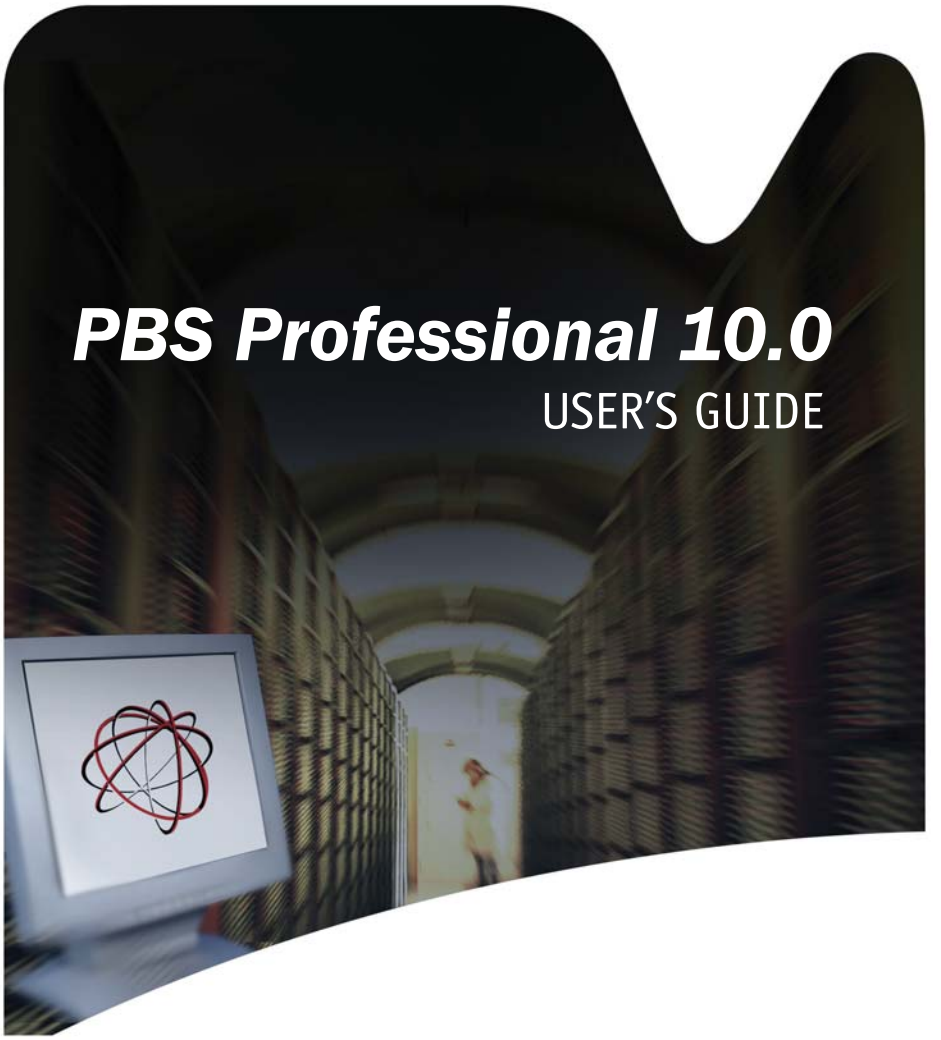


PBS Professional 10.0

USER'S GUIDE

GridWorks™

Enabling On-Demand Computing™

A division of Altair

Altair®

PBS Professional™

10.0

User's Guide

UNIX®, Linux® and Windows®

PBS Professional™ User's Guide, Altair® PBS Professional™ 10.0, Updated: 11/7/08. Edited by: Anne Urban

For more information, copies of these books, and for product sales, contact Altair at:

Web: www.pbsgridworks.com

Email: pbs-sales@pbspro.com

Technical Support

Table 1-1:

Location	Telephone	e-mail
North America	+1 248 614 2425	pbssupport@altair.com
China	+86 (0)21 5393 0011	support@altair.com.cn
France	+33 (0)1 4133 0990	francesupport@altair.com
Germany	+49 (0)7031 6208 22	hwsupport@altair.de
India	+91 80 658 8540 +91 80 658 8542	pbs-support@india.altair.com
Italy	+39 0832 315573 +39 800 905595	support@altairtorino.it
Japan	+81 3 5396 1341	pbs@altairjp.co.jp
Korea	+82 31 728 8600	support@altair.co.kr
Scandinavia	+46 (0)46 286 2050	support@altair.se
UK	+44 (0) 2476 323 600	support@uk.altair.com

Table of Contents

Acknowledgements

PBS Professional is the enhanced commercial version of the PBS software originally developed for NASA. The NASA version had a number of corporate and individual contributors over the years, for which the PBS developers and PBS community is most grateful. Below we provide formal legal acknowledgements to corporate and government entities, then special thanks to individuals.

The NASA version of PBS contained software developed by NASA Ames Research Center, Lawrence Livermore National Laboratory, and MRJ Technology Solutions. In addition, it included software developed by the NetBSD Foundation, Inc., and its contributors as well as software developed by the University of California, Berkeley and its contributors.

Other contributors to the NASA version of PBS include Bruce Kelly and Clark Streeter of NERSC; Kent Crispin and Terry Heidelberg of LLNL; John Kochmar and Rob Pennington of *Pittsburgh Supercomputing Center*; and Dirk Grunwald of *University of Colorado, Boulder*. The ports of PBS to the Cray T3e and the IBM SP SMP were funded by *DoD USAERDC*; the port of PBS to the Cray SV1 was funded by *DoD MSIC*.

No list of acknowledgements for PBS would possibly be complete without special recognition of the first two beta test sites. Thomas Milliman of the *Space Sciences Center* of the *University of New Hampshire* was the first beta tester. Wendy Lin of *Purdue University* was the second beta tester and holds the honor of submitting more problem reports than anyone else outside of NASA.

Preface

Intended Audience

PBS Professional is the professional workload management system from Altair that provides a unified queuing and job management interface to a set of computing resources. This document provides the user with the information required to use PBS Professional, including creating, submitting, and manipulating batch jobs; querying status of jobs, queues, and systems; and otherwise making effective use of the computer resources under the control of PBS.

Related Documents

The following publications contain information that may also be useful to the user of PBS:

PBS Professional Quick Start Guide: a short overview of the installation of PBS Professional.

PBS Professional Installation & Upgrade Guide: Contains administrator's information on installing and upgrading PBS Professional.

PBS Professional Administrator's Guide: Contains administrator's information required to configure and manage PBS, as well as a discussion of how PBS components interoperate.

PBS Professional External Reference Specification: discusses the PBS application programming interface (API), security within PBS, and inter-daemon/service communication.

Ordering Software and Publications

To order additional copies of this and other PBS publications, or to purchase additional software licenses, contact an authorized reseller, or the PBS Sales Department. Contact information is included on the copyright page of this document.

Document Conventions

PBS documentation uses the following typographic conventions.

abbreviation	In a PBS command can be abbreviated (such as sub-commands to qmgr) the shortest acceptable abbreviation is underlined.
`command`	This fixed-width font is used to denote literal commands, filenames, error messages, and program output.
`input`	Literal user input is shown in this bold fixed-width font.
`manpage(x)`	Following UNIX tradition, manual page references include the corresponding section number in parentheses appended to the man page name.
terms	Words or terms being defined, as well as variable names, are in italics.

Chapter 1

Introduction

This book, the **User's Guide** to PBS Professional is intended as your knowledgeable companion to the PBS Professional software. The information herein pertains to PBS in general, with specific information for PBS Professional 10.0.

1.1 Book Organization

This book is organized into 10 chapters, plus two appendices. Depending on your intended use of PBS, some chapters will be critical to you, and others may be safely skipped.

Chapter 1
gives an overview of this book, PBS, and the PBS team.

Chapter 2
discusses the various components of PBS and how they interact, followed by definitions of terms used in PBS and in distributed workload management.

Chapter 3
introduces PBS, describing both user interfaces and suggested settings to the user's environment.

Chapter 4
describes the structure and components of a PBS job, and explains how to create and submit a PBS job.

Chapter 5
introduces the `xpbs` graphical user interface, and shows how to submit a PBS job using `xpbs`.

Chapter 6
describes how to check status of a job, and request status of queues, vnodes, systems, or PBS Servers.

Chapter 7
discusses commonly used commands and features of PBS, and explains how to use each one.

Chapter 8
describes and explains how to use the more advanced features of PBS.

Chapter 9
describes and explains the job array features in PBS.

Chapter 10
explains how PBS interacts with multi-vnode and parallel applications, and illustrates how to run such applications under PBS.

Appendix A
provides a quick reference summary of PBS environment variables.

Appendix B
includes information for converting from NQS/NQE to PBS.

1.2 Supported Platforms

For a list of supported platforms, see the Release Notes.

1.3 What is PBS Professional?

PBS Professional is the professional version of the Portable Batch System (PBS), a flexible workload management system, originally developed to manage aerospace computing resources at NASA. PBS has since become the leader in supercomputer workload management and the *de facto* standard on Linux clusters.

Today, growing enterprises often support hundreds of users running thousands of jobs across different types of machines in different geographical locations. In this distributed heterogeneous environment, it can be extremely difficult for administrators to collect detailed, accurate usage data, or to set system-wide resource priorities. As a result, many computing resources are left under-utilized, while others are over-utilized. At the same time, users are confronted with an ever expanding array of operating systems and platforms. Each year, scientists, engineers, designers, and analysts must waste countless hours learning the nuances of different computing environments, rather than being able to focus on their core priorities. PBS Professional addresses these problems for computing-intensive industries such as science, engineering, finance, and entertainment.

Now you can use the power of PBS Professional to better control your computing resources. This allows you to unlock the potential in the valuable assets you already have, while at the same time, reducing dependency on system administrators and operators, freeing them to focus on other actives. PBS Professional can also help you effectively manage growth by tracking real usage levels across your systems and enhancing utilization of future purchases.

1.4 History of PBS

In the past, UNIX systems were used in a completely interactive manner. Background jobs were just processes with their input disconnected from the terminal. However, as UNIX moved onto larger and larger machines, the need to be able to schedule tasks based on available resources increased in importance. The advent of networked compute servers, smaller general systems, and workstations led to the requirement of a networked batch scheduling capability. The first such UNIX-based system was the Network Queueing System (NQS) funded by NASA Ames Research Center in 1986. NQS quickly became the *de facto* standard for batch queueing.

Over time, distributed parallel systems began to emerge, and NQS was inadequate to handle the complex scheduling requirements presented by such systems. In addition, computer system managers wanted greater control over their compute resources, and users wanted a single interface to the systems. In the early 1990's NASA needed a solution to this problem, but found nothing on the market that adequately addressed their needs. So NASA led an international effort to gather requirements for a next-generation resource management system. The requirements and functional specification were later adopted as an IEEE POSIX standard (1003.2d). Next, NASA funded the development of a new resource management system compliant with the standard. Thus the Portable Batch System (PBS) was born.

PBS was quickly adopted on distributed parallel systems and replaced NQS on traditional supercomputers and server systems. Eventually the entire industry evolved toward distributed parallel systems, taking the form of both special purpose and commodity clusters. Managers of such systems found that the capabilities of PBS mapped well onto cluster systems. (For information on converting from NQS to PBS, see Appendix B.)

The PBS story continued when MRJ-Veridian (the R&D contractor that developed PBS for NASA) released the Portable Batch System Professional Edition (PBS Pro), a commercial, enterprise-ready, workload management solution. Three years later, the MRJ-Veridian PBS Products business unit was acquired by Altair Engineering, Inc. Altair set up the PBS Products unit as a subsidiary company named Altair Grid Technologies focused on PBS Professional and related Grid software. This unit then became part of Altair Engineering.

1.5 About the PBS Team

The PBS Professional product is developed by the same team that originally designed PBS for NASA. In addition to the core engineering team, Altair Engineering includes individuals who have supported PBS on computers around the world, including some of the largest supercomputers in existence. The staff includes internationally-recognized experts in resource-management and job-scheduling, supercomputer optimization, message-passing programming, parallel computation, and distributed high-performance computing. In addition, the PBS team includes co-architects of the NASA Metacenter (the first full-production geographically distributed meta-computing grid), co-architects of the Department of Defense MetaQueueing (prototype Grid) Project, co-architects of the NASA Information Power Grid, and co-chair of the Global Grid Forum's Scheduling Group.

1.6 About Altair Engineering

Through engineering, consulting and high performance computing technologies, Altair Engineering increases innovation for more than 1,500 clients around the globe. Founded in 1985, Altair's unparalleled knowledge and expertise in product development and manufacturing extend throughout North America, Europe and Asia. Altair specializes in the development of high-end, open CAE software solutions for modeling, visualization, optimization and process automation.

1.7 Why Use PBS?

PBS Professional provides many features and benefits to both the computer system user and to companies as a whole. A few of the more important features are listed below to give the reader both an indication of the power of PBS, and an overview of the material that will be covered in later chapters in this book.

Enterprise-wide Resource Sharing provides transparent job scheduling on any PBS system by any authorized user. Jobs can be submitted from any client system both local and remote, crossing domains where needed.

Multiple User Interfaces provides a graphical user interface for submitting batch and interactive jobs; querying job, queue, and system status; and monitoring job progress. PBS also provides a traditional command line interface.

Security and Access Control Lists permit the administrator to allow or deny access to PBS systems on the basis of username, group, host, and/or network domain.

Job Accounting offers detailed logs of system activities for charge-back or usage analysis per user, per group, per project, and per compute host.

Automatic File Staging provides users with the ability to specify any files that need to be copied onto the execution host before the job runs, and any that need to be copied off after the job completes. The job will be scheduled to run only after the required files have been successfully transferred.

Parallel Job Support works with parallel programming libraries such as MPI, PVM and HPF. Applications can be scheduled to run within a single multi-processor computer or across multiple systems.

System Monitoring includes a graphical user interface for system monitoring. Displays vnode status, job placement, and resource utilization information for both stand-alone systems and clusters.

Job-Interdependency enables the user to define a wide range of interdependencies between jobs. Such dependencies include execution order, and execution conditioned on the success or failure of another specific job (or set of jobs).

Computational Grid Support provides an enabling technology for metacomputing and computational grids.

Comprehensive API includes a complete Application Programming Interface (API) for sites who desire to integrate PBS with other applications, or who wish to support unique job scheduling requirements.

Automatic Load-Leveling provides numerous ways to distribute the workload across a cluster of machines, based on hardware configuration, resource availability, keyboard activity, and local scheduling policy.

Distributed Clustering allows customers to utilize physically distributed systems and clusters, even across wide-area networks.

Common User Environment offers users a common view of the job submission, job querying, system status, and job tracking over all systems.

Cross-System Scheduling ensures that jobs do not have to be targeted to a specific computer system. Users may submit their job, and have it run on the first available system that meets their resource requirements.

Job Priority allows users the ability to specify the priority of their jobs; defaults can be provided at both the queue and system level.

Username Mapping provides support for mapping user account names on one system to the appropriate name on remote server systems. This allows PBS to fully function in environments where users do not have a consistent username across all hosts.

Fully Configurable. PBS was designed to be easily tailored to meet the needs of different sites. Much of this flexibility is due to the unique design of the scheduler module which permits significant customization.

Broad Platform Availability is achieved through support of Windows and every major version of UNIX and Linux, from workstations and servers to supercomputers. New platforms are being supported with each new release.

System Integration allows PBS to take advantage of vendor-specific enhancements on different systems (such as supporting cpusets on SGI systems).

Job Arrays are a mechanism for containerizing related work, making it possible to submit, query, modify and display a set of jobs as a single unit.

Chapter 2

Concepts and Terms

PBS is a distributed workload management system. As such, PBS handles the management and monitoring of the computational workload on a set of one or more computers. Modern workload management solutions like PBS Professional include the features of traditional batch queueing but offer greater flexibility and control than first generation batch systems (such as NQS).

Workload management systems have three primary roles:

Queuing

The collecting together of work or tasks to be run on a computer. Users submit tasks or "jobs" to the resource management system where they are queued up until the system is ready to run them.

Scheduling

The process of selecting which jobs to run, when, and where, according to a predetermined policy. Sites balance competing needs and goals on the system(s) to maximize

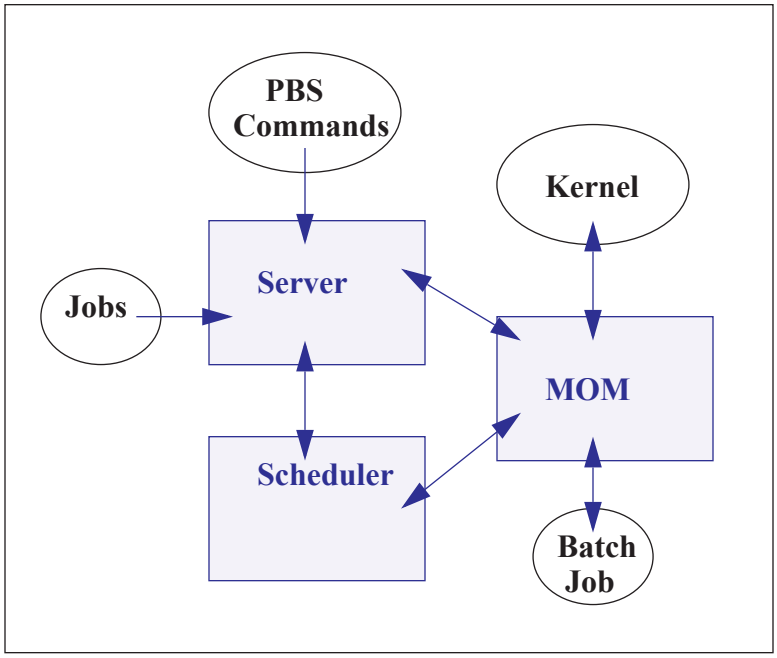
efficient use of resources (both computer time and people time).

Monitoring

The act of tracking and reserving system resources and enforcing usage policy. This includes both software enforcement of usage limits and user or administrator monitoring of scheduling policies to see how well they are meeting stated goals.

2.1 PBS Components

PBS consist of two major component types: user-level commands and system daemons/services. A brief description of each is given here to help you understand how the pieces fit together, and how they affect you.



Commands

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

There are three command classifications: user commands, which any authorized user can use, operator commands, and manager (or administrator) commands. Operator and manager commands which require specific access privileges are discussed in the **PBS Professional Administrator's Guide**.

Server

The *Job Server* daemon/service is the central focus for PBS. Within this document, it is generally referred to as *the Server* or by the execution name *pbs_server.* All commands and the other daemons/services communicate with the Server via an *Internet Protocol* (IP) network. The Server's main function is to provide the basic batch services such as receiving/creating a batch job, modifying the job, and running the job. Normally, there is one Server managing a given set of resources. However if the Server Failover feature is enabled, there will be two Servers.

Job Executor (MOM)

The *Job Executor* or *MOM* is the daemon/service which actually places the job into execution. This process, *pbs_mom,* is informally called *MOM* as it is the mother of all executing jobs. (MOM is a reverse-engineered acronym that stands for Machine Oriented Mini-server.) MOM places a job into execution when it receives a copy of the job from a Server. MOM creates a new session that is as identical to a user login session as is possible. (For example under UNIX, if the user's login shell is `csh`, then MOM creates a session in which `.login` is run as well as `.cshrc`.) MOM also has the responsibility for returning the job's output to the user when directed to do so by the Server. One MOM runs on each computer which will execute PBS jobs.

Scheduler

The *Job Scheduler* daemon/service, *pbs_sched*, implements the site's policy controlling when each job is run and on which resources. The Scheduler communicates with the various MOMs to query the state of system resources and with the Server for availability of jobs to execute. The interface to the Server is through the same API as used by the client commands. Note that the Scheduler interfaces with the Server with the same privilege as the PBS manager.

2.2 Defining PBS Concepts and Terms

The following section defines important terms and concepts of PBS. The reader should review these definitions before beginning the planning process prior to installation of PBS. The terms are defined in an order that best allows the definitions to build on previous terms.

Node

No longer used. A *node* to PBS is a computer system with a single *operating system* (OS) image, a unified virtual memory space, one or more CPUs and one or more IP addresses. Frequently, the term *execution host* is used for node. A computer such as the SGI Origin 3000, which contains multiple CPUs running under a single OS, is one node. Systems like the IBM SP and Linux clusters, which contain separate computational units each with their own OS, are collections of nodes.

If a host has more than one virtual processor, the VPs may be assigned to different jobs or used to satisfy the requirements of a single job (*exclusive*). This ability to temporarily allocate the entire host to the exclusive use of a single job is important for some multi-host parallel applications. Note that PBS enforces a one-to-one allocation scheme of cluster host VPs ensuring that the VPs are not over-allocated or over-subscribed between multiple jobs. (See also *vnode* and *virtual processors*.)

Vnode

A virtual node, or *vnode*, is an abstract object representing a set of resources which form a usable part of a machine. This could be an entire host, or a nodeboard or a blade. A single host can be made up of multiple vnodes. Each vnode can be managed and scheduled independently. Each vnode in a complex must have a unique name. Vnodes can share resources, such as node-locked licenses.

Host

A machine with its own operating system, made up of one or more vnodes. Also, all vnodes with the same value for *resources_available.host*. A single host can be made up of multiple vnodes.

Chunk

A set of resources allocated as a unit to a job. Specified inside a selection directive. All pars of a chunk come from the same host. In a typical MPI job, there is one chunk per MPI process.

Cluster

Generally, a very homogeneous set of systems that are viewed as one unit. Typically, the word "cluster" means "Linux cluster", although it is also being used to mean "Windows cluster".

Complex

A PBS complex consists of the machines running one primary Server+Scheduler (plus, optionally, a secondary backup Server+Scheduler) and all the machines on which the MOMs (attached to this Server+Scheduler) are running. In general, it can be a very heterogeneous mix of system architectures, operating systems, and can include several clusters.

Exclusive VP

An exclusive VP is one that is used by one and only one job at a time. A set of VPs is assigned exclusively to a job for the duration of that job. This is typically done to improve the performance of message-passing programs.

Load Balance

A policy wherein jobs are distributed across multiple time-shared hosts to even out the workload on each host. Being a policy, the distribution of jobs across execution hosts is solely a function of the Job Scheduler.

Queue

A *queue* is a named container for jobs within a Server. There are two types of queues defined by PBS, *routing* and *execution*. A *routing queue* is a queue used to move jobs to other queues including those that exist on different PBS Servers. A job must reside in an *execution queue* to be eligible to run and remains in an execution queue during the time it is running. In spite of the name, jobs in a queue need not be processed in queue order (first-come first-served or *FIFO*).

Vnode Attribute

Vnodes have attributes associated with them that provide control information. The attributes defined for vnodes are: state, the list of jobs to which the vnode is allocated, properties, `max_running`, `max_user_run`, `max_group_run`, and both assigned and available resources ("`resources_assigned`" and "`resources_available`").

PBS Professional

PBS consists of one Server (`pbs_server`), one Scheduler (`pbs_sched`), and one or more MOMs (`pbs_mom`). The PBS System can be set up to distribute the workload to one large system, multiple systems, a cluster of hosts, or any combination of these.

Virtual Processor (VP)

A vnode may be declared to consist of one or more *virtual processors (VPs).* The term virtual is used because the number of VPs declared does not have to equal the number of real processors (CPUs) on the physical vnode. The default number of virtual processors on a vnode is the number of currently functioning physical processors; the PBS Manager can change the number of VPs as required by local policy.

The remainder of this chapter provides additional terms, listed in alphabetical order.

Account

An *account* is arbitrary character string, which may have meaning to one or more hosts in the batch system. Frequently, account is used by sites for accounting or charge-back purposes.

Administrator

See Manager.

API

PBS provides an *Application Programming Interface (API)* which is used by the commands to communicate with the Server. This API is described in the **PBS Professional External Reference Specification**. A site may make use of the API to implement new commands if so desired.

Advance Reservation

An *advance reservation* is a set of resources available for jobs for a specific amount of time in the future. Both the amount of resources and the amount of time are fixed for the life cycle of the reservation. Advance reservations are created manually by a user.

Attribute

An *attribute* is a data item whose value affects the operation or behavior of the object and can be set by the owner of the object.

Batch or Batch Processing

This refers to the capability of running jobs outside of the interactive login environment.

Complex

A *complex* is a collection of hosts managed by one batch system. It may be made up of vnodes that are allocated to only one job at a time or of vnodes that have many jobs executing at once on each vnode or a combination of these two scenarios.

Destination

This is the location within PBS where a job is sent. A destination may be a single queue at a single Server or it may map into multiple possible locations, tried in turn until one accepts the job.

Destination Identifier

This is a string that names the destination. It is composed of two parts and has the format queue@server where server is the name of a PBS Server and queue is the string identifying a queue on that Server.

Directive

A means by which the user specifies to PBS the value of a variable such as number of CPUs, the name of a job, etc. The default start of a directive is "#PBS". PBS directives either specify resource requirements or attribute values. See page 74.

File Staging

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

Group ID (GID)

This unique number represents a specific group (see Group).

Group

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

Hold

A restriction which prevents a job from being selected for processing. There are three types of holds. One is applied by the job owner, another is applied by a PBS *Operator*, and a third applied by the system itself or the PBS *Manager*. (See also Operator and Manager in this glossary.)

Job or Batch Job

The basic execution object managed by the batch subsystem. A job is a collection of related processes which is managed as a whole. A job can often be thought of as a shell script running a set of tasks.

Manager

A *manager* is authorized to use all capabilities of PBS. The Manager may act upon the Server, queues, or jobs. The Manager is also called the administrator.

Occurrence of a Standing Reservation

An individual instance of a standing reservation.

Operator

A person authorized to use some but not all of the restricted capabilities of PBS is an *operator*.

Owner

The user who submitted a specific job to PBS.

PBS_HOME

Refers to the path under which PBS was installed on the local system. Your local system administrator can provide the specific location.

POSIX

This acronym refers to the various standards developed by the "Technical Committee on Operating Systems and Application Environments of the IEEE Computer Society" under standard P1003.

Requeue

The process of stopping a running (executing) job and putting it back into the queued ("Q") state. This includes placing the job as close as possible to its former position in that queue.

Rerunnable

If a PBS job can be terminated and its execution restarted from the beginning without harmful side effects, the job is rerunnable.

Stage In

This process refers to moving a file or files to the execution host prior to the PBS job beginning execution.

Stage Out

This process refers to moving a file or files off of the execution host after the PBS job completes execution.

Standing Reservation

A recurring advance reservation where each occurrence has the same resource specification.

User

Each system *user* is identified by a unique character string (the user name) and by a unique number (the user id).

Task

Task is a POSIX session started by MOM on behalf of a job.

User ID (UID)

Privilege to access system resources and services is typically established by the *user id*, which is a numeric identifier uniquely assigned to each user (see User).

Job Array

A collection of jobs submitted under a single job id. These jobs can be modified, queried and displayed as a set.

Chapter 3

Getting Started With PBS

This chapter introduces the user to PBS Professional. It describes new user-level features in this release, explains the different user interfaces, introduces the concept of a PBS "job", and shows how to set up your environment for running batch jobs with PBS.

3.1 New Features in This Release

The path to the PBS binaries may have changed for your system. If the old path was not one of `/opt/pbs`, `/usr/pbs`, or `/usr/local/pbs`, you may need to add `/opt/pbs/default` to your PATH environment variable.

3.2 New Features in Recent Releases

PBS Professional has new features. The sort_priority option to job_sort_key is replaced with the job_priority option. The following is a list of recent new features and changes in PBS Professional.

3.2.1 Job-Specific Staging and Execution Directories (9.2)

PBS can now provide a staging and execution directory for each job. Jobs have new attributes `sandbox` and `jobdir`, the MOM has a new option `$jobdir_root`, and there is a new environment variable called `PBS_JOBDIR`. If the job's `sandbox` attribute is set to `PRIVATE`, PBS creates a job-specific staging and execution directory. If the job's `sandbox` attribute is unset or is set to `HOME`, PBS uses the user's home directory for staging and execution, which is how previous versions of PBS behaved. See section 8.7 "Input/Output File Staging" on page 167.

3.2.2 Standing Reservations (9.2)

PBS now provides a facility for making standing reservations. A standing reservation is a series of advance reservations. The pbs_rsub command is used to create both advance and standing reservations. See section 8.9 "Advance and Standing Reservation of Resources" on page 182.

3.3 Deprecations

The **sort_priority** option to job_sort_key is deprecated and is replaced with the job_priority option.

The **-l nodes=nodespec** form is replaced by the -l select= and -l place= statements.

The **nodes** resource is no longer used.

The **-l resource=rescspec** form is replaced by the -l select= statement.

The **time-shared** node type is no longer used, and

the **:ts** suffix is obsolete.

The **cluster** node type is no longer used.

The resource **arch** is only used inside of a select statement.

The resource **host** is only used inside of a select statement.

The **nodect** resource is obsolete. The ncpus resource should be used instead. Sites which currently have default values or limits based on nodect should change them to be based on ncpus.

The **neednodes** resource is obsolete.

The **ssinodes** resource is obsolete.

Properties are replaced by boolean resources.

The **ppn** resource is deprecated.

3.4 Using PBS

From the user's perspective, a workload management system allows you to make more efficient use of your time. You specify the tasks you need executed. The system takes care of running these tasks and returning the results to you. If the available computers are full, then the workload management system holds your work and runs it when the resources are available.

With PBS you create a *batch job* which you then submit to PBS. A batch job is a file (a shell script under UNIX or a `cmd` batch file under Windows) containing the set of commands you want to run on some set of execution machines. It also contains directives which specify the characteristics (attributes) of the job, and resource requirements (e.g. memory or CPU time) that your job needs. Once you create your PBS job, you can reuse it if you wish. Or, you can modify it for subsequent runs. For example, here is a simple PBS batch job:

UNIX:

```
#!/bin/sh
#PBS -l walltime=1:00:00
#PBS -l mem=400mb,ncpus=4
./my_application
```

Windows:

```
#PBS -l walltime=1:00:00
#PBS -l mem=400mb,ncpus=4
my_application
```

Don't worry about the details just yet; the next chapter will explain how to create a batch job of your own.

3.5 PBS Interfaces

PBS provides two user interfaces: a command line interface (CLI) and a graphical user interface (GUI). The CLI lets you type commands at the system prompt. The GUI is a graphical point-and-click interface. The "user commands" are discussed in this book; the "administrator commands" are discussed in the **PBS Professional Administrator's Guide**. The subsequent chapters of this book will explain how to use both the CLI and GUI versions of the user commands to create, submit, and manipulate PBS jobs.

.

Table 3-1: PBS Professional User and Manager Commands

User Commands		Administrator Commands	
Command	Purpose	Command	Purpose
nqs2pbs	Convert from NQS	pbs-report	Report job statistics
pbs_rdel	Delete a Reservation		
pbs_rstat	Status a Reservation	pbs_hostn	Report host name(s)

Table 3-1: PBS Professional User and Manager Commands

User Commands		Administrator Commands	
pbs_ password	Update per user / per server password[1]	pbs_migrate _users	Migrate per user / per server passwords [1]
pbs_rsub	Submit a Reservation	pbs_probe	PBS diagnostic tool
pbsdsh	PBS distributed shell	pbs_rcp	File transfer tool
qalter	Alter job	pbs_tclsh	TCL with PBS API
qdel	Delete job	pbsfs	Show fairshare usage
qhold	Hold a job	pbsnodes	Vnode manipulation
qmove	Move job	printjob	Report job details
qmsg	Send message to job	qdisable	Disable a queue
qorder	Reorder jobs	qenable	Enable a queue
qrls	Release hold on job	qmgr	Manager interface
qselect	Select jobs by criteria	qrerun	Requeue running job
qsig	Send signal to job	qrun	Manually start a job
qstat	Status job, queue, Server	qstart	Start a queue
qsub	Submit a job	qstop	Stop a queue
tracejob	Report job history	qterm	Shutdown PBS
xpbs	Graphical User Interface	xpbsmon	GUI monitoring tool

Notes:

1 Available on Windows only.

3.6 User's PBS Environment

In order to have your system environment interact seamlessly with PBS, there are several items that need to be checked. In many cases, your system administrator will have already set up your environment to work with PBS.

In order to use PBS to run your work, the following are needed:

- User must have access to the resources/hosts that the site has configured for PBS
- User must have a valid account (username and group) on the execution hosts
- User must be able to transfer files between hosts (e.g. via `rcp` or `scp`)
- User's time zone environment variable must be set correctly in order to use advance and standing reservations. See section 8.9.8.1 "Setting the Submission Host's Time Zone" on page 204.

The subsequent sections of this chapter discuss these requirements in detail, and provide various setup procedures.

3.7 Usernames Under PBS

By default PBS will use your login identifier as the username under which to run your job. This can be changed via the "`-u`" option to `qsub`. See section 4.13.14 "Specifying Job User ID" on page 87. The user submitting the job must be authorized to run the job under the execution user name (whether explicitly specified or not).

IMPORTANT:

PBS enforces a maximum username length of 15 characters. If a job is submitted to run under a username longer than this limit, the job will be rejected.

3.8 Setting Up Your UNIX/Linux Environment

A user's job may not run if the user's start-up files (i.e `.cshrc`, `.login`, or `.profile`) contain commands which attempt to set terminal characteristics. Any such command sequence within these files should be skipped by testing for the environment variable **`PBS_ENVIRONMENT`**. This can be done as shown in the following sample `.login`:

```
...
setenv MANPATH /usr/man:/usr/local/man:$MANPATH
if ( ! $?PBS_ENVIRONMENT ) then
```

You should also be aware that commands in your startup files should not generate output when run under PBS. As in the previous example, commands that write to stdout should not be run for a PBS job. This can be done as shown in the following sample `.login`:

```
setenv MANPATH \
 /usr/man:/usr/local/man:$MANPATH
if ( ! $?PBS_ENVIRONMENT ) then
      do terminal settings here
      run command with output here
endif
```

When a PBS job runs, the “exit status” of the last command executed in the job is reported by the job’s shell to PBS as the “exit status” of the job. (We will see later that this is important for job dependencies and job chaining.) However, the last command executed might not be the last command in your job. This can happen if your job’s shell is `csh` on the execution host and you have a `.logout` there. In that case, the last command executed is

from the .logout and not your job. To prevent this, you need to preserve the job's exit status in your .logout file, by saving it at the top, then doing an explicit exit at the end, as shown below:

```
set EXITVAL = $status
previous contents of .logout here
exit $EXITVAL
```

Likewise, if the user's login shell is csh the following message may appear in the standard output of a job:

Warning: no access to tty, thus no job control in this shell

This message is produced by many csh versions when the shell determines that its input is not a terminal. Short of modifying csh, there is no way to eliminate the message. Fortunately, it is just an informative message and has no effect on the job.

An interactive job comes complete with a pseudotty suitable for running those commands that set terminal characteristics. But more importantly, it does not caution the user that starting something in the background that would persist after the user has exited from the interactive environment might cause trouble for some moms. They could believe that once the interactive session terminates, all the user's processes are gone with it. For example, applications like ssh-agent background themselves into a new session and would prevent a CPU set-enabled mom from deleting the CPU set for the job. This in turn might cause subsequent failed attempts to run new jobs, resulting in them being placed in a held state.

3.8.1 Setting MANPATH on SGI Systems

The PBS "man pages" (UNIX manual entries) are installed on SGI systems under /usr/bsd, or for the Altix, in /usr/pbs/man. In order to find the PBS man pages, users will need to ensure that /usr/bsd is set within their MANPATH. The following example illustrates this for the C shell:

```
setenv MANPATH \
 /usr/man:/usr/local/man:/usr/ \
 bsd:$MANPATH
```

3.9 Setting Up Your Windows Environment

This section discusses the setup steps needed for running PBS Professional in a Microsoft Windows environment, including host and file access, passwords, and restrictions on home directories.

3.9.1 Windows User's HOMEDIR

Each Windows user is assumed to have a home directory (HOMEDIR) where his/her PBS jobs are initially started.

If a user has not been explicitly assigned a home directory, then PBS will use this Windows-assigned default as the base location for the user's default home directory. More specifically, the actual home path will be:

```
[PROFILE_PATH]\My Documents\PBS Pro
```

For instance, if a *userA* has not been assigned a home directory, it will default to a local home directory of:

```
\Documents and Settings\userA\My
 Documents\PBS Pro
```

UserA's job will use the above path as its working directory.

Note that Windows can return as PROFILE_PATH one of the following forms:

```
\Documents and Settings\username
\Documents and Settings\username.local-host
 name
\Documents and Settings\username.local-host
 name.00N
where N is a number
\Documents and Settings\username.domain-name
```

3.9.2 Windows Usernames and Job Submission

A PBS job is run from a user account and the associated username string must conform to the POSIX-1 standard for portability. That is, the username must contain only alphanumeric characters, dot (.), underscore (_), and/or hyphen "-". The hyphen must not be the first letter of the username. If "@" appears in the username, then it will assumed to be in the context of a Windows domain account: `username@domainname`. An exception to the above rule is the space character, which is allowed. If a space character appears in a username string, then it will be displayed quoted and must be specified in a quoted manner. The following example requests the job to run under account "Bob Jones".

```
qsub -u "Bob Jones" my_job
```

3.9.3 Windows rhosts File

The Windows `rhosts` file is located in the user's [`PROFILE_PATH`], for example: `\Documents and Settings\`*`username`*`\.rhosts`, with the format:

hostname username

IMPORTANT:

Be sure the `.rhosts` file is owned by user or an administrator-type group, and has write access granted only to the owning user or an administrator or group.

This file can also determine if a remote user is allowed to submit jobs to the local PBS Server, if the mapped user is an Administrator account. For example, the following entry in user `susan`'s `.rhosts` file on the server would permit user `susan` to run jobs submitted from her workstation wks031:

```
wks031 susan
```

Furthermore, in order for Susan's output files from her job to be returned to her automatically by PBS, she would need to add an entry to her `.rhosts` file on her workstation naming the execution host `Host1`.

```
Host1 susan
```

If instead, Susan has access to several execution hosts, she would need to add all of them to her `.rhosts` file:

```
Host1 susan
Host2 susan
Host3 susan
```

Note that Domain Name Service (DNS) on Windows may return different permutations for a full hostname, thus it is important to list all the names that a host may be known. For instance, if Host4 is known as "Host4", "Host4.<subdomain>", or "Host4.<subdomain>.<domain>" you should list all three in the `.rhosts` file.

```
Host4 susan
Host4.subdomain susan
Host4.subdomain.domain susan
```

As discussed in the previous section, usernames with embedded white space must also be quoted if specified in any `hosts.equiv` or `.rhosts` files, as shown below.

```
Host5.subdomain.domain "Bob Jones"
```

3.9.4 Windows Mapped Drives and PBS

In Windows XP, when you map a drive, it is mapped "locally" to your session. The mapped drive cannot be seen by other processes outside of your session. A drive mapped on one session cannot be un-mapped in another session even if it's the same user. This has implications for running jobs under PBS. Specifically if you map a drive, `chdir` to it, and submit a job from that location, the vnode that executes the job may not be able to deliver the files back to the same location from which you issued `qsub`. The workaround is to use the "`-o`" or "`-e`" options to `qsub` and specify a local (non-mapped) directory location for the job output and error files. For details see section 4.13.2 "Redirecting Output and Error Files" on page 81.

3.10 Environment Variables

There are a number of environment variables provided to the PBS job. Some are taken from the user's environment and carried with the job. Others are created by PBS. Still others can be explicitly created by the user for exclusive use by PBS jobs. All PBS-provided environment variable names start with the characters "PBS_". Some are then followed by a capital O ("PBS_O_") indicating that the variable is from the job's originating environment (i.e. the user's). Appendix A gives a full listing of all environment variables provided to PBS jobs and their meaning. The following short example lists some of the more useful variables, and typical values.

```
PBS_O_HOME=/u/user1
PBS_O_LOGNAME=user1
PBS_O_PATH=/usr/new/bin:/usr/local/bin:/bin
PBS_O_SHELL=/sbin/csh
PBS_O_HOST=cray1
PBS_O_WORKDIR=/u/user1
PBS_O_QUEUE=submit
PBS_JOBID=16386.cray1
PBS_QUEUE=crayq
PBS_ENVIRONMENT=PBS_INTERACTIVE
```

There are a number of ways that you can use these environment variables to make more efficient use of PBS. In the example above we see **PBS_ENVIRONMENT**, which we used earlier in this chapter to test if we were running under PBS. Another commonly used variable is **PBS_O_WORKDIR** which contains the name of the directory from which the user submitted the PBS job.

There are also two environment variables that you can set to affect the behavior of PBS. The environment variable **PBS_DEFAULT** defines the name of the default PBS Server. Typically, it corresponds to the system name of the host on which the Server is running. If **PBS_DEFAULT** is not set, the default is defined by an administrator established file (usually `/etc/pbs.conf` on UNIX, and `[PBS Destination Folder]\pbs.conf` on Windows).

The environment variable **PBS_DPREFIX** determines the prefix string which identifies directives in the job script. The default prefix string is "#PBS"; however the Windows user may wish to change this as discussed in section 4.11 "Changing the Job's PBS Directive" on page 74.

3.11 Temporary Scratch Space: TMPDIR

PBS creates an environment variable, TMPDIR, which contains the full path name to a temporary "scratch" directory created for each PBS job. The directory will be removed when the job terminates.

Under Windows, TMP will also be set to the value of %TMPDIR%. The temporary directory will be created under either \winnt\temp or \windows\temp, unless an alternative directory was specified by the administrator in the MOM configuration file.

Users can access the job-specific temporary space, by changing directory to it inside their job script. For example:

UNIX:

```
cd $TMPDIR
```

Windows:

```
cd %TMPDIR%
```

Chapter 4

Submitting a PBS Job

This chapter describes virtual nodes, how to submit a PBS job, how to use resources for jobs, how to place your job on vnodes, job attributes, and several related areas.

4.1 Vnodes: Virtual Nodes

A virtual node, or vnode, is an abstract object representing a set of resources which form a

usable part of a machine. This could be an entire host, or a nodeboard or a blade. A single host can be made up of multiple vnodes. Each vnode can be managed and scheduled independently. PBS views hosts as being composed of one or more vnodes. Jobs run on one or more vnodes. See the pbs_node_attributes(7B) man page.

4.1.1 Relationship Between Hosts, Nodes, and Vnodes

A host is any computer. Execution hosts used to be called nodes. However, some machines such as the Altix can be treated as if they are made up of separate pieces containing CPUs, memory, or both. Each piece is called a vnode. Some hosts have a single vnode and some have multiple vnodes. PBS treats all vnodes alike in most respects. Chunks cannot be split across hosts, but they can be split across vnodes on the same host.

Resources that are defined at the host level are applied to vnodes. A host-level resource is shared among the vnodes on that host. This sharing is managed by the MOM.

4.1.2 Vnode Types

What were called nodes are now called vnodes. All vnodes are treated alike, and are treated the same as what were called “time-shared nodes”. The types “time-shared” and “cluster” are deprecated. The `:ts` suffix is deprecated. It is silently ignored, and not preserved during rewrite. The vnode attribute `ntype` is only used to distinguish between PBS and Globus vnodes. It is read-only.

4.2 PBS Resources

Resources can be available on the server and queues, and on vnodes. Jobs can request resources. Resources are allocated to jobs, and some resources such as memory are consumed by jobs. The scheduler matches requested resources with available resources, according to rules defined by the administrator. PBS can enforce limits on resource usage by jobs.

PBS provides built-in resources, and in addition, allows the administrator to define custom resources. The administrator can specify which resources are available on a given vnode, as well as at the server or queue level (e.g.

floating licenses.) Vnodes can share resources. The administrator can also specify default arguments for qsub. These arguments can include resources. See the qsub(1B) man page.

Resources made available by defining them via resources_available at the server level are only used as job-wide resources. These resources (e.g. walltime, server_dyn_res) are requested using -l RESOURCE=VALUE. Resources made available at the host (vnode) level are only used as chunk resources, and can only be requested within chunks using -l select=RESOURCE=VALUE. Resources such as mem and ncpus can only be used at the vnode level.

Resources are allocated to jobs both by explicitly requesting them and by applying specified defaults. Jobs explicitly request resources either at the vnode level in chunks defined in a selection statement, or in job-wide resource requests. See the pbs_resources(7B) manual page.

Jobs are assigned limits on the amount of resources they can use. These limits apply to how much the job can use on each vnode (per-chunk limit) and to how much the whole job can use (job-wide limit). Limits are derived from both requested resources and applied default resources.

Each chunk's per-chunk limits determine how much of any resource can be used in that chunk. Per-chunk resource usage limits are the amount of per-chunk resources requested, both from explicit requests and from defaults.

Job resource limits set a limit for per-job resource usage. Job resource limits are derived in this order from:

1. explicitly requested job-wide resources (e.g. -l resource=value)
2. the select specification (e.g. -l select =...)
3. the queue’s default_resources.RES
4. the server’s default_resources.RES
5. the queue’s resources_max.RES
6. the server’s resources_max.RES

The server’s default_chunk.RES does **not** affect job-wide limits.

The resources requested for chunks in the select specification are summed, and this sum is used for a job-wide limit. Job resource limits from sums of all chunks override those from job-wide defaults and resource requests.

Various limit checks are applied to jobs. If a job's job resource limit exceeds queue or server restrictions, it will not be put in the queue or accepted by the server. If, while running, a job exceeds its limit for a consumable or time-based resource, it will be terminated.

A “consumable” resource is one that is reduced by being used, for example, ncpus, licenses, or mem. A “non-consumable” resource is not reduced through use, for example, walltime or a boolean resource.

Resources are tracked in server, queue, vnode and job attributes. Servers, queues and vnodes have two attributes, resources_available.RESOURCE and resources_assigned.RESOURCE. The resources_available.RESOURCE attribute tracks the total amount of the resource available at that server, queue or vnode, without regard to how much is in use. The resources_assigned.RESOURCE attribute tracks how much of that resource has been assigned to jobs at that server, queue or vnode. Jobs have an attribute called resources_used.RESOURCE which tracks the amount of that resource used by that job.

The administrator can set server and queue defaults for resources used in chunks. See the PBS Professional Administrator’s Guide and the pbs_server_attributes(7B) and pbs_queue_attributes(7B) manual pages.

4.2.0.1 Unset Resources

When job resource requests are being matched with available resources, a numerical resource that is unset on a host is treated as if it were zero, and an unset string cannot be matched. An unset Boolean resource is treated as if it is set to “False”. An unset resource at the server or queue is treated as if it were infinite.

4.2.0.2 Resource Names and Values

The resource name is any string made up of alphanumeric characters, where the first character is alphabetic. Resource names must start with an alphabetic character and can contain alphanumeric, underscore (“_”), and dash (“-”) characters.

If a string resource value contains spaces or shell metacharacters, enclose the string in quotes, or otherwise escape the space and metacharacters. Be sure to use the correct quotes for your shell and the behavior you want. If the string resource value contains commas, the string must be enclosed in

an additional set of quotes so that the command (e.g. qsub, qalter) will parse it correctly. If the string resource value contains quotes, plus signs, equal signs, colons or parentheses, the string resource value must be enclosed in yet another set of additional quotes.

4.2.1 Resource Types

Resources have the following data types:

boolean

Boolean-valued resource. Should be defined only at the vnode level, for manageability. Non-consumable. Name of resource is a string. Allowable values (case insensitive): True|T|Y|1|False|F|N|0

float

Float. Allowable values: [+-] 0-9 [[0-9] ...][.][[0-9] ...]

long

Long integer. Allowable values: 0-9 [[0-9] ...]

size

Number of bytes (default) or words. It is expressed in the form `integer[suffix]`. The suffix is a multiplier defined in the following table. The size of a word is the word size on the execution host.

Table 4-1:

b or w	bytes or words.
kb or kw	Kilo (2^{10}, 1024) bytes or words.
mb or mw	Mega (2^{20}, 1,048,576) bytes or words.
gb or gw	Giga (2^{30}, 1,073,741,824) bytes or words.
tb or tw	Tera (2^{40}, or 1024 gigabytes) bytes or words.

string

String. Non-consumable. Allowable values: Any printable character, including the space character., except the tab or other white space and the ampersand (“&”) character. The first character must be alphanumeric or underscore. Only one of the two types of quote characters, " or ', may appear in any given value.

Values:[_a-zA-Z0-9][[-_a-zA-Z0-9 ! " # $ % ´ ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ‘ { | } ~] ...]

string array

Comma-separated list of strings. Strings in string arrays may not contain commas. Non-consumable. Resource request will succeed if request matches one of the values. Resource request can contain only one string.

time

specifies a maximum time period the resource can be used. Time is expressed in seconds as an integer, or in the form:

[hours:]minutes:]seconds[.milliseconds]

4.2.2 Built-in Resources

The table below lists the built-in resources that can be requested by PBS jobs on any system.

Table 4-2: PBS Resources

Resource	Description
arch	System architecture. Can be requested only inside of a select statement. One architecture can be defined for a vnode. One architecture can be requested per vnode. Allowable values and effect on job placement are site-dependent. Type: string.
cput	Amount of CPU time used by the job for all processes on all vnodes. Establishes a job resource limit. Can be requested only outside of a select statement. Non-consumable. Type: time.

Table 4-2: PBS Resources

Resource	Description
file	Size of any single file that may be created by the job. Can be requested only outside of a select statement. Type: size.
host	Name of execution host. Can be requested only inside of a select statement. Automatically set to the short form of the hostname in the Mom attribute. Cannot be changed. Site-dependent. Type: string.
mem	Amount of physical memory i.e. workingset allocated to the job, either job-wide or vnode-level. Can be requested only inside of a select statement. Consumable. Type: size.
mpiprocs	Number of MPI processes for this chunk. Defaults to 1 if ncpus > 0, 0 otherwise. Can be requested only inside of a select statement. Type: integer. The number of lines in PBS_NODEFILE is the sum of the values of mpiprocs for all chunks requested by the job. For each chunk with mpiprocs=P, the host name for that chunk is written to the PBS_NODEFILE P times.
mpparch	MPP compute node system type. Can be requested only outside of a select statement. Allowable values: XT or X2. Type: string.
mppdepth	Depth (number of threads) of each processor. Specifies the number of processors that each processing element will use. Can be requested only outside of a select statement. Default: 1. Type: integer.
mpphost	MPP host. Can be requested only outside of a select statement. Type: string.

Table 4-2: PBS Resources

Resource	Description
mpplabels	List of node labels. Runs the application only on those nodes with the specified labels. Format: comma-separated list of labels and/or a range of labels. Any lists containing commas should be enclosed in quotes escaped by backslashes. For example: `#PBS -l mpplabels=\"red,blue\"` or **`qsub -l mpplabels=\"red,blue\"`** Can be requested only outside of a select statement. Type: string.
mppmem	The maximum memory for all applications. The per-processing-element maximum resident set size memory limit. Can be requested only outside of a select statement. Type: size.
mppnodes	Manual placement list consisting of a comma-separated list of nodes (node1,node2), a range of nodes (node1-node2), or a combination of both formats. Node values are expressed as decimal numbers. The first number in a range must be less than the second number (i.e., 8-6 is invalid). A complete node list is required. Any lists containing commas should be enclosed in quotes escaped by backslashes. For example: `#PBS -l mppnodes=\"40-48,52-60,84,86,88,90\"` or **`qsub -l mppnodes=\"40-48,52-60,84,86,88,90\"`** Can be requested only outside of a select statement. Type: integer.
mppnppn	Number of processing elements (PEs) per node. Can be requested only outside of a select statement. Type: integer.

Table 4-2: PBS Resources

Resource	Description
mppwidth	Number of processing elements (PEs) for the job. Can be requested only outside of a select statement. Type: integer.
ncpus	Number of processors requested. Cannot be shared across vnodes. Can be requested only inside of a select statement. Consumable. Type: integer.
nice	Nice value under which the job is to be run. Host-dependent. Can be requested only outside of a select statement. Type: integer.
nodect	Deprecated. Number of chunks in resource request from selection directive, or number of hosts requested from node specification. Otherwise defaults to value of 1. Can be requested only outside of a select statement. Read-only. Type: integer.
ompthreads	Number of OpenMP threads for this chunk. Defaults to ncpus if not specified. Can be requested only inside of a select statement. Type: integer. For the MPI process with rank 0, the environment variables NCPUS and OMP_NUM_THREADS are set to the value of ompthreads. For other MPI processes, behavior is dependent on MPI implementation.
pcput	Amount of CPU time allocated to any single process in the job. Establishes a job resource limit. Non-consumable. Can be requested only outside of a select statement. Type: time.
pmem	Amount of physical memory (workingset) for use by any single process of the job. Establishes a job resource limit. Can be requested only outside of a select statement. Consumable. Type: size
pvmem	Amount of virtual memory for use by the job. Establishes a job resource limit. Can be requested only outside of a select statement. Not consumable. Type: size.

Table 4-2: PBS Resources

Resource	Description
software	Site-specific software specification. Can be requested only outside of a select statement. Allowable values and effect on job placement are site-dependent. Type: string.
vmem	Amount of virtual memory for use by all concurrent processes in the job. Establishes a per-chunk limit. Can be requested only inside of a select statement. Consumable. Type: size.
vnode	Name of virtual node (vnode) on which to execute. For use inside chunks only. Site-dependent. Can be requested only inside of a select statement. Type: string. See the pbs_node_attributes(7B) man page.
walltime	Actual elapsed (wall-clock, except during Daylight Savings transitions) time during which the job can run. Establishes a job resource limit. Can be requested only outside of a select statement. Non-consumable. Default: 5 years. Type: time.

4.3 PBS Jobs

4.3.1 Rules for Submitting Jobs

- The "place" specification cannot be used without the "select" specification. See section 4.6 “Placing Jobs on Vnodes” on page 59.
- A "select" specification cannot be used with a "nodes" specification.
- A "select" specification cannot be used with old-style resource requests such as -lncpus, -lmem, -lvmem, -larch, -lhost.
- The built-in resource "software" is not a vnode-level resource. See “PBS Resources” on page 34.
- A PBS job can be submitted at the command line or via xpbs.
- At the command line, the user can create a job script, and submit it.

During submission it is possible to override elements in the job script. Alternatively, PBS will read from input typed at the command line.

4.3.2 PBS Job Script

A PBS job script consists of:

- An optional shell specification (UNIX)
- PBS directives
- Tasks -- programs or commands

To submit a PBS job, the user can type

```
qsub <name of script>
```

4.3.2.1 Specifying the Shell

UNIX Users:

Since the job file under UNIX is a "shell script", the first line of the job file specifies which shell to use to execute the script. The Bourne shell (`sh`) is the default, but you can change this to your favorite shell. This first line can be omitted if it is acceptable for the job file to be interpreted using the Bourne shell. The remainder of the examples in this manual will assume these conditions are true. If this is not true for your site, simply add the shell specifier.

Windows Users:

Windows does not use a shell specification. This line will not appear for a Windows job.

4.3.2.2 PBS Directives

PBS directives are at the top of the script file. They are used to request resources or set attributes. A directive begins with the default string "#PBS". Attributes can also be set using options to the qsub command, which will override directives.

4.3.2.3 The User's Tasks

These can be programs or commands. This is where the user specifies an application to be run.

IMPORTANT:

In Windows, if you use `notepad` to create a job script, the last line does not automatically get newline-terminated. Be sure to put one explicitly, otherwise, PBS job will get the following error message:

More?

when the Windows command interpreter tries to execute that last line.

4.3.3 Setting Job Attributes

Job attributes can be set either by using directives or by giving options to the qsub command. These two methods have the same functionality. Options to the qsub command will override PBS directives, which override defaults. Some job attributes have default values preset in PBS. Some job attributes' default values are set at the user's site.

4.4 Submitting a PBS Job

There are a few ways to submit a PBS job using the command line. The first is to create a job script and submit it using qsub.

4.4.1 Submitting a Job Script

For example, with job script "myjob", the user can submit it by typing

```
qsub myjob
16387.foo.exampledomain
```

PBS returns a *job identifier* (e.g. "16387.foo.exampledomain" in the example above.) Its format will be:

```
sequence-number.servername
```

or, for a job array,

```
sequence-number[].servername.domain
```

You'll need the job identifier for any actions involving the job, such as checking job status, modifying the job, tracking the job, or deleting the job.

If "my_job" contains the following, the user is naming the job "testjob", and running a program called "myprogram".

```
#!/bin/sh
#PBS -N testjob
./myprogram
```

The largest possible job ID is the 7-digit number 9,999,999. After this has been reached, job IDs start again at zero.

4.4.1.1 Overriding Directives

PBS directives in a script can be overridden by using the equivalent options to qsub. For example, to override the PBS directive naming the job, and name it "newjob", the user could type

```
qsub -N newjob my_job
```

4.4.1.2 Submitting a Simple Job

Jobs can also be submitted without specifying values for attributes. The simplest way to submit a job is to type

```
qsub myjobscript <ret>
```

If `myjobscript` contains

```
#!/bin/sh
./myapplication
```

the user has simply told PBS to run myapplication.

4.4.1.3 Jobs Without a Job Script

It is possible to submit a job to PBS without first creating a job script file. If you run the `qsub` command, with the resource requests on the command line, and then press "enter" without naming a job file, PBS will read input from the keyboard. (This is often referred to as a "here document".) You can direct `qsub` to stop reading input and submit the job by typing on a line by itself a `control-d` (UNIX) or `control-z`, then `enter` (Windows).

Note that, under UNIX, if you enter a `control-c` while `qsub` is reading input, `qsub` will terminate the process and the job will not be submitted. Under Windows, however, often the `control-c` sequence will, depending on the command prompt used, cause `qsub` to submit the job to PBS. In such case, a `control-break` sequence will usually terminate the `qsub` command.

```
qsub <ret>
    [directives]
    [tasks]
    ctrl-D
```

4.4.1.4 Passing Arguments to Job Scripts

If you need to pass arguments to a job script, you can either use the -v option to qsub, where you set and use environment variables, or use standard input. When using standard input, any #PBS directives in the job script will be ignored. You can replace directives with the equivalent options to qsub. To use standard input, you can either use this form:

```
echo "jobscript.sh -a foo -b bar" | qsub -l select=...
```

or you can use this form:

```
qsub [option] [option] ... <ret>
./jobscript.sh foo      <^d>
152.mymachine
```

With this form, you can type the #PBS directives on lines the name of the job script. If you do not use the -n option to qsub, or specify it via a #PBS directive (second form only), the job will be named STDIN.

4.5 Requesting Resources

PBS provides built-in resources, and allows the administrator to define custom resources. The administrator can specify which resources are available on a given vnode, as well as at the queue or server level (e.g. floating licenses.) See “PBS Resources” on page 34 for a listing of built-in resources.

Resources defined at the queue or server level apply to an entire job. If they are defined at the vnode level, they apply only to the part of the job running on that vnode.

Jobs request resources, which are allocated to the job, along with any defaults specified by the administrator.

Custom resources are used for application licenses, scratch space, etc., and are defined by the administrator. See “Customizing PBS Resources” on page 255 of the PBS Professional Administrator’s Guide. Custom resources are used the same way built-in resources are used.

Jobs request resources in two ways. They can use the *select statement* to define *chunks* and specify the quantity of each chunk. A chunk is a set of resources that are to be allocated as a unit. Jobs can also use a job-wide resource request, which uses `resource=value` pairs, outside of the select statement.

The qsub, qalter and pbs_rsub commands are used to request resources. However, custom resources which were created to be invisible or unrequestable cannot be requested. See section 4.5.14 “Resource Permissions” on page 58.

The `-l nodes=` form is deprecated, and if it is used, it will be converted into a request for chunks and job-wide resources. Most jobs submitted with "-lnodes" will continue to work as expected. These jobs will be automatically converted to the new syntax. However, job tasks may execute in an unexpected order, because vnodes may be assigned in a different order. Jobs submitted with old syntax that ran successfully on versions of PBS Professional prior to 8.0 can fail because a limit that was per-chunk is now job-wide. This is an example of a job submitted using -l nodes=X -lmem=M that would fail because the mem limit is now job-wide. If the following conditions are true:

a. PBS Professional 9.0 or later using standard MPICH

b. The job is submitted with qsub -lnodes=5 -lmem=10GB

c. The master process of this job tries to use more than 2GB

The job will be killed, where in <= 7.0 the master process could use 10GB before being killed. 10GB is now a job-wide limit, divided up into a 2GB limit per chunk.

For more information see the `qsub(1B)`, `qalter(1B)`, `pbs_rsub(1B)` and `pbs_resources(7B)` manual pages.

Do not use an old-style resource or node specification ("-lnodes=") with "-lselect" or "-lplace". This will produce an error.

Each kind of resource plays a specific role, which is either inside chunks or outside of them, but not both. Some resources, e.g. ncpus, can only be used at the host (chunk) level. The rest, e.g. walltime, can only be used at the job-wide level. Therefore, no resource can be requested both inside and outside of a selection statement. Keep in mind that requesting, for example, -lncpus is the old form, which cannot be mixed with the new form.

4.5.1 Allocation

Resources are allocated to jobs both because jobs explicitly request them and because specified default resources are applied to jobs. Jobs explicitly request resources either at the vnode level in *chunks* defined in a *selection statement*, or in *job-wide* resource requests, outside of a selection statement. An explicit resource request can appear in the following, in order of precedence:

1. qalter
2. qsub
3. PBS job script directives

4.5.2 Requesting Resources in Chunks

A *chunk* declares the value of each resource in a set of resources which are to be allocated as a unit to a job. It is the smallest set of resources that will be allocated to a job. All of a chunk must be taken from a single host. A chunk request is a vnode-level request. Chunks are described in a selection statement, which specifies how many of each kind of chunk. A selection statement has this form:

-l select=[N:]chunk[+[N:]chunk ...]

If N is not specified, it is taken to be 1.

A chunk is one or more resource_name=value statements separated by a colon, e.g.:

```
ncpus=2:mem=10GB:host=Host1
ncpus=1:mem=20GB:arch=linux
```

Example of multiple chunks in a selection statement:

```
-l select= 2:ncpus=1:mem=10GB
   +3:ncpus=2:mem=8GB:arch=solaris
```

Each job submission can have only one "-l select" statement.

Host-level resources can only be requested as part of a chunk. Server or queue resources cannot be requested as part of a chunk. They must be requested outside of the selection statement.

4.5.3 Requesting Job-wide Resources

A *job-wide* resource request is for resource(s) at the server or queue level. Job-wide resources are requested outside of a selection statement, in this form:

-l keyword=value[,keyword=value ...]

where *keyword* identifies either a consumable resource or a time-based resource such as walltime.

Job-wide resources are used for requesting floating licenses or other resources not tied to specific vnodes, such as cput and walltime.

Job-wide resources can only be requested outside of chunks.

4.5.4 Boolean Resources

A resource request can specify whether a boolean resource should be true or false. For example, if some vnodes have green=true and some are red=true, a selection statement for two vnodes, each with one CPU, all green and no red, would be:

```
-l select=2:green=true:red=false:ncpus=1
```

The next example Windows script shows a job-wide request for walltime and a chunk request for ncpus and memory.

```
#PBS -l walltime=1:00:00
#PBS -l select=ncpus=4:mem=400mb
#PBS -j oe

date /t
.\my_application
date /t
```

Keep in mind the difference between requesting a vnode-level boolean and a job-wide boolean.

```
qsub -l select=1:green=True
```

will request a vnode with green set to True. However,

```
qsub -l green=True
```

will request green set to True on the server and/or queue.

4.5.5 Default Resources

Jobs get default resources, both job-wide and per-chunk, with the following order of precedence, from

1. Default `qsub` arguments
2. Default queue resources
3. Default server resources

For each chunk in the job's selection statement, first queue chunk defaults are applied, then server chunk defaults are applied. If the chunk request does not specify a resource listed in the defaults, the default is added. For a resource RESOURCE, a chunk default is called "default_chunk.RESOURCE".

For example, if the queue in which the job is enqueued has the following defaults defined:

```
default_chunk.ncpus=1
default_chunk.mem=2gb
```

a job submitted with this selection statement:

```
select=2:ncpus=4+1:mem=9gb
```

will have this specification after the default_chunk elements are applied:

```
select=2:ncpus=4:mem=2gb+1:ncpus=1:mem=9gb.
```

In the above, *mem=2gb* and *ncpus=1* are inherited from default_chunk.

The job-wide resource request is checked against queue resource defaults, then against server resource defaults. If a default resource is defined which is not specified in the resource request, it is added to the resource request.

4.5.6 Requesting Application Licenses

Application licenses are set up as resources defined by the administrator. PBS doesn't actually check out the licenses, the application being run inside the job's session does that.

4.5.6.1 Floating Licenses

PBS queries the license server to find out how many floating licenses are available at the beginning of each scheduling cycle. If you wish to request a site-wide floating license, it will typically have been set up as a server-level (job-wide) resource. To request an application license called AppF, use:

```
qsub -l AppF=<number of licenses> <other qsub
        arguments>
```

If only certain hosts can run the application, they will typically have a host-level boolean resource set to True. To request the application license and the vnodes on which to run the application, use:

```
qsub  -l AppF=<number of licenses>
         <other qsub arguments>
      -l select=haveAppF=True
```

PBS doesn't actually check out the licenses, the application being run inside the job's session does that.

4.5.6.2 Node-locked Licenses

Per-host node-locked licenses are typically set up as either a boolean resource on the vnode(s) that are licensed for the application. The resource request should include one license for each host. To request a host with a per-host node-locked license for AppA in one chunk:

```
qsub -l select=1:runsAppA=1 <jobscript>
```

Per-use node-locked licenses are typically set up so that the host(s) that run the application have the number of licenses that can be used at one time. The number of licenses the job requests should be the same as the number of instances of the application that will be run. To request a host with a per-use node-locked license for AppB, where you'll run one instance of AppB on two CPUs in one chunk:

```
qsub -l select=1:ncpus=2:AppB=1
```

Per-CPU node-locked licenses are set up so that the host has one license for each licensed CPU. You must request one license for each CPU. To request a host with a node-locked license for AppC, where you'll run a job using two CPUs in one chunk:

```
qsub -l select=1:ncpus=2:AppC=2
```

4.5.7 Requesting Scratch Space

Scratch space on a machine is set up as a host-level dynamic resource. The resource will have a name such as "dynscratch". To request 10MB of scratch space in one chunk, a resource request would include:

```
-l select=1:ncpus=N:dynscratch=10MB
```

4.5.8 Note About Submitting Jobs

The default for walltime is 5 years. The scheduler uses walltime to predict when resources will become available. Therefore it is useful to request a reasonable walltime for each job.

4.5.9 Submitting Jobs with Resource Specification (Old Syntax)

If neither a node specification nor a selection directive is specified, then a selection directive will be created requesting 1 chunk with resources specified by the job, and with those from the queue or server default resource list. These are: ncpus, mem, arch, host, and software, as well as any other default resources specified by the administrator.

For example, a job submitted with

```
qsub -l ncpus=4:mem=123mb:arch=linux
```

will have the following selection directive created:

```
select=1:ncpus=4:mem=123mb:arch=linux
```

Do not mix old style resource or node specification with the select and place statements. Do not use one in a job script and the other on the command line. This will result in an error.

4.5.10 Moving Jobs From One Queue to Another

If the job is moved from the current queue to a new queue, any default resources in the job's resource list that were contributed by the current queue are removed. This includes a select specification and place directive generated by the rules for conversion from the old syntax. If a job's resource is unset (undefined) and there exists a default value at the new queue or server, that default value is applied to the job's resource list. If either select or place is missing from the job's new resource list, it will be automatically generated, using any newly inherited default values.

Example:

Given the following set of queue and server default values:

Server

```
resources_default.ncpus=1
```

Queue QA

resources_default.ncpus=2

default_chunk.mem=2gb

Queue QB

default_chunk.mem=1gb

no default for ncpus

The following illustrate the equivalent select specification for jobs submitted into queue QA and then moved to (or submitted directly to) queue QB:

```
qsub -l ncpus=1 -lmem=4gb
```

In QA: `select=1:ncpus=1:mem=4gb`

- No defaults need be applied

In QB: `select=1:ncpus=1:mem=4gb`

- No defaults need be applied

`qsub -l ncpus=1`

In QA: `select=1:ncpus=1:mem=2gb`

- Picks up 2gb from queue default chunk and 1 ncpus from qsub

In QB: `select=1:ncpus=1:mem=1gb`

- Picks up 1gb from queue default chunk and 1 ncpus from qsub

`qsub -lmem=4gb`

In QA: `select=1:ncpus=2:mem=4gb`

- Picks up 2 ncpus from queue level job-wide resource default and 4gb mem from qsub

In QB: `select=1:ncpus=1:mem=4gb`

- Picks up 1 ncpus from server level job-wide default and 4gb mem from qsub

`qsub -l nodes=4`

In QA: `select=4:ncpus=1:mem=2gb`

- Picks up a queue level default memory chunk of 2gb.

(This is not 4:ncpus=2 because in prior versions, "nodes=x"
implied 1 CPU per node unless otherwise explicitly stated.)

In QB: `select=4:ncpus=1:mem=1gb`
(In prior versions, "nodes=x" implied 1 CPU per node unless
otherwise explicitly stated, so the ncpus=1 is not inherited
from the server default.)

`qsub -l mem=16gb -l nodes=4`

In QA: `select=4:ncpus=1:mem=4gb`
(This is not 4:ncpus=2 because in prior versions, "nodes=x"
implied 1 CPU per node unless otherwise explicitly stated.)

In QB: `select=4:ncpus=1:mem=4gb`
(In prior versions, "nodes=x" implied 1 CPU per node unless
otherwise explicitly stated, so the ncpus=1 is not inherited
from the server default.)

4.5.11 Resource Request Conversion Dependent on Where Resources are Defined

A job's resource request is converted from old-style to new according to various rules, one of which is that the conversion is dependent upon where resources are defined. For example: The boolean resource "Red" is defined on the server, and the boolean resource "Blue" is defined at the host level. A job requests "qsub -l Blue=True". This looks like an old-style resource request, and PBS checks to see where Blue is defined. Since Blue is defined at the host level, the request is converted into "-l select=1:Blue=True". However, if a job requests "qsub -l Red=True", while this looks like an old-style resource request, PBS does not convert it to a chunk request because Red is defined at the server.

4.5.12 Jobs Submitted with Undefined Resources

Any job submitted with undefined resources, specified either with "-l select" or with "-l nodes", will not be rejected at submission. The job will be aborted upon being enqueued in an execution queue if the resources are still undefined. This preserves backward compatibility.

4.5.13 Limits on Resource Usage

Each chunk's per-chunk limits determine how much of any resource can be used in that chunk. Per-chunk resource usage limits are established by per-chunk resources, both from explicit requests and from defaults.

Job resource limits set a limit for per-job resource usage. Job resource limits are established both by requesting job-wide resources and by summing per-chunk consumable resources. Job resource limits from sums of all chunks, including defaults, override those from job-wide defaults. Limits include both explicitly requested resources and default resources.

If a job's job resource limit exceeds queue or server restrictions, it will not be put in the queue or accepted by the server. If, while running, a job exceeds its limit for a consumable or time-based resource, it will be terminated. See The **PBS Professional Administrator's Guide**.

Job limits are created from the directive for each consumable resource.

For example,

```
qsub -lselect=2:ncpus=3:mem=4gb:arch=linux
```

will have the following job limits set:

ncpus=6 and mem=8gb

4.5.14 Resource Permissions

Custom resources can be created so that they are invisible, or cannot be requested or altered. If a resource is invisible it also cannot be requested or altered. The function of some PBS commands depends upon whether a resource can be viewed, requested or altered. These commands are those which view or request resources or modify resource requests:

pbsnodes
: Users cannot view restricted host-level custom resources.

pbs_rstat
: Users cannot view restricted reservation resources.

pbs_rsub
: Users cannot request restricted custom resources for reservations.

qalter
: Users cannot alter a restricted resource.

qmgr
: Users cannot print or list a restricted resource.

qselect
: Users cannot specify restricted resources via -l resource_list.

qsub
: Users cannot request a restricted resource.

qstat

Users cannot view a restricted resource.

4.6 Placing Jobs on Vnodes

The *place statement* controls how the job is placed on the vnodes from which resources may be allocated for the job. The place statement can be specified, in order of precedence, via:

1. Explicit placement request in qalter
2. Explicit placement request in qsub
3. Explicit placement request in PBS job script directives
4. Default `qsub` place statement
5. Queue default placement rules
6. Server default placement rules
7. Built-in default conversion and placement rules

The place statement may be not be used without the select statement.

The place statement has this form:

-l place=[arrangement][: sharing][: grouping]

where

arrangement is one of `free | pack | scatter`

sharing is one of `excl | shared`

grouping can have only one instance of `group=resource`

and where

Table 4-3: Placement Modifiers

Modifier	Meaning
free	Place job on any vnode(s).

Table 4-3: Placement Modifiers

Modifier	Meaning
pack	All chunks will be taken from one host.
scatter	Only one chunk will be taken from a host.
exclusive	Only this job uses the vnodes chosen.
shared	This job can share the vnodes chosen.
group=resource	Chunks will be grouped according to a resource. All vnodes in the group must have a common value for the resource, which can be either the built-in resource host or a site-defined vnode-level resource.

Note that vnodes can have sharing attributes that override job placement requests. See the `pbs_node_attributes(7B)` man page.

Grouping by resource name will override `node_group_key`. To run a job on a single host, use "-lplace=pack".

4.6.1 Vnodes Allocated to a Job

The nodes file contains the names of the vnodes allocated to a job. The nodes file's name is given by the environment variable PBS_NODEFILE. The order in which hosts appear in the file is the order in which chunks are specified in the selection directive. The order in which hostnames appear in the file is hostA X times, hostB Y times, where X is the number of MPI processes on hostA, Y is the number of MPI processes on hostB, etc. See the definition of the resources "mpiprocs" and "ompthreads" in "PBS Resources" on page 34. See also "The mpiprocs Resource" on page 237.

4.6.2 PBS_NODEFILE

The file containing the vnodes allocated to a job lists vnode names. This file's name is given by the environment variable PBS_NODEFILE. For jobs which request vnodes via the -lselect= option, the nodes file will contain the names of the allocated vnodes with each name repeated M times, where M is the number of mpiprocs specified for that vnode. For example,

```
qsub -l select=3:ncpus=2 -lplace=scatter
```

will result in this PBS_NODEFILE:

```
vnodeA
vnodeB
vnodeC
```

And

```
qsub -l select=3:ncpus=2:mpiprocs=2
```

will result in this PBS_NODEFILE:

```
vnodeA
vnodeA
vnodeB
vnodeB
vnodeC
vnodeC
```

For jobs which requested a set of nodes via the -lnodes=nodespec option to qsub, each vnode allocated to the job will be listed N times, where N is the total number of CPUs allocated from the vnode divided by the number of threads requested. For example, qsub -lnodes=4:ncpus=3:ppn=2 will result in each of the four vnodes being written twice (6 CPUs divided by 3 from ncpus.) The file will contain the name of the first vnode twice, followed by the second vnode twice, etc.

4.6.3 Resources Allocated from a Vnode

The resources allocated from a vnode are only those specified in the job's *schedselect*. This job attribute is created internally by starting with the select specification and applying any server and queue default_chunk resource defaults that are missing from the select statement. The schedselect job attribute contains only vnode-level resources. The exec_vnode job attribute shows which resources are allocated from which vnodes.

4.6.3.1 Resources Assigned to a Job

The Resource_List attribute is the list of resources requested via qsub, with job-wide defaults applied. Vnode-level resources from Resource_List are used in the converted select when the user doesn't specify a select statement. The converted select statement is used to fill in gaps in schedselect.

Values for ncpus or mem in the job's Resource_List come from three places:

(1) Resources specified via qsub,

(2) the sum of the values in the select specification (not including default_chunk), or

(3) resources inherited from queue and/or server resources_default.

Case 3 applies only when the user does not specify -l select, but uses -lnodes or -lncpus instead.

The Resource_List.mem is a job-wide memory limit which, if memory enforcement is enabled, the entire job (the sum of all of the job's usage) cannot exceed.

Examples:

The queue has the following:

```
resources_default.mem=200mb
default_chunk.mem=100mb
```

A job requesting -l select=2:ncpus=1:mem=345mb will take 345mb from each of two vnodes and have a job-wide limit of 690mb (2 * 345). The job's Resource_List.mem will show 690mb.

A job requesting -l select=2:ncpus=2 will take 100mb (default_chunk) value from each vnode and have a job wide limit of 200mb (2 * 100mb). The job's Resource_List.mem will show 200mb.

A job requesting -l ncpus=2 will take 200mb (inherited from resources_default and used to create the select spec) from one vnode and a job-wide limit of 200mb. The job's Resource_List.mem will show 200mb.

A job requesting -l nodes=2 will inherit the 200mb from resources_default.mem which will be the job-wide limit. The memory will be taken from the two vnodes, half (100mb) from each. The generated select spec is 2:ncpus=1:mem=100mb. The job's Resource_List.mem will show 200mb.

4.7 Submitting Jobs Using Select & Place: Examples

Unless otherwise specified, the vnodes allocated to the job will be allocated as shared or exclusive based on the setting of the vnode's sharing attribute. Each of the following shows how you would use -l select= and -l place=.

1. A job that will fit in a single host such as an Altix but not in any of the vnodes, packed into the fewest vnodes:

   -l select=1:ncpus=10:mem=20gb

   -l place=pack

   In earlier versions, this would have been:

   -lncpus=10,mem=20gb

2. Request four chunks, each with 1 CPU and 4GB of memory taken from anywhere.

   -l select=4:ncpus=1:mem=4GB

   -l place=free

3. Allocate 4 chunks, each with 1 CPU and 2GB of memory from between one and four vnodes which have an arch of "linux".

-l select=4:ncpus=1:mem=2GB:arch=linux -l place=free

4. Allocate four chunks on 1 to 4 vnodes where each vnode must have 1 CPU, 3GB of memory and 1 node-locked dyna license available for each chunk.

   -l select=4:dyna=1:ncpus=1:mem=3GB -l place=free

5. Allocate four chunks on 1 to 4 vnodes, and 4 floating dyna licenses. This assumes “dyna” is specified as a server dynamic resource.

   -l dyna=4 -l select=4:ncpus=1:mem=3GB -l place=free

6. This selects exactly 4 vnodes where the arch is linux, and each vnode will be on a separate host. Each vnode will have 1 CPU and 2GB of memory allocated to the job.

   -lselect=4:mem=2GB:ncpus=1:arch=linux -lplace=scatter

7. This will allocate 3 chunks, each with 1 CPU and 10GB of memory. This will also reserve 100mb of scratch space if scratch is to be accounted . Scratch is assumed to be on a file system common to all hosts. The value of “place” depends on the default which is “place=free”.

   -l scratch=100mb -l select=3:ncpus=1:mem=10GB

8. This will allocate 2 CPUs and 50GB of memory on a host named zooland. The value of “place” depends on the default which defaults to “place=free”:

   -l select=1:ncpus=2:mem=50gb:host=zooland

9. This will allocate 1 CPU and 6GB of memory and one host-locked swlicense from each of two hosts:

   -l select=2:ncpus=1:mem=6gb:swlicense=1

   -lplace=scatter

10. Request free placement of 10 CPUs across hosts:

    -l select=10:ncpus=1

    -l place=free

11. Here is an odd-sized job that will fit on a single Altix, but not on any one node-board. We request an odd number of CPUs that are not

shared, so they must be "rounded up":

-l select=1:ncpus=3:mem=6gb

-l place=pack:excl

12. Here is an odd-sized job that will fit on a single Altix, but not on any one node-board. We are asking for small number of CPUs but a large amount of memory:

-l select=1:ncpus=1:mem=25gb

-l place=pack:excl

13. Here is a job that may be run across multiple Altix systems, packed into the fewest vnodes:

-l select=2:ncpus=10:mem=12gb

-l place=free

14. Submit a job that must be run across multiple Altix systems, packed into the fewest vnodes:

-l select=2:ncpus=10:mem=12gb

-l place=scatter

15. Request free placement across nodeboards within a single host:

-l select=1:ncpus=10:mem=10gb

-l place=group=host

16. Request free placement across vnodes on multiple Altixes:

-l select=10:ncpus=1:mem=1gb

-l place=free

17. Here is a small job that uses a shared cpuset:

-l select=1:ncpus=1:mem=512kb

-l place=pack:shared

18. Request a special resource available on a limited set of nodeboards, such as a graphics card:

-l select= 1:ncpus=2:mem=2gb:graphics=True

+ 1:ncpus=20:mem=20gb:graphics=False

-l place=pack:excl

19. Align SMP jobs on c-brick boundaries:

-l select=1:ncpus=4:mem=6gb

-l place=pack:group=cbrick

20. Align a large job within one router, if it fits within a router:

-l select=1:ncpus=100:mem=200gb

-l place=pack:group=router

21. Fit large jobs that do not fit within a single router into as few available routers as possible. Here, RES is the resource used for node grouping:

-l select=1:ncpus=300:mem=300gb

-l place=pack:group=<RES>

22. To submit an MPI job, specify one chunk per MPI task. For a 10-way MPI job with 2gb of memory per MPI task:

-l select=10:ncpus=1:mem=2gb

23. To submit a non-MPI job (including a 1-CPU job or an OpenMP or shared memory) job, use a single chunk. For a 2-CPU job requiring 10gb of memory:

-l select=1:ncpus=2:mem=10gb

4.7.1 Examples Using Old Syntax

1. Request CPUs and memory on a single host using old syntax:

-l ncpus=5,mem=10gb

will be converted into the equivalent:

-l select=1:ncpus=5:mem=10gb

-l place=pack

2. Request CPUs and memory on a named host along with custom resources including a floating license using old syntax:

-l ncpus=1,mem=5mb,host=sunny,opti=1,arch=solaris

is converted to the equivalent:

-l select=1:ncpus=1:mem=5gb:host=sunny:arch=solaris

-l place=pack

-l opti=1

3. Request one host with a certain property using old syntax:

-lnodes=1:property

is converted to the equivalent:

-l select=1:ncpus=1:property=True

-l place=scatter

4. Request 2 CPUs on each of four hosts with a given property using old syntax:

-lnodes=4:property:ncpus=2

is converted to the equivalent:

-l select=4: ncpus=2:property=True

-l place=scatter

5. Request 1 CPU on each of 14 hosts asking for certain software, licenses and a job limit amount of memory using old syntax:

-lnodes=14:mpi-fluent:ncpus=1 -lfluent=1,fluent-all=1,

fluent-par=13

-l mem=280mb

is converted to the equivalent:

-l select=14:ncpus=1:mem=20mb:mpi_fluent=True

-l place=scatter

-l fluent=1,fluent-all=1,fluent-par=13

6. Requesting licenses using old syntax:

-lnodes=3:dyna-mpi-Linux:ncpus=2 -ldyna=6,mem=100mb,

software=dyna

is converted to the equivalent:

-l select=3:ncpus=2:mem=33mb: dyna-mpi-Linux=True

-l place=scatter

-l software=dyna

-l dyna=6

7. Requesting licenses using old syntax:

-l ncpus=2,app_lic=6,mem=200mb -l software=app

is converted to the equivalent:

-l select=1:ncpus=2:mem=200mb

-l place=pack

-l software=app

-l app_lic=6

8. Additional example using old syntax:

-lnodes=1:fserver+15:noserver

is converted to the equivalent:

-l select=1:ncpus=1:fserver=True + 15:ncpus=1:noserver=True

-l place=scatter

but could also be more easily specified with something like:

-l select=1:ncpus=1:fserver=True + 15:ncpus=1:fserver=False

-l place=scatter

9. Allocate 4 vnodes, each with 6 CPUs with 3 MPI processes per vnode, with each vnode on a separate host. The memory allocated would be one-fourth of the memory specified by the queue or server default if one existed. This results in a different placement of the job from version 5.4:

-l nodes=4:ppn=3:ncpus=2

is converted to:

-l select=4:ncpus=6:mpiprocs=3 -l place=scatter

10. Allocate 4 vnodes, from 4 separate hosts, with the property blue. The amount of memory allocated from each vnode is 2560MB (= 10GB / 4) rather than 10GB from each vnode.

-l nodes=4:blue:ncpus=2 -l mem=10GB

is converted to:

-l select=4:blue=True:ncpus=2:mem=2560mb \

-lplace=scatter

4.8 Backward Compatibility

For backward compatibility, a legal node specification or resource specification will be converted into selection and placement directives. Specifying "cpp" is part of the old syntax, and should be replaced with "ncpus". Do not mix old style resource or node specification syntax with select and place statements. If a job is submitted using -l select on the command line, and it contains an old-style specification in the job script, that will result in an error.

When a nodespec is converted into a select statement, the job will have the environment variables NCPUS and OMP_NUM_THREADS set to the value of ncpus in the first piece of the nodespec. This may produce incompatibilities with prior versions when a complex node specification using different values of ncpus and ppn in different pieces is converted.

4.8.1 Node Specification Conversion

Node specification format:

-lnodes=[N:*spec_list* | *spec_list*]

[[+N:*spec_list* | +*spec_list*] ...]

[#*suffix* ...][-lncpus=Z]

where:

spec_list has syntax: *spec*[:*spec* ...]

spec is any of: hostname | property | ncpus=X | cpp=X | ppn=P

suffix is any of: property | excl | shared

N and P are positive integers

X and Z are non-negative integers

The node specification is converted into selection and placement directives as follows:

Each *spec_list* is converted into one chunk, so that N:*spec_list* is converted into N chunks.

If spec is hostname :

The chunk will include host=hostname

If spec matches any vnode's resources_available.host value:

The chunk will include host=hostname

If spec is property :

The chunk will include property=true

Property must be a site-defined vnode-level boolean resource.

If spec is ncpus=X or cpp=X :

The chunk will include ncpus=X

If no spec is ncpus=X and no spec is cpp=X :

The chunk will include ncpus=P

If spec is ppn=P :

The chunk will include mpiprocs=P

If the nodespec is

```
-lnodes=N:ppn=P
```

It is converted to

-lselect=N:ncpus=P:mpiprocs=P

Example:

-lnodes=4:ppn=2

is converted into

-lselect=4:ncpus=2:mpiprocs=2

If -lncpus=Z is specified and no spec contains ncpus=X and no spec is cpp=X :

Every chunk will include ncpus=W,

where W is Z divided by the total number of chunks.

(Note: W must be an integer; Z must be evenly divisible by the number of chunks.)

If property is a suffix :

All chunks will include property=true

If excl is a suffix :

The placement directive will be -lplace=scatter:excl

If shared is a suffix :

The placement directive will be -lplace=scatter:shared

If neither excl nor shared is a suffix :

The placement directive will be -lplace=scatter

Example:

-l nodes=3:green:ncpus=2:ppn=2+2:red

is converted to:

-l select=3:green=true:ncpus=4:mpiprocs=2+2 \

:red=true:ncpus=1

-l place=scatter

Node specification syntax for requesting properties is deprecated. The boolean resource syntax "property=true" is only accepted in a selection directive. It is erroneous to mix old and new syntax.

4.8.2 Resource Specification Conversion

The resource specification is converted to select and place statements after any defaults have been applied.

Resource specification format:

-l resource=value[:resource=value ...]

The resource specification is converted to:

-lselect=1[:*resource=value* ...]

-lplace=pack

with one instance of *resource=value* for each of the following vnode-level resources in the resource request:

built-in resources: ncpus | mem | vmem | arch | host

site-defined vnode-level resources

4.9 How PBS Parses a Job Script

The `qsub` command scans the lines of the script file for directives. Scanning will continue until the first executable line, that is, a line that is not blank, not a directive line, nor a line whose first non white space character is "`#`". If directives occur on subsequent lines, they will be ignored.

A line in the script file will be processed as a directive to `qsub` if and only if the string of characters starting with the first non white space character on the line and of the same length as the directive prefix matches the directive prefix (i.e. "`#PBS`"). The remainder of the directive line consists of the options to `qsub` in the same syntax as they appear on the command line. The option character is to be preceded with the "`-`" character.

If an option is present in both a directive and on the command line, that option and its argument, if any, will be ignored in the directive. The command line takes precedence. If an option is present in a directive and not on the command line, that option and its argument, if any, will be taken from there.

4.10 A Sample PBS Job

Let's look at an example PBS job in detail:

UNIX:

```
#!/bin/sh
#PBS -l walltime=1:00:00
#PBS -l select=mem=400mb
#PBS -j oe

date
./my_application
date
```

Windows:

```
#PBS -l walltime=1:00:00
#PBS -l select=mem=400mb
#PBS -j oe

date /t
my_application
date /t
```

On line one in the example above Windows does not show a shell directive. (The default on Windows is the batch command language.) Also note that it is possible under both Windows and UNIX to specify to PBS the scripting language to use to interpret the job script (see the "-S" option to `qsub` in section 4.13.9 "Specifying Scripting Language to Use" on page 85). The Windows script will be a .exe or .bat file.

Lines 2-8 of both files are almost identical. The primary differences will be in file and directory path specification (such as the use of drive letters and slash vs. backslash as the path separator).

Lines 2-4 are PBS directives. PBS reads down the shell script until it finds the first line that is not a valid PBS directive, then stops. It assumes the rest of the script is the list of commands or tasks that the user wishes to run. In this case, PBS sees lines 6-8 as being user commands.

The section "Job Submission Options" on page 79 describes how to use the `qsub` command to submit PBS jobs. Any option that you specify to the `qsub` command line (except "`-I`") can also be provided as a PBS directive inside the PBS script. PBS directives come in two types: resource requirements and attribute settings.

In our example above, lines 2-3 specify the "`-l`" resource list option, followed by a specific resource request. Specifically, lines 2-3 request 1 hour of wall-clock time as a job-wide request, and 400 megabytes (MB) of memory in a chunk. .

Line 4 requests that PBS *join* the `stdout` and `stderr` output streams of the job into a single stream.

Finally lines 6-8 are the command lines for executing the program(s) we wish to run. You can specify as many programs, tasks, or job steps as you need.

4.11 Changing the Job's PBS Directive

By default, the text string "`#PBS`" is used by PBS to determine which lines in the job file are PBS directives. The leading "`#`" symbol was chosen because it is a comment delimiter to all shell scripting languages in common use on UNIX systems. Because directives look like comments, the scripting language ignores them.

Under Windows, however, the command interpreter does not recognize the '`#`' symbol as a comment, and will generate a benign, non-fatal warning when it encounters each "`#PBS`" string. While it does not cause a problem for the batch job, it can be annoying or disconcerting to the user. Therefore Windows users may wish to specify a different PBS directive, via either the

PBS_DPREFIX environment variable, or the "-C" option to qsub. For example, we can direct PBS to use the string "REM PBS" instead of "#PBS" and use this directive string in our job script:

```
REM PBS -l wall-
time=1:00:00
REM PBS -l
select=mem=400mb
REM PBS -j oe
```

Given the above job script, we can submit it to PBS in one of two ways:

```
set PBS_DPREFIX=REM PBS
qsub my_job_script
```

or

```
qsub -C "REM PBS" my_job_script
```

For additional details on the "-C" option to qsub, see section 4.13 "Job Submission Options" on page 79.

4.12 Windows Jobs

4.12.1 Submitting Windows Jobs

Any .bat files that are to be executed within a PBS job script have to be prefixed with "call" as in:

---[job_b.bat]----------

```
@echo off
call E:\step1.bat
call E:\step2.bat
```

Without the "call", only the first .bat file gets executed and it doesn't return control to the calling interpreter.

An example:

A job script that contains:

--[job_a.bat]---------

```
@echo off
E:\step1.bat
E:\step2.bat
```

should now be:

--[job_a.bat]---------

```
@echo off
call E:\step1.bat
call E:\step2.bat
```

Under Windows, comments in the job script must be in ASCII characters.

4.12.2 Passwords

When running PBS in a password-protected Windows environment, you will need to specify to PBS the password needed in order to run your jobs. There are two methods of doing this: (1) by providing PBS with a password once to be used for all jobs (“single signon method”), or (2) by specifying the password for each job when submitted (“per job method”). Check with your system administrator to see which method was configured at your site.

4.12.2.1 Single-Signon Password Method

To provide PBS with a password to be used for all your PBS jobs, use the `pbs_password` command. This command can be used whether or not you have jobs enqueued in PBS. The command usage syntax is:

pbs_password [-s server] [-r] [-d] [user]

When no options are given to pbs_password, the password credential on the default PBS server for the current user, i.e. the user who executes the command, is updated to the prompted password. Any user jobs previously held due to an invalid password are not released.

The available options to `pbs_password` are:

-r

Any user jobs previously held due to an invalid password are released.

-s server

Allows user to specify server where password will be changed.

-d

Deletes the password.

user

The password credential of user user is updated to the prompted password. If user is not the current user, this action is only allowed if:

1. The current user is root or admin.

2. User user has given the current user explicit access via

the ruserok() mechanism:

a. The hostname of the machine from which the current

user is logged in appears in the server's hosts.equiv

file, or

b. The current user has an entry in user's

HOMEDIR\.rhosts file.

Note that `pbs_password` encrypts the password obtained from the user before sending it to the PBS Server. The `pbs_password` command does not change the user's password on the current host, only the password that is cached in PBS.

4.12.2.2 Per-job Password Method

If you are running in a password-protected Windows environment, but the single-signon method has not been configured at your site, then you will need to supply a password with the submission of each job. You can do this via the `qsub` command, with the `-Wpwd` option, and supply the password when prompted.

```
qsub -Wpwd="<password>" job.script
```

see 8.12

The password specified will be shown on screen and will be passed onto the program, which will then encrypt it and save it securely for use by the job. The password should be enclosed in double quotes. If you only type the pair of double quotes, you will be prompted for the password.

The password can also be specified in `xpbs` using the "`SUBMIT-PASSWORD`" entry box in the Submit window. The password you type in will not be shown on the screen.

IMPORTANT:

Both the `-Wpwd` option to `qsub`, and the `xpbs SUBMIT-PASSWORD` entry box can only be used when submitting jobs to Windows. The UNIX `qsub` does not support the `-Wpwd` option; and if you type a password into the `xpbs SUBMIT-PASSWORD` entry box under UNIX, the job will be rejected.

Keep in mind that in a multi-host job, the password supplied will be propagated to all the sister hosts. This requires that the password be the same on the user's accounts on all the hosts. The use of domain accounts for a multi-host job will be ideal in this case.

IMPORTANT:

Because of enhanced security features found in Windows 2003 Server, you may not be able to run non-passworded jobs.

Accessing network share drives/resources within a job session also requires that you submit the job with a password via `qsub -W pwd=""` or the "`SUBMIT-PASSWORD`" entry box in `xpbs`.

Furthermore, if the job is submitted *without* a password, do not use the native `rcp` command from within the job script, as it will generate the error: “unable to get user name”. Instead, please use `pbs_rcp`.

4.13 Job Submission Options

There are many options to the `qsub` command. The table below gives a quick summary of the available options; the rest of this chapter explains how to use each one.

Table 4-4: Options to the `qsub` Command

Option	Function and Page Reference
-A account_string	“Specifying a Local Account” on page 90
-a date_time	“Deferring Execution” on page 86
-C “DPREFIX”	“Changing the Job’s PBS Directive” on page 74
-c interval	“Specifying Job Checkpoint Interval” on page 86
-e path	“Redirecting Output and Error Files” on page 81
-h	“Holding a Job (Delaying Execution)” on page 86
-I	“Interactive-batch Jobs” on page 92
-J X-Y[:Z]	“Job Array” on page 211
-j *join*	“Merging Output and Error Files” on page 90
-k *keep*	“Retaining Output and Error Files on Execution Host” on page 91
-l resource_list	section 4.3.1 “Rules for Submitting Jobs” on page 42
-M user_list	“Setting Email Recipient List” on page 83
-m MailOptions	“Specifying Email Notification” on page 83

Table 4-4: Options to the `qsub` Command

Option	Function and Page Reference
-N name	“Specifying a Job Name” on page 84
-o path	“Redirecting Output and Error Files” on page 81
-p priority	“Setting a Job’s Priority” on page 85
-q destination	“Specifying Queue and/or Server” on page 81
-r value	“Marking a Job as “Rerunnable” or Not” on page 84
-S path_list	“Specifying Scripting Language to Use” on page 85
-u user_list	“Specifying Job User ID” on page 87
-V	“Exporting Environment Variables” on page 82
-v variable_list	“Expanding Environment Variables” on page 83
-W depend=list	“Specifying Job Dependencies” on page 164
-W group_list=list	“Specifying Job Group ID” on page 89
-W stagein=list	“Input/Output File Staging” on page 167
-W stageout=list	“Input/Output File Staging” on page 167
-W cred=dce	“Running PBS in a UNIX DCE Environment” on page 208
-W block=opt	“Requesting qsub Wait for Job Completion” on page 163
-W pwd=”password”	“Per-job Password Method” on page 78 and “Running PBS in a UNIX DCE Environment” on page 208
-W umask=nnn	“Changing UNIX Job umask” on page 163
-z	“Suppressing Job Identifier” on page 92

4.13.1 Specifying Queue and/or Server

The "`-q destination`" option to `qsub` allows you to specify a particular destination to which you want the job submitted. The *destination* names a queue, a Server, or a queue at a Server. The `qsub` command will submit the script to the Server defined by the *destination* argument. If the *destination* is a routing queue, the job may be routed by the Server to a new destination. If the `-q` option is not specified, the `qsub` command will submit the script to the default queue at the default Server. (See also the discussion of **PBS_DEFAULT** in "Environment Variables" on page 30.) The destination specification takes the following form:

```
-q [queue[@host]]
```

Examples:

```
qsub -q queue my_job
qsub -q @server my_job
#PBS -q queueName
qsub -q queueName@serverName my_job
qsub -q queueName@serverName.domain.com my_job
```

4.13.2 Redirecting Output and Error Files

PBS, by default, always copies the standard output (stdout) and standard error (stderr) files back to $PBS_O_WORKDIR on the submission host when a job finishes. When qsub is run, it sets $PBS_O_WORKDIR to the current working directory where the qsub command is executed.

The "`-o path`" and "`-e path`" options to `qsub` allows you to specify the name of the files to which the stdout and the stderr file streams should be written. The path argument is of the form: `[hostname:]path_name` where *hostname* is the name of a host to which the file will be returned and *path_name* is the path name on that host. You may

specify relative or absolute paths. If you specify only a file name, it is assumed to be relative to your home directory. Do not use variables in the path. The following examples illustrate these various options.

```
#PBS -o /u/user1/myOutputFile
#PBS -e /u/user1/myErrorFile
```

```
qsub -o myOutputFile my_job
qsub -o /u/user1/myOutputFile my_job
qsub -o myWorkstation:/u/user1/myOutputFile my_job
qsub -e myErrorFile my_job
qsub -e /u/user1/myErrorFile my_job
qsub -e myWorkstation:/u/user1/myErrorFile my_job
```

Note that if the PBS client commands are used on a Windows host, then special characters like spaces, backslashes (\), and colons (:) can be used in command line arguments such as for specifying pathnames, as well as drive letter specifications. The following are allowed:

```
qsub -o \temp\my_out job.scr
qsub -e "host:e:\Documents and
    Settings\user\Desktop\output"
```

The error output of the above job is to be copied onto the `e:` drive on *`host`* using the path "`\Documents and Settings\user\Desktop\output`". The quote marks are required when arguments to `qsub` contain spaces.

4.13.3 Exporting Environment Variables

The "`-V`" option declares that all environment variables in the `qsub` command's environment are to be exported to the batch job.

```
qsub -V my_job
#PBS -V
```

4.13.4 Expanding Environment Variables

The "`-v variable_list`" option to `qsub` allows you to specify additional environment variables to be exported to the job. *variable_list* names environment variables from the `qsub` command environment which are made available to the job when it executes. The *variable_list* is a comma separated list of strings of the form `variable` or `variable=value`. These variables and their values are passed to the job.

```
qsub -v DISPLAY,myvariable=32 my_job
```

4.13.5 Specifying Email Notification

The "`-m MailOptions`" defines the set of conditions under which the execution server will send a mail message about the job. The *MailOptions* argument is a string which consists of either the single character "`n`", or one or more of the characters "`a`", "`b`", and "`e`". If no email notification is specified, the default behavior will be the same as for "`-m a`" .

a
send mail when job is *aborted* by batch system

b
send mail when job *begins* execution

e
send mail when job *ends* execution

n
do not send mail

Examples:

```
qsub -m ae my_job
#PBS -m b
```

4.13.6 Setting Email Recipient List

The "`-M user_list`" option declares the list of users to whom mail is sent by the execution server when it sends mail about the job. The *user_list* argument is of the form:

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

If unset, the list defaults to the submitting user at the `qsub` host, i.e. the job owner.

qsub -M *user1@mydomain.com* my_job

IMPORTANT:

PBS on Windows can only send email to addresses that specify an actual hostname that accepts port 25 (sendmail) requests. For the above example on Windows you will need to specify:

qsub -M user1@host.mydomain.com

where "`host.mydomain.com`" accepts port 25 connections.

4.13.7 Specifying a Job Name

The "`-N name`" option declares a name for the job. The *name* specified may be up to and including 15 characters in length. It must consist of printable, non-whitespace characters with the first character alphabetic, and contain no "special characters". If the `-N` option is not specified, the job name will be the base name of the job script file specified on the command line. If no script file name was specified and the script was read from the standard input, then the job name will be set to STDIN.

qsub -N *myName* my_job

```
#PBS -N myName
```

4.13.8 Marking a Job as "Rerunnable" or Not

The "`-r y|n`" option declares whether the job is rerunnable. To rerun a job is to terminate the job and requeue it in the execution queue in which the job currently resides. The *value* argument is a single character, either "`y`" or "`n`". If the argument is "`y`", the job is rerunnable. If the argument is "`n`", the job is not rerunnable. The default value is "`y`", rerunnable.

qsub -r *n* my_job

```
#PBS -r n
```

4.13.9 Specifying Scripting Language to Use

The "`-S path_list`" option declares the path and name of the scripting language to be used in interpreting the job script. The option argument *path_list* is in the form: `path[@host][,path[@host],...]` Only one path may be specified for any host named, and only one path may be specified without the corresponding host name. The path selected will be the one with the host name that matched the name of the execution host. If no matching host is found, then the path specified without a host will be selected, if present. If the `-S` option is not specified, the option argument is the null string, or no entry from the *path_list* is selected, then PBS will use the user's login shell on the execution host.

```
#PBS -S /bin/bash@mars,/usr/bin/bash@jupiter
```

IMPORTANT:

Using this option under Windows is more complicated because if you change from the default shell of `cmd`, then a valid `PATH` is not automatically set. Thus if you use the "`-S`" option under Windows, you must explicitly set a valid `PATH` as the first line of your job script.

4.13.10 Setting a Job's Priority

The "`-p priority`" option defines the priority of the job. The *priority* argument must be an integer between -1024 (lowest priority) and +1023 (highest priority) inclusive. The default is no priority which is equivalent to a priority of zero.

This option allows the user to specify a priority for their jobs. However, this option is dependant upon the local scheduling policy. By default the "sort jobs by job-priority" feature is disabled. If your local PBS administrator has enabled it, then all queued jobs will be sorted based on the user-specified priority. (If you need an absolute ordering of your own jobs, see "Specifying Job Dependencies" on page 164.)

```
qsub -p 120 my_job
#PBS -p -300
```

4.13.11 Deferring Execution

The "`-a date_time`" option declares the time after which the job is eligible for execution. The *date_time* argument is in the form: `[[[[CC]YY]MM]DD]hhmm[.SS]` where `CC` is the first two digits of the year (the century), `YY` is the second two digits of the year, `MM` is the two digits for the month, `DD` is the day of the month, `hh` is the hour, `mm` is the minute, and the optional `SS` is the seconds. If the month, `MM`, is not specified, it will default to the current month if the specified day `DD`, is in the future. Otherwise, the month will be set to next month. Likewise, if the day, `DD`, is not specified, it will default to today if the time `hhmm` is in the future. Otherwise, the day will be set to tomorrow. For example, if you submit a job at 11:15am with a time of "`1110`", the job will be eligible to run at 11:10am tomorrow. Other examples include:

qsub -a *0700* my_job

```
#PBS -a 10220700
```

4.13.12 Holding a Job (Delaying Execution)

The "`-h`" option specifies that a *user hold* be applied to the job at submission time. The job will be submitted, then placed in a hold state. The job will remain ineligible to run until the hold is released. (For details on releasing a held job see "Holding and Releasing Jobs" on page 151.)

qsub -h my_job

```
#PBS -h
```

4.13.13 Specifying Job Checkpoint Interval

4.13.13.1 Checkpointable Jobs

A job is checkpointable if either of the following is true:

- its application supports checkpointing and there are checkpoint scripts
- the OS supports checkpointing.

Checkpoint scripts are set up by the local system administrator.

4.13.13.2 Checkpoint Interval

The "`-c interval`" option defines the interval (in minutes) at which the job will be checkpointed, if the job is checkpointable. If the job is not checkpointable, this option is ignored.

The *interval* argument is specified as:

n
: No checkpointing is to be performed.

s
: Checkpointing is to be performed only when the Server executing the job is shutdown.

c
: Checkpointing is to be performed at the default minimum time for the MOM executing the job.

c=minutes
: Checkpointing is to be performed at an interval of *minutes*, which is the integer number of minutes of CPU time used by the job. This value must be greater than zero. The MOM's polling cycle controls the minimum frequency for checkpointing.

u
: Checkpointing is unspecified, thus resulting in the same behavior as "s".

If "`-c`" is not specified, the checkpoint attribute is set to the value "`u`".

```
qsub -c c my_job
#PBS -c c=10
```

Checkpointing is not supported for job arrays.

4.13.14 Specifying Job User ID

PBS requires that a user's name be consistent across a server and its execution hosts, but not across a submission host and a server. A user may have access to more than one server, and may have a different username on each server. In this environment, if a user wishes to submit a job to any of the

available servers, the username for each server is specified. The wildcard username will be used if the job ends up at yet another server not specified, but only if that wildcard username is valid.

For example, our user is UserS on the submission host HostS, UserA on server ServerA, and UserB on server ServerB, and is UserC everywhere else. Note that this user must be UserA on all ExecutionA and UserB on all ExecutionB machines. Then our user can use "qsub -u UserA@ServerA,UserB@ServerB,UserC" for the job. The job owner will always be UserS.

4.13.14.1 qsub -u: User ID with UNIX

The server's flatuid attribute determines whether it assumes that identical usernames mean identical users. If true, it assumes that if UserS exists on both the submission host and the server host, then UserS can run jobs on that server. If not true, the server calls ruserok() which uses /etc/hosts.equiv and .rhosts to authorize UserS to run as UserS.

Table 4-5: UNIX User ID and flatuid

Value of flatuid	Submission host username/server host username	
	Same: UserS/UserS	Different: UserS/UserA
True	Server assumes user has permission to run job	Server checks whether UserS can run job as UserA
Not true	Server checks whether UserS can run job as UserS	Server checks whether UserS can run job as UserA

Note that if different names are listed via the `-u` option, then they are checked regardless of the value of `flatuid`.

4.13.14.2 qsub -u: User ID with Windows

Under Windows, if a user has a non-admin account, the server's hosts.equiv file is used to determine whether that user can run a job on a given server. For an admin account, [PROFILE_PATH].\rhosts is used, and the server's acl_roots attribute must be set to allow job submissions.

Usernames containing spaces are allowed as long as the username length is no more than 15 characters, and the usernames are quoted when used in the command line.

Table 4-6: Requirements for Admin User to Submit Job

Location/Action	Submission host username/Server host username	
	Same: UserS/UserS	Different: UserS/ UserA
[PROFILE_PATH]\ .rhosts contains	For UserS on ServerA, add <HostS> UserS	For UserA on ServerA, add <HostS> UserS
set ServerA's acl_roots attribute	qmgr> set server acl_roots=UserS	qmgr> set server acl_roots=UserA

Table 4-7: Requirements for Non-admin User to Submit Job

File	Submission host username/Server host username	
	Same: UserS/UserS	Different: UserS/UserA
hosts.equiv on ServerA	<HostS>	<HostS> UserS

4.13.15 Specifying Job Group ID

The "`-W group_list=g_list`" option defines the group name under which the job is to run on the execution system. The *g_list* argument is of the form:

```
group[@host][,group[@host],...]
```

Only one group name may be given per specified host. Only one of the group specifications may be supplied without the corresponding host specification. That group name will used for execution on any host not named in the argument list. If not set, the *group_list* defaults to the primary group

of the user under which the job will be run. Under Windows, the primary group is the first group found for the user by PBS when querying the accounts database.

qsub -W *group_list=grpA,grpB@jupiter* my_job

4.13.16 Specifying a Local Account

The "`-A account_string`" option defines the account string associated with the job. The *account_string* is an opaque string of characters and is not interpreted by the Server which executes the job. This value is often used by sites to track usage by locally defined account names.

IMPORTANT:

Under Unicos, if the Account string is specified, it must be a valid account as defined in the system "User Data Base", UDB.

qsub -A *Math312* my_job

```
#PBS -A accountNumber
```

4.13.17 Merging Output and Error Files

The "`-j` ***join***" option declares if the standard error stream of the job will be merged with the standard output stream of the job. A *join* argument value of `oe` directs that the two streams will be merged, intermixed, as standard output. A *join* argument value of `eo` directs that the two streams will be merged, intermixed, as standard error. If the *join* argument is `n` or the option is not specified, the two streams will be two separate files.

qsub -j *oe* my_job

```
#PBS -j eo
```

4.13.18 Retaining Output and Error Files on Execution Host

The "-k *keep*" option defines which (if either) of standard output (STDOUT) or standard error (STDERR) of the job will be retained in the job's staging and execution directory on the primary execution host. If set, this option overrides the path name for the corresponding file. If not set, neither file is retained on the execution host. The argument is either the single letter "e" or "o", or the letters "e" and "o" combined in either order. Or the argument is the letter "n". If "-k" is not specified, neither file is retained.

e

The standard error file is to be retained in the job's staging and execution directory on the primary execution host. The job's name will be the default file name given by: *job_name*.**e***sequence* where job_name is the name specified for the job, and *sequence* is the sequence number component of the job identifier.

o

The standard output file is to be retained in the job's staging and execution directory on the primary execution host. The file name will be the default file name given by: *job_name*.**o***sequence* where job_name is the name specified for the job, and *sequence* is the sequence number component of the job identifier.

eo, oe

Both standard output and standard streams are retained on the primary execution host, in the job's staging and execution directory.

n

Neither file is retained.

qsub -k oe my_job

#PBS -k eo

4.13.19 Suppressing Job Identifier

The "-z" option directs the qsub command to not write the job identifier assigned to the job to the command's standard output.

```
qsub -z my_job
#PBS -z
```

4.13.20 Interactive-batch Jobs

PBS provides a special kind of batch job called *interactive-batch*. An interactive-batch job is treated just like a regular batch job (in that it is queued up, and has to wait for resources to become available before it can run). Once it is started, however, the user's terminal input and output are connected to the job in a matter similar to a login session. It appears that the user is logged into one of the available execution machines, and the resources requested by the job are reserved for that job. Many users find this useful for debugging their applications or for computational steering. The "-I" option declares that the job is an interactive-batch job.

IMPORTANT:

Interactive-batch jobs are not supported on Windows.

IMPORTANT:

Interactive-batch jobs do not support job arrays.

If the -I option is specified on the command line, the job is an interactive job. If a script is given, it will be processed for directives, but any executable commands will be discarded. When the job begins execution, all input to the job is from the terminal session in which qsub is running. The -I option is ignored in a script directive.

When an interactive job is submitted, the qsub command will not terminate when the job is submitted. qsub will remain running until the job terminates, is aborted, or the user interrupts qsub with a SIGINT (the control-C key). If qsub is interrupted prior to job start, it will query if the user wishes to exit. If the user responds "yes", qsub exits and the job is aborted.

Once the interactive job has started execution, input to and output from the job pass through `qsub`. Keyboard-generated interrupts are passed to the job. Lines entered that begin with the tilde ('~') character and contain special sequences are interpreted by `qsub` itself. The recognized special sequences are:

~.
: `qsub` terminates execution. The batch job is also terminated.

~susp
: If running under the UNIX C shell, suspends the `qsub` program. "susp" is the suspend character, usually CNTL-Z.

~asusp
: If running under the UNIX C shell, suspends the input half of `qsub` (terminal to job), but allows output to continue to be displayed. "asusp" is the auxiliary suspend character, usually control-Y.

4.14 Job Attributes

A PBS job has the following attributes, which may be set by the various options to `qsub` (for details see section 4.13 "Job Submission Options" on page 79).

Account_Name
: Reserved for local site accounting. If specified (using the `-A` option to `qsub`) this value is carried within the job for its duration, and is included in the job accounting records.

  Python attribute value type: str

block
: When true, specifies that `qsub` will wait for the job to complete, and return the exit value of the job. Default: false. Set via the *-W block* option to qsub. If `qsub` receives one of the signals: SIGHUP, SIGINT, SIGQUIT or SIGTERM, it will print the following message on stderr:
  `qsub: wait for job <jobid> interrupted by signal <signal>`

Python attribute value type: int

Checkpoint

If the job is checkpointable, the checkpoint attribute determines when checkpointing will be performed. The legal values for checkpoint are described under the `qalter` and `qsub` commands. See section 4.13.13.1 “Checkpointable Jobs” on page 86.

Python attribute value type: pbs.checkpoint

depend

The type of inter-job dependencies specified by the job owner.

Python attribute value type: pbs.depend

Error_Path

The final path name for the file containing the job’s standard error stream. See the `qsub` and `qalter` command description for more detail.

Python attribute value type: str

Execution_Time

The time after which the job may execute. The time is maintained in seconds since Epoch. If this time has not yet been reached, the job will not be scheduled for execution and the job is said to be in *wait* state.

Python attribute value type: long

group_list

A list of *group_names@hosts* which determines the group under which the job is run on a given host. When a job is to be placed into execution, the Server will select a group name according to the rules specified for use of the `qsub` command.

Python attribute value type: pbs.group_list

Hold_Types

The set of holds currently applied to the job. If the set is not null, the job will not be scheduled for execution and is said to be in the *hold* state. Note, the *hold* state takes precedence over the *`wait`* state.

n no hold

o other hold

p bad password

s system hold

u user hold

Python attribute value type: pbs.hold_types

Job_Name

The name assigned to the job by the `qsub` or `qalter` command.

Python attribute value type: str

Join_Path

If the `Join_Path` attribute is `oe`, then the job's standard error stream will be merged, inter-mixed, with the job's standard output stream and placed in the file determined by the `Output_Path` attribute. The `Error_Path` attribute is maintained, but ignored. However, if the `Join_Path` attribute is `eo`, then the job's standard output stream will be merged, inter-mixed, with the job's standard error stream and placed in the file determined by the `Error_Path` attribute, and the `Output_Path` attribute will be ignored.

Python attribute value type: pbs.join_path

Keep_Files

If `Keep_Files` contains the values "`o`" `KEEP_OUTPUT` and/or "`e`" `KEEP_ERROR` the corresponding streams of the batch job will be retained on the execution host upon job termination. `Keep_Files` overrides the `Output_Path` and `Error_Path` attributes.

Python attribute value type: pbs.keep_files

Mail_Points

Identifies when the Server will send email about the job.

Python attribute value type: pbs.mail_points

Mail_Users

The set of users to whom mail may be sent when the job makes certain state changes.

Python attribute value type: pbs.mail_users

no_stdio_sockets

Flag to indicate whether a multi-host job should have the standard output and standard error streams of tasks running on other hosts returned to mother superior via sockets. These sockets may cause a job to be not checkpointable. Default: false (sockets are created.)

Python attribute value type: bool

Output_Path

The final path name for the file containing the job's standard output stream. See the `qsub` and `qalter` command description for more detail.

Python attribute value type: str

Priority

The job scheduling priority assigned by the user.

Python attribute value type: int

Rerunnable

The rerunnable flag given by the user.

Python attribute value type: bool

Resource_List

The resource list is a set of resources required by the job. The value also establishes the limit of usage of that resource. If not set, the value for a resource may be determined by a queue or Server default established by the administrator.

Python attribute value type: dictionary: resources_available["<resource name>"] = <resource val> where <resource name> is any built-in or custom resource

sandbox

When set to PRIVATE, PBS creates job-specific staging and execution directories under the directory specified in the $jobdir_root MOM configuration option. When set to HOME or not set, PBS will use the job owner's home directory for staging and execution. User-settable via

qsub -Wsandbox=<value> or via a PBS directive. Not set by default. See the $jobdir_root MOM configuration option in pbs_mom.8B.

Python attribute value type: str; valid values: *PRIVATE*, *HOME, O_WORKDIR*

Shell_Path_List

A set of absolute paths of the program to process the job's script file.

Python attribute value type: pbs.path_list

stagein

The list of files to be staged in prior to job execution. Format: local_path@remote_host:remote_path

Python attribute value type: pbs.staging_list

stageout

The list of files to be staged out after job execution. Format: local_path@remote_host:remote_path

Python attribute value type: pbs.staging_list

umask

The initial umask of the job is set to the value of this attribute when the job is created. This may be changed by umask commands in the shell initialization files such as .profile or .cshrc.

Default value: 077

Python attribute value type: int

User_List

The list of *user@host* which determines the username under which the job is run on a given host.

Python attribute value type: pbs.user_list

Variable_List

This is the list of environment variables passed with the *Queue Job* batch request.

Python attribute value type: dictionary:
Variable_List["<variable name>"] = <value>

Note that PBS environment variables listed in the qsub(1B) man page are not settable.

Python attribute value type: dictionary: Variable_List[“<variable name>”] = <value> where <resource name> is any built-in or custom resource

comment

An attribute for displaying comments about the job from the system. Visible to any client. Under Windows, comments can contain only ASCII characters.

Python attribute value type: str

The following attributes are read-only, they are established by the Server and are visible to the user but cannot be set or changed by a user.

accounting_id

Accounting ID for tracking accounting data not produced by PBS.

Python attribute value type: str

accrue_type

Indicates what kind of time the job is accruing. Can be one of *initial_time*, *eligible_time*, *ineligible_time*, or *run_time*. Viewable only by Manager.

Python attribute value type: int

alt_id

For a few systems, the session id is insufficient to track which processes belong to the job. Where a different identifier is required, it is recorded in this attribute. If set, it will also be recorded in the end-of-job accounting record.

Python attribute value type: str

array

boolean; true if applied to a job array

Python attribute value type: bool

array_id

string; applies to subjob; job array identifier for given subjob

Python attribute value type: str

array_index

string; applies to subjob; index number of given subjob

Python attribute value type: int

array_indices_remaining

string; applies to job array; list of indices of subjobs still queued. Range or list of ranges

Python attribute value type: pbs.range

array_indices_submitted

string; applies to job array; complete list of indices of subjobs given at submission time. Given as a range.

Python attribute value type: pbs.range

array_state_count

string; applies to job array; lists number of subjobs in each state

Python attribute value type: pbs.state_count

ctime

The time that the job was created.

Python attribute value type: long

eligible_time

The amount of wall clock wait time a job has accrued because the job is blocked waiting for resources. For a job currently accruing eligible_time, if we were to add enough of the right type of resources, the job would start immediately. Viewable via qstat -f by job owner, Manager and Operator. Settable by Operator or Manager.

Python attribute value type: long

etime

The time that the job became eligible to run, i.e. in a queued state while residing in an execution queue.

Python attribute value type: long

exec_host

If the job is running, string set to the name of each vnode on which the job is executing, along with the vnode-level, consumable resources allocated from that vnode.

Format:

”(vnode:ncpus=N:mem=M+vnode:ncpus=N:mem=M[+...])”, where *vnode* is the name of a vnode, *N* is the number of CPUs on that vnode allocated to the job and *M* is the amount of memory on that vnode allocated to the job. Other resources may show up as well.

Python attribute value type: pbs.exec_host

exec_vnode

If the job is running, this is set to the name of each node used by the job with the node-level, consumable resources allocated from that node. Each chunk's worth of nodes is enclosed in parentheses, and chunks are connected by plus signs. So for a job which requested two chunks that were satisfied by resources from three nodes, exec_vnode could look like

(vnodeA:ncpus=N:mem=X)+(nodeB:ncpus=P:mem=Y+nodeC:mem=Z).

Python attribute value type: pbs.exec_vnode

egroup

If the job is queued in an execution queue, this attribute is set to the group name under which the job is to be run. [This attribute is available only to the batch administrator.]

Python attribute value type: str

euser

If the job is queued in an execution queue, this attribute is set to the user name under which the job is to be run. [This attribute is available only to the batch administrator.]

Python attribute value type: str

hashname

The name used as a basename for various files, such as the job file, script file, and the standard output and error of the job. [This attribute is available only to the batch administrator.]

Python attribute value type: str

interactive

True if the job is an interactive PBS job.

Python attribute value type: int

jobdir

Path of the job's staging and execution directory on the primary execution host. Viewable via qstat -f.

Python attribute value type: str

Job_Owner

The login name on the submitting host of the user who submitted the batch job.

Python attribute value type: str

job_state

The state of the job.

Python attribute value type: a PBS job state constant. See section 9.8.7.5 “Job State Objects” on page 475 of the PBS Professional Administrator’s Guide.

mtime

The time that the job was last modified, changed state, or changed locations.

Python attribute value type: long

qtime

The time that the job entered the current queue.

Python attribute value type: long

queue

The name of the queue in which the job currently resides.

Python attribute value type: pbs.queue

queue_rank

The job’s position in the queue. Set by server. Read-only. Requires operator or administrator privilege to view. Integer.

Python attribute value type: int

resources_used

The amount of resources used by the job. This is provided as part of job status information if the job is running.

Python attribute value type: dictionary:
resources_used[“<resource name>”] = <resource val>
where <resource name> is any built-in or custom resource

server

The name of the server which is currently managing the job.

Python attribute value type: pbs.server

session_id

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

Python attribute value type: int

stime

The time when the job started execution. Set by the server. Displayed in date/time format.

Python attribute value type: long

Chapter 5

Using the xpbs GUI

The PBS graphical user interface is called **xpbs**, and provides a user-friendly, point and click interface to the PBS commands. xpbs utilizes the tcl/tk graphics tool suite, while providing the user with most of the same functionality as the PBS CLI commands. In this chapter we introduce xpbs, and show how to create a PBS job using xpbs.

5.1 Starting xpbs

If PBS is installed on your local workstation, or if you are running under Windows, you can launch xpbs by double-clicking on the xpbs icon on the desktop. You can also start xpbs from the command line with the following command.

UNIX:

```
xpbs &
```

Windows:

xpbs.exe

Doing so will bring up the main xpbs window, as shown below.

5.1.1 Running xpbs Under UNIX

Before running xpbs for the first time under UNIX, you may need to configure your workstation for it. Depending on how PBS is installed at your site, you may need to allow xpbs to be displayed on your workstation. However, if the PBS client commands are installed locally on your workstation, you can skip this step. (Ask your PBS administrator if you are unsure.)

The most secure method of running xpbs remotely and displaying it on your local XWindows session is to redirect the XWindows traffic through ssh (secure shell), via setting the "X11Forwarding yes" parameter in the sshd_config file. (Your local system administrator can provide details on this process if needed.)

An alternative, but less secure, method is to direct your X-Windows session to permit the xpbs client to connect to your local X-server. Do this by running the xhost command with the name of the host from which you will be running xpbs, as shown in the example below:

```
xhost + server.mydomain.com
```

Next, on the system from which you will be running xpbs, set your X-Windows **DISPLAY** variable to your local workstation. For example, if using the C-shell:

```
setenv DISPLAY myWorkstation:0.0
```

However, if you are using the Bourne or Korn shell, type the following:

```
export DISPLAY=myWorkstation:0.0
```

5.2 Using xpbs: Definitions of Terms

The various panels, boxes, and regions (collectively called “widgets”) of `xpbs` and how they are manipulated are described in the following sections. A *listbox* can be multi-selectable (a number of entries can be selected/highlighted using a mouse click) or single-selectable (one entry can be highlighted at a time).

For a multi-selectable listbox, the following operations are allowed:

- left-click to select/highlight an entry.
- shift-left-click to contiguously select more than one entry.
- control-left-click to select multiple non-contiguous entries.
- click the *Select All / Deselect All* button to select all entries or deselect all entries at once.
- double clicking an entry usually activates some action that uses the selected entry as a parameter.

An *entry* widget is brought into focus with a left-click. To manipulate this widget, simply type in the text value. Use of arrow keys and mouse selection of text for deletion, overwrite, copying and pasting with sole use of mouse buttons are permitted. This widget has a scrollbar for horizontally scanning a long text entry string.

A *matrix of entry boxes* is usually shown as several rows of entry widgets where a number of entries (called fields) can be found per row. The matrix is accompanied by up/down arrow buttons for paging through the rows of data, and each group of fields gets one scrollbar for horizontally scanning long entry strings. Moving from field to field can be done using the <Tab> (move forward), <Cntrl-f> (move forward), or <Cntrl-b> (move backward) keys.

A *spinbox* is a combination of an entry widget and a horizontal scrollbar. The entry widget will only accept values that fall within a defined list of valid values, and incrementing through the valid values is done by clicking on the up/down arrows.

A *button* is a rectangular region appearing either raised or pressed that invokes an action when clicked with the left mouse button. When the button appears pressed, then hitting the <RETURN> key will automatically select the button.

A *text region* is an editor-like widget. This widget is brought into focus with a left-click. To manipulate this widget, simply type in the text. Use of arrow keys, backspace/delete key, mouse selection of text for deletion or overwrite, and copying and pasting with sole use of mouse buttons are permitted. This widget has a scrollbar for vertically scanning a long entry.

5.3 Introducing the xpbs Main Display

The main window or display of `xpbs` is comprised of five collapsible subwindows or *panels*. Each panel contains specific information. Top to bottom, these panels are: the Menu Bar, Hosts panel, Queues panel, Jobs panel, and the Info panel.

5.3.1 xpbs Menu Bar

The Menu Bar is composed of a row of command buttons that signal some action with a click of the left mouse button. The buttons are:

Manual Update
: forces an update of the information on hosts, queues, and jobs.

Auto Update
: sets an automatic update of information every user-specified number of minutes.

Track Job
: for periodically checking for returned output files of jobs.

Preferences
: for setting parameters such as the list of Server host(s) to query.

Help
: contains some help information.

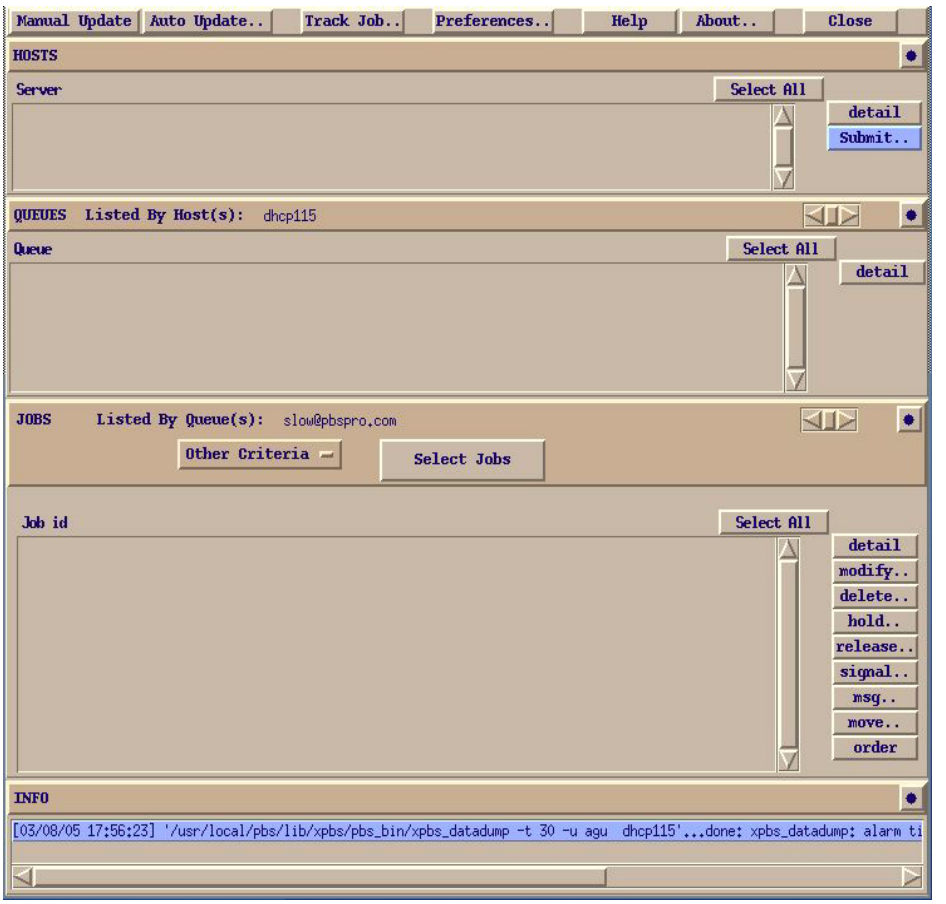
About

gives general information about the xpbs GUI.

Close

for exiting `xpbs` plus saving the current setup information.

.



5.3.2 xpbs Hosts Panel

The Hosts panel is composed of a leading horizontal HOSTS bar, a listbox, and a set of command buttons. The HOSTS bar contains a minimize/maximize button, identified by a dot or a rectangular image, for displaying or iconizing the Hosts region. The listbox displays information about favorite

Server host(s), and each entry is meant to be selected via a single left-click, shift-left-click for contiguous selection, or control-left-click for non-contiguous selection.

To the right of the Hosts Panel are buttons that represent actions that can be performed on selected host(s). Use of these buttons will be explained in detail below.

detail

Provides information about selected Server host(s). This functionality can also be achieved by double clicking on an entry in the Hosts listbox.

submit

For submitting a job to any of the queues managed by the selected host(s).

terminate

For terminating (shutting down) PBS Servers on selected host(s). (Visible via the "`-admin`" option only.)

IMPORTANT:

Note that some buttons are only visible if `xpbs` is started with the "`-admin`" option, which requires manager or operator privilege to function.

The middle portion of the Hosts Panel has abbreviated column names indicating the information being displayed, as the following table shows:

Table 5-1: xpbs Server Column Headings

Heading	Meaning
Max	Maximum number of jobs permitted
Tot	Count of jobs currently enqueued in any state
Que	Count of jobs in the Queued state
Run	Count of jobs in the Running state
Hld	Count of jobs in the Held state
Wat	Count of jobs in the Waiting state
Trn	Count of jobs in the Transiting state

Table 5-1: xpbs Server Column Headings

Heading	Meaning
Ext	Count of jobs in the Exiting state
Status	Status of the corresponding Server
PEsInUse	Count of Processing Elements (CPUs, PEs, Vnodes) in Use

5.3.3 xpbs Queues Panel

The Queues panel is composed of a leading horizontal QUEUES bar, a listbox, and a set of command buttons. The QUEUES bar lists the hosts that are consulted when listing queues; the bar also contains a minimize/maximize button for displaying or iconizing the Queues panel. The listbox displays information about queues managed by the Server host(s) selected from the Hosts panel; each listbox entry can be selected as described above for the Hosts panel.

To the right of the Queues Panel area are buttons for actions that can be performed on selected queue(s).

detail
: provides information about selected queue(s). This functionality can also be achieved by double clicking on a Queue listbox entry.

stop
: for stopping the selected queue(s). (`-admin` only)

start
: for starting the selected queue(s). (`-admin` only)

disable
: for disabling the selected queue(s). (`-admin` only)

enable
: for enabling the selected queue(s). (`-admin` only)

The middle portion of the Queues Panel has abbreviated column names indicating the information being displayed, as the following table shows:

Table 5-2: xpbs Queue Column Headings

Heading	Meaning
Max	Maximum number of jobs permitted
Tot	Count of jobs currently enqueued in any state
Ena	Is queue enabled? yes or no
Str	Is queue started? yes or no
Que	Count of jobs in the Queued state
Run	Count of jobs in the Running state
Hld	Count of jobs in the Held state
Wat	Count of jobs in the Waiting state
Trn	Count of jobs in the Transiting state
Ext	Count of jobs in the Exiting state
Type	Type of queue: execution or route
Server	Name of Server on which queue exists

5.3.4 xpbs Jobs Panel

The Jobs panel is composed of a leading horizontal JOBS bar, a listbox, and a set of command buttons. The JOBS bar lists the queues that are consulted when listing jobs; the bar also contains a minimize/maximize button for displaying or iconizing the Jobs region. The listbox displays information about jobs that are found in the queue(s) selected from the Queues listbox; each listbox entry can be selected as described above for the Hosts panel.

The region just above the Jobs listbox shows a collection of command buttons whose labels describe criteria used for filtering the Jobs listbox contents. The list of jobs can be selected according to the owner of jobs (Owners), job state (Job_States), name of the job (Job_Name), type of hold

placed on the job (Hold_Types), the account name associated with the job (Account_Name), checkpoint attribute (Checkpoint), time the job is eligible for queueing/execution (Queue_Time), resources requested by the job (Resources), priority attached to the job (Priority), and whether or not the job is rerunnable (Rerunnable).

The selection criteria can be modified by clicking on any of the appropriate command buttons to bring up a selection box. The criteria command buttons are accompanied by a *Select Jobs* button, which when clicked, will update the contents of the Jobs listbox based on the new selection criteria. Note that only jobs that meet ***all*** the selected criteria will be displayed.

Finally, to the right of the Jobs panel are the following command buttons, for operating on selected job(s):

detail
: provides information about selected job(s). This functionality can also be achieved by double-clicking on a Jobs listbox entry.

modify
: for modifying attributes of the selected job(s).

delete
: for deleting the selected job(s).

hold
: for placing some type of hold on selected job(s).

release
: for releasing held job(s).

signal
: for sending signals to selected job(s) that are running.

msg
: for writing a message into the output streams of selected job(s).

move
: for moving selected job(s) into some specified destination.

order
: for exchanging order of two selected jobs in a queue.

run
: for running selected job(s). (-admin only)

rerun

for requeueing selected job(s) that are running. (-admin only)

The middle portion of the Jobs Panel has abbreviated column names indicating the information being displayed, as the following table shows:

Table 5-3: xpbs Job Column Headings

Heading	Meaning
Job id	Job Identifier
Name	Name assigned to job, or script name
User	User name under which job is running
PEs	Number of Processing Elements (CPUs) requested
CputUse	Amount of CPU time used
WalltUse	Amount of wall-clock time used
S	State of job
Queue	Queue in which job resides

5.3.5 xpbs Info Panel

The Info panel shows the progress of the commands executed by `xpbs`. Any errors are written to this area. The INFO panel also contains a minimize/maximize button for displaying or iconizing the Info panel.

5.3.6 xpbs Keyboard Tips

There are a number of shortcuts and key sequences that can be used to speed up using `xpbs`. These include:

Tip 1.

All buttons which appear to be depressed in the dialog box/subwindow can be activated by pressing the return/enter key.

Tip 2.

Pressing the tab key will move the blinking cursor from one text field to another.

Tip 3.

To contiguously select more than one entry: left-click then drag the mouse across multiple entries.

Tip 4.

To non-contiguously select more than one entry: hold the control-left-click on the desired entries.

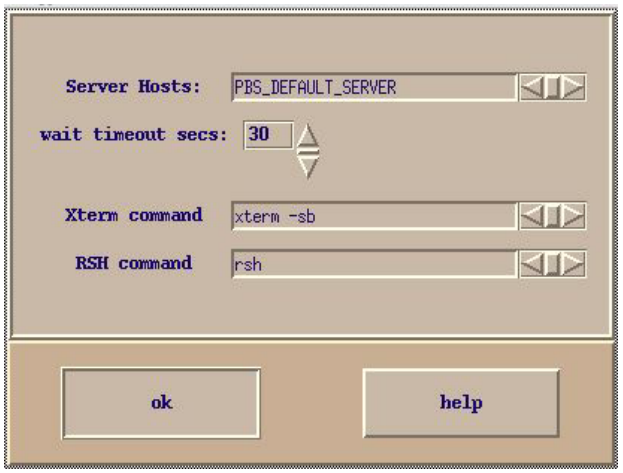
5.4 Setting xpbs Preferences

The "Preferences" button is in the Menu Bar at the top of the main `xpbs` window. Clicking it will bring up a dialog box that allows you to customize the behavior of xpbs:

1. Define Server hosts to query
2. Select wait timeout in seconds
3. Specify `xterm` command (for interactive jobs, UNIX only)

4.

Specify which `rsh`/`ssh` command to use



5.5 Relationship Between PBS and `xpbs`

`xpbs` is built on top of the PBS client commands, such that all the features of the command line interface are available through the GUI. Each "task" that you perform using `xpbs` is converted into the necessary PBS command and then run.

Table 5-4: `xpbs` Buttons and PBS Commands

Location	Command Button	PBS Command
Hosts Panel	detail	`qstat -B -f` *selected server_host(s)*
Hosts Panel	submit	`qsub` *options selected Server(s)*
Hosts Panel	terminate *	`qterm` *selected server_host(s)*
Queues Panel	detail	`qstat -Q -f` *selected queue(s)*
Queues Panel	stop *	`qstop` *selected queue(s)*
Queues Panel	start *	`qstart` *selected queue(s)*

Table 5-4: xpbs Buttons and PBS Commands

Location	Command Button	PBS Command
Queues Panel	enable *	`qenable` *selected queue(s)*
Queues Panel	disable *	`qdisable` *selected queue(s)*
Jobs Panel	detail	`qstat -f` *selected job(s)*
Jobs Panel	modify	`qalter` *selected job(s)*
Jobs Panel	delete	`qdel` *selected job(s)*
Jobs Panel	hold	`qhold` *selected job(s)*
Jobs Panel	release	`qrls` *selected job(s)*
Jobs Panel	run	`qrun` *selected job(s)*
Jobs Panel	rerun	`qrerun` *selected job(s)*
Jobs Panel	signal	`qsig` *selected job(s)*
Jobs Panel	msg	`qmsg` *selected job(s)*
Jobs Panel	move	`qmove` *selected job(s)*
Jobs Panel	order	`qorder` *selected job(s)*

* Indicates command button is visible only if `xpbs` is started with the "`-admin`" option.

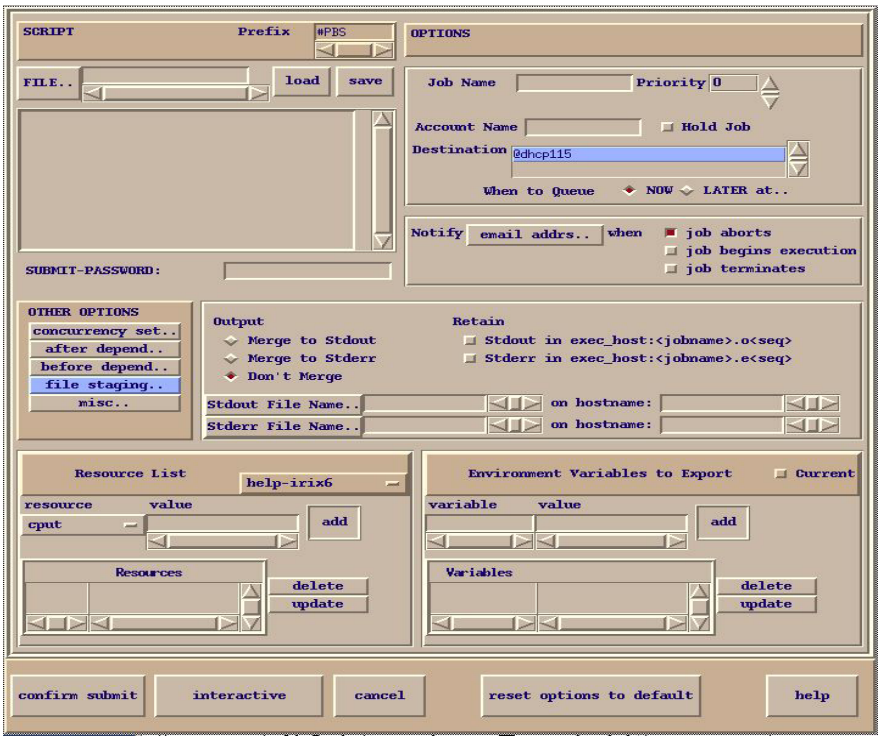
5.6 How to Submit a Job Using xpbs

To submit a job using `xpbs`, perform the following steps:

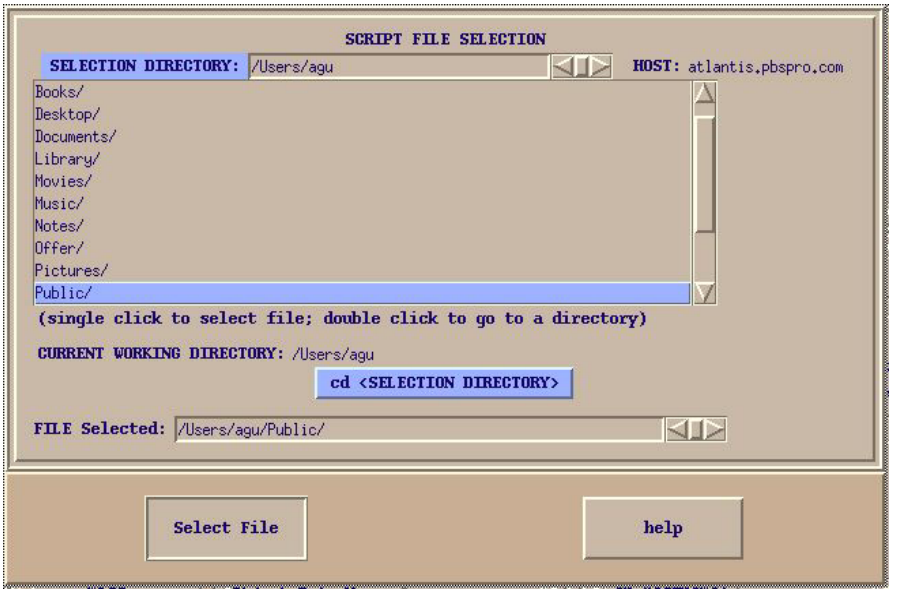
First, select a host from the HOSTS listbox in the main `xpbs` display to which you wish to submit the job.

Next, click on the *Submit* button located next to the HOSTS panel. The *Submit* button brings up the Submit Job Dialog box (see below) which is composed of four distinct regions. The Job Script File region is at the upper left. The OPTIONS region containing various widgets for setting job

attributes is scattered all over the dialog box. The OTHER OPTIONS is located just below the Job Script file region, and COMMAND BUTTONS region is at the bottom.



The job script region is composed of a header box, the text box, FILE entry box, and two buttons labeled *load* and *save.* If you have a script file containing PBS options and executable lines, then type the name of the file on the FILE entry box, and then click on the *load* button. Alternatively, you may click on the *FILE* button, which will display a File Selection browse window, from which you may point and click to select the file you wish to open. The File Selection Dialog window is shown below. Clicking on the *Select File* button will load the file into `xpbs`, just as does the *load* button described above.



The various fields in the Submit window will get loaded with values found in the script file. The script file text box will only be loaded with executable lines (non-PBS) found in the script. The job script header box has a *Prefix* entry box that can be modified to specify the PBS directive to look for when parsing a script file for PBS options.

If you don't have a existing script file to load into `xpbs`, you can start typing the executable lines of the job in the file text box.

Next, review the Destination listbox. This box shows the queues found in the host that you selected. A special entry called "`@host`" refers to the default queue at the indicated host. Select appropriately the destination queue for the job.

Next, define any required resources in the Resource List subwindow.

The resources specified in the "Resource List" section will be job-wide resources only. In order to specify chunks or job placement, use a script.

To run an array job, use a script. You will not be able to query individual subjobs or the whole job array using xpbs. Type the script into the "File: entry" box. Do not click the "Load" button. Instead, use the "Submit" button.

Finally, review the optional settings to see if any should apply to this job.

For example:

- Use the one of the buttons in the "Output" region to merge output and error files.
- Use "Stdout File Name" to define standard output file and to redirect output
- Use the "Environment Variables to Export" subwindow to have current environment variables exported to the job.
- Use the "Job Name" field in the OPTIONS subwindow to give the job a name.
- Use the "Notify email address" and one of the buttons in the OPTIONS subwindow to have PBS send you mail when the job terminates.

Now that the script is built you have four options of what to do next:

Reset options to default

Save the script to a file

Submit the job as a batch job

Submit the job as an interactive-batch job (UNIX only)

Reset clears all the information from the submit job dialog box, allowing you to create a job from a fresh start.

Use the FILE. field (in the upper left corner) to define a filename for the script. Then press the *Save* button. This will cause a PBS script file to be generated and written to the named file.

Pressing the *Confirm Submit* button at the bottom of the Submit window will submit the PBS job to the selected destination. `xpbs` will display a small window containing the job identifier returned for this job. Clicking *OK* on this window will cause it and the Submit window to be removed from your screen.

On UNIX systems (not Windows) you can alternatively submit the job as an interactive-batch job, by clicking the *Interactive* button at the bottom of the Submit Job window. Doing so will cause an X-terminal window (`xterm`) to be launched, and within that window a PBS interactive-batch job submitted. The path for the xterm command can be set via the prefer-

ences, as discussed above in section 5.4 “Setting xpbs Preferences” on page 113. For further details on usage, and restrictions, see “Interactive-batch Jobs” on page 92.)

5.7 Exiting xpbs

Click on the *Close* button located in the Menu bar to leave `xpbs`. If any settings have been changed, `xpbs` will bring up a dialog box asking for a confirmation in regards to saving state information. The settings will be saved in the `.xpbsrc` configuration file, and will be used the next time you run `xpbs`, as discussed in the following section.

5.8 The xpbs Configuration File

Upon exit, the `xpbs` state may be written to the `.xpbsrc` file in the user’s home directory. (See also section 3.9.1 “Windows User's HOMEDIR” on page 27.) Information saved includes: the selected host(s), queue(s), and job(s); the different jobs listing criteria; the view states (i.e. minimized/maximized) of the Hosts, Queues, Jobs, and INFO regions; and all settings in the Preferences section. In addition, there is a system-wide `xpbs` configuration file, maintained by the PBS Administrator, which is used in the absence of a user’s personal `.xpbsrc` file.

5.9 xpbs Preferences

The resources that can be set in the `xpbs` configuration file, `~/.xpbsrc`, are:

***serverHosts**
: List of Server hosts (space separated) to query by `xpbs`. A special keyword **PBS_DEFAULT_SERVER** can be used which will be used as a placeholder for the value obtained

from the `/etc/pbs.conf` file (UNIX) or "`[PBS Destination Folder]\pbs.conf`" file (Windows).

***timeoutSecs**
Specify the number of seconds before timing out waiting for a connection to a PBS host.

***xtermCmd**
The xterm command to run driving an interactive PBS session.

***labelFont**
Font applied to text appearing in labels.

***fixlabelFont**
Font applied to text that label fixed-width widgets such as listbox labels. This must be a fixed-width font.

***textFont**
Font applied to a text widget. Keep this as fixed-width font.

***backgroundColor**
The color applied to background of frames, buttons, entries, scrollbar handles.

***foregroundColor**
The color applied to text in any context.

***activeColor**
The color applied to the background of a selection, a selected command button, or a selected scroll bar handle.

***disabledColor**
Color applied to a disabled widget.

***signalColor**
Color applied to buttons that signal something to the user about a change of state. For example, the

color of the *Track Job* button when returned output files are detected.

***shadingColor**
A color shading applied to some of the frames to emphasize focus as well as decoration.

***selectorColor**
The color applied to the selector box of a radiobutton or checkbutton.

***selectHosts**
List of hosts (space separated) to automatically select/highlight in the HOSTS listbox.

***selectQueues**
List of queues (space separated) to automatically select/highlight in the QUEUES listbox.

***selectJobs**
List of jobs (space separated) to automatically select/highlight in the JOBS listbox.

***selectOwners**
List of owners checked when limiting the jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as "Owners: <list_of_owners>". See `-u` option in `qselect(1B)` for format of <list_of_owners>.

***selectStates**
List of job states to look for (do not space separate) when limiting the jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as "Job_States: <states_string>". See `-s` option in `qselect(1B)` for format of <states_string>.

***selectRes**
List of resource amounts (space separated) to consult when limiting the jobs appearing on the Jobs

listbox in the main `xpbs` window. Specify value as "Resources: <res_string>". See `-l` option in `qselect(1B)` for format of <res_string>.

***selectExecTime**
The Execution Time attribute to consult when limiting the list of jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as "Queue_Time: <exec_time>". See `-a` option in `qselect(1B)` for format of <exec_time>.

***selectAcctName**

The name of the account that will be checked when limiting the jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as "Account_Name: <account_name>". See `-A` option in `qselect(1B)` for format of <account_name>.

***selectCheckpoint**

The checkpoint attribute relationship (including the logical operator) to consult when limiting the list of jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as "Checkpoint: <checkpoint_arg>". See `-c` option in `qselect(1B)` for format of <checkpoint_arg>.

***selectHold**

The hold types string to look for in a job when limiting the jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as "Hold_Types: <hold_string>". See `-h` option in `qselect(1B)` for format of <hold_string>.

***selectPriority**

The priority relationship (including the logical operator) to consult when limiting the list of jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as "Priority: <priority_value>". See `-p` option in `qselect(1B)` for format of <priority_value>.

***selectRerun**

The rerunnable attribute to consult when limiting the list of jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as "Rerunnable: <rerun_val>". See `-r` option in `qselect(1B)` for format of <rerun_val>.

***selectJobName**

Name of the job that will be checked when limiting the jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as "Job_Name: <jobname>". See `-N` option in `qselect(1B)` for format of <jobname>.

***iconizeHostsView**

A boolean value (true or false) indicating whether or not to iconize the HOSTS region.

***iconizeQueuesView**

A boolean value (true or false) indicating whether or not to iconize the QUEUES region.

***iconizeJobsView**

A boolean value (true or false) indicating whether or not to iconize the JOBS region.

***iconizeInfoView**

A boolean value (true or false) indicating whether or not to iconize the INFO region.

***jobResourceList**

A curly-braced list of resource names as according to architecture known to `xpbs`. The format is as follows:

{ <arch-type1> resname1 resname2 ... resnameN }

{ <arch-type2> resname1 resname2 ... resnameN }

{ <arch-typeN> resname1 resname2 ... resnameN }

Chapter 6

Checking Job / System Status

This chapter introduces several PBS commands useful for checking status of jobs, queues, and PBS Servers. Examples for use are included, as are instructions on how to accomplish the same task using the `xpbs` graphical interface.

6.1 The `qstat` Command

The **`qstat`** command is used to the request the status of jobs, queues, and the PBS Server. The requested status is written to standard output stream (usually the user's terminal). When requesting job status, any jobs for which the user does not have view privilege are not displayed. For detailed usage information, see the qstat(1B) man page or the PBS Professional External Reference Specification.

6.1.1 Checking Job Status

Executing the `qstat` command without any options displays job information in the default format. (An alternative display format is also provided, and is discussed below.) The default display includes the following information:

- The job identifier assigned by PBS
- The job name given by the submitter
- The job owner
- The CPU time used
- The job state
- The queue in which the job resides

The job state is abbreviated to a single character:

Table 6-1: Job States

State	Description
B	Job arrays only: job array has started
E	Job is exiting after having run
H	Job is held. A job is put into a held state by the server or by a user or administrator. A job stays in a held state until it is released by a user or administrator.
Q	Job is queued, eligible to run or be routed
R	Job is running
S	Job is suspended by server. A job is put into the suspended state when a higher priority job needs the resources.
T	Job is in transition (being moved to a new location)
U	Job is suspended due to workstation becoming busy
W	Job is waiting for its requested execution time to be reached, or the job's specified stagein request has failed for some reason.

Table 6-1: Job States

State	Description
X	Subjobs only; subjob is finished (expired.)

The following example illustrates the default display of `qstat`.

```
qstat
Job id    Name         User         Time Use S Queue
--------- ------------ ------------ -------- - -----
16.south  aims14       user1               0 H workq
18.south  aims14       user1               0 W workq
26.south  airfoil      barry        00:21:03 R workq
27.south  airfoil      barry        21:09:12 R workq
28.south  myjob        user1               0 Q workq
29.south  tns3d        susan               0 Q workq
30.south  airfoil      barry               0 Q workq
31.south  seq_35_3     donald              0 Q workq
```

An alternative display (accessed via the "`-a`" option) is also provided that includes extra information about jobs, including the following additional fields:

Session ID

Number of vnodes requested

Number of parallel tasks (or CPUs)

Requested amount of memory

Requested amount of wallclock time

Walltime or CPU time, whichever submitter specified, if job is running.

```
qstat -a

                                                  Req'd  Elap
Job ID   User   Queue Jobname Ses NDS TSK Mem Time S Time
-------- ------ ----- ------- --- --- --- --- ---- - ----
16.south user1  workq aims14  --  --   1  --  0:01 H  --
18.south user1  workq aims14  --  --   1  --  0:01 W  --
```

```
51.south barry  workq airfoil 930 --   1  --  0:13 R 0:01
52.south user1  workq myjob  --  --   1  --  0:10 Q  --
53.south susan  workq tns3d   --  --   1  --  0:20 Q  --
54.south barry  workq airfoil --  --   1  --  0:13 Q  --
55.south donald workq seq_35_ --  --   1  --  2:00 Q  --
```

Other options which utilize the alternative display are discussed in subsequent sections of this chapter.

6.1.2 Viewing Specific Information

When requesting queue or Server status `qstat` will output information about each destination. The various options to `qstat` take as an operand either a job identifier or a destination. If the operand is a job identifier, it must be in the following form:

sequence_number[.server_name][@server]

where `sequence_number.server_name` is the job identifier assigned at submittal time, see `qsub`. If the `.server_name` is omitted, the name of the default Server will be used. If `@server` is supplied, the request will be for the job identifier currently at that Server.

If the operand is a destination identifier, it takes one of the following three forms:

queue

@server

queue@server

If *`queue`* is specified, the request is for status of all jobs in that queue at the default Server. If the *`@server`* form is given, the request is for status of all jobs at that Server. If a full destination identifier, *`queue@server`*, is given, the request is for status of all jobs in the named *`queue`* at the named *`server`*.

IMPORTANT:

If a PBS Server is not specified on the `qstat` command line, the default Server will be used. (See discussion of **`PBS_DEFAULT`** in “Environment Variables” on page 30.)

6.1.3 Checking Server Status

The "**-B**" option to `qstat` displays the status of the specified PBS Batch Server. One line of output is generated for each Server queried. The three letter abbreviations correspond to various job limits and counts as follows: Maximum, Total, Queued, Running, Held, Waiting, Transiting, and Exiting. The last column gives the status of the Server itself: active, idle, or scheduling.

```
qstat -B
Server      Max  Tot  Que  Run  Hld  Wat  Trn  Ext Status
----------- --- ---- ---- ---- ---- ---- ---- ---- ------
fast.domain   0   14   13    1    0    0    0    0 Active
```

When querying jobs, Servers, or queues, you can add the "-f" option to qstat to change the display to the *full* or *long* display. For example, the Server status shown above would be expanded using "-f" as shown below:

```
qstat -Bf
Server: fast.mydomain.com
   server_state = Active
   scheduling = True
   total_jobs = 14
   state_count = Transit:0 Queued:13 Held:0 Waiting:0
      Running:1 Exiting:0
   managers = user1@fast.mydomain.com
   default_queue = workq
   log_events = 511
   mail_from = adm
   query_other_jobs = True
   resources_available.mem = 64mb
   resources_available.ncpus = 2
   resources_default.ncpus = 1
   resources_assigned.ncpus = 1
   resources_assigned.nodect = 1
   scheduler_iteration = 600
   pbs_version = PBSPro_10.0.41640
```

6.1.4 Checking Queue Status

The "**-Q**" option to qstat displays the status of all (or any specified) queues at the (optionally specified) PBS Server. One line of output is generated for each queue queried. The three letter abbreviations correspond to

limits, queue states, and job counts as follows: Maximum, Total, Enabled Status, Started Status, Queued, Running, Held, Waiting, Transiting, and Exiting. The last column gives the type of the queue: *routing* or *execution*.

```
qstat -Q
Queue Max Tot Ena Str Que Run Hld Wat Trn Ext Type
----- --- --- --- --- --- --- --- --- --- --- ---------
workq   0  10 yes yes   7   1   1   1   0   0 Execution
```

The full display for a queue provides additional information:

```
 qstat -Qf
Queue: workq
   queue_type = Execution
   total_jobs = 10
   state_count = Transit:0 Queued:7 Held:1 Waiting:1
      Running:1 Exiting:0
   resources_assigned.ncpus = 1
   hasnodes = False
   enabled = True
   started = True
```

6.1.5 Viewing Job Information

We saw above that the "-f" option could be used to display full or long information for queues and Servers. The same applies to jobs. By specifying the "-f" option and a job identifier, PBS will print all information

known about the job (e.g. resources requested, resource limits, owner, source, destination, queue, etc.) as shown in the following example. (See “Job Attributes” on page 93 for a description of attribute.)

```
qstat -f 89
Job Id: 89.south
   Job_Name = tns3d
   Job_Owner = susan@south.mydomain.com
   resources_used.cput = 00:00:00
   resources_used.mem = 2700kb
   resources_used.ncpus = 1
   resources_used.vmem = 5500kb
   resources_used.walltime = 00:00:00
   job_state = R
   queue = workq
   server = south
   Checkpoint = u
   ctime = Thu Aug 23 10:11:09 2004
   Error_Path = south:/u/susan/tns3d.e89
   exec_host = south/0
   Hold_Types = n
   Join_Path = oe
   Keep_Files = n
   Mail_Points = a
   mtime = Thu Aug 23 10:41:07 2004
   Output_Path = south:/u/susan/tns3d.o89
   Priority = 0
   qtime = Thu Aug 23 10:11:09 2004
   Rerunnable = True
   Resource_List.mem = 300mb
   Resource_List.ncpus = 1
   Resource_List.walltime = 00:20:00
   session_id = 2083
```

```
Variable_List = PBS_O_HOME=/u/
susan,PBS_O_LANG=en_US,
PBS_O_LOGNAME=susan,PBS_O_PATH=/bin:/usr/
bin,PBS_O_SHELL=/bin/csh,PBS_O_HOST=south,
PBS_O_WORKDIR=/u/susan,PBS_O_SYSTEM=Linux,
PBS_O_QUEUE=workq
euser = susan
egroup = myegroup
queue_type = E
comment = Job run on host south - started at 10:41
etime = Thu Aug 23 10:11:09 2004
```

6.1.6 List User-Specific Jobs

The "**-u**" option to `qstat` displays jobs owned by any of a list of user names specified. The syntax of the list of users is:

user_name[@host][,user_name[@host],...]

Host names are not required, and may be "wild carded" on the left end, e.g. "`*.mydomain.com`". *user_name* without a "`@host`" is equivalent to "`user_name@*`", that is at any host.

qstat -u user1

```
                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
16.south user1  workq aims14   --   --   1  -- 0:01 H  --
18.south user1  workq aims14   --   --   1  -- 0:01 W  --
52.south user1  workq my_job   --   --   1  -- 0:10 Q  --
```

qstat -u user1,barry

```
51.south barry  workq airfoil  930  --   1  -- 0:13 R 0:01
52.south user1  workq my_job   --   --   1  -- 0:10 Q  --
54.south barry  workq airfoil  --   --   1  -- 0:13 Q  --
```

6.1.7 List Running Jobs

The "**-r**" option to `qstat` displays the status of all running jobs at the (optionally specified) PBS Server. Running jobs include those that are running and suspended. One line of output is generated for each job reported, and the information is presented in the alternative display.

6.1.8 List Non-Running Jobs

The "**-i**" option to `qstat` displays the status of all non-running jobs at the (optionally specified) PBS Server. Non-running jobs include those that are queued, held, and waiting. One line of output is generated for each job reported, and the information is presented in the alternative display (see description above).

6.1.9 Display Size in Gigabytes

The "**-G**" option to `qstat` displays all jobs at the requested (or default) Server using the alternative display, showing all size information in gigabytes (GB) rather than the default of smallest displayable units. Note that if the size specified is less than 1 GB, then the amount if rounded up to 1 GB.

6.1.10 Display Size in Megawords

The "**-M**" option to `qstat` displays all jobs at the requested (or default) Server using the alternative display, showing all size information in megawords (MW) rather than the default of smallest displayable units. A word is considered to be 8 bytes.

6.1.11 List Hosts Assigned to Jobs

The "**-n**" option to `qstat` displays the hosts allocated to any running job at the (optionally specified) PBS Server, in addition to the other information presented in the alternative display. The host information is printed

immediately below the job (see job 51 in the example below), and includes the host name and number of virtual processors assigned to the job (i.e. "south/0", where "south" is the host name, followed by the virtual processor(s) assigned.). A text string of "--" is printed for non-running jobs. Notice the differences between the queued and running jobs in the example below:

```
qstat -n

                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
16.south user1  workq aims14   --   --   1  -- 0:01 H  --
    --
18.south user1  workq aims14   --   --   1  -- 0:01 W  --
    --
51.south barry  workq airfoil  930  --   1  -- 0:13 R
    0:01 south/0
52.south user1  workq my_job   --   --   1  -- 0:10 Q  --
    --
```

6.1.12 Display Job Comments

The "**-s**" option to qstat displays the job comments, in addition to the other information presented in the alternative display. The job comment is printed immediately below the job. By default the job comment is updated by the Scheduler with the reason why a given job is not running, or when the job began executing. A text string of "--" is printed for jobs whose comment has not yet been set. The example below illustrates the different type of messages that may be displayed:

```
qstat -s

                                                  Req'd  Elap
Job ID   User  Queue Jobname Sess NDS TSK Mem Time S Time
-------- ----- ----- ------- ---- --- --- --- ---- - ----
16.south user1 workq aims14   --   --   1  -- 0:01 H  --
    Job held by user1 on Wed Aug 22 13:06:11 2004
18.south user1 workq aims14   --   --   1  -- 0:01 W  --
```

```
    Waiting on user requested start time
51.south barry workq airfoil  930  --   1  -- 0:13 R 0:01
    Job run on host south - started Thu Aug 23 at 10:56
52.south user1 workq my_job   --   --   1  -- 0:10 Q  --
    Not Running: No available resources on nodes
57.south susan workq solver   --   --   2  -- 0:20 Q  --
    --
```

6.1.13 Display Queue Limits

The "**-q**" option to `qstat` displays any limits set on the requested (or default) queues. Since PBS is shipped with no queue limits set, any visible limits will be site-specific. The limits are listed in the format shown below.

```
qstat -q
server: south

Queue  Memory CPU Time Walltime Node Run Que Lm  State
------ ------ -------- -------- ---- --- --- --  -----
workq    --      --       --     --    1   8 --   E R
```

6.1.14 Show State of Job, Job Array or Subjob

The "-t" option to qstat will show the state of a job, a job array object, and all non-X subjobs. In combination with "-J", qstat will show only the state of subjobs.

6.1.15 Viewing Job Status in Wide Format

The `–w qstat` option displays job status in wide format. The total width of the display is extended from 80 characters to 120 characters. The Job ID column can be up to 30 characters wide, while the Username, Queue, and Jobname column can be up to 15 characters wide. The SessID column can be up to eight characters wide, and the NDS column can be up to four characters wide.

Note: You can use this option only with the `–a, -n`, or `–s qstat` options.

6.1.16 Show state of Job Arrays

The “-J” option to qstat will show only the state of job arrays. In combination with “-t”, qstat will show only the state of subjobs.

6.1.17 Print Job Array Percentage Completed

The “-p” option to qstat prints the default display, with a column for Percentage Completed. For a job array, this is the number of subjobs completed and deleted, divided by the total number of subjobs.

6.1.18 Getting Information on Jobs Moved to Another Server

If your site is using peer scheduling, your job may be moved to a server that is not your default server. When that happens, you will need to give the job ID as an argument to qstat. If you use only “qstat”, your job will not appear to exist. For example: you submit a job to ServerA, and it returns the jobid as “123.ServerA”. Then 123.ServerA is moved to ServerB. In this case, use

```
qstat 123
```

or

```
qstat 123.ServerA
```

to get information about your job. ServerA will query ServerB for the information. To list all jobs at ServerB, you can use:

```
qstat @ServerB
```

If you use “qstat” without the job ID, the job will not appear to exist.

6.1.19 Viewing Resources Allocated to a Job

The exec_vnode attribute displayed via `qstat` shows the allocated resources on each vnode.

The exec_vnode line looks like:

```
exec_vnode = hostA:ncpus=1
```

For example, a job requesting

```
-l select=2:ncpus=1:mem=1gb+1:ncpus=4:mem=2gb
```

would get an exec_vnode of

```
exec_vnode =
   (VNA:ncpus=1:mem=1gb)+(VNB:ncpus=1:mem=1gb)\
   +(VNC:ncpus=4:mem=2gb)
```

Note that the vnodes and resources required to satisfy a chunk are grouped by parentheses. In the example above, if two vnodes on a single host were required to satisfy the last chunk, the exec_vnode might be:

```
exec_vnode =
   (VNA:ncpus=1:mem=1gb)+(VNB:ncpus=1:mem=1gb)
   +(VNC1:ncpus=2:mem=1gb+VNC2:ncpus=2:mem=1gb)
```

You cannot use the `qstat` command to view any custom resource which has been created to be invisible or unrequestable, whether this resource is on a queue, the server, or is a job attribute. See section 4.5.14 “Resource Permissions” on page 58.

6.2 Viewing Job / System Status with xpbs

The main display of `xpbs` shows a brief listing of all selected Servers, all queues on those Servers, and any jobs in those queues that match the *selection criteria* (discussed below). Servers are listed in the HOST panel near the top of the display.

To view detailed information about a given Server (i.e. similar to that produced by "qstat -fB") select the Server in question, then click the "Detail" button. Likewise, for details on a given queue (i.e. similar to that produced by "qstat -fQ") select the queue in question, then click its corresponding "Detail" button. The same applies for jobs as well (i.e. "qstat -f"). You can view detailed information on any displayed job by selecting it, and then clicking on the "Detail" button. Note that the list of jobs displayed will be dependent upon the Selection Criteria currently selected. This is discussed in the xpbs portion of the next section.

6.3 The qselect Command

The **qselect** command provides a method to list the job identifier of those jobs, job arrays or subjobs which meet a list of selection criteria. Jobs are selected from those owned by a single Server. When qselect successfully completes, it will have written to standard output a list of zero or more job identifiers which meet the criteria specified by the options. Each option acts as a filter restricting the number of jobs which might be listed. With no options, the qselect command will list all jobs at the Server which the user is authorized to list (query status of). The -u option may be used to limit the selection to jobs owned by this user or other specified users.

When an option is specified with a optional *op* component to the option argument, then *op* specifies a relation between the value of a certain job attribute and the value component of the option argument. If an *op* is allowable on an option, then the description of the option letter will indicate that *op* is allowable. The only acceptable strings for the *op* component, and the relation the string indicates, are shown in the following list:

.eq.
: The value represented by the attribute of the job is equal to the value represented by the option argument.

.ne.
: The value represented by the attribute of the job is not equal to the value represented by the option argument.

.ge.
: The value represented by the attribute of the job is greater than or equal to the value represented by the option argument.

.gt.
: The value represented by the attribute of the job is greater than the value represented by the option argument.

.le.
: The value represented by the attribute of the job is less than or equal to the value represented by the option argument.

.lt.
: The value represented by the attribute of the job is less than the value represented by the option argument.

The available options to `qselect` are:

-a [op]date_time
: Restricts selection to a specific time, or a range of times. The `qselect` command selects only jobs for which the value of the *Execution_Time* attribute is related to the *date_time* argument by the optional *op* operator. The *date_time* argument is in the POSIX date format:

  [[CC]YY]MMDDhhmm[.SS]

  where the `MM` is the two digits for the month, `DD` is the day of the month, `hh` is the hour, `mm` is the minute, and the optional `SS` is the seconds. `CC` is the century and `YY` the year. If op is not specified, jobs will be selected for which the *Execution_Time* and *date_time* values are equal.

-A account_string
: Restricts selection to jobs whose *Account_Name* attribute matches the specified account_string.

-c [op] interval
: Restricts selection to jobs whose *Checkpoint* interval attribute matches the specified relationship. The values of the *Checkpoint* attribute are defined to have the following ordered relationship:

  n > s > c=minutes > c > u

If the optional op is not specified, jobs will be selected whose *Checkpoint* attribute is equal to the interval argument.

-h hold_list

Restricts the selection of jobs to those with a specific set of hold types. Only those jobs will be selected whose *Hold_Types* attribute exactly match the value of the *hold_list* argument. The *hold_list* argument is a string consisting of one or more occurrences the single letter `n`, or one or more of the letters `u`, `o`, `p`, or `s` in any combination. If letters are duplicated, they are treated as if they occurred once. The letters represent the hold types:

Table 6-2: Hold Types

Letter	Meaning
n	none
u	user
o	operator
p	bad password (Windows only)
s	system

-J

Shows only job array identifiers

-l resource_list

Restricts selection of jobs to those with specified resource amounts. Only those jobs will be selected whose *Resource_List* attribute matches the specified relation with each resource and value listed in the *resource_list* argument. The relation operator *op* **must** be present. The *resource_list* is in the following format:

resource_name**op**value[,resource_name**op**val,...]

You cannot use the `qselect` command to select jobs based on a custom resource which has been created to be invisible or unrequestable. See section 4.5.14 “Resource Permissions” on page 58.

-N name

Restricts selection of jobs to those with a specific name.

-p [op]priority

Restricts selection of jobs to those with a priority that matches the specified relationship. If *op* is not specified, jobs are selected for which the job Priority attribute is equal to the priority.

-q destination

Restricts selection to those jobs residing at the specified destination. The destination may be one of the following three forms:

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

If the `-q` option is not specified, jobs will be selected from the default Server. If the destination describes only a queue, only jobs in that queue on the default batch Server will be selected. If the destination describes only a Server, then jobs in all queues on that Server will be selected. If the destination describes both a queue and a Server, then only jobs in the named queue on the named Server will be selected.

-r rerun

Restricts selection of jobs to those with the specified *Rerunnable* attribute. The option argument must be a single character. The following two characters are supported by PBS: `y` and `n`.

-s states

Restricts job selection to those in the specified states. The *states* argument is a character string which consists of any combination of the characters: `B`,`E`, `H`, `Q`, `R`, `S`, `T`, `U`, `W` and `X`. The characters in the *states* argument list states shown in the table titled "Job States" on page 126.

Jobs will be selected which are in any of the specified *states*.

-t

Shows job, job array and subjob identifiers.

-u user_list

Restricts selection to jobs owned by the specified user names. This provides a means of limiting the selection to jobs owned by one or more users. The syntax of the *user_list* is:

user_name[@host][,user_name[@host],...]

Host names may be wild carded on the left end, e.g. "`*.mydomain.com`". *User_name* without a "`@host`" is equivalent to "`user_name@*`", i.e. at any host. Jobs will be selected which are owned by the listed users at the corresponding hosts.

For example, say you want to list all jobs owned by user "barry" that requested more than 16 CPUs. You could use the following `qselect` command syntax:

```
qselect -u barry -l ncpus.gt.16
121.south
133.south
154.south
```

Notice that what is returned is the job identifiers of jobs that match the selection criteria. This may or may not be enough information for your purposes. Many users will use shell syntax to pass the list of job identifiers directly into `qstat` for viewing purposes, as shown in the next example (necessarily different between UNIX and Windows).

UNIX:

```
qstat -a  ' qselect -u barry -l ncpus.gt.16 '
                                                  Req'd     Elap
Job ID    User  Queue Jobname Sess NDS TSK Mem Time S Time
--------  ----- ----- ------- ---- --- --- --- ---- - ----
121.south barry workq airfoil  --   --  32  -- 0:01 H   --
133.south barry workq trialx   --   --  20  -- 0:01 W   --
154.south barry workq airfoil 930   --  32  -- 1:30 R 0:32
```

Windows (type the following at the `cmd` prompt, all on one line):

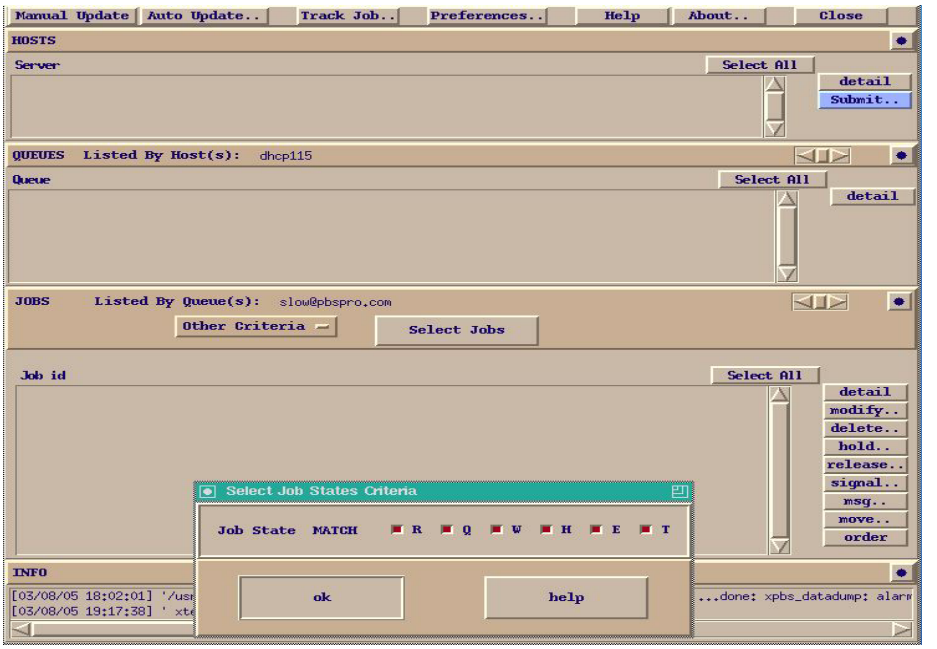
```
for /F "usebackq" %j in (`qselect -u barry -l
   ncpus.gt.16`)  do
( qstat -a %j )
121.south
133.south
154.south
```

Note: This technique of using the output of the `qselect` command as input to `qstat` can also be used to supply input to other PBS commands as well.

6.4 Selecting Jobs Using xpbs

The `xpbs` command provides a graphical means of specifying job selection criteria, offering the flexibility of the `qselect` command in a point and click interface. Above the JOBS panel in the main `xpbs` display is the *Other Criteria* button. Clicking it will bring up a menu that lets you choose and select any job selection criteria you wish.

The example below shows a user clicking on the *Other Criteria* button, then selecting *Job States*, to reveal that all job states are currently selected. Clicking on any of these job states would remove that state from the selection criteria.



You may specify as many or as few selection criteria as you wish. When you have completed your selection, click on the *Select Jobs* button above the HOSTS panel to have `xpbs` refresh the display with the jobs that match your selection criteria. The selected criteria will remain in effect until you change them again. If you exit `xpbs`, you will be prompted if you wish to save your configuration information; this includes the job selection criteria.

6.5 Using `xpbs` TrackJob Feature

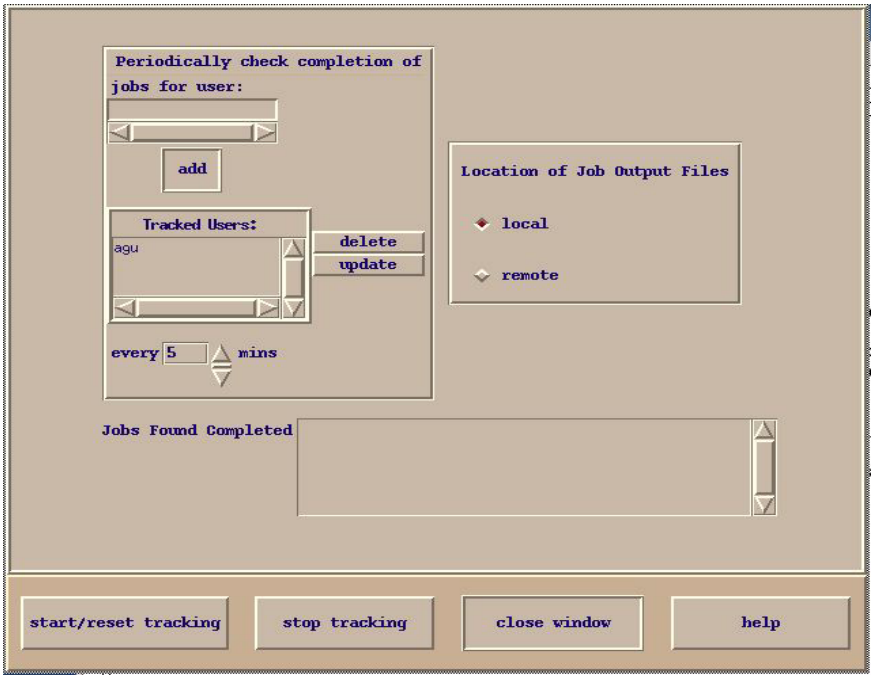
The `xpbs` command includes a feature that allows you to track the progress of your jobs. When you enable the *Track Job* feature, `xpbs` will monitor your jobs, looking for the output files that signal completion of the job. The *Track Job* button will flash red on the `xpbs` main display, and if you then click it, `xpbs` will display a list of all completed jobs (that you

were previously tracking). Selecting one of those jobs will launch a window containing the standard output and standard error files associated with the job.

IMPORTANT:

The Track Job feature is not currently available on Windows.

To enable `xpbs` job tracking, click on the *Track Job* button at the top center of the main `xpbs` display. Doing so will bring up the Track Job dialog box shown below.



From this window you can name the users whose jobs you wish to monitor. You also need to specify where you expect the output files to be: either local or remote (e.g. will the files be retained on the Server host, or did you request them to be delivered to another host?). Next, click the *start/reset tracking button* and then the *close window* button. Note that you can disable job tracking at any time by clicking the *Track Job* button on the main `xpbs` display, and then clicking the *stop tracking* button.

Chapter 7

Working With PBS Jobs

This chapter introduces the reader to various commands useful in working with PBS jobs. Covered topics include: modifying job attributes, holding and releasing jobs, sending messages to jobs, changing order of jobs within a queue, sending signals to jobs, and deleting jobs. In each section below, the command line method for accomplishing a particular task is presented first, followed by the `xpbs` method.

7.1 Modifying Job Attributes

Most attributes can be changed by the owner of the job (or a manager or operator) while the job is still queued. However, once a job begins execution, the only resources that can be modified are `cputime` and `walltime`. These can only be reduced.

When the qalter "-l" option is used to alter the resource list of a queued job, it is important to understand the interactions between altering the select directive and job limits.

If the job was submitted with an explicit "-l select=", then vnode-level resources must be qaltered using the "-l select=" form. In this case a vnode level resource RES cannot be qaltered with the "-l RES" form.

For example:

Submit the job:

% qsub -l select=1:ncpus=2:mem=512mb jobscript

Job's ID is 230

qalter the job using "-l RES" form:

% qalter -l ncpus=4 230

Error reported by qalter:

qalter: Resource must only appear in "select"

specification when select is used: ncpus 230

qalter the job using the "-l select=" form:

% qalter -l select=1:ncpus=4:mem=512mb 230

No error reported by qalter:

%

7.1.1 Changing the Selection Directive

If the selection directive is altered, the job limits for any consumable resource in the directive are also modified.

For example, if a job is queued with the following resource list:

select=2:ncpus=1:mem=5gb, ncpus=2, mem=10gb

and the selection directive is altered to request

select=3:ncpus=2:mem=6gb

then the job limits are reset to ncpus=6 and mem=18gb

7.1.2 Changing the Job-wide Limit

However, if the job-wide limit is modified, the corresponding resources in the selection directive are not modified. It would be impossible to determine where to apply the changes in a compound directive.

Reducing a job-wide limit to a new value less than the sum of the resource in the directive is strongly discouraged. This may produce a situation where the job is aborted during execution for exceeding its limits. The actual effect of such a modification is not specified.

If a job is queued, requested modifications must still fit within the queue's and server's job resource limits. If a requested modification to a resource would exceed the queue's or server's job resource limits, the resource request will be rejected.

Resources are modified by using the `-l` option, either in chunks inside of selection statements, or in job-wide modifications using `resource_name=value` pairs. The selection statement is of the form:

```
-l select=[N:]chunk[+[N:]chunk ...]
```

where N specifies how many of that chunk, and a chunk is of the form:

```
resource_name=value[:resource_name=value ...]
```

Job-wide resource_name=value modifications are of the form:

```
-l resource_name=value[,resource_name=value
    ...]
```

It is an error to use a boolean resource as a job-wide limit.

Placement of jobs on vnodes is changed using the place statement:

```
-l place=modifier[:modifier]
```

where modifier is any combination of group, excl, and/or one of free|pack|scatter.

The usage syntax for `qalter` is:

qalter job-resources job-list

The following examples illustrate how to use the `qalter` command. First we list all the jobs of a particular user. Then we modify two attributes as shown (increasing the wall-clock time from 20 to 25 minutes, and changing the job name from “airfoil” to “engine”):

```
qstat -u barry
                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
51.south barry  workq airfoil  930  --   1  -- 0:16 R 0:01
54.south barry  workq airfoil  --   --   1  -- 0:20 Q  --

qalter -l walltime=20:00 -N engine 54
qstat -a 54
                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
54.south barry  workq engine   --   --   1  -- 0:25 Q  --
```

To alter a job attribute via `xpbs`, first select the job(s) of interest, and the click on *modify* button. Doing so will bring up the *Modify Job Attributes* dialog box. From this window you may set the new values for any attribute you are permitted to change. Then click on the *confirm modify* button at the lower left of the window.

The qalter command can be used on job arrays, but not on subjobs or ranges of subjobs. When used with job arrays, any job array identifiers must be enclosed in double quotes, e.g.:

```
qalter -l walltime=25:00 “1234[].south”
```

You cannot use the `qalter` command (or any other command) to alter a custom resource which has been created to be invisible or unrequestable. See section 4.5.14 “Resource Permissions” on page 58.

For more information, see the `qalter(1B)` manual page.

7.2 Holding and Releasing Jobs

PBS provides a pair of commands to hold and release jobs. To hold a job is to mark it as ineligible to run until the hold on the job is “released”.

The **`qhold`** command requests that a Server place one or more holds on a job. A job that has a hold is not eligible for execution. There are three types of holds: *user*, *operator*, and *system*. A user may place a *user* hold upon any job the user owns. An “operator”, who is a user with “operator privilege”, may place either an *user* or an *operator* hold on any job. The PBS Manager may place any hold on any job. The usage syntax of the `qhold` command is:

qhold [-h hold_list] job_identifier ...

Note that for a job array the `job_identifier` must be enclosed in double quotes.

The *`hold_list`* defines the type of holds to be placed on the job. The *`hold_list`* argument is a string consisting of one or more of the letters `u`, `p`, `o`, or `s` in any combination, or the letter `n`. The hold type associated with each letter is:

Table 7-1:

Letter	Meaning
n	none - no hold type specified
u	user - the user may set and release this hold type
p	password - set if job fails due to a bad password; can be unset by the user
o	operator; require operator privilege to unset

Table 7-1:

Letter	Meaning
s	system - requires manager privilege to unset

If no `-h` option is given, the *user* hold will be applied to the jobs described by the *`job_identifier`* operand list. If the job identified by *`job_identifier`* is in the queued, held, or waiting states, then all that occurs is that the hold type is added to the job. The job is then placed into held state if it resides in an execution queue.

If the job is running, then the following additional action is taken to interrupt the execution of the job. If the job is checkpointable, requesting a hold on a running job will cause (1) the job to be checkpointed, (2) the resources assigned to the job to be released, and (3) the job to be placed in the held state in the execution queue. If the job is not checkpointable, `qhold` will only set the requested hold attribute. This will have no effect unless the job is requeued with the `qrerun` command. See section 4.13.13.1 “Checkpointable Jobs” on page 86.

The qhold command can be used on job arrays, but not on subjobs or ranges of subjobs. On job arrays, the qhold command can be applied only in the ‘Q’, ‘B’ or ‘W’ states. This will put the job array in the ‘H’, held, state. If any subjobs are running, they will run to completion. Job arrays cannot be moved in the ‘H’ state if any subjobs are running.

Checkpointing is not supported for job arrays. Even on systems that support checkpointing, no subjobs will be checkpointed -- they will run to completion.

Similarly, the **`qrls`** command releases a hold on a job. However, the user executing the `qrls` command must have the necessary privilege to release a given hold. The same rules apply for releasing a hold as exist for setting a hold.

The qrls command can only be used with job array objects, not with subjobs or ranges. The job array will be returned to its pre-hold state, which can be either ‘Q’, ‘B’, or ‘W’.

The usage syntax of the `qrls` command is:

qrls [-h hold_list] job_identifier ...

For job arrays, the `job_identifier` must be enclosed in double quotes.

The following examples illustrate how to use both the `qhold` and `qrls` commands. Notice that the state ("S") column shows how the state of the job changes with the use of these two commands.

```
qstat -a 54

                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
54.south barry  workq  engine  --   --   1  -- 0:20 Q  --

qhold 54
qstat -a 54

                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
54.south barry  workq  engine  --   --   1  -- 0:20 H  --

qrls -h u 54
qstat -a 54

                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
54.south barry  workq  engine  --   --   1  -- 0:20 Q  --
```

If you attempted to release a hold on a job which is not on hold, the request will be ignored. If you use the `qrls` command to release a hold on a job that had been previously running, and subsequently checkpointed, the hold will be released, and the job will return to the queued (Q) state (and be eligible to be scheduled to run when resources come available).

To hold (or release) a job using `xpbs`, first select the job(s) of interest, then click the *hold* (or *release*) button.

7.3 Deleting Jobs

PBS provides the **`qdel`** command for deleting jobs from the system. The `qdel` command deletes jobs in the order in which their job identifiers are presented to the command. A job that has been deleted is no longer subject to management by PBS. A batch job may be deleted by its owner, a PBS operator, or a PBS administrator.

Example:

```
qdel 51
qdel 1234[].server
```

Job array identifiers must be enclosed in double quotes.

Mail is sent for each job deleted unless you specify otherwise. Use the following option to `qdel` to prevent more email than you want from being sent:

```
-Wsuppress_email=<N>
```

`N` must be a non-negative integer. Make `N` the largest number of emails you wish to receive. PBS will send one email for each deleted job, up to `N`. Note that a job array is one job, so deleting a job array results in one email being sent.

To delete a job using `xpbs`, first select the job(s) of interest, then click the *delete* button.

7.4 Sending Messages to Jobs

To send a message to a job is to write a message string into one or more output files of the job. Typically this is done to leave an informative message in the output of the job. Such messages can be written using the **`qmsg`** command.

IMPORTANT:

A message can only be sent to running jobs.

The usage syntax of the qmsg command is:

qmsg [-E][-O] message_string job_identifier

Example:

```
qmsg -O "output file message" 54
qmsg -O "output file message" "1234[].server"
```

Job array identifiers must be enclosed in double quotes.

The -E option writes the message into the error file of the specified job(s). The -O option writes the message into the output file of the specified job(s). If neither option is specified, the message will be written to the error file of the job.

The first operand, *message_string*, is the message to be written. If the string contains blanks, the string must be quoted. If the final character of the string is not a newline, a newline character will be added when written to the job's file. All remaining operands are *job_identifiers* which specify the jobs to receive the message string. For example:

```
qmsg -E "hello to my error (.e) file" 55
qmsg -O "hello to my output (.o) file" 55
qmsg "this too will go to my error (.e) file" 55
```

To send a message to a job using xpbs, first select the job(s) of interest, then click the *msg* button. Doing so will launch the *Send Message to Job* dialog box. From this window, you may enter the message you wish to send and indicate whether it should be written to the standard output or the standard error file of the job. Click the *Send Message* button to complete the process.

7.5 Sending Signals to Jobs

The **qsig** command requests that a signal be sent to executing PBS jobs. The signal is sent to the session leader of the job. Usage syntax of the qsig command is:

qsig [-s signal] job_identifier

Job array job_identifiers must be enclosed in double quotes.

If the -s option is not specified, SIGTERM is sent. If the -s option is specified, it declares which *signal* is sent to the job. The *signal* argument is either a signal name, e.g. SIGKILL, the signal name without the SIG prefix, e.g. KILL, or an unsigned signal number, e.g. 9. The signal name SIGNULL is allowed; the Server will send the signal 0 to the job which will have no effect. Not all signal names will be recognized by qsig. If it doesn't recognize the signal name, try issuing the signal number instead. The request to signal a batch job will be rejected if:

- The user is not authorized to signal the job.
- The job is not in the running state.
- The requested signal is not supported by the execution host.
- The job is exiting.

Two special signal names, "suspend" and "resume", (note, all lower case), are used to suspend and resume jobs. When suspended, a job continues to occupy system resources but is not executing and is not charged for wall-time. Manager or operator privilege is required to suspend or resume a job.

The three examples below all send a signal 9 (SIGKILL) to job 34:

```
qsig -s SIGKILL 34
qsig -s KILL 34
```

IMPORTANT:

On most UNIX systems the command "kill -l" (that's 'minus ell') will list all the available signals.

To send a signal to a job using xpbs, first select the job(s) of interest, then click the *signal* button. Doing so will launch the *Signal Running Job* dialog box.

From this window, you may click on any of the common signals, or you may enter the signal number or signal name you wish to send to the job. Click the *Signal* button to complete the process.

7.6 Changing Order of Jobs

PBS provides the **qorder** command to change the order of two jobs, within or across queues. To order two jobs is to exchange the jobs' positions in the queue or queues in which the jobs reside. If job1 is at position 3 in queue A and job2 is at position 4 in queue B, qordering them will result in job1 being in position 4 in queue B and job2 being in position 3 in queue A. The two jobs must be located at the same Server, and both jobs must be owned by the user. No attribute of the job (such as priority) is changed. The impact of changing the order within the queue(s) is dependent on local job scheduling policy; contact your systems administrator for details.

IMPORTANT:

A job in the running state cannot be reordered.

Usage of the `qorder` command is:

qorder job_identifier1 job_identifier2

Job array identifiers must be enclosed in double quotes.

Both operands are *job_identifiers* which specify the jobs to be exchanged.

```
qstat -u bob

                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
54.south bob    workq twinkie  --   --   1  -- 0:20 Q  --
63[].south bob  workq airfoil  --   --   1  -- 0:13 Q  --

qorder 54 "63[]"
qstat -u bob

                                                  Req'd     Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
63[].south bob  workq airfoil  --   --   1  -- 0:13 Q  --
54.south bob    workq twinkie  --   --   1  -- 0:20 Q  --
```

To change the order of two jobs using `xpbs`, select the two jobs, and then click the *order* button.

The qorder command can only be used with job array objects, not on subjobs or ranges. This will change the queue order of the job array in association with other jobs or job arrays in the queue.

7.7 Moving Jobs Between Queues

PBS provides the **`qmove`** command to move jobs between different queues (even queues on different Servers). To move a job is to remove the job from the queue in which it resides and instantiate the job in another queue.

IMPORTANT:

A job in the running state cannot be moved.

The usage syntax of the `qmove` command is:

qmove destination job_identifier(s)

Job array `job_identifiers` must be enclosed in double quotes.

The first operand is the new destination for

queue

@server

queue@server

If the ***`destination`*** operand describes only a queue, then `qmove` will move jobs into the queue of the specified name at the job's current Server. If the ***`destination`*** operand describes only a Server, then `qmove` will move jobs into the default queue at that Server. If the ***`destination`*** operand describes both a queue and a Server, then `qmove` will move the jobs into the specified queue at the specified Server. All following operands are ***`job_identifiers`*** which specify the jobs to be moved to the new ***`destination`***.

To move jobs between queues or between Servers using xpbs, select the job(s) of interest, and then click the move button. Doing so will launch the Move Job dialog box from which you can select the queue and/or Server to which you want the job(s) moved.

The qmove command can only be used with job array objects, not with subjobs or ranges. Job arrays can only be moved from one server to another if they are in the 'Q', 'H', or 'W' states, and only if there are no running subjobs. The state of the job array object is preserved in the move. The job array will run to completion on the new server.

As with jobs, a qstat on the server from which the job array was moved will not show the job array. A qstat on the job array object will be redirected to the new server.

Note: The subjob accounting records will be split between the two servers.

7.8 Converting a Job into a Reservation Job

The pbs_rsub command can be used to convert a normal job into a reservation job that will run as soon as possible. PBS creates a reservation queue and a reservation, and moves the job into the queue. Other jobs can also be moved into that queue via qmove(1B) or submitted to that queue via qsub(1B). The reservation is called an ASAP reservation.

The format for converting a normal job into a reservation job is:

pbs_rsub [-l walltime=time] -W qmove=job_identifier

Example:

```
pbs_rsub -W qmove=54
pbs_rsub -W qmove="1234[].server"
```

The -R and -E options to pbs_rsub are disabled when using the -W qmove option.

For more information, see "Advance and Standing Reservation of Resources" on page 182, and the pbs_rsub(1B), qsub(1B) and qmove(1B) manual pages.

A job's default walltime is 5 years. Therefore an ASAP reservation's start time can be in 5 years, if all the jobs in the system have the default walltime.

You cannot use the `pbs_rsub` command (or any other command) to request a custom resource which has been created to be invisible or unrequestable. See section 4.5.14 "Resource Permissions" on page 58.

Chapter 8

Advanced PBS Features

This chapter covers the less commonly used commands and more complex topics which will add substantial functionality to your use of PBS. The reader is advised to read chapters 5 - 7 of this manual first.

8.1 New Features

8.1.1 Job-Specific Staging and Execution Directories

PBS can now provide a staging and execution directory for each job. Jobs have new attributes `sandbox` and `jobdir`, the MOM has a new option `$jobdir_root`, and there is a new environment variable called `PBS_JOBDIR`. If the job's `sandbox` attribute is set to `PRIVATE`, PBS creates a job-specific staging and execution directory. If the job's `sand-`

box attribute is unset or is set to HOME, PBS uses the user's home directory for staging and execution, which is how previous versions of PBS behaved. Note that where local pathnames used in staging used to be relative to the user's home directory, they are now relative to the staging and execution directory. See section 8.7 "Input/Output File Staging" on page 167.

8.1.2 Standing Reservations

PBS now provides a facility for making standing reservations. A standing reservation is a series of advance reservations. The pbs_rsub command is used to create both advance and standing reservations. See section 8.9 "Advance and Standing Reservation of Resources" on page 182.

8.2 UNIX Job Exit Status

On UNIX systems, the exit status of a job is normally the exit status of the shell executing the job script. If a user is using `csh` and has a `.logout` file in the home directory, the exit status of `csh` becomes the exit status of the last command in `.logout`. This may impact the use of job dependencies which depend on the job's exit status. To preserve the job's exit status, the user may either remove `.logout` or edit it as shown in this example:

```
set EXITVAL = $status
[ .logout's original content ]
```

Doing so will ensure that the exit status of the job persists across the invocation of the `.logout` file.

The exit status of a job array is determined by the status of each of the completed subjobs. It is only available when all valid subjobs have completed. The individual exit status of a completed subjob is passed to the epilogue, and is available in the 'E' accounting log record of that subjob. See "Job Array Exit Status" on page 233.

8.3 Changing UNIX Job umask

The "`-W umask=`*nnn*" option to `qsub` allows you to specify, on UNIX systems, what `umask` PBS should use when creating and/or copying your `stdout` and `stderr` files, and any other files you direct PBS to transfer on your behalf.

IMPORTANT:

This feature does not apply to Windows.

The following example illustrates how to set your `umask` to 022 (i.e. to have files created with write permission for owner only: `-rw-r--r--`).

```
qsub -W umask=022 my_job
#PBS -W umask=022
```

8.4 Requesting qsub Wait for Job Completion

The "`-W block=true`" option to `qsub` allows you to specify that you want `qsub` to wait for the job to complete (i.e. "block") and report the exit value of the job. If job submission fails, no special processing will take place. If the job is successfully submitted, `qsub` will block until the job terminates or an error occurs.

If `qsub` receives one of the signals: `SIGHUP`, `SIGINT`, or `SIGTERM`, it will print a message and then exit with the exit status 2. If the job is deleted before running to completion, or an internal PBS error occurs, an error message describing the situation will be printed to this error stream and `qsub` will exit with an exit status of 3. Signals `SIGQUIT` and `SIGKILL` are not trapped and thus will immediately terminate the `qsub` process, leaving the associated job either running or queued. If the job runs to completion, `qsub` will exit with the exit status of the job. (See also section 8.2 "UNIX Job Exit Status" on page 162 for further discussion of the job exit status.)

For job arrays, blocking qsub waits until the entire job array is complete, then returns the exit status of the job array.

8.5 Specifying Job Dependencies

PBS allows you to specify dependencies between two or more jobs. Dependencies are useful for a variety of tasks, such as:

1. Specifying the order in which jobs in a set should execute
2. Requesting a job run only if an error occurs in another job
3. Holding jobs until a particular job starts or completes execution

The "`-W depend=dependency_list`" option to `qsub` defines the dependency between multiple jobs. The *dependency_list* has the format:

type:arg_list[,type:arg_list ...]

where except for the `on` type, the `arg_list` is one or more PBS job IDs in the form:

jobid[:jobid ...]

There are several types:

after:arg_list
: This job may be scheduled for execution at any point after all jobs in arg_list have started execution.

afterok:arg_list
: This job may be scheduled for execution only after all jobs in arg_list have terminated with no errors. See "Warning about exit status with csh" in EXIT STATUS.

afternotok:arg_list
: This job may be scheduled for execution only after all jobs in arg_list have terminated with errors. See "Warning about exit status with csh" in EXIT STATUS.

afterany:arg_list
: This job may be scheduled for execution after all jobs in arg_list have finished execution, with or without errors.

before:arg_list
: Jobs in arg_list may begin execution once this job has begun execution.

beforeok:arg_list
: Jobs in arg_list may begin execution once this job terminates without errors. See "Warning about exit status with csh" in EXIT STATUS.

beforenotok:arg_list
: If this job terminates execution with errors, the jobs in arg_list may begin. See "Warning about exit status with csh" in EXIT STATUS.

beforeany:arg_list
: Jobs in arg_list may begin execution once this job terminates execution, with or without errors.

on:count
: This job may be scheduled for execution after `count` dependencies on other jobs have been satisfied. This type is used in conjunction with one of the `before` types listed. `count` is an integer greater than 0.

Job IDs in the arg_list of `before` types must have been submitted with a type of `on`.

To use the `before` types, the user must have the authority to alter the jobs in `arg_list`. Otherwise, the dependency is rejected and the new job aborted.

Error processing of the existence, state, or condition of the job on which the newly submitted job is a deferred service, i.e. the check is performed after the job is queued. If an error is detected, the new job will be deleted by the server. Mail will be sent to the job submitter stating the error.

Suppose you have three jobs (job1, job2, and job3) and you want job3 to start *after* job1 and job2 have *ended*. The first example below illustrates the options you would use on the `qsub` command line to specify these job dependencies.

```
qsub job1
16394.jupiter
qsub job2
16395.jupiter
qsub -W depend=afterany:16394:16395 job3
16396.jupiter
```

As another example, suppose instead you want job2 to start *only if* job1 ends with no errors (i.e. it exits with a no error status):

```
qsub job1
16397.jupiter
qsub -W depend=afterok:16397 job2
16396.jupiter
```

Similarly, you can use `before` dependencies, as the following example exhibits. Note that unlike `after` dependencies, `before` dependencies require the use of the `on` dependency.

```
qsub -W depend=on:2 job1
16397.jupiter
qsub -W depend=beforeany:16397 job2
16398.jupiter
qsub -W depend=beforeany:16397 job3
16399.jupiter
```

You can use `xpbs` to specify job dependencies as well. On the *Submit Job* window, in the other options section (far left, center of window) click on one of the three dependency buttons: “after depend”, “before depend”, or “concurrency”. These will launch a “Dependency” window in which you will be able to set up the dependencies you wish.

8.5.1 Job Array Dependencies

Job dependencies are supported:

- Between jobs and jobs
- Between job arrays and job arrays
- Between job arrays and jobs
- Between jobs and job arrays

Note: Job dependencies are not supported for subjobs or ranges of subjobs.

8.6 Delivery of Output Files

To transfer output files or to transfer staged-in or staged-out files to/from a remote destination, PBS uses either `rcp` or `scp` depending on the configuration options. The version of `rcp` used by PBS always exits with a non-zero exit status for any error. Thus MOM knows if the file was delivered or not. The secure copy program, `scp`, is also based on this version of `rcp` and exits with the proper exit status.

If using `rcp`, the copy of output or staged files can fail for (at least) two reasons.

- The user lacks authorization to access the specified system. (See discussion in "User's PBS Environment" on page 24.)
- Under UNIX, if the user's `.cshrc` outputs any characters to standard output, e.g. contains an echo command, `pbs_rcp` will fail.

If using *Secure Copy (*`scp`), then PBS will first try to deliver output or stagein/out files using `scp`. If `scp` fails, PBS will try again using `rcp` (assuming that `scp` might not exist on the remote host). If `rcp` also fails, the above cycle will be repeated after a delay, in case the problem is caused by a temporary network problem. All failures are logged in MOM's log, and an email containing the errors is sent to the job owner.

For delivery of output files on the local host, PBS uses the `cp` command (UNIX) or the `xcopy` command (Windows). Local and remote delivery of output may fail for the following additional reasons:

- A directory in the specified destination path does not exist.
- A directory in the specified destination path is not searchable by the user.
- The target directory is not writable by the user.

8.7 Input/Output File Staging

File staging is a way to specify which files should be copied onto the execution host before the job starts, and which should be copied off the execution host when it finishes.

8.7.1 Staging and Execution Directory: User's Home vs. Job-specific

The job's staging and execution directory is the directory to which files are copied before the job runs, and from which output files are copied after the job has finished. This directory is either your home directory or a job-specific directory created by PBS just for this job. If you use job-specific staging and execution directories, you don't need to have a home directory on each execution host, as long as those hosts are configured properly. In addition, each job gets its own staging and execution directory, so you can more easily avoid filename collisions.

This table lists the differences between using your home directory for staging and execution and using a job-specific staging and execution directory created by PBS.

Table 8-1: Differences Between User's Home and Job-specific Directory for Staging and Execution

Question Regarding Action, Requirement, or Setting	User's Home Directory	Job-specific Directory
Does PBS create a job-specific staging and execution directory?	No	Yes
User's home directory must exist on execution host(s)?	Yes	No
Standard out and standard error automatically deleted when qsub -k option is used?	No	Yes
When are staged-out files are deleted?	Successfully staged-out files are deleted; others go to "undelivered"	Only after all are successfully staged out
Staging and execution directory deleted after job finishes?	No	Yes
How is job's sandbox attribute set?	`HOME` or not set	PRIVATE

8.7.2 Using Job-specific Staging and Execution Directories

8.7.2.1 Setting the Job's Staging and Execution Directory

The job's `sandbox` attribute controls whether PBS creates a unique job-specific staging and execution directory for this job. If the job's `sandbox` attribute is set to `PRIVATE`, PBS creates a unique staging and execution directory for the job. If `sandbox` is unset, or is set to HOME, PBS uses the user's home directory as the job's staging and execution directory. By default, the `sandbox` attribute is not set.

The user can set the `sandbox` attribute via `qsub`, or through a PBS directive. For example:

```
qsub -Wsandbox=PRIVATE
```

The job's `sandbox` attribute cannot be altered while the job is executing.

Table 8-2: Effect of Job's `sandbox` Attribute on Location of Staging and Execution Directory

Job's `sandbox` attribute	Effect
not set	Job's staging and execution directory is the user's home directory
HOME	Job's staging and execution directory is the user's home directory
PRIVATE	Job's staging and execution directory is a job-specific directory created by PBS. If the qsub -k option is used, output and error files are retained on the primary execution host in the staging and execution directory. This directory is removed, along with all of its contents, when the job finishes.

8.7.2.2 The Job's `jobdir` Attribute and the PBS_JOBDIR Environment Variable

The job's `jobdir` attribute is a read-only attribute, set to the pathname of the job's staging and execution directory on the primary host. The user can view this attribute by using `qstat -f`, only while the job is executing. The value of `jobdir` is not retained if a job is rerun; it is undefined whether `jobdir` is visible or not when the job is not executing.

The environment variable `PBS_JOBDIR` is set to the pathname of the staging and execution directory on the primary execution host. `PBS_JOBDIR` is added to the job script process, any job tasks, and the prologue and epilogue.

8.7.3 Attributes and Environment Variables Affecting Staging

The following attributes and environment variables affect staging and execution.

Table 8-3: Attributes and Environment Variables Affecting Staging

Job's Attribute or Environment Variable	Effect
`sandbox` attribute	Determines whether PBS uses user's home directory or creates job-specific directory for staging and execution. User-settable per job via `qsub -W` or through a PBS directive.
`stagein` attribute	Sets list of files or directories to be staged in. User-settable per job via `qsub -W` or through a PBS directive.
`stageout` attribute	Sets list of files or directories to be staged out. User-settable per job via `qsub -W` or through a PBS directive.

Table 8-3: Attributes and Environment Variables Affecting Staging

Job's Attribute or Environment Variable	Effect
`Keep_Files` attribute	Determines whether output and/or error files remain on execution host. User-settable per job via `qsub -k` or through a PBS directive. If the `Keep_Files` attribute is set to `o` and/or `e` (output and/or error files remain in the staging and execution directory) and the job's `sandbox` attribute is set to `PRIVATE`, standard out and/or error files are removed, when the staging directory is removed at job end along with its contents.
`jobdir` attribute	Set to pathname of staging and execution directory on primary execution host. Read-only; viewable via `qstat -f`.
`PBS_JOBDIR` environment variable	Set to pathname of staging and execution directory on primary execution host. Added to environments of job script process, job tasks, and prologue and epilogue.
`TMPDIR` environment variable	Location of job-specific scratch directory.

8.7.4 Specifying Files To Be Staged In or Staged Out

You can specify files to be staged in before the job runs and staged out after the job runs by using `-W stagein=file_list` and `-W stageout=file_list`. You can use these as options to qsub, or as directives in the job script.

The *file_list* takes the form:

local_path@hostname:remote_path[,...]

for both stagein and stageout.

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

The '@' character separates the local specification from the remote specification.

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

IMPORTANT:

It is advisable to use an absolute pathname for the remote_path. Remember that the path to your home directory may be different on each machine, and that when using `sandbox = PRIVATE,` you may or may not have a home directory on all execution machines.

For stagein, the direction of travel is **from** remote_path **to** local_path.

For stageout, the direction of travel is **from** local_path **to** remote_path.

The following example shows how to use a directive to stagein a file named `grid.dat` located in the directory `/u/user1` on the host called `serverA`. The staged-in file is copied to the staging and execution directory and given the name `dat1`. Since `local_path` is evaluated relative to the staging and execution directory, it is not necessary to specify a full pathname for `dat1`. Always use a relative pathname for `local_path` when the job's staging and execution directory is created by PBS.

```
#PBS -W stagein=dat1@serverA:/u/user1/grid.dat ...
```

To use the `qsub` option to stage in the file residing on myhost, in `/Users/myhome/mydata/data1`, calling it `input_data1` in the staging and execution directory:

```
qsub -W stagein=input_data1@myhost:/Users/myhome/
   mydata/data1
```

To stage more than one file or directory, use a comma-separated list of paths, and enclose the list in double quotes. For example, to stage two files data1 and data2 in:

```
qsub -W stagein="input1@hostA:/myhome/data1,
   \input2@hostA:/myhome/data1"
```

- Under Windows, special characters such as spaces, backslashes (\), colons (:), and drive letter specifications are valid pathnames. For example, the following will stagein the grid.dat file on drive D at hostB to a local file ("dat1") on drive C.:

```
qsub -W stagein="dat1@hostB:D\Documents and Set-
   tings\grid.dat"
```

8.7.4.1 Copying Directories Into and Out Of the Staging and Execution Directory

You can stage directories into and out of the staging and execution directory the same way you stage files. The *remote_path* and *local_path* for both stagein and stageout can be a directory. If you stagein or stageout a directory, PBS copies that directory along with all of its files and subdirectories. At the end of the job, the directory, including all files and subdirectories, is deleted. This can create a problem if multiple jobs are using the same directory.

8.7.4.2 Wildcards In File Staging

You can use wildcards when staging files and directories, according to the following rules.

- The asterisk "*" matches one or more characters.
- The question mark "?" matches a single character.
- All other characters match only themselves.
- Wildcards inside of quote marks are expanded.
- Wildcards cannot be used to match UNIX files that begin with period "." or Windows files that have the "SYSTEM" or "HIDDEN" attributes.
- When using the qsub command line on UNIX, you must prevent the shell from expanding wildcards. For some shells, you can enclose

the pathnames in double quotes. For some shells, you can use a backspace before the wildcard.

- Wildcards can only be used in the source side of a staging specification. This means they can be used in the remote_path specification for stagein, and in the local_path specification for stageout.
- When staging using wildcards, the destination must be a directory. If the destination is not a directory, the result is undefined. So for example, when staging out all .out files, you must specify a directory for remote_path.
- Wildcards can only be used in the final path component, i.e. the basename.
- When wildcards are used during stagein, PBS will not automatically delete staged files at job end. Note that if PBS created the staging and execution directory, that directory and all its contents are deleted at job end.

Examples:

1. Stage out all files from the execution directory to a specific directory:
   - UNIX

```
-W stageout=*@myworkstation:/user/project1/
case1
```

   - Windows

```
-W stageout=*@mypc:E:\project1\case1
```

2. Stage out specific types of result files and disregard the scratch and other temporary files after the job terminates. The result files that are interesting for this example end in '.dat':
   - UNIX

```
-W stageout=*.dat@myworkstation:project3/data
```

   - Windows

```
-W stageout=*.dat@mypc:C:\project\data
```

3. Stage in all files from an application data directory to a subdirectory:
   - UNIX

-W stagein=jobarea@myworkstation:crashtest1/*

- Windows

  -W stagein=jobarea@mypc:E:\crashtest1*

4. Stage in data from files and directories matching "wing*":

   - UNIX

     -W stagein=.@myworkstation:848/wing*

   - Windows

     -W stagein=.@mypc:E:\flowcalc\wing*

5. Stage in .bat and .dat files to jobarea:

   - UNIX:

     -W stagein=jobarea@myworkstation:/users/me/crash1.?at

   - Windows:

     -W stagein=jobarea@myworkstation:C:\me\crash1.?at

8.7.4.3 Caveats

When using a job-specific staging and execution directory, do not use an absolute path in `local_path`.

8.7.4.4 Output Filenames

The name of the job defaults to the script name, if no name is given via `qsub -N`, via a PBS directive, or via stdin. For example, if the sequence number is 1234,

```
#PBS -N fixgamma
```

gives stdout the name fixgamma.o1234 and stderr the name fixgamma.e1234.

For information on submitting jobs, see section 4.4 "Submitting a PBS Job" on page 44.

8.7.5 Example of Using Job-specific Staging and Execution Directories

In this example, you want the file "jay.fem" to be delivered to the job-specific staging and execution directory given in PBS_JOBDIR, by being copied from the host "submithost". The job script is executed in PBS_JOBDIR and "jay.out" is staged out from PBS_JOBDIR to your home directory on the submittal host (i.e., "hostname"):

```
qsub -Wsandbox=PRIVATE -Wstagein=jay.fem@submit-
   host:jay.fem -Wstageout=jay.out@submithost:jay.out
```

8.7.6 Summary of the Job's Lifecycle

This is a summary of the steps performed by PBS. The steps are not necessarily performed in this order.

- On each execution host, if specified, PBS creates a job-specific staging and execution directory.
- PBS sets PBS_JOBDIR and the job's jobdir attribute to the path of the job's staging and execution directory.
- On each execution host allocated to the job, PBS creates a job-specific temporary directory.
- PBS sets the TMPDIR environment variable to the pathname of the temporary directory.
- If any errors occur during directory creation or the setting of variables, the job is requeued.
- PBS stages in any files or directories.
- The prologue is run on the primary execution host, with its current working directory set to PBS_HOME/mom_priv, and with PBS_JOBDIR and TMPDIR set in its environment.
- The job is run as the user on the primary execution host.
- The job's associated tasks are run as the user on the execution host(s).
- The epilogue is run on the primary execution host, with its current working directory set to the path of the job's staging and execution

directory, and with PBS_JOBDIR and TMPDIR set in its environment.

- PBS stages out any files or directories.
- PBS removes any staged files or directories.
- PBS removes any job-specific staging and execution directories and their contents, and all TMPDIRs and their contents.
- PBS writes the final job accounting record and purges any job information from the Server's database.

8.7.7 Detailed Description of Job's Lifecycle

8.7.7.1 Creation of TMPDIR

For each host allocated to the job, PBS creates a job-specific temporary scratch directory for the job. If the temporary scratch directory cannot be created, the job is aborted.

8.7.7.2 Choice of Staging and Execution Directories

If the job's `sandbox` attribute is set to `PRIVATE`, PBS creates job-specific staging and execution directories for the job. If the job's `sandbox` attribute is set to `HOME`, or is unset, PBS uses the user's home directory for staging and execution.

8.7.7.2.1 Job-specific Staging and Execution Directories

If the staging and execution directory cannot be created the job is aborted. If PBS fails to create a staging and execution directory, see the system administrator.

You should not depend on any particular naming scheme for the new directories that PBS creates for staging and execution.

8.7.7.2.2 User's Home Directory as Staging and Execution Directory

The user must have a home directory on each execution host. The absence of the user's home directory is an error and causes the job to be aborted.

8.7.7.3 Setting Environment Variables and Attributes

PBS sets PBS_JOBDIR and the job's jobdir attribute to the pathname of the staging and execution directory. The TMPDIR environment variable is set to the pathname of the job-specific temporary scratch directory.

8.7.7.4 Staging Files Into Staging and Execution Directories

PBS evaluates local_path and remote_path relative to the staging and execution directory given in PBS_JOBDIR, whether this directory is the user's home directory or a job-specific directory created by PBS. PBS copies the specified files and/or directories to the job's staging and execution directory.

8.7.7.5 Running the Prologue

The MOM's prologue is run on the primary host as root, with the current working directory set to PBS_HOME/mom_priv, and with PBS_JOBDIR and TMPDIR set in its environment.

8.7.7.6 Job Execution

PBS runs the job script on the primary host as the user. PBS also runs any tasks created by the job as the user. The job script and tasks are executed with their current working directory set to the job's staging and execution directory, and with PBS_JOBDIR and TMPDIR set in their environment.

8.7.7.7 Standard Out and Standard Error

The job's stdout and stderr files are created directly in the job's staging and execution directory on the primary execution host.

8.7.7.7.1 Job-specific Staging and Execution Directories

If the qsub -k option is used, the stdout and stderr files will **not** be automatically copied out of the staging and execution directory at job end - they will be deleted when the directory is automatically removed.

8.7.7.7.2 User's Home Directory as Staging and Execution Directory

If the `-k` option to `qsub` is used, standard out and/or standard error files are retained on the primary execution host instead of being returned to the submission host, and are not deleted after job end.

8.7.7.8 Running the Epilogue

PBS runs the epilogue on the primary host as root. The epilogue is executed with its current working directory set to the job's staging and execution directory, and with `PBS_JOBDIR` and `TMPDIR` set in its environment.

8.7.7.9 Staging Files Out and Removing Execution Directory

When PBS stages files out, it evaluates `local_path` and `remote_path` relative to `PBS_JOBDIR`. Files that cannot be staged out are saved in PBS_HOME/undelivered. See section 10.5.6 "Non-delivery of Output" on page 533 of the PBS Professional Administrator's Guide.

8.7.7.9.1 Job-specific Staging and Execution Directories

If PBS created job-specific staging and execution directories for the job, it cleans up at the end of the job. The staging and execution directory and all of its contents are removed, on all execution hosts.

8.7.7.10 Removing TMPDIRs

PBS removes all TMPDIRs, along with their contents.

8.7.8 Staging with Job Arrays

File staging is supported for job arrays. See "File Staging" on page 217.

8.7.9 Using xpbs for File Staging

Using `xpbs` to set up file staging directives may be easier than using the command line. On the *Submit Job* window, in the miscellany options section (far left, center of window) click on the *file staging* button. This will launch the *File Staging* dialog box (shown below) in which you will be able to set up the file staging you desire.

The *File Selection Box* will be initialized with your current working directory. If you wish to select a different directory, double-click on its name, and `xpbs` will list the contents of the new directory in the *File Selection Box*. When the correct directory is displayed, simply click on the name of the file you wish to stage (in or out). Its name will be written in the *File Selected* area.

Next, click either of the *Add file selected...* buttons to add the named file to the stagein or stageout list. Doing so will write the file name into the corresponding area on the lower half of the *File Staging* window. Now you need to provide location information. For stagein, type in the path and filename where you want the named file placed. For stageout, specify the hostname and pathname where you want the named file delivered. You may repeat this process for as many files as you need to stage.

When you are done selecting files, click the *OK* button.

8.7.10 Stagein and Stageout Failure

When stagein fails, the job is placed in a 30-minute wait to allow the user time to fix the problem. Typically this is a missing file or a network outage. Email is sent to the job owner when the problem is detected. Once the problem has been resolved, the job owner or the Operator may remove the wait by resetting the time after which the job is eligible to be run via the `-a` option to `qalter`. The server will update the job's comment with information about why the job was put in the wait state. When the job is eligible to run, it may run on different vnodes.

When stageout encounters an error, there are three retries. PBS waits 1 second and tries

again, then waits 11 seconds and tries a third time, then finally waits another 21 seconds

and tries a fourth time. Email is sent to the job owner if all attempts fail. Files that cannot be staged out are saved in PBS_HOME/undelivered. See section 10.5.6 “Non-delivery of Output” on page 533 of the PBS Professional Administrator’s Guide.

8.8 The pbsdsh Command

The **pbsdsh** command allows you to distribute and execute a task on each of the vnodes assigned to your job. (`pbsdsh` uses the PBS Task Manager API, see `tm`(3), to distribute the program on the allocated vnodes.)

IMPORTANT:

The `pbsdsh` command is not available under Windows.

Usage of the `pbsdsh` command is:

pbsdsh [-c N] [-o] [-s] [-v] -- program [program args]

pbsdsh [-n N] [-o] [-s] [-v] -- program [program args]

Note that the double dash must come after the options and before the program and arguments. The double dash is only required for Linux.

The available options are:

-c N

The program is spawned on the first *N* vnodes allocated. If the value of *N* is greater than the number of vnodes, it will “wrap” around, running multiple copies on the vnodes. This option is mutually exclusive with `-n`.

-n N

The program is spawned on a single vnode which is the *N*-th vnode allocated. This option is mutually exclusive with `-c`.

-o

The program will not wait for the tasks to finish.

-s

If this option is given, the program is run sequentially on each vnode, one after the other.

-v

Verbose output about error messages and task exit status is produced.

When run without the -c or the -n option, pbsdsh will spawn the program on all vnodes allocated to the PBS job. The execution take place concurrently--all copies of the task execute at (about) the same time.

The following example shows the pbsdsh command inside of a PBS batch job. The options indicate that the user wants pbsdsh to run the *myapp* program with one argument (*app-arg1*) on all four vnodes allocated to the job (i.e. the default behavior).

```
#!/bin/sh
#PBS -l select=4:ncpus=1
#PBS -l walltime=1:00:00


pbsdsh ./myapp app-arg1
```

The pbsdsh command runs one task for each line in the PBS_NODEFILE. Each MPI rank will get a single line in the PBS_NODEFILE, so if you are running multiple MPI ranks on the same host, you will still get multiple pbsdsh tasks on that host.

8.9 Advance and Standing Reservation of Resources

An *advance reservation* is a reservation for a set of resources for a specified time. The reservation is only available to a specific user or group of users.

A *standing reservation* is an advance reservation which recurs at specified times. For example, the user can reserve 8 CPUs and 10GB every Wednesday and Thursday from 5pm to 8pm, for the next three months.

An *instance* of a standing reservation is also called an *occurrence* of the standing reservation. The *soonest occurrence* of a standing reservation is the occurrence which is currently active, or if none is active, then it is the next occurrence.

An occurrence of a standing reservation behaves like an advance reservation, with the following exceptions:

- While a job can be submitted to a specific advance reservation, it can only be submitted to the standing reservation as a whole, not to a specific occurrence. You can only specify *when* the job is eligible to run. See the `qsub(1B)` man page.
- When an advance reservation ends, it and all of its jobs, running or queued, are deleted, but when an occurrence ends, only its running jobs are deleted.

Each occurrence of a standing reservation has reserved resources which satisfy the resource request, but each instance may have its resources drawn from a different source. A query for the resources assigned to a standing reservation will return the resources assigned to the soonest occurrence, shown in the `resv_nodes` attribute reported by `pbs_rstat`.

The time for which a reservation is requested is in the time zone at the submission host.

8.9.1 Introduction to Creating and Using Reservations

The user creates both advance and standing reservations using the `pbs_rsub` command. PBS either confirms that the reservation can be made, or rejects the request. Once the reservation is confirmed, PBS creates a queue for the reservation's jobs. Jobs are then submitted to this queue.

When a reservation is confirmed, it means that the reservation will not conflict with currently running jobs, other confirmed reservations, or dedicated time, and that the requested resources are available for the reservation. A reservation request that fails these tests is rejected. All instances of a standing reservation must be acceptable in order for the standing reservation to be confirmed.

The `pbs_rsub` command returns a *reservation ID*, which is the reservation name. For an advance reservation, this reservation ID has the format:

R<unique integer>.<server name>

For a standing reservation, this reservation ID refers to the entire series, and has the format:

S<unique integer>.<server name>

The user specifies the resources for a reservation using the same syntax as for a job. Jobs in reservations are placed the same way non-reservation jobs are placed in placement sets.

The `xpbs` GUI cannot be used for creation, querying, or deletion of reservations.

8.9.2 Creating Advance Reservations

You create an advance reservation using the `pbs_rsub` command. PBS must be able to calculate the start and end times of the reservation, so you must specify two of the following three options:

D	Duration
E	End time
R	Start time

8.9.2.1 Examples of Creating Advance Reservations

The following example shows the creation of an advance reservation asking for 1 vnode, 30 minutes of wall-clock time, and a start time of 11:30. Since an end time is not specified, PBS will calculate the end time based on the reservation start time and duration.

```
pbs_rsub -R 1130 -D 00:30:00
```

PBS returns the reservation ID:

```
R226.south UNCONFIRMED
```

The following example shows an advance reservation for 2 CPUs from 8:00 p.m. to 10:00 p.m.:

```
pbs_rsub -R 2000.00 -E 2200.00 -l select=1:ncpus=2
```

PBS returns the reservation ID:

```
R332.south UNCONFIRMED
```

8.9.3 Creating Standing Reservations

You create standing reservations using the `pbs_rsub` command. You **must** specify a start and end date when creating a standing reservation. The recurring nature of the reservation is specified using the `-r` option to `pbs_rsub`. The `-r` option takes the `recurrence_rule` argument, which specifies the standing reservation's occurrences. The recurrence rule uses iCalendar syntax, and uses a subset of the parameters described in RFC 2445.

The recurrence rule can take two forms:

"FREQ= freq_spec; COUNT= count_spec; interval_spec"

In this form, you specify how often there will be occurrences, how many there will be, and which days and/or hours apply.

"FREQ= freq_spec; UNTIL= until_spec; interval_spec"

In this form, the user specifies how often there will be occurrences, when the occurrences will end, and which days and/or hours apply.

freq_spec
: This is the frequency with which the reservation repeats. Valid values are `WEEKLY | DAILY | HOURLY`

  When using a freq_spec of `WEEKLY`, you may use an interval_spec of `BYDAY` and/or `BYHOUR`. When using a freq_spec of `DAILY`, you may use an interval_spec of `BYHOUR`. When using a freq_spec of `HOURLY`, do not use an interval_spec.

count_spec
: The exact number of occurrences. Number up to 4 digits in length. Format: integer.

interval_spec
: Specifies the interval at which there will be occurrences. Can be one or both of `BYDAY=<days>` or `BYHOUR=<hours>`. Valid values are `BYDAY = MO | TU | WE | TH | FR | SA | SU` and `BYHOUR = 0 | 1 | 2 | ... | 23`. When using both, separate them with a semicolon. Separate days or hours with a comma.

For example, to specify that there will be recurrences on Tuesdays and Wednesdays, at 9 a.m. and 11 a.m., use `BYDAY=TU,WE;BYHOUR=9,11`

`BYDAY` should be used with `FREQ=WEEKLY`. `BYHOUR` should be used with `FREQ=DAILY` or `FREQ=WEEKLY`.

until_spec

Occurrences will start up to but not after this date and time. This means that if occurrences last for an hour, and normally start at 9 a.m., then a time of 9:05 a.m on the day specified in the until_spec means that an occurrence will start on that day.

Format: `YYYYMMDD[THHMMSS]`

Note that the year-month-day section is separated from the hour-minute-second section by a capital `T`.

Default: 3 years from time of reservation creation.

8.9.3.1 Setting Reservation Start Time and Duration

In a standing reservation, the arguments to the `-R` and `-E` options to `pbs_rsub` can provide more information than they do in an advance reservation. In an advance reservation, they provide the start and end time of the reservation. In a standing reservation, they can provide the start and end time, but they can also be used to compute the duration and the offset from the interval start.

The difference between the values of the arguments for `-R` and `-E` is the duration of the reservation. For example, if you specify

```
-R 0930 -E 1145
```

the duration of your reservation will be two hours and fifteen minutes. If you specify

```
-R 150800 -E 170830
```

the duration of your reservation will be two days plus 30 minutes.

The interval_spec can be used to specify the day or the hour at which the interval starts. If you specify

```
-R 0915 -E 0945 ... BYHOUR=9,10
```

the duration is 30 minutes, and the offset is 15 minutes from the start of the interval. The interval start is at 9 and again at 10. Your reservation will run from 9:15 to 9:45, and again at 10:15 and 10:45. Similarly, if you specifiy

```
-R 0800 -E -1000 ... BYDAY=WE,TH
```

the duration is two hours and the offset is 8 hours from the start of the interval. Your reservation will run Wednesday from 8 to 10, and again on Thursday from 8 to 10.

Elements specified in the recurrence rule override those specified in the arguments to the `-R` and `-E` options. Therefore if you specify

```
-R 0730 -E 0830 ... BYHOUR=9
```

the duration is one hour, but the hour element (9:00) in the recurrence rule has overridden the hour element specified in the argument to `-R` (7:00). The offset is still 30 minutes after the interval start. Your reservation will run from 9:30 to 10:30. Similarly, if the 16th is a Monday, and you specify

```
-R 160800 -E 170900 ... BYDAY=TU;BYHOUR=11
```

the duration 25 hours, but both the day and the hour elements have been overridden. Your reservation will run on Tuesday at 11, for 25 hours, ending Wednesday at 12. However, if you specify

```
-R 160810 -E 170910 ... BYDAY=TU;BYHOUR=11
```

the duration is 25 hours, and the offset from the interval start is 10 minutes. Your reservation will run on Tuesday at 11:10, for 25 hours, ending Wednesday at 12:10. The minutes in the offset weren't overridden by anything in the recurrence rule.

The values specified for the arguments to the `-R` and `-E` options can be used to set the start and end times in a standing reservation, just as they are in an advance reservation. To do this, don't override their elements inside the recurrence rule. If you specify

```
-R 0930 -E 1030 ... BYDAY=MO,TU
```

you haven't overridden the hour or minute elements. Your reservation will run Monday and Tuesday, from 9:30 to 10:30.

8.9.3.2 Requirements for Creating Standing Reservations

- The user must specify a start and end date. See the -R and -E options to the pbs_rsub command in section 8.9.4 “The pbs_rsub Command” on page 189.
- The user must set the submission host’s PBS_TZID environment variable. The format for PBS_TZID is a timezone location. Example: America/Los_Angeles, America/Detroit, Europe/Berlin, Asia/Calcutta. See section 8.9.8.1 “Setting the Submission Host’s Time Zone” on page 204.
- The recurrence rule must be one unbroken line. See the -r option to pbs_rsub in section 8.9.4 “The pbs_rsub Command” on page 189.
- The recurrence rule must be enclosed in double quotes.
- Vnodes that have been configured to accept jobs only from a specific queue (vnode-queue restrictions) cannot be used for advance or standing reservations. See your PBS administrator to determine whether some vnodes have been configured to accept jobs only from specific queues.

8.9.3.3 Examples of Creating Standing Reservations

For a reservation that runs every day from 8am to 10am, for a total of 10 occurrences:

```
pbs_rsub -R 0800 -E 1000 - r"FREQ=DAILY;COUNT=10"
```

Every weekday from 6am to 6pm until December 10, 2008:

```
pbs_rsub -R 0600 -E 1800 -r "FREQ=WEEKLY;
   BYDAY=MO,TU,WE,TH,FR; UNTIL=20081210"
```

Every week from 3pm to 5pm on Monday, Wednesday, and Friday, for 9 occurrences, i.e., for three weeks:

```
pbs_rsub -R 1500 -E 1700 -r
   "FREQ=WEEKLY;BYDAY=MO,WE,FR; COUNT=3"
```

8.9.4 The pbs_rsub Command

The pbs_rsub command returns a reservation ID string, and the current status of the reservation.

These are the options to the pbs_rsub command:

-D duration
: Specifies reservation duration. If the start time and end time are the only times specified, this duration time is calculated.

  Format: Either a total number of seconds of walltime, or a colon-delimited timestring, e.g. HH:MM:SS or MM:SS.

  Default: none.

-E end_time
: Specifies the reservation end time. If start time and duration are the only times specified, the end time value is calculated.

  Format: Datetime. See “Formats:” on page 196 for a description of the datetime string.

  Default: none.

-g group_list
: The *group_list* is a comma-separated list of group names. The server uses entries on this list, along with an ordered set of rules, to associate a group name with the reservation.

  Format: group@hostname[,group@hostname ...]

-G auth_group_list
: Comma-separated list of names of groups who can or cannot submit jobs to this reservation. Group names are interpreted in the context of the server's host, not the context of the host from which the job is submitted. This list becomes the acl_groups list for the reservation's queue. Refer to the Authorized_Groups reservation attribute on the pbs_resv_attributes(7B) man page.

  Format: [+|-]group_name[,[+|-]group_name ...]

  Default: All groups are authorized to submit jobs.

-H auth_host_list

Comma-separated list of hosts from which jobs can and cannot be submitted to this reservation. This list becomes the `acl_hosts` list for the reservation's queue. See the `Authorized_Hosts` reservation attribute on the `pbs_resv_attributes(7B)` man page.

Format: `[+|-]hostname[,[+|-]hostname ...]`

Default: All hosts are authorized to submit jobs.

-l block_time

Specifies interactive mode. The `pbs_rsub` command will block, up to `block_time` seconds, while waiting for the reservation to be confirmed or denied.

If `block_time` is positive, and the reservation isn't confirmed or denied in the specified time, the ID string for the reservation is returned with the status "`UNCONFIRMED`".

If `block_time` is negative, and the reservation is neither confirmed nor denied in the specified time, the reservation is automatically deleted.

Format: integer.

Default: not interactive.

-l placement

The placement specifies how a job will be placed on vnodes. The place statement has this form:

-l place=[arrangement][: sharing][: grouping]

where

arrangement is one of *free* | *pack* | *scatter*

sharing is one of *excl* | *share*

grouping can have only one instance of *group=resource*

and where

free: Place job on any vnode(s).

pack: All chunks will be taken from one host.

scatter: Only one chunk with any MPI processes will be taken from a host. A chunk with no MPI processes may be taken from the same node as another chunk.

excl: Only this job uses the vnodes chosen.

share: This job can share the vnodes chosen.

group=resource: Chunks will be grouped according to a resource. All nodes in the group must have a common value for the resource, which can be either the built-in resource host or a site-defined node-level resource.

Note that nodes can have sharing attributes that override job placement requests. See the pbs_node_attributes(7B) man page.

-l resource_request

The `resource_request` specifies the resources required for the reservation. These resources will be used for the limits on the queue that is dynamically created for the reservation. The aggregate amount of resources for currently running jobs from this queue will not exceed these resource limits. Jobs in the queue that request more of a resource than the queue limit for that resource are not allowed to run. Also, the queue inherits the value of any resource limit set on the server, and these are used for the job if the reservation request itself is silent about that resource. A non-privileged user cannot submit a reservation requesting a custom resource which has been created to be invisible or read-only for users.

Resources are requested by using the `-l` option, either in chunks inside of selection statements, or in job-wide requests using `resource_name=value` pairs. The selection statement is of the form:

-l select=[N:]chunk[+[N:]chunk ...]

where `N` specifies how many of that chunk, and a chunk is of the form:

resource_name=value [:resource_name=value ...]

Job-wide resource_name=value requests are of the form:

-l resource_name=value [,resource_name=value ...]

-m mail_events

Specifies whether mail is sent to user_list and when. The argument `mail_events` is a string, either "`n`", for no

mail, or composed of any combination of "a", "b", "e", or "c". Must be enclosed in double quotes.

Table 8-4:

"n"	do not send mail
"a"	notify if the reservation is terminated for any reason
"b"	notify when the reservation period begins
"e"	notify when the reservation period ends
"c"	notify when the reservation is confirmed

Format: String.

Default: "ac".

-M mail_list

The list of users to whom mail is sent whenever the reservation transitions to one of the states specified in the -m *mail_events* option.

Format:

user[@hostname][,user[@hostname]...]

Default: reservation's owner.

-N reservation_name

This specifies a name for the reservation.

Format: String up to 15 characters in length. It must consist of printable, non-white space characters with the first character alphabetic.

Default: None.

-q destination

Specifies the destination server at which to create the reservation. The default server is used if this option is not specified.

-r recurrence_rule

Specifies rule for recurrence of standing reservations. This rule must conform to iCalendar syntax, and is specified using a subset of parameters from RFC 2445. Valid syntax for `recurrence_rule` takes one of two forms:

"FREQ= freq_spec; COUNT=count_spec; interval_spec"

or

"FREQ= freq_spec; UNTIL=until_spec; interval_spec"

where

Table 8-5: Recurrence Rule Parameters

Specification	Description	Format & Valid Values
freq_spec	Frequency with which the standing reservation repeats.	Valid values are: `WEEKLY` | `DAILY` |HOURLY
count_spec	The exact number of occurrences. Number up to 4 digits in length.	Format: integer.

Table 8-5: Recurrence Rule Parameters

Specification	Description	Format & Valid Values
interval_spec	Specifies interval. Format uses one or both of `BYDAY=` and `BYHOUR=`. When using both, separate them with a semicolon.	BYDAY = MO|TU|WE|TH|FR|`SA`|`SU`
		BYHOUR = 0|1|2|...|23
until_spec	Occurrences will start up to but not after date and time specified.	Format: `YYYYMMDD[THHMMSS]` Note that the year-month-day section is separated from the hour-minute-second section by a capital `T`.

Requirements:

- The recurrence rule must be on one unbroken line and must be enclosed in double quotes.
- A start and end date must be used when specifying a recurrence rule. See the `R` and `E` options.
- The `PBS_TZID` environment variable must be set at the submission host.

-R start_time

Reservation start time. If the reservation's end time and duration are the only times specified, this start time is calculated.

If the day, `DD`, is not specified, it will default to today if the time `hhmm` is in the future. Otherwise, the day will be set to tomorrow. For example, if you submit a reservation having

a specification `-R 1110` at 11:15am, it will be interpreted as being for 11:10am tomorrow. If the month portion, `MM`, is not specified, it defaults to the current month provided that the specified day `DD`, is in the future. Otherwise, the month will be set to next month. Similar comments apply to the two other optional, left hand components.

Format: Datetime.

-u user_list

Comma-separated list of user names. Not used. Refer to the `User_List` reservation attribute on the `pbs_resv_attributes(7B)` man page.

Format: *user[@host][,user[@host] ...]*

Default: None.

-U auth_user_list

Comma-separated list of users who are and are not allowed to submit jobs to this reservation. This list becomes the `acl_users` attribute for the reservation's queue. Refer to the `Authorized_Users` reservation attribute on the `pbs_resv_attributes(7B)` man page.

Format: *[+|-]user@host[,[+|-]user@host...]*

Default: Job owner only.

-W attribute_value_list

This allows you to define other attributes for the reservation.

The following attribute is supported:

qmove=jobid

Converts the normal job with job ID *jobid* into a reservation job that will run as soon as possible. Creates the reservation and reservation queue and places the job in the queue. Uses the resources requested by the job to create the reservation.

In creating the reservation, resources requested through the `pbs_rsub` command override existing job resources. Therefore, if the existing job resources are greater than those requested for the reservation, the job will be rejected by the reservation.

The -R and -E options to pbs_rsub are disabled when using the qmove=jobid attribute.

See "Converting a Job into a Reservation Job" on page 159.

--version

The pbs_rsub command returns its PBS version information and exits. This option can only be used alone.

Formats:

Datetime

[[[[CC]YY]MM]DD]hhmm[.SS]

8.9.4.1 Getting Confirmation of a Reservation

By default the pbs_rsub command does not immediately notify you whether the reservation is confirmed or denied. Instead you receive email with this information. You can specify that the pbs_rsub command should wait for confirmation by using the -I *<block_time>* option. The pbs_rsub command will wait up to *<block_time>* seconds for the reservation to be confirmed or denied and then notify you of the outcome. If *block_time* is negative and the reservation is not confirmed in that time, the reservation is automatically deleted.

To find out whether the reservation has been confirmed, use the pbs_rstat command. It will display the state of the reservation. CO and RESV_CONFIRMED indicate that it is confirmed. If the reservation does not appear in the output from pbs_rstat, that means that the reservation was denied.

To ensure that you receive mail about your reservations, set the reservation's Mail_Users attribute via the -M *<email address>* option to pbs_rsub. By default, you will get email when the reservation is terminated or confirmed. If you want to receive email about events other than those, set the reservation's Mail_Points attribute via the -m *<mail events>* option. For more information, see the pbs_rsub(1B) and pbs_resv_attributes(7B) man pages.

8.9.5 Viewing the Status of a Reservation

The following table shows the list of possible states for a reservation. The states that you will usually see are `CO, UN, BD`, and `RN`, although a reservation usually remains unconfirmed for too short a time to see that state.

Table 8-6: Reservation States

Code	State	Description
NO	RESV_NONE	No reservation yet
UN	RESV_UNCONFIRMED	Reservation not confirmed
CO	RESV_CONFIRMED	Reservation confirmed
WT	RESV_WAIT	Unused
TR	RESV_TIME_TO_RUN	Transitory state; reservation's start time has arrived
RN	RESV_RUNNING	Time period from reservation's start time to end time is being traversed
FN	RESV_FINISHED	Transitory state; reservation's end time has arrived and reservation will be deleted
BD	RESV_BEING_DELETED	Transitory state; reservation is being deleted
DE	RESV_DELETED	Transitory state; reservation has been deleted
DJ	RESV_DELETING_JOBS	Jobs remaining after reservation's end time being deleted

To view the status of a reservation, use the `pbs_rstat` command. It will display the status of all reservations at the PBS server. For a standing reservation, the `pbs_rstat` command will display the status of the soonest occurrence. Duration is shown in seconds. The `pbs_rstat` command

will not display a custom resource which has been created to be invisible. See section 4.5.14 “Resource Permissions” on page 58. This command has three options:

Table 8-7: Options to pbs_rstat Command

Option	Meaning	Description
B	Brief	Lists only the names of the reservations
S	Short	Lists in table format the name, queue name, owner, state, and start, duration and end times of each reservation
F	Full	Lists the name and all non-default-value attributes for each reservation.
<none>	Default	Default is S option

The full listing for a standing reservation is identical to the listing for an advance reservation, with the following additions:

- A line that specifies the recurrence rule:

  ```
  reserve_rrule = FREQ=WEEKLY;BYDAY=MO;COUNT=5
  ```

- An entry for the vnodes reserved for the soonest occurrence of the standing reservation. This entry also appears for an advance reservation, but will be different for each occurrence:

  ```
  resv_nodes=(vnode_name:...)
  ```

- A line that specifies the total number of occurrences of the standing reservation:

  ```
  reserve_count = 5
  ```

- The index of the soonest occurrence:

  ```
  reserve_index = 1
  ```

- The timezone at the site of submission of the reservation is appended to the reservation Variable list. For example, in California:

  ```
  Variable_List=<other variables>PBS_TZID=America/
     Los_Angeles
  ```

To get the status of a reservation at a server other than the default server, set the PBS_SERVER environment variable to the name of the server you wish to query, then use the pbs_rstat command. Your PBS commands will treat the new server as the default server, so you may wish to unset this environment variable when you are finished.

You can also get information about the reservation's queue by using the qstat command. See section 6.1 "The qstat Command" on page 125 and the qstat(1B) man page.

8.9.5.1 Examples of Viewing Reservation Status Using pbs_rstat

In our example, we have one advance reservation and one standing reservation. The advance reservation is for today, for two hours, starting at noon. The standing reservation is for every Thursday, for one hour, starting at 3:00 p.m. Today is Monday, April 28th, and the time is 1:00, so the advance reservation is running, and the soonest occurrence of the standing reservation is Thursday, May 1, at 3:00 p.m.

Example brief output:

```
pbs_rstat -B
Name: R302.south
Name: S304.south
```

Example short output:

```
pbs_rstat -S

Name      Queue User State Start / Duration / End
--------------------------------------------------------
R302.south R302 user1 RN Today 12:00 / 7200/  Today 14:00
S304.south S304 user1 CO May 1 2008 15:00/3600/May 1 2008 16:00
```

Example full output:

```
pbs_rstat -F
```

```
Name: R302.south
Reserve_Name = NULL
Reserve_Owner = user1@south.mydomain.com
reserve_state = RESV_RUNNING
reserve_substate = 2
reserve_start = Mon Apr 28 12:00:00 2008
reserve_end = Mon Apr 28 14:00:00 2008
reserve_duration = 7200
queue = R302
Resource_List.ncpus = 2
Resource_List.nodect = 1
Resource_List.walltime = 02:00:00
Resource_List.select = 1:ncpus=2
Resource_List.place = free
resv_nodes = (south:ncpus=2)
Authorized_Users = user1@south.mydomain.com
server = south
ctime = Mon Apr 28 11:00:00 2008
Mail_Users = user1@mydomain.com
mtime = Mon Apr 28 11:00:00 2008
Variable_List =
   PBS_O_LOGNAME=user1,PBS_O_HOST=south.mydomain.com

Name: S304.south
Reserve_Name = NULL
Reserve_Owner = user1@south.mydomain.com
reserve_state = RESV_CONFIRMED
reserve_substate = 2
reserve_start = Thu May 1 15:00:00 2008
reserve_end = Thu May 1 16:00:00 2008
reserve_duration = 3600
```

```
queue = S304
Resource_List.ncpus = 2
Resource_List.nodect = 1
Resource_List.walltime = 01:00:00
Resource_List.select = 1:ncpus=2
Resource_List.place = free
resv_nodes = (south:ncpus=2)
reserve_rrule = FREQ=WEEKLY;BYDAY=MO;COUNT=5
reserve_count = 5
reserve_index = 2
Authorized_Users = user1@south.mydomain.com
server = south
ctime = Mon Apr 28 11:01:00 2008
Mail_Users = user1@mydomain.com
mtime = Mon Apr 28 11:01:00 2008
Variable_List =
   PBS_O_LOGNAME=user1,PBS_O_HOST=south.mydo-
   main.com,PBS_TZID=America/Los_Angeles
```

8.9.6 Deleting Reservations

You can delete an advance or standing reservation by using the `pbs_rdel` command. For a standing reservation, you can only delete the entire reservation, including all occurrences. When you delete a reservation, all of the jobs that have been submitted to the reservation are also deleted. A reservation can be deleted by its owner or by a PBS Operator or Manager. For example, to delete `S304.south`:

```
pbs_rdel S304.south
```

or

```
pbs_rdel S304
```

8.9.7 Submitting a Job to a Reservation

Jobs can be submitted to the queue associated with a reservation, or they can be moved from another queue into the reservation queue. You submit a job to a reservation by using the `-q` <*queue*> option to the `qsub` command to specify the reservation queue. For example, to submit a job to the soonest occurrence of a standing reservation named `S123.south`, submit to its queue `S123`:

```
qsub -q S123 <script>
```

You move a job into a reservation queue by using the `qmove` command. For more information, see the `qsub(1B)` and `qmove(1B)` man pages. For example, to `qmove` job `22.myhost` from `workq` to `S123`, the queue for the reservation named `S123.south`:

```
qmove S123 22.myhost
```

or

```
qmove S123 22
```

A job submitted to a standing reservation without a restriction on when it can run will be run, if possible, during the soonest occurrence. In order to submit a job to a specific occurrence, use the `-a` <*start time*> option to the `qsub` command, setting the start time to the time of the occurrence that you want. You can also use a `cron` job to submit a job at a specific time. See the `qsub(1B)` and `cron(8)` man pages.

8.9.7.1 Running Jobs in a Reservation

A confirmed reservation will accept jobs into its queue at any time. Jobs are only scheduled to run from the reservation once the reservation period arrives.

The jobs in a reservation are not allowed to use, in aggregate, more resources than the reservation requested. A reservation job is started only if its requested walltime will fit within the reservation period. So for example if the reservation runs from 10:00 to 11:00, and the job's walltime is 4 hours, the job will not be started.

When an advance reservation ends, any running or queued jobs in that reservation are deleted.

When an occurrence of a standing reservation ends, any running jobs in that reservation are killed. Any jobs still queued for that reservation are kept in the queued state. They are allowed to run in future occurrences. When the last occurrence of a standing reservation ends, all jobs remaining in the reservation are deleted, whether queued or running.

A job in a reservation cannot be preempted.

8.9.7.2 Access to Reservations

By default, the reservation accepts jobs only from the user who created the reservation, and accepts jobs submitted from any group or host. You can specify a list of users and groups whose jobs will and will not be accepted by the reservation by setting the reservation's `Authorized_Users` and `Authorized_Groups` attributes using the `-U` *auth_user_list* and `-G` *auth_group_list* options to `pbs_rsub`. You can specify the hosts from which jobs can and cannot be submitted by setting the reservation's `Authorized_Hosts` attribute using the `-H` *auth_host_list* option to `pbs_rsub`.

The administrator can also specify which users and groups can and cannot submit jobs to a reservation, and the list of hosts from which jobs can and cannot be submitted.

For more information, see the `pbs_rsub(1B)` and `pbs_resv_attributes(7B)` man pages.

8.9.7.3 Viewing Status of a Job Submitted to a Reservation

You can view the status of a job that has been submitted to a reservation or to an occurrence of a standing reservation by using the `qstat` command. See section 6.1 "The qstat Command" on page 125 and the `qstat(1B)` man page.

For example, if a job named `MyJob` has been submitted to the soonest occurrence of the standing reservation named `S304.south,` it is listed under `S304`, the name of the queue:

```
qstat


Job id      Name       User          Time Use  S  Queue
```

```
----------  ---------  ------------  -------- --  -----
139.south   MyJob      user1                0  Q  S304
```

8.9.8 Reservation Caveats and Errors

8.9.8.1 Setting the Submission Host's Time Zone

The environment variable PBS_TZID must be set at the submission host. The time for which a reservation is requested is the time defined at the submission host. The format for PBS_TZID is a timezone location, rather than a timezone POSIX abbreviation. Examples of values for PBS_TZID are:

```
America/Los_Angeles
America/Detroit
Europe/Berlin
Asia/Calcutta
```

8.9.8.2 Reservation Errors

The following table describes the error messages that apply to reservations:

Table 8-8: Reservation Errors

Description of Error	Server Log Error Code	Error Message
Invalid syntax when specifying a standing reservation	15133	"pbs_rsub error: Undefined iCalendar syntax"
Recurrence rule has both a COUNT and an UNTIL parameter	15134	"pbs_rsub error: Undefined iCalendar syntax. COUNT or UNTIL is required"

Table 8-8: Reservation Errors

Description of Error	Server Log Error Code	Error Message
Recurrence rule missing valid `COUNT` or `UNTIL` parameter	15134	"pbs_rsub error: Undefined iCalendar syntax. A valid COUNT or UNTIL is required"
Problem with the start and/or end time of the reservation, such as: Given start time is earlier than current date and time Missing start time or end time End time is earlier than start time	15086	"pbs_rsub: Bad time specification(s)"
Reservation duration exceeds 24 hours and the recurrence frequency, `FREQ`, is set to `DAILY`	15129	"pbs_rsub error: DAILY recurrence duration cannot exceed 24 hours"
Reservation duration exceeds 7 days and the frequency `FREQ` is set to `WEEKLY`	15128	"pbs_rsub error: WEEKLY recurrence duration cannot exceed 1 week"
Reservation duration exceeds 1 hour and the frequency `FREQ` is set to `HOURLY` or the BY-rule is set to `BYHOUR` and occurs every hour, such as `BYHOUR=9,10`	15130	"pbs_rsub error: HOURLY recurrence duration cannot exceed 1 hour"
The PBS_TZID environment variable is not set correctly at the submission host; rejection at submission host	None	"pbs_rsub error: a valid `PBS_TZID` timezone environment variable is required"

Table 8-8: Reservation Errors

Description of Error	Server Log Error Code	Error Message
The PBS_TZID environment variable is not set correctly at the submission host; rejection at Server	15135	"Unrecognized `PBS_TZID` environment variable"

8.9.8.3 Time Required Between Reservations

Leave enough time between reservations for the reservations and jobs in them to clean up. A job consumes resources even while it is in the `E` or exiting state. This can take longer when large files are being staged. If the job is still running when the reservation ends, it may take up to two minutes to be cleaned up. The reservation itself cannot finish cleaning up until its jobs are cleaned up. This will delay the start time of jobs in the next reservation unless there is enough time between the reservations for cleanup.

8.9.9 Reservation Information in the Accounting Log

The PBS Server writes an accounting record for each reservation in the job accounting file. The accounting record for a reservation is similar to that for a job. The accounting record for any job belonging to a reservation will include the reservation ID. See section 6.15.2 "Accounting Log Format" on page 367 in the PBS Professional Administrator's Guide.

8.10 Dedicated Time

Dedicated time is one or more specific time periods defined by the administrator. These are not repeating time periods. Each one is individually defined.

During dedicated time, the only jobs PBS starts are those in special dedicated time queues. PBS schedules non-dedicated jobs so that they will not run over into dedicated time. Jobs in dedicated time queues are also scheduled so that they will not run over into non-dedicated time. PBS will attempt to backfill around the dedicated-non-dedicated time borders.

PBS uses walltime to schedule within and around dedicated time. If a job is submitted without a walltime to a non-dedicated-time queue, it will not be started until all dedicated time periods are over. If a job is submitted to a dedicated-time queue without a walltime, it will never run.

To submit a job to be run during dedicated time, use the -q <queue name> option to qsub and give the name of the dedicated-time queue you wish to use as the queue name. Queues are created by the administrator; see your administrator for queue name(s).

8.11 Using Comprehensive System Accounting

PBS supports Comprehensive System Accounting (CSA) on SGI Altix machines that are running SGI's ProPack 4.0 or greater and have the Linux job container facility available. CSA provides accounting information about user jobs, called user job accounting.

CSA works the same with and without PBS. To run user job accounting, either the user must specify the file to which raw accounting information will be written, or an environment variable must be set. The environment variable is "ACCT_TMPDIR". This is the directory where a temporary file of raw accounting data is written.

To run user job accounting, the user issues the CSA command "`ja <filename>`" or, if the environment variable "ACCT_TMPDIR" is set, "`ja`". In order to have an accounting report produced, the user issues the command "`ja -<options>`" where the options specify that a report will be written and what kind. To end user job accounting, the user issues the command "`ja -t`"; the -t option can be included in the previous set of options. See the manpage on `ja` for details.

The starting and ending ja commands must be used before and after any other commands the user wishes to monitor. Here are examples of command line and a script:

On the command line:

```
qsub -N myjobname -l ncpus=1
    ja myrawfile
    sleep 50
    ja -c > myreport
    ja -t myrawfile
ctrl-D
```

Accounting data for the user's job (sleep 50) is written to myreport.

If the user creates a file foo with these commands:

```
#PBS -N myjobname
#PBS -l ncpus=1
ja myrawfile
sleep 50
ja -c > myreport
ja -t myrawfile
```

The user could run this script via qsub:

```
qsub foo
```

This does the same thing, via the script "foo".

8.12 Running PBS in a UNIX DCE Environment

PBS Professional includes optional support for UNIX-based DCE. (By optional, we mean that the customer may acquire a copy of PBS Professional with the standard security and authentication module replaced with the DCE module.)

There are two -W options available with qsub which will enable a dcelogin context to be set up for the job when it eventually executes. The user may specify either an encrypted password or a forwardable/renewable Kerberos V5 TGT.

Specify the "-W cred=dce" option to qsub if a forwardable, renewable, Kerberos V5, TGT (ticket granting ticket) with the user as the listed principal is what is to be sent with the job. If the user has an established credentials cache and a non-expired, forwardable, renewable, TGT is in the cache, that information is used.

The other choice, "-W cred=dce:pass", causes the qsub command to interact with the user to generate a DES encryption of the user's password. This encrypted password is sent to the PBS Server and MOM processes, where it is placed in a job-specific file for later use by pbs_mom in acquiring a DCE login context for the job. The information is destroyed if the job terminates, is deleted, or aborts.

IMPORTANT:

The "-W pwd='' " option to qsub has been superseded by the above two options, and therefore should no longer be used. ### really?

Any acquired login contexts and accompanying DCE credential caches established for the job get removed on job termination or deletion.

```
qsub -Wcred=dce <other qsub options> job-script
```

IMPORTANT:

The "-W cred" option to qsub is not available under Windows.

8.13 Running PBS in a UNIX Kerberos Environment

PBS Professional includes optional support for Kerberos-only (i.e. no DCE) environment. (By optional, we mean that the customer may acquire a copy of PBS Professional with the standard security and authentication module replaced with the KRB5 module.) This is not supported under Windows.

To use a forwardable/renewable Kerberos V5 TGT specify the "`-W cred=krb5`" option to `qsub`. This will cause `qsub` to check the user's credential cache for a valid forwardable/renewable TGT which it will send to the Server and then eventually to the execution MOM. While it's at the Server and the MOM, this TGT will be periodically refreshed until either the job finishes or the maximum refresh time on the TGT is exceeded, whichever comes first. If the maximum refresh time on the TGT is exceeded, no KRB5 services will be available to the job, even though it will continue to run.

8.14 Support for Large Page Mode on AIX

A process running as part of a job can use large pages. The memory reported in `resources_used.mem` may be larger with large page sizes.

You can set an environment variable to request large memory pages:

```
LDR_CNTRL="LARGE_PAGE_DATA=M"
LDR_CNTRL="LARGE_PAGE_DATA=Y"
```

For more information see the man page for `setpcred`. This can be viewed with the command "man setpcred" on an AIX machine.

You can run a job that requests large page memory in "mandatory mode":

```
% qsub
export LDR_CNTRL="LARGE_PAGE_DATA=M"
/path/to/exe/bigprog
^D
```

You can run a job that requests large page memory in "advisory mode":

```
% qsub
export LDR_CNTRL="LARGE_PAGE_DATA=Y"
/path/to/exe/bigprog
^D
```

Chapter 9

Job Arrays

This chapter describes job arrays and their use. A job array represents a collection of jobs which only differ by a single index parameter. The purpose of a job array is twofold. It offers the user a mechanism for grouping related work, making it possible to submit, query, modify and display the set as a single unit. Second, it offers a way to possibly improve performance, because the batch system can use certain known aspects of the collection for speedup.

9.1 Definitions

Subjob

Individual entity within a job array (e.g. **1234[7]**, where **1234[]** is the job array itself, and **7** is the index) which has many properties of a job as well as additional semantics (defined below.)

Sequence_number

The numeric part of a job or job array identifier, e.g. **1234.**

Subjob index

The unique index which differentiates one subjob from another. This must be a non-negative integer.

Job array identifier

The identifier returned upon success when submitting a job array. The format is **sequence_number[]** or

sequence_number[].server.domain.com.

Job array range

A set of subjobs within a job array. When specifying a range, indices used must be valid members of the job array's indices.

9.1.1 Description

A job array is a compact representation of one or more jobs, called subjobs when part of a Job array, which have the same job script, and have the same values for all attributes and resources, with the following exceptions:

- each subjob has a unique index
- Job Identifiers of subjobs only differ by their indices
- the state of subjobs can differ

All subjobs within a job array have the same scheduling priority.

A job array is submitted through a single command which returns, on success, a "job array identifier" with a server-unique sequence number. Subjob indices are specified at submission time. These can be:

- a contiguous range, e.g. 1 through 100
- a range with a stepping factor, e.g. every second entry in 1 through 100 (1, 3, 5, ... 99)

A job array identifier can be used

- by itself to represent the set of all subjobs of the job array
- with a single index (a “job array identifier”) to represent a single subjob
- with a range (a “job array range”) to represent the subjobs designated by the range

9.1.2 Identifier Syntax

Job arrays have three identifier syntaxes:

- The job array object itself : 1234[].server or 1234[]
- A single subjob of a job array with index M: 1234[M].server or 1234[M]
- A range of subjobs of a job array: 1234[X-Y:Z].server or 1234[X-Y:Z]

Examples:

1234[].server.domain.com	Full job array identifier
1234[]	Short job array identifier
1234[73]	Subjob identifier of the 73rd index of job array 1234[]
1234	Error, if 1234[] is a job array
1234.server.domain.com	Error, if 1234[].server.domain.com is a job array

The sequence number (1234 in 1234[].server) is unique, so that jobs and job arrays cannot share a sequence number.

Note: Since some shells, for example csh and tcsh, read “[“ and “]” as shell metacharacters, job array names and subjob names will need to be enclosed in double quotes for all PBS commands.

Example:

```
qdel “1234.myhost[5]”
qdel “1234.myhost[]”
```

Single quotes will work, except where you are using shell variable substitution.

9.2 qsub: Submitting a Job Array

To submit a job array, qsub is used with the option **-J range**, where **range** is of the form **X-Y[:Z]**. **X** is the starting index, **Y** is the ending index, and **Z** is the optional **stepping factor**. X and Y must be whole numbers, and Z must be a positive integer. Y must be greater than X. If Y is not a multiple of the stepping factor above X, (i.e. it won't be used as an index value) the highest index used will be the next below Y. For example, 1-100:2 gives 1, 3, 5, ... 99.

Blocking qsub waits until the entire job array is complete, then returns the exit status of the job array.

Interactive submission of job arrays is not allowed.

Examples:

Example 1: To submit a job array of 10,000 subjobs, with indices 1, 2, 3, ... 10000:

```
$ qsub -J 1-10000 job.scr
1234[].server.domain.com
```

Example 2: To submit a job array of 500 subjobs, with indices 500, 501, 502, ... 1000:

```
$ qsub -J 500-1000 job.scr
1235[].server.domain.com
```

Example 3: To submit a job array with indices 1, 3, 5 ... 999:

```
$ qsub -J 1-1000:2 job.scr
1236[].server.domain.com
```

9.2.1 Interactive Job Submission

Job arrays do not support interactive submission.

9.3 Job Array Attributes

Job arrays and subjobs have all of the attributes of a job. In addition, they have the following when appropriate. These attributes are read-only.

Table 9-1: Job Array Attributes

Name	Type	Applies To	Value
array	boolean	job array	True if item is job array
array_id	string	subjob	Subjob's job array identifier
array_index	string	subjob	Subjob's index number
array_state_count	string	job array	Similar to state_count attribute for server and queue objects. Lists number of subjobs in each state.
array_indices_remaining	string	job array	List of indices of subjobs still queued. Range or list of ranges, e.g. 500, 552, 596-1000
array_indices_submitted	string	job array	Complete list of indices of subjobs given at submission time. Given as range, e.g. 1-100

9.4 Job Array States

Job array states map closely to job states except for the ‘B’ state. The ‘B’ state applies to job arrays and indicates that at least one subjob has left the queued state and is running or has run, but not all subjobs have run. Job arrays will never be in the ‘R’, ‘S’ or ‘U’ states.

Table 9-2: Job Array States

State	Indication
B	The job array has started
W	The job array has a wait time in the future
H	The job array is held
T	The job array is in transit between servers
Q	The job array is queued, or has been qrerun
E	All subjobs are finished and the server is cleaning up the job array

9.4.1 Subjob States

Subjobs can be in one of six states, listed here.

Table 9-3: Subjob States

State	Indication
Q	Queued
R	Running
E	Ending
X	Expired or deleted; subjob has completed execution or been deleted
S	Suspended

Table 9-3: Subjob States

State	Indication
U	Suspended by keyboard activity

9.5 PBS Environmental Variables

Table 9-4: PBS Environmental Variables

Environment Variable Name	Used For	Description
$PBS_ARRAY_INDEX	subjobs	Subjob index in job array, e.g. 7
$PBS_ARRAY_ID	subjobs	Identifier for a job array. Sequence number of job array, e.g. 1234[].server
$PBS_JOBID	Jobs, subjobs	Identifier for a job or a subjob. For subjob, sequence number and subjob index in brackets, e.g. 1234[7].server

9.6 File Staging

File staging for job arrays is like that for jobs, with an added variable to specify the subjob index. This variable is **^array_index^**. This is the name of the variable that will be used for the actual array index. The stdout and stderr files follow the naming convention for jobs, but include the identifier of the job array, which includes the subscripted index. As with jobs, the stagein and stageout keywords require the -W option to qsub.

9.6.1 Specifying Files To Be Staged In or Staged Out

You can specify files to be staged in before the job runs and staged out after the job runs by using `-W stagein=file_list` and `-W stageout=file_list`. You can use these as options to qsub, or as directives in the job script.

The *file_list* takes the form:

local_path@hostname:remote_path[,...]

for both stagein and stageout.

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

The '@' character separates the local specification from the remote specification.

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

IMPORTANT:

It is advisable to use an absolute pathname for the remote_path. Remember that the path to your home directory may be different on each machine, and that when using `sandbox = PRIVATE,` you may or may not have a home directory on all execution machines.

For stagein, the direction of travel is **from** remote_path **to** local_path.

For stageout, the direction of travel is **from** local_path **to** remote_path.

When staging more than one filename, separate the filenames with a comma and enclose the entire list in double quotes.

Examples:

Remote_path: store:/film

Data files used as input: frame1, frame2, frame3

Local_path: pix

Executable: a.out

For this example, `a.out` produces `frame2.out` from `frame2.`

```
#PBS -W stagein=pix/in/frame^array_index^@store:/film/
   frame^array_index^
#PBS- W stageout=pix/out/frame^array_index^.out
   @store:/film/frame^array_index^.out
#PBS -J 1-3 a.out frame$PBS_ARRAY_INDEX ./in ./out
```

Note that the stageout statement is all one line, broken here for readability.

The result will be that the user's directory named "film" contains the original files `frame1, frame2, frame3`, plus the new files `frame1.out, frame2.out` and `frame3.out`.

9.6.1.1 Scripts

Example 1: In this example, we have a script named ArrayScript which calls scriptlet1 and scriptlet2.

All three scripts are located in `/homedir/testdir`.

```
#!/bin/sh
#PBS -N ArrayExample
#PBS -J 1-2
echo "Main script: index " $PBS_ARRAY_INDEX
/homedir/testdir/scriptlet$PBS_ARRAY_INDEX
```

In our example, scriptlet1 and scriptlet2 simply echo their names. We run ArrayScript using the qsub command:

```
qsub ArrayScript
```

Example 2: In this example, we have a script called `StageScript`. It takes two input files, `dataX` and `extraX`, and makes an output file,

newdataX, as well as echoing which iteration it is on. The dataX and extraX files will be staged from inputs to work, then newdataX will be staged from work to outputs.

```
#!/bin/sh
#PBS -N StagingExample
#PBS -J 1-2
#PBS -W stagein="/homedir/work/data^array_index^
   @host1:/homedir/inputs/data^array_index^, \
   /homedir/work/extra^array_index^ \
   @host1:/homedir/inputs/extra^array_index^"
#PBS -W stageout=/homedir/work/newdata^array_index^
   @host1:/homedir/outputs/newdata^array_index^
echo "Main script: index " $PBS_ARRAY_INDEX
cd /homedir/work
cat data$PBS_ARRAY_INDEX extra$PBS_ARRAY_INDEX \
   >> newdata$PBS_ARRAY_INDEX
```

Local path (execution directory):

```
/homedir/work
```

Remote host (data storage host):

```
host1
```

Remote path for inputs (original data files dataX and extraX):

```
/homedir/inputs
```

Remote path for results (output of computation newdataX):

```
/homedir/outputs
```

StageScript resides in /homedir/testdir. In that directory, we can run it by typing:

```
qsub StageScript
```

It will run in /homedir, our home directory, which is why the line

```
"cd /homedir/work"
```

is in the script.

Example 3: In this example, we have the same script as before, but we will run it in a staging and execution directory created by PBS. StageScript takes two input files, `dataX` and `extraX`, and makes an output file, `newdataX`, as well as echoing which iteration it is on. The `dataX` and `extraX` files will be staged from `inputs` to the staging and execution directory, then `newdataX` will be staged from the staging and execution directory to `outputs`.

```
#!/bin/sh
#PBS -N StagingExample
#PBS -J 1-2
#PBS -W stagein="data^array_index^\
   @host1:/homedir/inputs/data^array_index^, \
   extra^array_index^ \
   @host1:/homedir/inputs/extra^array_index^"
#PBS -W stageout=newdata^array_index^\
   @host1:/homedir/outputs/newdata^array_index^
echo "Main script: index " $PBS_ARRAY_INDEX
cat data$PBS_ARRAY_INDEX extra$PBS_ARRAY_INDEX \
   >> newdata$PBS_ARRAY_INDEX
```

Local path (execution directory):

created by PBS; we don't know the name

Remote host (data storage host):

`host1`

Remote path for inputs (original data files `dataX` and `extraX`):

`/homedir/inputs`

Remote path for results (output of computation `newdataX`):

`/homedir/outputs`

`StageScript` resides in `/homedir/testdir`. In that directory, we can run it by typing:

```
qsub StageScript
```

It will run in the staging and execution directory created by PBS. See section 8.7 “Input/Output File Staging” on page 167.

9.6.1.2 Output Filenames

The name of the job array will default to the script name if no name is given via qsub -N.

For example, if the sequence number were 1234,

```
#PBS -N fixgamma
```

would give stdout for index number 7 the name fixgamma.o1234.7 and stderr the name fixgamma.e1234.7. The name of the job array can also be given through stdin.

9.6.2 Job Array Staging Syntax on Windows

In Windows the stagein and stageout string must be contained in double quotes when using ^array_index^.

Example of a stagein:

```
qsub -W stagein="foo.^array_index^@host-
   1:C:\WINNT\Temp\foo.^array_index^" -J 1-5
   stage_script
```

Example of a stageout:

```
qsub -W stageut="C:\WINNT\Temp\foo.^array_index^@host-
   1:Q:\my_username\foo. ^array_index^_out" -J 1-5
   stage_script
```

9.7 PBS Commands

9.7.1 PBS Commands Taking Job Arrays as Arguments

Note: Some shells such as csh and tcsh use the square bracket (“[“, “]”) as a metacharacter. When using one of these shells, and a PBS command taking subjobs, job arrays or job array ranges as arguments, the subjob, job array or job array range must be enclosed in double quotes.

The following table shows PBS commands that take job arrays, subjobs or ranges as arguments. The cells in the table indicate which objects are acted upon. In the table,

Array[] =	the job array object
Array[Range] =	the set of subjobs of the job array with indices in range given
Array[Index] =	the individual subjob of the job array with the index given
Array[RUNNING] =	the set of subjobs of the job array which are currently running
Array[QUEUED] =	the set of subjobs of the job array which are currently queued
Array[REMAINING] =	the set of subjobs of the job array which are queued or running
Array[DONE]=	the set of subjobs of the job array which have finished running

Table 9-5: PBS Commands Taking Job Arrays as Arguments

	Argument to Command		
Command	Array[]	Array[Range]	Array[Index]
`qstat`	Array[]	Array[Range]	Array[Index]
`qdel`	Array[] & Array[REMAINING]	Array[Range] where Array[REMAINING]	Array[Index]
`qalter`	Array[]	erroneous	erroneous
`qorder`	Array[]	erroneous	erroneous
`qmove`	Array[] & Array[QUEUED]	erroneous	erroneous
`qhold`	Array[] & Array[QUEUED]	erroneous	erroneous
`qrls`	Array[] & Array[QUEUED]	erroneous	erroneous
`qrerun`	Array[RUNNING] & Array[DONE]	Array[Range] where Array[RUNNING]	Array[Index]
`qrun`	erroneous	Array[Range] where Array[QUEUED]	Array[Index]
`tracejob`	erroneous	erroneous	Array[Index]
`qsig`	Array[RUNNING]	Array[Range] where Array[RUNNING]	Array[Index]
`qmsg`	erroneous	erroneous	erroneous

9.7.2 qstat: Status of a Job Array

The qstat command is used to query the status of a Job Array. The default output is to list the Job Array in a single line, showing the Job Array Identifier. Options can be combined. To show the state of all running subjobs, use -t -r. To show the state only of subjobs, not job arrays, use -t -J.

Table 9-6: Job Array and Subjob Options to qstat

Option	Result
-t	Shows state of job array object and subjobs. Will also show state of jobs.
-J	Shows state only of job arrays.
-p	Prints the default display, with column for Percentage Completed. For a job array, this is the number of subjobs completed or deleted divided by the total number of subjobs. For a job, it is time used divided by time requested.

Examples:

We run an example job and an example job array, on a machine with 2 processors:

demoscript:

```
#!/bin/sh
#PBS -N JobExample
sleep 60
```

arrayscript:

```
#!/bin/sh
#PBS -N ArrayExample
#PBS -J 1-5
sleep 60
```

We run these scripts using qsub.

qsub arrayscript

1235[].host

qsub demoscript

1236.host

Then:

qstat

```
Job id      Name         User        Time Use S Queue
----------- ------------ ----------  -------- - -----
1235[].host ArrayExample user1              0 B workq
1236.host   JobExample   user1              0 Q workq
```

qstat -J

```
Job id      Name         User        Time Use S Queue
----------- ------------ ----------  -------- - -----
1235[].host ArrayExample user1              0 B workq
```

qstat -p

```
Job id      Name         User        % done  S Queue
----------- ------------ ----------  ------- - -----
1235[].host ArrayExample user1            0  B workq
1236.host   JobExample   user1           --  Q workq
```

qstat -t

```
Job id       Name         User        Time Use S Queue
-----------  ------------ ----------  -------- - -----
1235[].host  ArrayExample user1              0 B workq
1235[1].host ArrayExample user1       00:00:00 R workq
```

```
1235[2].host ArrayExample user1       00:00:00 R workq
1235[3].host ArrayExample user1              0 Q workq
1235[4].host ArrayExample user1              0 Q workq
1235[5].host ArrayExample user1              0 Q workq
1236.host    JobExample   user1              0 Q workq
```

qstat -Jt

```
Job id       Name         User   Time Use S Queue
------------ ------------ -----  -------- - -----
1235[1].host ArrayExample user1  00:00:00 R workq
1235[2].host ArrayExample user1  00:00:00 R workq
1235[3].host ArrayExample user1         0 Q workq
1235[4].host ArrayExample user1         0 Q workq
1235[5].host ArrayExample user1         0 Q workq
```

After the first two subjobs finish:

qstat -Jtp

```
Job id       Name         User   % done S Queue
------------ ------------ -----  ------ - -----
1235[1].host ArrayExample user1     100 X workq
1235[2].host ArrayExample user1     100 X workq
1235[3].host ArrayExample user1      -- R workq
1235[4].host ArrayExample user1      -- R workq
1235[5].host ArrayExample user1      -- Q workq
```

qstat -pt

```
Job id       Name         User   % done S Queue
------------ ------------ -----  ------ - -----
1235[].host  ArrayExample user1      40 B workq
```

```
1235[1].host ArrayExample user1     100 X workq
1235[2].host ArrayExample user1     100 X workq
1235[3].host ArrayExample user1      -- R workq
1235[4].host ArrayExample user1      -- R workq
1235[5].host ArrayExample user1      -- Q workq
1236.host    JobExample   user1      -- Q workq
```

Now if we wait until only the last subjob is still running:

qstat -rt

```
                                                        Req'd  Req'd   Elap
Job ID     Username Queue Jobname   SessID NDS TSK Memory Time S Time
---------- ------  ----- ---------  ---- --- --- ------ ---- - -----
1235[5].host user1  workq ArrayExamp 3048  --   1    --    --  R 00:00
1236.host    user1  workq JobExample 3042  --   1    --    --  R 00:00
```

qstat -Jrt

```
                                                      Req'd  Req'd   Elap
Job ID     Username Queue Jobname  SessID NDS TSK Memory Time S Time
---------- -------- ----- --------- ---- --- --- ------ ---- - -----
1235[5].host   user1 workq ArrayExamp 048 --   1    --    --  R 00:01
```

9.7.3 qdel: Deleting a Job Array

The `qdel` command will take a job array identifier, subjob identifier or job array range. The indicated object(s) are deleted, including any currently running subjobs. Running subjobs are treated like running jobs. Subjobs not running will be deleted and never run. Only one email is sent per deleted job array, so deleting a job array of 5000 subjobs results in one email being sent.

9.7.4 qalter: Altering a Job Array

The qalter command can only be used on a job array object, not on subjobs or ranges. Job array attributes are the same as for jobs.

9.7.5 qorder: Ordering Job Arrays in the Queue

The qorder command can only be used with job array objects, not on subjobs or ranges. This will change the queue order of the job array in association with other jobs or job arrays in the queue.

9.7.6 qmove: Moving a Job Array

The qmove command can only be used with job array objects, not with subjobs or ranges. Job arrays can only be moved from one server to another if they are in the 'Q', 'H', or 'W' states, and only if there are no running subjobs. The state of the job array object is preserved in the move. The job array will run to completion on the new server.

As with jobs, a qstat on the server from which the job array was moved will not show the job array. A qstat on the job array object will be redirected to the new server.

Note: The subjob accounting records will be split between the two servers.

9.7.7 qhold: Holding a Job Array

The qhold command can only be used with job array objects, not with subjobs or ranges. A hold can be applied to a job array only from the 'Q', 'B' or 'W' states. This will put the job array in the 'H', held, state. If any subjobs are running, they will run to completion. No queued subjobs will be started while in the 'H' state.

9.7.8 qrls: Releasing a Job Array

The qrls command can only be used with job array objects, not with subjobs or ranges. If the job array was in the 'Q' or 'B' state, it will be returned to that state. If it was in the 'W' state, it will be returned to that state unless its waiting time was reached, it will go to the 'Q' state.

9.7.9 qrerun: Requeueing a Job Array

The qrerun command will take a job array identifier, subjob identifier or job array range. If a job array identifier is given as an argument, it is returned to its initial state at submission time, or to its altered state if it has been qaltered. All of that job array's subjobs are requeued, which includes those that are currently running, and completed and deleted. If a subjob or range is given, those subjobs are requeued as jobs would be.

9.7.10 qrun: Running a Job Array

The qrun command takes a subjob or a range of subjobs, not a job array object. If a single subjob is given as the argument, it is run as a job would be. If a range of subjobs is given as the argument, the non-running subjobs within that range will be run.

9.7.11 tracejob on Job Arrays

The tracejob command can be run on job arrays and individual subjobs. When tracejob is run on a job array or a subjob, the same information is displayed as for a job, with additional information for a job array. Note that subjobs do not exist until they are running, so tracejob will not show any information until they are. When tracejob is run on a job array, the information displayed is only that for the job array object, not the subjobs. Job arrays themselves do not produce any MOM log information. Running tracejob on a job array will give information about why a subjob did not start.

9.7.12 qsig: Signaling a Job Array

If a job array object, subjob or job array range is given to qsig, all currently running subjobs within the specified set will be sent the signal.

9.7.13 qmsg: Sending Messages

The qmsg command is not supported for job arrays.

9.8 Other PBS Commands Supported for Job Arrays

9.8.1 qselect: Selection of Job Arrays

The default behavior of qselect is to return the job array identifier, without returning subjob identifiers.

Note: qselect will not return any job arrays when the state selection (-s) option restricts the set to ‘R’, ‘S’, ‘T’ or ‘U’, because a job array will never be in any of these states. However, qselect can be used to return a list of subjobs by using the -t option.

Options to qselect can be combined. For example, to restrict the selection to subjobs, use both the -J and the -T options. To select only running subjobs, use -J -T -sR.

Table 9-7: Options to qselect for Job Arrays

Option	Selects	Result
(none)	jobs, job arrays	Shows job and job array identifiers
-J	job arrays	Shows only job array identifiers
-T	jobs, subjobs	Shows job and subjob identifiers

9.9 Job Arrays and xpbs

xpbs does not support job arrays.

9.10 More on Job Arrays

9.10.1 Job Array Run Limits

Jobs and subjobs are treated the same way by job run limits. For example, if *max_user_run* is set to 5, a user can have a maximum of 5 subjobs and/or jobs running.

9.10.2 Starving

A job array's starving status is based on the queued portion of the array. This means that if there is a queued subjob which is starving, the job array is starving. A running subjob retains its starving status when it was started.

9.10.3 Job Array Dependencies

Job dependencies are supported:

between job arrays and job arrays

between job arrays and jobs

between jobs and job arrays

Note: Job dependencies are not supported for subjobs or ranges of subjobs.

9.10.4 Accounting

Job accounting records for job arrays and subjobs are the same as for jobs. When a job array has been moved from one server to another, the subjob accounting records are split between the two servers, except that there will be no ‘Q’ records for subjobs.

9.10.5 Checkpointing

Checkpointing is not supported for job arrays. On systems that support checkpointing, subjobs are not checkpointed, instead they run to completion.

9.10.6 Prologues and Epilogues

If defined, prologues and epilogues will run at the beginning and end of each subjob, but not for job arrays.

9.10.7 Job Array Exit Status

The exit status of a job array is determined by the status of each of the completed subjobs. It is only available when all valid subjobs have completed. The individual exit status of a completed subjob is passed to the epilogue, and is available in the ‘E’ accounting log record of that subjob.

Table 9-8:

Exit Status	Meaning
0	All subjobs of the job array returned an exit status of 0. No PBS error occurred. Deleted subjobs are not considered
1	At least 1 subjob returned a non-zero exit status. No PBS error occurred.
2	A PBS error occurred.

9.10.8 Scheduling Job Arrays

All subjobs within a job array have the same scheduling priority.

9.10.8.1 Preemption

Individual subjobs may be preempted by higher priority work.

9.10.8.2 Peer Scheduling

Peer scheduling does not support job arrays.

9.10.8.3 Fairshare

Subjobs are treated like jobs with respect to fairshare ordering, fairshare accounting and fairshare limits. If running enough subjobs of a job array causes the priority of the owning entity to change, additional subjobs from that job array may not be the next to start.

9.10.8.4 Placement Sets and Node Grouping

All nodes associated with a single subjob should belong to the same placement set or node group. Different subjobs can be put on different placement sets or node groups.

Chapter 10

Multiprocessor Jobs

10.1 Job Placement

Placement sets allow partitioning by multiple resources, so that a vnode may be in one set that share a value for one resource, and another set that share a different value for a different resource. See the **PBS Professional Administrator's Guide**.

If a job requests grouping by a resource, i.e. *place=group=resource*, then the chunks are placed as requested and complex-wide node grouping is ignored.

If a job is to use node grouping but the required number of vnodes is not defined in any one group, grouping is ignored. This behavior is unchanged.

10.2 Submitting SMP Jobs

To submit a job which should run on one host and which requires a certain number of cpus and amount of memory, submit the job with:

```
qsub -l select=ncpus=N:mem=M -l place=group=host
```

When the job is run, the PBS_NODEFILE will contain one entry, the name of the selected execution host. Generally this is ignored for SMP jobs as all processes in the job are run on the host where the job script is run. The job will have two environment variables, NCPUS and OMP_NUM_THREADS, set to N, the number of CPUs allocated.

10.3 Submitting MPI Jobs

The preferred method for submitting an MPI job is by specifying one chunk per MPI task. For example, for a 10-way MPI job with 2gb of memory per MPI task, you would use:

```
qsub -l select=10:ncpus=1:mem=2gb
```

If you have a cluster of small systems with for example 2 CPUs each, and you wish to submit an MPI job that will run on four separate hosts, then submit:

```
qsub -l select=4:ncpus=1 -l place=scatter
```

The PBS_NODEFILE file will contain one entry for each of the hosts allocated to the job. In the example above, it would contain 4 lines. The variables NCPUS and OMP_NUM_THREADS will be set to one.

If you do not care where the four MPI processes are run, you may submit:

```
qsub -l select=4:ncpus=1 -l place=free
```

and the job will run on 2, 3, or 4 hosts depending on what is available.

For this example, PBS_NODEFILE will contain 4 entries, either four separate hosts, or 3 hosts one of which is repeated once, or 2 hosts, etc. NCPUS and OMP_NUM_THREADS will be set 1 or 2 depending on the number of cpus allocated from the first listed host.

10.3.1 The mpiprocs Resource

The number of MPI processes for a job is controlled by the value of the resource mpiprocs. The mpiprocs resource controls the contents of the PBS_NODEFILE on the host which executes the top PBS task for the PBS job (the one executing the PBS job script.) See “Built-in Resources” on page 38. The PBS_NODEFILE contains one line per MPI process with the name of the host on which that process should execute. The number of lines in PBS_NODEFILE is equal to the sum of the values of mpiprocs over all chunks requested by the job. For each chunk with mpiprocs=P, (where P > 0), the host name (the value of the allocated vnode's resources_available.host) is written to the PBS_NODEFILE exactly P times.

If a user wishes to run two MPI processes on each of 3 hosts and have them "share" a single processor on each host, the user would request

```
-lselect=3:ncpus=1:mpiprocs=2
```

The PBS_NODEFILE would contain the following list:

```
VnodeA
VnodeA
VnodeB
VnodeB
VnodeC
VnodeC
```

If you want 3 chunks, each with 2 CPUs and running 2 MPI process, use:

```
-l select=3:ncpus=2:mpiprocs=2...
```

The PBS_NODEFILE would contain the following list:

```
VnodeA
VnodeA
VnodeB
VnodeB
VnodeC
VnodeC
```

10.4 OpenMP Jobs with PBS

PBS Professional supports OpenMP applications by setting the OMP_NUM_THREADS variable automatically based on the resource request of a job in the environment of the job. The OpenMP run-time will pick up the value of OMP_NUM_THREADS and create threads appropriately.

The OMP_NUM_THREADS value can be set explicitly by using the ompthreads pseudo-resource for any chunk within the select statement. If ompthreads is not used, then OMP_NUM_THREADS is set to the value of the ncpus resource of that chunk. If neither ncpus nor ompthreads is used within the select statement, then OMP_NUM_THREADS is set to 1.

To submit an OpenMP job is as a single chunk, for a 2-CPU job requiring 10gb of memory, you would use:

```
qsub -l select=1:ncpus=2:mem=10gb
```

You might be running an OpenMP application on a host and wish to run fewer threads than the number of CPUs requested. This might be because the threads need exclusive access to shared resources in a multi-core processor system, such as to a cache shared between cores, or to the memory shared between cores. If you want one chunk, with 16 CPUs and 8 threads:

```
qsub -l select=1:ncpus=16:ompthreads=8
```

You might be running an OpenMP application on a host and wish to run more threads than the number of CPUs requested (because each thread is I/O bound perhaps). If you want one chunk, with eight CPUs and 16 threads:

```
qsub -l select=1:ncpus=8:ompthreads=16
```

10.5 Hybrid MPI-OpenMP Jobs

For jobs that are both MPI and multi-threaded, the number of threads per chunk, for all chunks, is set to the number of threads requested (explicitly or implicitly) in the first chunk, except for MPIs that have been integrated with the PBS TM API. For these MPIs (LAM MPI), you can specify the

number of threads separately for each chunk. This means that for most MPIs, OMP_NUM_THREADS and NCPUS will default to the number of ncpus requested on the first chunk, and for integrated MPIs, you can set the ompthreads resource separately for each chunk.

Should you have a job that is both MPI and multi-threaded, you can request one chunk for each MPI process, or set mpiprocs to the number of MPI processes you want on each chunk.

For example, to request 4 chunks, each with 1 MPI process, 2 CPUs and 2 threads:

```
qsub -l select=4:ncpus=2
```

or

```
qsub -l select=4:ncpus=2:ompthreads=2
```

To request 4 chunks, each with 2 CPUs and 4 threads:

```
qsub -l select=4:ncpus=2:ompthreads=4
```

To request 16 MPI processes each with 2 threads on machines with 2 processors:

```
qsub -l select=16:ncpus=2
```

To request two chunks, each with 8 CPUs and 8 MPI tasks and four threads:

```
qsub -l select=2:ncpus=8:mpiprocs=8:ompthreads=4
```

Example:

```
qsub -l select=4:ncpus=2
```

This request is satisfied by 4 CPUs from VnodeA, 2 from VnodeB and 2 from VnodeC, so the following is written to the PBS_NODEFILE:

```
VnodeA
VnodeA
VnodeB
VnodeC
```

The OpenMP environment variables are set (for the 4 PBS tasks corresponding to the 4 MPI processes) as follows:

- For PBS task #1 on VnodeA: OMP_NUM_THREADS=2 NCPUS=2
- For PBS task #2 on VnodeA: OMP_NUM_THREADS=2 NCPUS=2
- For PBS task #3 on VnodeB: OMP_NUM_THREADS=2 NCPUS=2
- For PBS task #4 on VnodeC: OMP_NUM_THREADS=2 NCPUS=2

Example:

```
qsub -l select=3:ncpus=2:mpiprocs=2:ompthreads=1
```

This is satisfied by 2 CPUs from each of three vnodes (VnodeA, VnodeB, and VnodeC), so the following is written to the PBS_VNODEFILE:

```
VnodeA
VnodeA
VnodeB
VnodeB
VnodeC
VnodeC
```

The OpenMP environment variables are set (for the 6 PBS tasks corresponding to the 6 MPI processes) as follows:

- For PBS task #1 on VnodeA: OMP_NUM_THREADS=1 NCPUS=1
- For PBS task #2 on VnodeA: OMP_NUM_THREADS=1 NCPUS=1
- For PBS task #3 on VnodeB: OMP_NUM_THREADS=1 NCPUS=1
- For PBS task #4 on VnodeB: OMP_NUM_THREADS=1 NCPUS=1
- For PBS task #5 on VnodeC: OMP_NUM_THREADS=1 NCPUS=1
- For PBS task #6 on VnodeC: OMP_NUM_THREADS=1 NCPUS=1

To run two threads on each of N chunks, each running a process, all on the same Altix:

```
qsub -l select=N:ncpus=2 -l place=pack
```

This starts N processes on a single host, with two OpenMP threads per process, because OMP_NUM_THREADS=2.

10.6 MPI Jobs with PBS

PBS creates one MPI process per chunk.

For most implementations of the Message Passing Interface (MPI), you would use the `mpirun` command to launch your application. For example, here is a sample PBS script for an MPI job:

```
#PBS -l select=arch=linux
#
mpirun -np 32 -machinefile $PBS_NODEFILE a.out
```

10.6.1 MPICH Jobs With PBS

For users of PBS with MPICH on Linux, the `mpirun` command has been changed slightly. The syntax and arguments are the same except for one option, which should not be set by the user:

-machinefile file
: PBS supplies the machinefile. If the user tries to specify it, PBS will print a warning that it is replacing the machinefile.

Example of using mpirun:

```
#PBS -l select=arch=linux
#
mpirun a.out
```

Under Windows the `-localroot` option to `MPICH`'s `mpirun` command may be needed in order to allow the job's processes to run more efficiently.

10.6.2 MPI Jobs Using LAM MPI

The `pbs_mpilam` command follows the convention of LAM's `mpirun`. The "nodes" here are LAM nodes. LAM's `mpirun` has two syntax forms:

pbs_mpilam/mpirun [global_options] [<where>] <program> [--args]

pbs_mpilam/mpirun [global_options] <schema file>

Where

<where> is a set of node and/or CPU identifiers indicating where to start <program>:

Nodes: n<list>, e.g., n0-3,5

CPUS: c<list>, e.g., c0-3,5

Extras: h (local node), o (origin node), N (all nodes), C (all CPUs)

<schema file> is an ASCII file containing a description of the programs which constitute an application.

The first form is fully supported by PBS: all user MPI processes are tracked. The second form is supported, but user MPI processes are not tracked.

CAUTION: Keep in mind that if the <where> argument and global option -np or -c are not specified in the command line, then pbs_mpilam will expect an ASCII schema file as argument.

10.6.3 MPI Jobs Using AIX, POE

PBS users of AIX machines running IBM's Parallel Operating Environment, or POE, can run jobs on the HPS using either IP or US mode. PBS will manage the HPS.

Under PBS, the poe command is slightly different. The syntax and arguments are the same except for the following:

Options:

-procs <numranks>
: If the -procs option or the MP_PROCS environment variable is not set by the user, a default of the number of entries in the file $PBS_NODEFILE is used. The maximum number of ranks that can be launched is the number of entries in $PBS_NODEFILE.

-hostfile <file>
: PBS supplies the hostfile to POE. Any specification for hostfile will be ignored.

-euilib {ip | us}
: If the command line option -euilib is set, it will take precedence over the MP_EUILIB environment variable. If the -euilib option is set to us, user mode is set for the job. If the option is set to any other value, that value is passed to poe.

-msg_api
: This option can only take the values "MPI" or "LAPI".

Environment variables:

MP_EUILIB
: If the MP_EUILIB environment variable is set to us, user mode is set for the job. If the variable is set to any other value, that value is passed to poe.

MP_HOSTFILE
: The MP_HOSTFILE environment variable is excised.

MP_PROCS
: If the -procs option or the MP_PROCS environment variable is not set by the user, a default of the number of entries in the file $PBS_NODEFILE is used.

MP_MSG_API
: This variable can only take the values "MPI" or "LAPI".

Notes:

Since PBS is tracking tasks started by poe, these tasks are counted towards a user's run limits. Running multiple poe jobs in the background will not work. Instead, run poe jobs one after the other or submit separate jobs. Otherwise HPS windows will be used by more than one task. The tracejob command will show any of various error messages.

For more information on using IBM's Parallel Operating Environment, see "IBM Parallel Environment for AIX 5L Hitchhiker's Guide".

10.6.3.1 Examples Using poe

Example 1: Using IP mode, run a single executable poe job with 4 ranks on hosts spread across the PBS-allocated nodes listed in $PBS_NODEFILE:

```
% cat $PBS_NODEFILE
host1
host2
host3
host4
% cat job.script
poe /path/mpiprog -euilib ip

% qsub -l select=4:ncpus=1 -lplace=scatter
    job.script
```

Example 2: Using US mode, run a single executable poe job with 4 ranks on hosts spread across the PBS-allocated nodes listed in $PBS_NODEFILE:

```
% cat $PBS_NODEFILE
host1
host2
host3
host4

% cat job.script
poe /path/mpiprog -euilib us

% qsub -l select=4:ncpus=1 -lplace=scatter
    job.script
```

Example 3: Using IP mode, run executables prog1 and prog2 with 2 ranks of prog1 on host1, 2 ranks of prog2 on host2 and 2 ranks of prog2 on host3.

```
% cat $PBS_NODEFILE
host1
host1
host2
host2
host3
host3
```

```
% cat job.script
echo prog1 > /tmp/poe.cmd
echo prog1 >> /tmp/poe.cmd
echo prog2 >> /tmp/poe.cmd
echo prog2 >> /tmp/poe.cmd
echo prog2 >> /tmp/poe.cmd
echo prog2 >> /tmp/poe.cmd
poe -cmdfile /tmp/poe.cmd -euilib ip
rm /tmp/poe.cmd
```

```
% qsub -l select=3:ncpus=2:mpiprocs=2 \
    -l place=scatter job.script
```

Example 4: Using US mode, run executables prog1 and prog2 with 2 ranks of prog1 on host1, 2 ranks of prog2 on host2 and 2 ranks of prog2 on host3.

```
% cat $PBS_NODEFILE
host1
host1
host2
host2
host3
host3
```

```
% cat job.script
echo prog1 > /tmp/poe.cmd
echo prog1 >> /tmp/poe.cmd
echo prog2 >> /tmp/poe.cmd
echo prog2 >> /tmp/poe.cmd
echo prog2 >> /tmp/poe.cmd
echo prog2 >> /tmp/poe.cmd
poe -cmdfile /tmp/poe.cmd -euilib us
rm /tmp/poe.cmd
```

```
% qsub -l select=3:ncpus=2:mpiprocs=2 \
    -l place=scatter job.script
```

10.6.3.2 If Your Complex Contains Machines Not on the HPS

If your complex contains machines that are not on the HPS, and you wish to run on the HPS, you must specify machines on the HPS. Your administrator will define a resource on each host on the HPS. To specify machines on the HPS, you must request the "hps" resource in your select statement. For this example, the resource is “hps”.

Using place=scatter: When "scatter" is used, the 4 chunks are on different hosts so each host has 1 hps resource:

```
% qsub -l select=4:ncpus=2:hps=1
```

Using place=pack: When "pack" is used, all the chunks are put on one host so a chunk with no resources and one "hps" must be specified:

```
% qsub -l select=4:ncpus=2+1:ncpus=0:hps=1
```

This ensures that the hps resource is only counted once. You could also use this:

```
% qsub -l select=1:ncpus=8:hps=1
```

For two chunks of 4 CPUs, one on one machine and one on another, you would use:

```
% qsub -l select=2:ncpus=4 -l place=scatter
```

10.6.4 PBS MPI Jobs on HP-UX and Linux

PBS is tightly integrated with the `mpirun` command on HP-UX so that resources can be tracked and processes managed. When running a PBS MPI job, you can use the same arguments to the `mpirun` command as you would outside of PBS. The `-h host` and `-l user` options will be ignored, and the `-np number` option will be modified to fit the available resources.

10.6.5 PBS Jobs with `MPICH-GM's mpirun` Using `rsh/ssh (mpirun.ch_gm)`

PBS provides an interface to `MPICH-GM's mpirun` using `rsh/ssh`. If executed inside a PBS job, this lets PBS track all `MPICH-GM` processes started via `rsh/ssh` so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard `mpirun.ch_gm` had been used.

You use the same command as you would use outside of PBS, either “mpirun.ch_gm” or “mpirun”.

10.6.5.1 Options

Inside a PBS job script, all of the options to the PBS interface are the same as `mpirun.ch_gm` except for the following:

-machinefile <file>
: The `file` argument contents are ignored and replaced by the contents of the `$PBS_NODEFILE`.

-np
: If not specified, the number of entries found in the `$PBS_NODEFILE` is used. The maximum number of ranks that can be launched is the number of entries in $PBS_NODEFILE.

-pg
: The use of the `-pg` option, for having multiple executables on multiple hosts, is allowed but it is up to user to make sure only PBS hosts are specified in the process group file; MPI processes spawned on non-PBS hosts are not guaranteed to be under the control of PBS.

10.6.5.2 Examples

Example 1: Run a single-executable `MPICH-GM` job with 64 processes spread out across the PBS-allocated hosts listed in `$PBS_NODEFILE`:

PBS_NODEFILE:

```
pbs-host1
pbs-host2
pbs-host3
```

```
qsub -l select=3:ncpus=1
mpirun.ch_gm -np 64 /path/myprog.x 1200
^D
<job-id>
```

Example 2: Run an MPICH-GM job with multiple executables on multiple hosts listed in the process group file "procgrp":

```
qsub -l select=3:ncpus=1
echo  "host1 1 user1 /x/y/a.exe arg1 arg2" > procgrp
echo  "host2 1 user1 /x/x/b.exe arg1 arg2" >> procgrp
mpirun.ch_gm -pg procgrp /path/mypro.x
rm -f procgrp
^D
<job-id>
```

When the job runs, mpirun.ch_gm will give this warning message:

```
warning: "-pg" is allowed but it is up to user to make
   sure only PBS hosts are specified; MPI processes
   spawned are not guaranteed to be under the control
   of PBS.
```

The warning is issued because if any of the hosts listed in procgrp are not under the control of PBS, then the processes on those hosts will not be under the control of PBS.

10.6.6 PBS Jobs with MPICH-MX's mpirun Using rsh/ssh (mpirun.ch_mx)

PBS provides an interface to MPICH-MX's mpirun using rsh/ssh. If executed inside a PBS job, this allows for PBS to track all MPICH-MX processes started by rsh/ssh so that PBS can perform accounting and has complete job control. If executed outside of a PBS job, it behaves exactly as if standard mpirun.ch_mx had been used.

You use the same command as you would use outside of PBS, either "mpirun.ch_mx" or "mpirun".

10.6.6.1 Options

Inside a PBS job script, all of the options to the PBS interface are the same as mpirun.ch_mx except for the following:

-machinefile <file>

The `file` argument contents are ignored and replaced by the contents of the `$PBS_NODEFILE`.

-np

If not specified, the number of entries found in the `$PBS_NODEFILE` is used. The maximum number of ranks that can be launched is the number of entries in $PBS_NODEFILE.

-pg

The use of the `-pg` option, for having multiple executables on multiple hosts, is allowed but it is up to user to make sure only PBS hosts are specified in the process group file; MPI processes spawned on non-PBS hosts are not guaranteed to be under the control of PBS.

10.6.6.2 Examples

Example 1: Run a single-executable `MPICH-MX` job with 64 processes spread out across the PBS-allocated hosts listed in `$PBS_NODEFILE`:

PBS_NODEFILE:

```
pbs-host1
pbs-host2
pbs-host3
```

```
qsub -l select=3:ncpus=1
mpirun.ch_mx -np 64 /path/myprog.x 1200
^D
<job-id>
```

Example 2: Run an MPICH-MX job with multiple executables on multiple hosts listed in the process group file "procgrp":

```
qsub -l select=2:ncpus=1
echo  "pbs-host1 1 username /x/y/a.exe arg1 arg2" >
   procgrp
echo  "pbs-host2 1 username /x/x/b.exe arg1 arg2" >>
   procgrp
mpirun.ch_mx -pg procgrp /path/myprog.x
rm -f procgrp
^D
<job-id>
```

mpirun.ch_mx will give the warning message:

```
warning: "-pg" is allowed but it is up to user to make
   sure only PBS hosts are specified; MPI processes
   spawned are not guaranteed to be under PBS-control
```

The warning is issued because if any of the hosts listed in procgrp are not under the control of PBS, then the processes on those hosts will not be under the control of PBS.

10.6.7 PBS Jobs with MPICH-GM's mpirun Using MPD (mpirun.mpd)

PBS provides an interface to MPICH-GM's mpirun using MPD. If executed inside a PBS job, this allows for PBS to track all MPICH-GM processes started by the MPD daemons so that PBS can perform accounting have and complete job control. If executed outside of a PBS job, it behaves exactly as if standard mpirun.mpd with MPD had been used.

You use the same command as you would use outside of PBS, either "mpirun.mpd" or "mpirun". If the MPD daemons are not already running, the PBS interface will take care of starting them for you.

10.6.7.1 Options

Inside a PBS job script, all of the options to the PBS interface are the same as `mpirun.mpd` with MPD except for the following:

-m <file>
: The `file` argument contents are ignored and replaced by the contents of the `$PBS_NODEFILE`.

-np
: If not specified, the number of entries found in the `$PBS_NODEFILE` is used. The maximum number of ranks that can be launched is the number of entries in $PBS_NODEFILE

-pg
: The use of the `-pg` option, for having multiple executables on multiple hosts, is allowed but it is up to user to make sure only PBS hosts are specified in the process group file; MPI processes spawned on non-PBS hosts are not guaranteed to be under the control of PBS.

10.6.7.2 MPD Startup and Shutdown

The script starts MPD daemons on each of the unique hosts listed in `$PBS_NODEFILE`, using either the `rsh` or `ssh` method based on the value of the environment variable `RSHCOMMAND`. The default is `rsh`. The script also takes care of shutting down the MPD daemons at the end of a run.

If the MPD daemons are not running, the PBS interface to `mpirun.mpd` will start GM's MPD daemons as this user on the allocated PBS hosts. The MPD daemons may have been started already by the administrator or by the user. MPD daemons are not started inside a PBS prologue script since it won't have the path of `mpirun.mpd` that the user executed (GM or MX), which would determine the path to the MPD binary.

10.6.7.3 Examples

Example 1: Run a single-executable MPICH-GM job with 64 processes spread out across the PBS-allocated hosts listed in $PBS_NODEFILE:

```
PBS_NODEFILE:
pbs-host1
pbs-host2
pbs-host3
qsub -l select=3:ncpus=1
[MPICH-GM-HOME]/bin/mpirun.mpd -np 64 /path/myprog.x
   1200
^D
<job-id>
```

If the GM MPD daemons are not running, the PBS interface to mpirun.mpd will start them as this user on the allocated PBS hosts. The daemons may have been previously started by the administrator or the user.

Example 2: Run an MPICH-GM job with multiple executables on multiple hosts listed in the process group file "procgrp":

Job script:

```
qsub -l select=3:ncpus=1
echo  "host1 1 user1 /x/y/a.exe arg1 arg2" > procgrp
echo  "host2 1 user1 /x/x/b.exe arg1 arg2" >> procgrp

[MPICH-GM-HOME]/bin/mpirun.mpd -pg procgrp /path/
   mypro.x 1200
rm -f procgrp
^D
<job-id>
```

When the job runs, mpirun.mpd will give the warning message:

```
warning: "-pg" is allowed but it is up to user to make
   sure only PBS hosts are specified; MPI processes
   spawned are not guaranteed to be under PBS-control.
```

The warning is issued because if any of the hosts listed in procgrp are not under the control of PBS, then the processes on those hosts will not be under the control of PBS.

10.6.8 PBS Jobs with MPICH-MX's mpirun Using MPD (mpirun.mpd)

PBS provides an interface to MPICH-MX's mpirun using MPD. If executed inside a PBS job, this allows for PBS to track all MPICH-MX processes started by the MPD daemons so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard mpirun.ch_mx with MPD was used.

You use the same command as you would use outside of PBS, either "mpirun.mpd" or "mpirun". If the MPD daemons are not already running, the PBS interface will take care of starting them for you.

10.6.8.1 Options

Inside a PBS job script, all of the options to the PBS interface are the same as `mpirun.ch_gm` with MPD except for the following:

-m <file>

The `file` argument contents are ignored and replaced by the contents of the `$PBS_NODEFILE`.

-np

If not specified, the number of entries found in the `$PBS_NODEFILE` is used. The maximum number of ranks that can be launched is the number of entries in $PBS_NODEFILE.

-pg

The use of the `-pg` option, for having multiple executables on multiple hosts, is allowed but it is up to user to make sure only PBS hosts are specified in the process group file; MPI processes spawned on non-PBS hosts are not guaranteed to be under the control of PBS.

10.6.8.2 MPD Startup and Shutdown

The PBS `mpirun` interface starts MPD daemons on each of the unique hosts listed in `$PBS_NODEFILE`, using either the `rsh` or `ssh` method, based on value of environment variable `RSHCOMMAND`. The default is `rsh`. The interface also takes care of shutting down the MPD daemons at the end of a run.

If the MPD daemons are not running, the PBS interface to `mpirun.mpd` will start MX's MPD daemons as this user on the allocated PBS hosts. The MPD daemons may already have been started by the administrator or by the user. MPD daemons are not started inside a PBS prologue script since it won't have the path of `mpirun.mpd` that the user executed (GM or MX), which would determine the path to the MPD binary.

10.6.8.3 Examples

Example 1: Run a single-executable `MPICH-MX` job with 64 processes spread out across the PBS-allocated hosts listed in `$PBS_NODEFILE`:

PBS_NODEFILE:

```
pbs-host1
pbs-host2
pbs-host3
```

```
qsub -l select=3:ncpus=1
[MPICH-MX-HOME]/bin/mpirun.mpd -np 64 /path/myprog.x
   1200
^D
<job-id>
```

If the MPD daemons are not running, the PBS interface to `mpirun.mpd` will start GM's MPD daemons as this user on the allocated PBS hosts. The MPD daemons may be already started by the administrator or by the user.

Example 2: Run an `MPICH-MX` job with multiple executables on multiple hosts listed in the process group file “procgrp”:

```
qsub -l select=2:ncpus=1
echo  "pbs-host1 1 username /x/y/a.exe \
    arg1 arg2" > procgrp
echo  "pbs-host2 1 username /x/x/b.exe \
    arg1 arg2" >> procgrp
[MPICH-MX-HOME]/bin/mpirun.mpd -pg procgrp \
    /path/myprog.x 1200
rm -f procgrp
^D
<job-id>
```

`mpirun.mpd` will print a warning message:

warning: “-pg” is allowed but it is up to user to make sure only PBS hosts are specified; MPI processes spawned are not guaranteed to be under PBS-control

The warning is issued because if any of the hosts listed in `procgrp` are not under the control of PBS, then the processes on those hosts will not be under the control of PBS.

10.6.9 PBS Jobs with `MPICH2's mpirun`

PBS provides an interface to `MPICH2's mpirun`. If executed inside a PBS job, this allows for PBS to track all `MPICH2` processes so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard `MPICH2's mpirun` had been used.

You use the same “mpirun” command as you would use outside of PBS.

When submitting PBS jobs that invoke the pbsrun wrapper script for MPICH2's mpirun, be sure to explicitly specify the actual number of ranks or MPI tasks in the qsub select specification. Otherwise, jobs will fail to run with "too few entries in the machinefile".

For instance, specification of the following in 7.1:

```
#PBS -l
   select=1:ncpus=1:host=hostA+1:ncpus=2:host=hostB
mpirun -np 3 /tmp/mytask
```

would result in a 7.1 $PBS_NODEFILE listing:

```
hostA
hostB
hostB
```

but in 8.0 or later it would be:

```
hostA
hostB
```

which would conflict with the "-np 3" specification in mpirun as only 2 MPD daemons will be started.

The correct way now is to specify either a) or b) as follows:

a. `#PBS -l`
`select=1:ncpus=1:host=`hostA+2:ncpus=1:host=hostB

b. #PBS -l
select=1:ncpus=1:host=hostA+1:ncpus=2:host=hostB:mpiprocs=2

which will cause $PBS_NODEFILE to list:

```
hostA
hostB
hostB
```

and an "mpirun -np 3" will then be consistent.

10.6.9.1 Options

If executed inside a PBS job script, all of the options to the PBS interface are the same as `MPICH2's mpirun` except for the following:

-host, -ghost
: For specifying the execution host to run on. Ignored.

-machinefile <file>
: The file argument contents are ignored and replaced by the contents of the `$PBS_NODEFILE.`

-localonly <x>
: For specifying the <x> number of processes to run locally. Not supported. The user is advised instead to use the equivalent arguments:

  `"-np <x> -localonly".`

-np
: If the user does not specify a `-np` option, then no default value is provided by the PBS wrapper scripts. It is up to the local mpirun to decide what the reasonable default value should be, which is usually 1. The maximum number of ranks that can be launched is the number of entries in $PBS_NODEFILE.

10.6.9.2 MPD Startup and Shutdown

The interface ensures that the MPD daemons are started on each of the hosts listed in the `$PBS_NODEFILE`. It also ensures that the MPD daemons are shut down at the end of MPI job execution.

10.6.9.3 Examples

Example 1: Run a single-executable MPICH2 job with 6 processes spread out across the PBS-allocated hosts listed in $PBS_NODEFILE:

PBS_NODEFILE:

```
pbs-host1
pbs-host2
pbs-host3
pbs-host1
pbs-host2
pbs-host3
```

Job.script:

```
# mpirun runs 6 processes mapped to each host
# listed in $PBS_NODEFILE
mpirun -np 6 /path/myprog.x 1200
```

Run job script:

```
qsub -l select=3:ncpus=2 job.script
```

```
<job-id>
```

2. Run an MPICH2 job with multiple executables on multiple hosts using $PBS_NODEFILE and mpiexec arguments in mpirun:

PBS_NODEFILE:

```
hostA
hostA
hostB
hostB
hostC
hostC
```

Job script:

```
#PBS -l select=3:ncpus=2
mpirun -np 2 /tmp/mpitest1 : \
   -np 2 /tmp/mpitest2 : \
   -np 2 /tmp/mpitest3
```

Run job:

```
qsub job.script
```

Example 3: Run an MPICH2 job with multiple executables on multiple hosts using `mpirun -configfile` option and `$PBS_NODEFILE`:

PBS_NODEFILE:

```
hostA
hostA
hostB
hostB
hostC
hostC
```

Job script:

```
#PBS -l select=3:ncpus=2
echo "-np 2 /tmp/mpitest1" > my_config_file
echo "-np 2 /tmp/mpitest2" >> my_config_file
echo "-np 2 /tmp/mpitest3" >> my_config_file
mpirun -configfile my_config_file
rm -f my_config_file
```

Run job:

```
qsub job.script
```

10.6.10 PBS Jobs with Intel MPI's `mpirun`

PBS provides an interface to Intel MPI's `mpirun`. If executed inside a PBS job, this allows for PBS to track all Intel MPI processes so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard Intel MPI's `mpirun` was used.

You use the same "mpirun" command as you would use outside of PBS.

When submitting PBS jobs that invoke the pbsrun wrapper script for Intel MPI, be sure to explicitly specify the actual number of ranks or MPI tasks in the qsub select specification. Otherwise, jobs will fail to run with "too few entries in the machinefile".

For instance, specification of the following in 7.1:

```
#PBS -l
   select=1:ncpus=1:host=hostA+1:ncpus=2:host=hostB
mpirun -np 3 /tmp/mytask
```

would result in a 7.1 $PBS_NODEFILE listing:

```
hostA
hostB
hostB
```

but in 8.0 or later it would be:

```
hostA
hostB
```

which would conflict with the "-np 3" specification in mpirun as only 2 MPD daemons will be started.

The correct way now is to specify either a) or b) as follows:

a. #PBS -l select=1:ncpus=1:host=hostA+2:ncpus=1:host=hostB

b. #PBS -l
select=1:ncpus=1:host=hostA+1:ncpus=2:host=hostB:mpiprocs=2

which will cause $PBS_NODEFILE to list:

```
hostA
hostB
hostB
```

and an "mpirun -np 3" will then be consistent.

10.6.10.1 Options

If executed inside a PBS job script, all of the options to the PBS interface are the same as for Intel MPI's `mpirun` except for the following:

-host, -ghost
: For specifying the execution host to run on. Ignored.

-machinefile <file>
: The file argument contents are ignored and replaced by the contents of the `$PBS_NODEFILE`.

mpdboot option --totalnum=*
: Ignored and replaced by the number of unique entries in `$PBS_NODEFILE`.

mpdboot option --file=*
: Ignored and replaced by the name of `$PBS_NODEFILE.` The argument to this option is replaced by `$PBS_NODEFILE`.

  Argument to `mpdboot` option `-f <mpd_hosts_file>` replaced by `$PBS_NODEFILE`.

-s
: If the PBS interface to Intel MPI's `mpirun` is called inside a PBS job, Intel MPI's `mpirun -s` argument to `mpdboot` is not supported as this closely matches the `mpirun` option "`-s <spec>`". The user can simply run a separate `mpdboot -s` before calling `mpirun`. A warning message is issued by the PBS interface upon encountering a `-s` option telling users of the supported form.

-np
: If the user does not specify a `-np` option, then no default value is provided by the PBS interface. It is up to the local `mpirun` to decide what the reasonable default value should

be, which is usually 1. The maximum number of ranks that can be launched is the number of entries in $PBS_NODEFILE.

10.6.10.2 MPD Startup and Shutdown

Intel MPI's `mpirun` takes care of starting/stopping the MPD daemons. The PBS interface to Intel MPI's `mpirun` always passes the arguments `-totalnum=<number of mpds to start>` and `-file=<mpd_hosts_file>` to the actual `mpirun`, taking its input from unique entries in `$PBS_NODEFILE`.

10.6.10.3 Examples

Example 1: Run a single-executable Intel MPI job with 6 processes spread out across the PBS-allocated hosts listed in `$PBS_NODEFILE`:

PBS_NODEFILE:

```
pbs-host1
pbs-host2
pbs-host3
pbs-host1
pbs-host2
pbs-host3
```

Job script:

```
# mpirun takes care of starting the MPD
# daemons on unique hosts listed in
# $PBS_NODEFILE, and also runs 6 processes
# mapped to each host listed in
# $PBS_NODEFILE; mpirun takes care of
# shutting down MPDs.
mpirun /path/myprog.x 1200
```

Run job script:

```
qsub -l select=3:ncpus=2 job.script
<job-id>
```

Example 2: Run an Intel MPI job with multiple executables on multiple hosts using `$PBS_NODEFILE` and `mpiexec` arguments to `mpirun`:

$PBS_NODEFILE

```
hostA
hostA
hostB
hostB
hostC
hostC
```

Job script:

```
# mpirun runs MPD daemons on hosts listed
# in $PBS_NODEFILE
# mpirun runs 2 instances of mpitest1
# on hostA; 2 instances of mpitest2 on
# hostB; 2 instances of mpitest3 on
# hostC.
# mpirun takes care of shutting down the
# MPDs at the end of MPI job run.
mpirun -np 2 /tmp/mpitest1 : -np 2 /tmp/mpitest2 : -np
   2 /tmp/mpitest3
```

Run job script:

```
qsub -l select=3:ncpus=2 job.script
 <job-id>
```

Example 3: Run an Intel MPI job with multiple executables on multiple hosts via the `-configfile` option and `$PBS_NODEFILE`:

$PBS_NODEFILE:

```
hostA
hostA
hostB
hostB
hostC
hostC
```

Job script:

```
echo "-np 2 /tmp/mpitest1" >> my_config_file
echo "-np 2 /tmp/mpitest2" >> my_config_file
echo "-np 2 /tmp/mpitest3" >> my_config_file

# mpirun takes care of starting the MPD daemons
# config file says run 2 instances of mpitest1
# on hostA; 2 instances of mpitest2 on
# hostB; 2 instances of mpitest3 on
# hostC.
# mpirun takes care of shutting down the MPD
# daemons.
mpirun -configfile my_config_file

# cleanup
rm -f my_config_file
```

Run job script:

```
qsub -l select=3:ncpus=2 job.script
<job-id>
```

10.6.11 PBS Jobs with `MVAPICH1's mpirun`

PBS provides an interface to `MVAPICH1's mpirun`. MVAPICH1 allows use of InfiniBand. If executed inside a PBS job, this allows for PBS to track all `MVAPICH1` processes so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard `MVAPICH1 mpirun` had been used.

You use the same "mpirun" command as you would use outside of PBS.

10.6.11.1 Options

If executed inside a PBS job script, all of the options to the PBS interface are the same as `MVAPICH1's mpirun` except for the following:

-map
: The map option is ignored.

`-machinefile <file>`
: The machinefile option is ignored.

`-exclude`
: The exclude option is ignored.

-np
: If the user does not specify a `-np` option, then PBS uses the number of entries found in the $PBS_NODEFILE. The maximum number of ranks that can be launched is the number of entries in $PBS_NODEFILE.

10.6.11.2 Examples

Example 1: Run a single-executable MVAPICH1 job with 6 ranks spread out across the PBS-allocated hosts listed in $PBS_NODEFILE:

PBS_NODEFILE:

```
pbs-host1
pbs-host1
pbs-host2
pbs-host2
pbs-host3
pbs-host3
```

Job.script:

```
# mpirun runs 6 processes mapped to each host listed
# in $PBS_NODEFILE
mpirun -np 6 /path/myprog
```

Run job script:

```
qsub -l select=3:ncpus=2:mpiprocs=2 job.script
<job-id>
```

10.6.12 PBS Jobs with MVAPICH2's mpiexec

PBS provides an interface to MVAPICH2's mpiexec. MVAPICH2 allows the use of InfiniBand. If executed inside a PBS job, this allows for PBS to track all MVAPICH2 processes so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard MVAPICH2's mpiexec had been used.

You use the same "mpiexec" command as you would use outside of PBS.

The maximum number of ranks that can be launched is the number of entries in $PBS_NODEFILE.

10.6.12.1 Options

If executed inside a PBS job script, all of the options to the PBS interface are the same as `MVAPICH2's mpiexec` except for the following:

-host
: The host option is ignored.

-machinefile <file>
: The file option is ignored.

-mpdboot
: If mpdboot is not called before mpiexec, it is called automatically before `mpiexec` runs so that an MPD daemon is started on each host assigned by PBS.

10.6.12.2 MPD Startup and Shutdown

The interface ensures that the MPD daemons are started on each of the hosts listed in the `$PBS_NODEFILE`. It also ensures that the MPD daemons are shut down at the end of MPI job execution.

10.6.12.3 Examples

Example 1: Run a single-executable `MVAPICH2` job with 6 ranks on hosts listed in `$PBS_NODEFILE`:

PBS_NODEFILE:

```
pbs-host1
pbs-host2
pbs-host3
```

Job.script:

```
mpiexec -np 6 /path/mpiprog
```

Run job script:

```
qsub -l select=3:ncpus=2 job.script
<job-id>
```

Example 2: Launch an MVAPICH2 MPI job with multiple executables on multiple hosts listed in the default file "mpd.hosts". Here, run executables prog1 and prog2 with 2 ranks of prog1 on host1, 2 ranks of prog2 on host2 and 2 ranks of prog2 on host3 all specified on the command line.

PBS_NODEFILE:

```
pbs-host1
pbs-host2
pbs-host3
```

Job.script:

```
mpiexec -n 2 prog1 : -n 2  prog2 : -n 2  prog2
```

Run job script:

```
qsub -l select=3:ncpus=2 job.script
<job-id>
```

Example 3: Launch an MVAPICH2 MPI job with multiple executables on multiple hosts listed in the default file "mpd.hosts". Run executables prog1 and prog2 with 2 ranks of prog1 on host1, 2 ranks of prog2 on host2 and 2 ranks of prog2 on host3 all specified using the -configfile option.

PBS_NODEFILE:

```
pbs-host1
pbs-host2
pbs-host3
```

Job.script:

```
echo "-n 2 -host host1 prog1" > /tmp/jobconf
echo "-n 2 -host host2 prog2" >> /tmp/jobconf
echo "-n 2 -host host3 prog2" >> /tmp/jobconf
mpiexec -configfile /tmp/jobconf
rm /tmp/jobconf
```

Run job script:

```
qsub -l select=3:ncpus=2 job.script
<job-id>
```

10.6.13 PBS Jobs with HP MPI

In order to override the default rsh, set PBS_RSHCOMMAND in your job script:

```
export PBS_RSHCOMMAND=<rsh_cmd>
```

10.7 MPI Jobs on the Altix

10.7.1 Jobs on an Altix Running ProPack 4/5

PBS has its own mpiexec for the Altix running ProPack 4 or greater. The PBS mpiexec has the standard mpiexec interface. The PBS mpiexec does require proper configuration of the Altix. See your administrator to find out whether your system is configured for the PBS mpiexec.

You can launch an MPI job on a single Altix, or across multiple Altixes. PBS will manage and track the processes. You can use CSA, if it is configured, to collect accounting information on your jobs. PBS will run the MPI tasks in the cpusets it manages.

You can run MPI jobs in the placement sets chosen by PBS. When a job is finished, PBS will clean up after it.

For MPI jobs across multiple Altixes, PBS will manage the multihost jobs. For example, if you have two Altixes named Alt1 and Alt2, and want to run two applications called mympi1 and mympi2 on them, you can put this in your job script:

```
mpiexec -host Alt1 -n 4 mympi1 : -host Alt2 -n 8 mympi2
```

You can specify the name of the array to use via the PBS_MPI_SGIARRAY environment variable.

To verify how many CPUs are included in a cpuset created by PBS, use:

```
> $ cpuset -d <set name> | egrep cpus
```

This will work either from within a job or not.

The alt_id returned by MOM has the form `cpuset=<name>`. <name> is the name of the cpuset, which is the $PBS_JOBID.

Jobs will share cpusets if the jobs request sharing and the cpusets' sharing attribute is not set to force_excl. Jobs can share the memory on a node-board if they have a CPU from that nodeboard. To fit as many small jobs as possible onto vnodes that already have shared jobs on them, request sharing in the job resource requests.

PBS will try to put a job that will fit in a single nodeboard on just one node-board. However, if the only CPUs available are on separate nodeboards, and those vnodes are not allocated exclusively to existing jobs, and the job can share a vnode, then the job will be run on the separate nodeboards.

If a job is suspended, its processes will be moved to the global cpuset. When the job is restarted, they are restored.

10.8 PVM Jobs with PBS

On a typical system, to execute a Parallel Virtual Machine (PVM) program you can use the `pvmexec` command. The pvmexec command expects a "hostfile" argument for the list of hosts on which to spawn the parallel job.

For example, here is a sample PBS script for a PVM job:

```
#PBS -N pvmjob
#
pvmexec a.out -inputfile data_in
```

To start the PVM daemons on the hosts listed in $PBS_NODEFILE, start the PVM console on the first host in the list, and print the hosts to the standard output file named “jobname.o<PBS jobID>, use “echo conf | pvm $PBS_NODEFILE”. To quit the PVM console but leave the PVM daemons running, use “quit”. To stop the PVM daemons, restart the PVM console, and quit, use “echo halt | pvm”.

To submit a PVM job to PBS, use

```
qsub  your_pvm_job
```

Here is an example script for your_pvm_job:

```
#PBS -N pvmjob
#PBS -V
cd $PBS_O_WORKDIR
echo conf   | pvm $PBS_NODEFILE
echo quit | pvm
./my_pvm_program
echo halt | pvm
```

10.9 Checkpointing SGI MPI Jobs

10.9.1 Jobs on an Altix

Jobs are suspended on the Altix using the PBS suspend feature. Jobs are checkpointed on the Altix using application-level checkpointing. There is no OS-level checkpoint. Suspended or checkpointed jobs will resume on the original nodeboards.

Appendix A: PBS Environment Variables

Table 11-1: PBS Environment Variables

Variable	Meaning
NCPUS	Number of threads, defaulting to number of CPUs, on the vnode
OMP_NUM_THREADS	Same as NCPUS.
PBS_ARRAY_ID	Identifier for job arrays. Consists of sequence number.
PBS_ARRAY_INDEX	Index number of subjob in job array.
PBS_ENVIRONMENT	Indicates job type: **PBS_BATCH** or **PBS_INTERACTIVE**

Table 11-1: PBS Environment Variables

Variable	Meaning
PBS_JOBCOOKIE	Unique identifier for inter-MOM job-based communication.
PBS_JOBID	The job identifier assigned to the job or job array by the batch system.
PBS_JOBDIR	Pathname of job-specific staging and execution directory
PBS_JOBNAME	The job name supplied by the user.
PBS_MOMPORT	Port number on which this job's MOMs will communicate.
PBS_NODEFILE	The filename containing a list of vnodes assigned to the job.
PBS_NODENUM	Logical vnode number of this vnode allocated to the job.
PBS_O_HOME	Value of **HOME** from submission environment.
PBS_O_HOST	The host name on which the `qsub` command was executed.
PBS_O_LANG	Value of **LANG** from submission environment
PBS_O_LOGNAME	Value of **LOGNAME** from submission environment
PBS_O_MAIL	Value of **MAIL** from submission environment
PBS_O_PATH	Value of **PATH** from submission environment
PBS_O_QUEUE	The original queue name to which the job was submitted.
PBS_O_SHELL	Value of **SHELL** from submission environment

Table 11-1: PBS Environment Variables

Variable	Meaning
PBS_O_SYSTEM	The operating system name where `qsub` was executed.
PBS_O_TZ	Value of **TZ** from submission environment
PBS_O_WORKDIR	The absolute path of directory where `qsub` was executed.
PBS_QUEUE	The name of the queue from which the job is executed.
PBS_TASKNUM	The task (process) number for the job on this vnode.
TMPDIR	The job-specific temporary directory for this job.

Chapter 11 Appendix A: PBS Environment Variables

Appendix B: Converting From NQS to PBS

For those converting to PBS from NQS or NQE, PBS includes a utility called **nqs2pbs** which converts an existing NQS job script so that it will work with PBS. (In fact, the resulting script will be valid to both NQS and PBS.) The existing script is copied and PBS directives ("#PBS") are inserted prior to each NQS directive (either "#QSUB" or "#Q$") in the original script.

```
nqs2pbs existing-NQS-script new-PBS-script
```

Section "Setting Up Your UNIX/Linux Environment" on page 25 discusses PBS environment variables.

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

12.1 Converting Date Specifications

Converting NQS date specifications to the PBS form may result in a warning message and an incomplete converted date. PBS does not support date specifications of “today”, “tomorrow”, or the name of the days of the week such as “Monday”. If any of these are encountered in a script, the PBS specification will contain only the time portion of the NQS specification (i.e. `#PBS -a hhmm[.ss]`). It is suggested that you specify the execution time on the `qsub` command line rather than in the script. All times are taken as local time. If any unrecognizable NQS directives are encountered, an error message is displayed. The new PBS script will be deleted if any errors occur.

Appendix C: License Agreement

Altair Engineering, Inc.

Software License Agreement

This License Agreement is a legal agreement between Altair Engineering, Inc. (“Altair”) and you (“Licensee”) governing the terms of use of the Altair Software. Before you may download or use the Software, your consent to the following terms and conditions is required by clicking on the 'I Accept” button. If you do not have the authority to bind your organization to these terms and conditions, you must click on the button that states “I do not accept” and then have an authorized party in your organization consent to these terms. In the event that your organization and Altair have a master software license agreement, mutually agreed upon in writing, in place at the time of your execution of this agreement, the terms of the master agreement shall govern.

Appendix C: License Agreement

1. DEFINITIONS. In addition to terms defined elsewhere in this Agreement, the following terms shall have the meanings defined below for purposes of this Agreement:

Documentation. Documentation provided by Altair on any media for use with the Software.

Execute. To load Software into a computer's RAM or other primary memory for execution by the computer.

Global Zone: Software is licensed based on three Global Zones: the Americas, Europe and Asia-Pacific. When Licensee has Licensed Workstations located in multiple Global Zones, which are connected to a single License (Network) Server, a premium is applied to the standard Software License pricing for a single Global Zone.

License Log File. A computer file providing usage information on the Software as gathered by the Software.

License Management System. The license management system that accompanies the Software and limits its use in accordance with the usage permitted under this Agreement, and which includes a License Log File.

L**icense (Network) Server**. A network file server that Licensee owns or leases located on Licensee's premises and identified by machine serial number on the Order Form.

License Units. A parameter used by the License Management System to determine the usage of the Software permitted under this Agreement at any one time.

Licensed Workstations. Single-user computers located in the same Global Zone(s) that Licensee owns or leases that are connected to the License (Network) Server via local area network or Licensee's private wide-area network.

Maintenance Release. Any release of the Software made generally available by Altair to its Licensees with annual leases, or those with perpetual licenses who have an active maintenance agreement in effect, that corrects programming errors or makes other minor changes to the Software. The fees for maintenance and support services are included in the annual license fee but perpetual licenses require a separate fee.

Order Form. Altair's standard form in either hard copy or electronic format that contains the specific parameters (such as identifying Licensee's contracting office, License Fees, Software, Support, and License (Network) Servers) of the transaction governed by this Agreement.

Proprietary Rights Notices. Patent, copyright, trademark or other proprietary rights notices applied to the Software, Documentation or the packaging or media of same.

Software. The software identified in the Order Form and any Updates or Maintenance Releases.

Suppliers. Any person, corporation or other legal entity which may provide software or documents which are included in the Software.

Support. The maintenance and support services provided by Altair pursuant to this Agreement.

Templates. Human readable ASCII files containing machine-interpretable commands for use with the Software.

Term. The initial term of this Agreement or any renewal term. Annual licenses shall have a 12-month term of use. Paid-up, or perpetual licenses, shall have a term of twenty-five years.

Update. A new version of the Software made generally available by Altair to its Licensee that includes additional features or functionalities but is substantially the same computer code as the existing Software.

2. **PAYMENT**. Licensee shall pay in full the fee for licensed Software and Support within thirty (30) days of receipt of the invoice. Past due fees shall bear interest at the maximum legal rate. Altair may condition its delivery of any Maintenance Release or Update to Licensee on Licensee's having paid all amounts then owed to Altair. Fees do not include taxes or duties and Licensee is responsible for paying (or for reimbursing Altair if Altair is required to pay) any federal, state or local taxes, or duties imposed on this License or the possession or use by Licensee of the Software excluding, however, all taxes on or measured by Altair's net income. Altair shall be entitled to its reasonable costs of collection (including attorneys fees and interest) if license fees are not paid to it on a timely basis.

3. **TERM**. Unless terminated earlier in accordance with the provisions of this Agreement, this Agreement will be in force for a period as stated on the Order Form. For annual licenses or Support provided for perpetual licenses, renewal shall be automatic for a successive year (“Renewal

Term"), upon mutual written execution of a new Order Form. All charges and fees for each Renewal Term shall be set forth in the Order Form executed for each Renewal Term. All Software procured by Licensee may be made coterminous at the request of Licensee and the consent of Altair.

4. **LICENSE GRANT**. Subject to the terms and conditions set forth in this Agreement, Altair hereby grants Licensee, and Licensee hereby accepts, a limited, non-exclusive, non-transferable license to: a) install the Software on the License (Network) Server(s) identified on the Order Form for use only at the sites identified on the Order Form; b) execute the Software on Licensed Workstations in accordance with the License Management System for use solely by Licensee's employees or its onsite Contractors who have agreed to be bound by the terms of this Agreement, for Licensee's internal business use on Licensed Workstations within the Global Zone(s) as identified on the Order Form and for the term identified on the Order Form; c) make backup copies of the Software, provided that Altair's Proprietary Rights Notices are reproduced on each such backup copy; d) freely modify and use Templates, provided that such modifications shall not be subject to Altair's warranties, indemnities, support or other Altair obligations under this Agreement; and e) copy and distribute Documentation inside Licensee's organization exclusively for use by Licensee's employees. A copy of the License Log File shall be made available to Altair automatically on no less than a monthly basis. In the event that Licensee uses a third party vendor to provide itself with information technology (IT) support, the IT company shall be permitted to access the Software only upon its agreement to abide by the terms of this Agreement. Licensee shall indemnify, defend and hold harmless Altair for the actions of its IT vendor(s).

5. **RESTRICTIONS ON USE**. Notwithstanding the foregoing license grant, Licensee shall not do (or allow others to do) any of the following: a) install, use, copy, modify, merge, or transfer copies of the Software or Documentation, except as expressly authorized in this Agreement; b) use any back-up copies of the Software for any purpose other than to replace the original copy provided by Altair in the event it is destroyed or damaged; c) disassemble, decompile or "unlock", reverse translate, reverse engineer, or in any manner decode the Software for any reason; d) sublicense, sell, lend, assign, rent, distribute, publicly display or publicly perform the Software or Documentation or Licensee's rights under this Agreement; e) allow use outside the Global Zone(s) or User Sites identified on the Order Form; f) allow third parties to access or use the Software, such as through a service bureau, wide area network, Internet location or

time-sharing arrangement except as expressly provided in Section 4(b); g) remove any Proprietary Rights Notices from the Software; h) disable or circumvent the License Management System provided with the Software; or (i) develop, test or support software of Licensee or third parties.

6. OWNERSHIP AND CONFIDENTIALITY. Licensee acknowledges that all applicable rights in patents, copyrights, trademarks, service marks, and trade secrets embodied in the Software and Documentation are owned by Altair and/or its Suppliers. Licensee further acknowledges that the Software and Documentation, and all copies thereof, are and shall remain the sole and exclusive property of Altair and/or its Suppliers. This Agreement is a license and not a sale of the Software. Altair retains all rights in the Software and Documentation not expressly granted to Licensee herein. Licensee acknowledges that the Software and accompanying Documentation are confidential and constitute valuable assets and trade secrets of Altair and/or its Suppliers. Licensee agrees to take the precautions necessary to protect and maintain the confidentiality of the Software and Documentation, and shall not disclose or make them available to any person or entity except as expressly provided in this Agreement. Licensee shall promptly notify Altair in the event any unauthorized person obtains access to the Software. If Licensee is required by any governmental authority or court of law to disclose Altair's confidential information, then Licensee shall immediately notify Altair before making such disclosure so that Altair may seek a protective order or other appropriate relief. Licensee's obligations set forth in Section 5 and Section 6 of this Agreement shall survive termination of this Agreement for any reason. Altair's Suppliers, as third party beneficiaries, shall be entitled to enforce the terms of this Agreement directly against Licensee as necessary to protect Supplier's intellectual property or other rights. Altair shall keep confidential all Licensee information provided to Altair in order that Altair may provide Support to Licensee shall be kept confidential and used only for the purpose of assisting Licensee in its use of the licensed Software.

7. MAINTENANCE AND SUPPORT. Maintenance. Altair will provide Licensee at no additional charge for annual licenses, and for a fee for paid-up licenses, with any Maintenance Releases and Updates of the Software or Documentation that are generally released by Altair during the term of this Agreement, except that this shall not apply to any Renewal Term for which full payment has not been received. Altair does not promise that there will be a certain number of Updates (or any Updates) during a particular year. If there is any question or dispute as to whether a particular release is a Maintenance Release, an Update or a new product, the categori-

zation of the release as determined by Altair shall be final. Licensee must install Maintenance Releases and Updates promptly after receipt from Altair. Maintenance Releases and Updates are Software subject to this Agreement. Altair shall only be obligated to provide support and maintenance for the most current release of the Software and its most recent prior release Support. Altair will provide support via telephone and email to Licensee at the fees, if any, as listed on the Order Form.. If Support has not been procured for any period of time for paid-up licenses, a reinstatement fee shall apply. Support consists of responses to questions from Licensee's personnel related to the use of the then-current and most recent prior release version of the Software. Licensee agrees to provide Altair will sufficient information to resolve technical issues as may be reasonably requested by Altair. Licensee agrees to the best of its abilities to read, comprehend and follow operating instructions and procedures as specified in, but not limited to, Altair's Documentation and other correspondence related to the Software, and to follow procedures and recommendations provided by Altair in an effort to correct problems. Licensee also agrees to notify Altair of a programming error, malfunction and other problems in accordance with Altair's then current problem reporting procedure. If Altair believes that a problem reported by Licensee may not be due to an error in the Software, Altair will so notify Licensee. Questions must be directed to Altair's specially designated telephone support numbers and email addresses. Support will also be available via email at Internet addresses designated by Altair. Support is available Monday through Friday (excluding holidays) from 8:00 a.m. to 5:00 p.m local time in the Global Zone where licensed. Exclusions. Altair shall have no obligation to maintain or support (a) altered, damaged or Licensee-modified Software, or any portion of the Software incorporated with or into other software; (b) any version of the Software other than the current version of the Software or the immediately previous version; (c) Software problems causes by Licensee's negligence, abuse or misapplication of Software other than as specified in the Documentation, or other causes beyond the reasonable control of Altair; or (d) Software installed on any hardware, operating system version or network environment that is not supported by Altair. Support also excludes configuration of hardware, non Altair Software, and networking services; consulting services; general solution provider related services; and general computer system maintenance.

8. **WARRANTY AND DISCLAIMER**. Altair warrants for a period of ninety (90) days after Licensee initially receives the Software that the Software will perform under normal use substantially as described in then

current Documentation and this Agreement. Supplier software included in the Software and provided to Licensee shall be warranted as stated by the Supplier. Copies of the Suppliers' terms and conditions for software are available on the Altair Support website. Support services shall be provided in a workmanlike and professional manner, in accordance with the prevailing standard of care for consulting support engineers at the time and place the services are performed.

ALTAIR DOES NOT REPRESENT OR WARRANT THAT THE SOFTWARE WILL MEET LICENSEE'S REQUIREMENTS OR THAT ITS OPERATION WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT IT WILL BE COMPATIBLE WITH ANY PARTICULAR HARDWARE OR SOFTWARE. ALTAIR EXCLUDES AND DIS**CLAIMS ALL EXPRESS AND IMPLIED WARRANTIES NOT STATED HEREIN, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT. THE ENTIRE RISK FOR THE PERFORMANCE, NON-PERFORMANCE OR RESULTS OBTAINED FROM USE OF THE SOFTWARE RESTS WITH LICENSEE AND NOT ALTAIR. ALTAIR MAKES NO WARRANTIES WITH RESPECT TO THE ACCURACY, COMPLETENESS, FUNCTIONALITY, SAFETY, PERFORMANCE, OR ANY OTHER ASPECT OF ANY DESIGN, PROTOTYPE OR FINAL PRODUCT DEVELOPED BY LICENSEE USING THE SOFTWARE**.

9. **INDEMNITY**. Altair will defend, at its expense, any claim made against Licensee based on an allegation that the Software infringes a patent or copyright (“Claim”); provided, however, that this indemnification does not include claims based on Supplier software, and that Licensee has not materially breached the terms of this Agreement, Licensee notifies Altair in writing within ten (10) days after Licensee first learns of the Claim; and Licensee cooperates fully in the defense of the claim. Altair shall have sole control over such defense; provided, however, that it may not enter into any settlement license binding upon Licensee without Licensee's consent, which shall not be unreasonably withheld. If a Claim is made, Altair may modify the Software to avoid the alleged infringement, provided, however, that such modifications do not materially diminish the Software's functionality. If such modifications are not commercially reasonably or technically possible, Altair may terminate this Agreement and refund to Licensee the prorated license fee that Licensee paid for the then current Term. Perpetual licenses shall be pro-rated over a 36-month term. Altair shall have no obligation under this Section 9, however, if the alleged

infringement arises from Altair's compliance with specifications or instructions prescribed by Licensee, modification of the Software by Licensee, use of the Software in combination with other software not provided by Altair and which use is not specifically described in the Documentation and if Licenses is not using the most current version of the Software, if such alleged infringement would not have occurred except for such exclusions listed here. This section 9 states Altair's entire liability to Licensee in the event a Claim is made.

10. LIMITATION OF REMEDIES AND LIABILITY. Licensee's exclusive remedy (and Altair's sole liability) for Software that does not meet the warranty set forth in Section 8 shall be, at Altair's option, either (i) to correct the nonconforming Software within a reasonable time so that it conforms to the warranty; or (ii) to terminate this Agreement and refund to Licensee the license fees that Licensee has paid for the then current Term for the nonconforming Software; provided, however that Licensee notifies Altair of the problem in writing within the applicable Warranty Period when the problem first occurs. Any corrected Software shall be warranted in accordance with Section 8 for ninety (90) days after delivery to Licensee. The warranties hereunder are void if the Software has been misused or improperly installed, or if Licensee has violated the terms of this Agreement.

Altair's entire liability for all claims arising under or related in any way to this Agreement (regardless of legal theory), except as provided in Section 9, shall be limited to direct damages, and shall not exceed, in the aggregate for all claims, the license and maintenance fees paid under this Agreement by Licensee in the 12 months prior to the claim on a prorated basis. ALTAIR AND ITS SUPPLIERS SHALL NOT BE LIABLE TO LICENSEE OR ANYONE ELSE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING HEREUNDER (INCLUDING LOSS OF PROFITS OR DATA, DEFECTS IN DESIGN OR PRODUCTS CREATED USING THE SOFTWARE, OR ANY INJURY OR DAMAGE RESULTING FROM SUCH DEFECTS, SUFFERED BY LICENSEE OR ANY THIRD PARTY) EVEN IF ALTAIR OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Licensee acknowledges that it is solely responsible for the adequacy and accuracy of the input of data, including the output generated from such data, and agrees to defend, indemnify, and hold harmless Altair and its Suppliers from any and all claims, including reasonable attorney's fees, resulting from, or in connection with Licensee's use of the Software. No action, regardless of form, arising out of the transactions under

this Agreement may be brought by either party against the other more than two (2) years after the cause of action has accrued, except for actions related to unpaid fees.

11. TERMINATION. Either party may terminate this Agreement upon thirty (30) days prior written notice upon the occurrence of a default or material breach by the other party of its obligations under this Agreement (except for a breach by Altair of the warranty set forth in Section 8 for which a remedy is provided under Section 10; or a breach by Licensee of Section 5 or Section 6 for which no cure period is provided and Altair may terminate this Agreement immediately) if such default or breach continues for more than thirty (30) days after receipt of such notice. Upon termination of this Agreement, Licensee must cease using the Software and, at Altair's option, return all copies to Altair, or certify it has destroyed all such copies of the Software and Documentation.

12. UNITED STATES GOVERNMENT RESTRICTED RIGHTS. This section applies to all acquisitions of the Software by or for the United States government. By accepting delivery of the Software, the government hereby agrees that the Software qualifies as “commercial” computer software as that term is used in the acquisition regulations applicable to this procurement and that the government's use and disclosure of the Software is controlled by the terms and conditions of this Agreement to the maximum extent possible. This Agreement supersedes any contrary terms or conditions in any statement of work, contract, or other document that are not required by statute or regulation. If any provision of this Agreement is unacceptable to the government, Vendor may be contacted at Altair Engineering, Inc., 1820 E. Big Beaver Road, Troy, MI 48083-2031; telephone (248) 614-2400. If any provision of this Agreement violates applicable federal law or does not meet the government's actual, minimum needs, the government agrees to return the Software for a full refund.

For procurements governed by DFARS Part 227.72 (OCT 1998), HyperWorks Software is provided with only those rights specified in this Agreement in accordance with the Rights in Commercial Computer Software or Commercial Computer Software Documentation policy at DFARS 227.7202-3(a) (OCT 1998). For procurements other than for the Department of Defense, use, reproduction, or disclosure of the Software is subject to the restrictions set forth in this Agreement and in the Commercial Computer Software - Restricted Rights FAR clause 52.227-19 (June 1987) and any restrictions in successor regulations thereto.

Portions of Altair's PBS Professional Software and Documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subdivision(c)(1)(ii) of the rights in the Technical Data and Computer Software clause in DFARS 252.227-7013, or in subdivision (c)(1) and (2) of the Commercial Computer Software-Restricted Rights clause at 48 CFR52.227-19, as applicable.

13. CHOICE OF LAW AND VENUE. This Agreement shall be governed by and construed under the laws of the state of Michigan, without regard to that state's conflict of laws principles except if the state of Michigan adopts the Uniform Computer Information Transactions Act drafted by the National Conference of Commissioners of Uniform State Laws as revised or amended as of June 30, 2002 ("UCITA") which is specifically excluded. This Agreement shall not be governed by the United Nations Convention on Contracts for the International Sale of Goods, the application of which is expressly excluded. Each Party waives its right to a jury trial in the event of any dispute arising under or relating to this Agreement. Each party agrees that money damages may not be an adequate remedy for breach of the provisions of this Agreement, and in the event of such breach, the aggrieved party shall be entitled to seek specific performance and/or injunctive relief (without posting a bond or other security) in order to enforce or prevent any violation of this Agreement.

14. GENERAL PROVISIONS. **Export Controls**. Licensee acknowledges that the Software may be subject to the export control laws and regulations of the United States and any amendments thereof. Licensee agrees that Licensee will not directly or indirectly export the Software into any country or use the Software in any manner except in compliance with all applicable U.S. export laws and regulations. **Notice**. All notices given by one party to the other under this Agreement shall be sent by certified mail, return receipt requested, or by overnight courier, to the respective addresses set forth in this Agreement or to such other address either party has specified in writing to the other. All notices shall be deemed given when actually received. **Assignment**. Neither party shall assign this Agreement without the prior written consent of other party, which shall not be unreasonably withheld. All terms and conditions of this Agreement shall be binding upon and inure to the benefit of the parties hereto and their respective successors and permitted assigns. **Waiver**. The failure of a party to enforce at any time any of the provisions of this Agreement shall not be construed to be a waiver of the right of the party thereafter to enforce any such provisions. **Severability**. If any provision of this Agreement is found

void and unenforceable, such provision shall be interpreted so as to best accomplish the intent of the parties within the limits of applicable law, and all remaining provisions shall continue to be valid and enforceable. **Headings**. The section headings contained in this Agreement are for convenience only and shall not be of any effect in constructing the meanings of the Sections. **Modification**. No change or modification of this Agreement will be valid unless it is in writing and is signed by a duly authorized representative of each party. **Conflict**. In the event of any conflict between the terms of this Agreement and any terms and conditions on a Purchase Order or comparable document, the terms of this Agreement shall prevail. Moreover, each party agrees any additional terms on any Purchase Order other than the transaction items of (a) item(s) ordered; (b) pricing; (c) quantity; (d) delivery instructions and (e) invoicing directions, are not binding on the parties. **Entire Agreement**. This Agreement and the Order Form(s) constitute the entire understanding between the parties related to the subject matter hereto, and supersedes all proposals or prior agreements, whether written or oral, and all other communications between the parties with respect to such subject matter. This Agreement may be executed in one or more counterparts, all of which together shall constitute one and the same instrument.

Index

Q

R

S

X