hook as follows:

```
qmgr -c "create hook mom_dyn_res event=exechost_periodic,freq=30"
qmgr -c "import hook mom_dyn_res application/x-python default mom_dyn_res.py"
```

The mom_dyn_res.py script:

```
# NOTE:
#  Update the dyn_res[] array below to include any other custom resources
#  to be included in the updates. Ensure that each resource added has an
#  entry in the scheduler's sched_config file.

import pbs
import os
import sys

# get_filesystem_avail_unprivileged: returns available size in kbytes
# (in pbs.size type) to unprivileged users, of the filesystem where
# 'dirname' resides.

def get_filesystem_avail_unprivileged( dirname ):
    o = os.statvfs(dirname)
    return pbs.size( "%skb" % ((o.f_bsize * o.f_bavail) / 1024) )

# get_filesystem_avail_privileged: returns available size in kbytes
# (in pbs.size type) to privileged users, of the filesystem where 'dirname'
# resides.

def get_filesystem_avail_privileged( dirname ):
    o = os.statvfs(dirname)
    return pbs.size( "%skb" % ((o.f_bsize * o.f_bfree) / 1024) )

try:
    # Define here the custom resources as key, and the function and its
    # argument for obtaining the value of the custom resource:
```

```
    #    Format: dyn_res[<resource_name>] = [<function_name>,
    #                    <function_argument>]
    # So "<function_name>(<function_argument>)" is called to return the
    # value for custom <resource_name>.
    dyn_res = {}
    dyn_res["scratch"] = [get_filesystem_avail_unprivileged, "/tmp"]
    dyn_res["home"]    = [get_filesystem_avail_unprivileged, "/home"]

    vnl = pbs.event().vnode_list
    local_node = pbs.get_local_nodename()

    for k in dyn_res.keys():
        vnl[local_node].resources_available[k] = dyn_res[k][0](dyn_res[k][1])

except SystemExit:
    pass

except:
    e = pbs.event()
    e.reject("%s hook failed with %s. Please contact Admin" % \
    (e.hook_name, sys.exc_info()[:2]))
```

Example 9-19: Log loads on vnodes

Hook type: exechost_periodic

You must create the custom resources r1m, r5m, and r15m on the vnodes.

```
#cat getload.py


import pbs
import sys
import os
load = os.getloadavg()
r1m = load[0]
r5m = load[1]
r15m = load[2]
e = pbs.event()
mynode = pbs.get_local_nodename()
v = e.vnode_list[mynode]
v.resources_available["r1m"] = r1m
v.resources_available["r5m"] = r5m
v.resources_available["r15m"] = r15m
pbs.logmsg(pbs.LOG_DEBUG,"getloadavg: vnode %s, r1m = %f, r5m = %f, r15m = %f" %
(repr(mynode), r1m, r5m, r15m))
```

Example 9-20: *Set job attributes and resources*

Hook type: exechost_periodic

```
% cat period.py
import pbs
E = pbs.event()
for k in e.job_list.keys():
    e.job_list[k].resources_used["mem"] = pbs.size("7gb")
    e.job_list[k].Variable_List["POLI"] = "negri"
    e.job_list[k].Hold_Types = pbs.hold_types("us")
```

Create the hook:

```
# qmgr -c "create hook period event=exechost_periodic,freq=30"
# qmgr -c "import hook period application/x-python default period.py"
```

Submit several jobs:

```
% qsub job.scr
<job-id1>
% qsub job.scr
<job-id2>
```

As the exechost_periodic hook executes, the jobs get the new values:

```
% qstat -f <job-id1>
…
Resources_used.mem = 7gb
Hold_Types = us
Variable_List = …POLI=negri…
2
% qstat -f <job-id1>
…
Resources_used.mem = 7gb
Hold_Types = us
Variable_List = …POLI=negri…
```

9.9 Multi-event Hooks

Example 9-21: Helper function for logging exceptions more completely and flexibly:

```
# The trace_hook function has been written to be portable between hooks.
def trace_hook(**kwargs):
    """Simple exception trace logger for PBS hooks
    loglevel=<int> (pbs.LOG_DEBUG): log level to pass to pbs.logmsg()
    reject=True: reject the job upon completion of logging trace
    trace_in_reject=<bool> (False): pass trace to pbs.event().reject()
    trace_in_reject=<str>: message to pass to pbs.event().reject() with trace
    Usage:
    try:
        your=code(here)
    except:
        trace_hook()
    """

    import pbs
    import sys

    if 'loglevel' in kwargs:
        loglevel=kwargs['loglevel']
    else:
        loglevel=pbs.LOG_ERROR
    if 'reject' in kwargs:
        reject=kwargs['reject']
    else:
        reject=True
    if 'trace_in_reject' in kwargs:
        trace_in_reject=kwargs['trace_in_reject']
    else:
        trace_in_reject=False

    # Associate hook events with the appropriate PBS constant. This is a list
    # of all hook events as of PBS Pro 13.0. If the event does not exist, it is
    # removed from the list.
    hook_events=['queuejob', 'modifyjob', 'movejob', 'runjob', 'execjob_begin',
                 'execjob_prologue', 'execjob_launch', 'execjob_attach',
                 'execjob_preterm', 'execjob_epilogue', 'execjob_end',
                 'resvsub', 'provision', 'exechost_periodic',
                 'exechost_startup']

    hook_event={}
    for he in hook_events:
        # Only set available hooks for the current version of PBS.
```

```
    if hasattr(pbs, he.upper()):
        event_code=eval('pbs.'+he.upper())
        hook_event[event_code]=he
        hook_event[he]=event_code
        hook_event[he.upper()]=event_code
        del event_code
    else:
        del hook_events[hook_events.index(he)]

trace={
    'line':      sys.exc_info()[2].tb_lineno,
    'module':    sys.exc_info()[2].tb_frame.f_code.co_name,
    'exception': sys.exc_info()[0].__name__,
    'message':   sys.exc_info()[1].message,
}
tracemsg='%s hook %s encountered an exception: Line %s in %s %s: %s' %(
    hook_event[pbs.event().type], pbs.event().hook_name,
    trace['line'], trace['module'], trace['exception'], trace['message']
)
rejectmsg="Hook Error: request rejected as filter hook '%s' encountered " \
    "an exception. Please inform Admin" % pbs.event().hook_name
if not isinstance(loglevel, int):
    pbs.logmsg(pbs.LOG_ERROR, 'trace_hook() called with invalid argument' \
        ' (loglevel=%s), setting to pbs.LOG_ERROR. ' % repr(loglevel))
    loglevel=pbs.LOG_ERROR

pbs.logmsg(loglevel, tracemsg)

if reject:
    tracemsg+=', request rejected'
    if isinstance(trace_in_reject, bool):
        if trace_in_reject:
            pbs.event().reject(tracemsg)
        else:
            pbs.event().reject(rejectmsg)
    else:
        pbs.event().reject(str(trace_in_reject)+'Line %s in %s %s:\n%s' % (
            trace['line'],trace['module'],trace['exception'],
            trace['message'] ))
```

Index

A

accept an action HG-5
action HG-5

B

built-in hook HG-5

C

configuration file
    hook HG-6
creating a hook HG-5

D

DIS HG-136

E

event HG-5
events
    exechost_periodic HG-95, HG-106, HG-107
    execjob_begin HG-96, HG-98
    execjob_end HG-105
    execjob_epilogue HG-104
    execjob_preterm HG-103
    execjob_prologue HG-97
    modifyjob HG-92
    movejob HG-93
    queuejob HG-91
    resvsub HG-90
    runjob HG-94
exechost_periodic HG-89
exechost_periodic events HG-95, HG-106, HG-107
execjob_attach HG-89
execjob_begin HG-88
execjob_begin events HG-96, HG-98
execjob_end HG-89
execjob_end events HG-104, HG-105
execjob_epilogue HG-89
execjob_launch HG-88
execjob_postsuspend HG-89
execjob_preresume HG-89
execjob_preterm HG-89
execjob_preterm events HG-103
execjob_prologue HG-88
execjob_prologue events HG-97
execution event hooks HG-6

F

failover and hooks HG-21
failure action HG-6

H

hook
    configuration file HG-6
    creating HG-5
    importing HG-6
hook configuration file HG-6

hooks
and failover HG-21
creating empty hooks HG-30
deleting HG-31
enabling and disabling HG-38
event types HG-15
events
exechost_periodic HG-95, HG-106, HG-107
execjob_begin HG-96, HG-98
execjob_end HG-104, HG-105
execjob_preterm HG-103
execjob_prologue HG-97
modifyjob HG-92
movejob HG-93
queuejob HG-91
resvsub HG-90
runjob HG-94
exechost_periodic events HG-95, HG-106, HG-107
execjob_begin events HG-96, HG-98
execjob_end events HG-104, HG-105
execjob_preterm events HG-103
execjob_prologue events HG-97
execution event HG-6
exporting HG-35
importing HG-34
interface objects HG-76
job attributes HG-55
log level objects HG-152
modifyjob events HG-92
MoM HG-6
movejob events HG-93
non-job event HG-6
overview of creating HG-30
pbs.exec_vnode HG-129
pbs.job HG-122
pbs.queue() HG-121
pbs.resv HG-131
pbs.server() HG-118
pbs.vchunk HG-131
pbs.vnode HG-133
pre-execution event HG-6
queuejob events HG-91
reservation attributes HG-59
resources HG-47
resvsub events HG-90
runjob events HG-94
setting order of execution HG-38
setting timeout HG-39
setting trigger events HG-31
simple how-to HG-11
vnode attributes HG-57

I

importing a hook HG-6
interface objects for hooks HG-76

J

job
attributes in hooks HG-55
job.array_indices_submitted HG-123
job.Checkpoint HG-124
job.delete() HG-128
job.depend HG-124
job.exec_host HG-124
job.exec_vnode HG-124
job.Execution_Time HG-124
job.group_list HG-124
job.Hold_Types HG-124
job.id HG-123
job.in_ms_mom() HG-128
job.is_checkpointed() HG-127
job.job_state HG-125
job.Mail_Points HG-126
job.Mail_Users HG-126
job.rerun() HG-129
job.resources_used HG-126
job.stagein HG-126
job.stageout HG-126
job.User_List HG-127

L

logging
hooks log level objects HG-152

M

modifyjob HG-88
modifyjob events HG-92
MoM hooks HG-6
movejob HG-88
movejob events HG-93

N

non-job event hooks HG-6

P

pbs module HG-6
pbs.acl() HG-143
pbs.args() HG-143
pbs.checkpoint( ) HG-144
pbs.depend() HG-144
pbs.duration() HG-144
pbs.email_list() HG-144
pbs.event().accept() HG-116
pbs.event().alarm HG-109

pbs.event().hook_name HG-111, HG-112
pbs.event().hook_type HG-111
pbs.event().pid HG-112
pbs.event().reject() HG-116
pbs.event().requestor HG-113
pbs.event().requestor_host HG-113
pbs.event().type HG-113
pbs.exec_host() HG-145
pbs.exec_vnode HG-129
pbs.exec_vnode() HG-145
pbs.get_local_nodename() HG-151
pbs.group_list() HG-145
pbs.hold_types() HG-145
pbs.job HG-122
pbs.job_sort_formula() HG-145
pbs.join_path() HG-146
pbs.keep_files() HG-146
pbs.license_count() HG-146
pbs.logmsg() HG-152
pbs.mail_points() HG-146
pbs.node_group_key() HG-146
pbs.path_list() HG-146
pbs.pbs_env() HG-146
pbs.pid HG-81
pbs.place() HG-147
pbs.queue HG-121
pbs.queue() HG-121
pbs.queue.job() HG-122
pbs.range() HG-148
pbs.reboot() HG-153
pbs.resv HG-131
pbs.route_destinations() HG-148
pbs.select() HG-148
pbs.server() HG-118
pbs.server(). HG-118
pbs.server().job() HG-119
pbs.server().jobs() HG-119
pbs.server().name HG-118
pbs.server().queue() HG-120
pbs.server().queues() HG-120
pbs.server().resv() HG-120
pbs.server().resvs() HG-120
pbs.server().scheduler_restart_cycle() HG-120
pbs.server().vnode() HG-120
pbs.server().vnodes() HG-120
pbs.size() HG-150
pbs.software() HG-150
pbs.staging_list() HG-150
pbs.state_count() HG-151
pbs.user_list() HG-151
pbs.vchunk HG-131
pbs.version() HG-151
pbs.vnode HG-133
PBS_AUTH_METHOD HG-136
PBS_BATCH_SERVICE_PORT HG-136
PBS_BATCH_SERVICE_PORT_DIS HG-136
PBS_COMM_LOG_EVENTS HG-136
PBS_COMM_ROUTERS HG-136
PBS_COMM_THREADS HG-136
PBS_CONF_SYSLOG HG-140
PBS_CONF_SYSLOGSEVR HG-140
PBS_CORE_LIMIT HG-136
PBS_DATA_SERVICE_PORT HG-136
PBS_ENCRYPT_METHOD HG-136
PBS_ENVIRONMENT HG-136
PBS_EXEC HG-136
PBS_HOME HG-136
PBS_LEAF_NAME HG-137
PBS_LEAF_ROUTERS HG-137
PBS_LOCALLOG HG-137
PBS_LOG_HIGHRES_TIMESTAMP HG-137
PBS_MAIL_HOST_NAME HG-137
PBS_MANAGER_SERVICE_PORT HG-137
PBS_MOM_HOME HG-137
PBS_MOM_NODE_NAME HG-137
PBS_MOM_SERVICE_PORT HG-137
PBS_OUTPUT_HOST_NAME HG-138
PBS_PRIMARY HG-138
PBS_RCP HG-138
PBS_REMOTE_VIEWER HG-138
PBS_SCHED_THREADS HG-138
PBS_SCHEDULER_SERVICE_PORT HG-138
PBS_SCP HG-138
PBS_SECONDARY HG-139
PBS_SERVER HG-139
PBS_SERVER_HOST_NAME HG-139
PBS_START_COMM HG-139
PBS_START_MOM HG-139
PBS_START_SCHED HG-139
PBS_START_SERVER HG-139
PBS_SUPPORTED_AUTH_METHODS HG-139
PBS_TMPDIR HG-140
pbshook HG-6
pre-execution event hooks HG-6
primary server HG-138

Q

queue. HG-121
queue.job() HG-122
queue.jobs() HG-122
queue.name HG-121
queuejob HG-87
queuejob events HG-91

R

rcp HG-138
reject an action HG-6

reservation
    attributes in hooks HG-59
resources
    in hooks HG-47
resv. HG-132
resv.resvid HG-132
resvsub HG-87
resvsub events HG-90
runjob HG-88
runjob events HG-94

S

scp HG-138
secondary server HG-139
server
    primary HG-138
    secondary HG-139
server hook HG-6
setting hook trigger events HG-31

V

vchunk.chunk_resources.keys() HG-131
vchunk.vnode_name HG-131
vnode
    attributes in hooks HG-57
vnode.topology_info HG-134