



Sun N1 Grid Engine 6.1 User's Guide



Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 820-0699
May 2007

Copyright 2007 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more U.S. patents or pending patent applications in the U.S. and in other countries.

U.S. Government Rights – Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

This distribution may include materials developed by third parties.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, the Solaris logo, the Java Coffee Cup logo, docs.sun.com, N1 Java, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and SunTM Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

Products covered by and information contained in this publication are controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical or biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists is strictly prohibited.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 2007 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. Tous droits réservés.

Sun Microsystems, Inc. détient les droits de propriété intellectuelle relatifs à la technologie incorporée dans le produit qui est décrit dans ce document. En particulier, et ce sans limitation, ces droits de propriété intellectuelle peuvent inclure un ou plusieurs brevets américains ou des applications de brevet en attente aux Etats-Unis et dans d'autres pays.

Cette distribution peut comprendre des composants développés par des tierces personnes.

Certains composants de ce produit peuvent être dérivées du logiciel Berkeley BSD, licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays; elle est licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, le logo Solaris, le logo Java Coffee Cup, docs.sun.com, N1 Java et Solaris sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui, en outre, se conforment aux licences écrites de Sun.

Les produits qui font l'objet de cette publication et les informations qu'il contient sont régis par la législation américaine en matière de contrôle des exportations et peuvent être soumis au droit d'autres pays dans le domaine des exportations et importations. Les utilisations finales, ou utilisateurs finaux, pour des armes nucléaires, des missiles, des armes chimiques ou biologiques ou pour le nucléaire maritime, directement ou indirectement, sont strictement interdites. Les exportations ou réexportations vers des pays sous embargo des Etats-Unis, ou vers des entités figurant sur les listes d'exclusion d'exportation américaines, y compris, mais de manière non exclusive, la liste de personnes qui font objet d'un ordre de ne pas participer, d'une façon directe ou indirecte, aux exportations des produits ou des services qui sont régis par la législation américaine en matière de contrôle des exportations et la liste de ressortissants spécifiquement désignés, sont rigoureusement interdites.

LA DOCUMENTATION EST FOURNIE "EN L'ETAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFACON.

Contents

Preface	15
1 Introduction to the N1 Grid Engine 6.1 Software	19
What Is Grid Computing?	19
Managing Workload by Managing Resources and Policies	22
How the System Operates	23
Matching Resources to Requests	23
Jobs and Queues	24
Usage Policies	24
Grid Engine System Components	26
Hosts	27
Daemons	27
Queues	28
Client Commands	29
QMON, the Grid Engine System's Graphical User Interface	30
2 Navigating the Grid Engine System	33
QMON Main Control Window	33
Launching the QMON Main Control Window	33
Customizing QMON	34
Users and User Categories	35
User Access Permissions	36
Managers, Operators, and Owners	38
Displaying Queues and Queue Properties	38
Displaying a List of Queues	38
Displaying Queue Properties	38
▼ How to Display Queue Properties With QMON	39

Interpreting Queue Property Information	40
Hosts and Host Functionality	41
Finding the Name of the Master Host	41
Displaying a List of Execution Hosts	42
Displaying a List of Administration Hosts	42
Displaying a List of Submit Hosts	42
Requestable Attributes	42
Displaying a List of Requestable Attributes	44
3 Submitting Jobs	47
Submitting a Simple Job	47
▼ How To Submit a Simple Job From the Command Line	48
▼ How To Submit a Simple Job With QMON	49
Submitting Batch Jobs	53
About Shell Scripts	53
Example of a Shell Script	54
Extensions to Regular Shell Scripts	55
Submitting Extended Jobs and Advanced Jobs	59
Submitting Extended Jobs With QMON	59
Submitting Extended Jobs From the Command Line	63
Submitting Advanced Jobs With QMON	63
Submitting Advanced Jobs From the Command Line	66
Defining Resource Requirements	68
Job Dependencies	71
Submitting Array Jobs	71
Submitting Interactive Jobs	73
Submitting Interactive Jobs With QMON	74
Submitting Interactive Jobs With qsh	75
Submitting Interactive Jobs With qlogin	75
Transparent Remote Execution	76
Remote Execution With qcrsh	76
Transparent Job Distribution With qtcsh	77
Parallel Makefile Processing With qmake	79
How Jobs Are Scheduled	82
Job Priorities	82

Ticket Policies	83
Queue Selection	83
4 Monitoring and Controlling Jobs and Queues	85
Monitoring and Controlling Jobs	85
Monitoring and Controlling Jobs With QMON	85
Monitoring and Controlling Jobs From the Command Line	94
Monitoring Jobs by Email	98
Monitoring and Controlling Queues	98
Monitoring and Controlling Queues With QMON	98
Controlling Queues With qmod	105
Using Job Checkpointing	106
User-Level Checkpointing	106
Kernel-Level Checkpointing	106
Migrating Checkpointing Jobs	107
Composing a Checkpointing Job Script	107
File System Requirements for Checkpointing	109
5 Accounting and Reporting	111
Starting the Accounting and Reporting Console	111
▼ How to Start the Accounting and Reporting Console	111
Creating and Running Simple Queries	113
▼ How to Create a Simple Query	113
▼ How to Create a View Configuration	117
Defining Data Series for Diagrams	119
▼ How to Run a Simple Query	123
▼ How to Edit a Simple Query	123
Creating and Running Advanced Queries	124
▼ How to Create an Advanced Query	124
▼ How to Run an Advanced Query	125
▼ How to Edit an Advanced Query	125
Latebindings for Advanced Queries	126

6 Automating Grid Engine Functions Through the Distributed Resource Management Application API	127
Introduction to Distributed Resource Management Application API (DRMAA)	127
Developing with the C Language Binding	128
Important Files for the C Language Binding	128
Including the DRMAA Header File	128
Compiling Your C Application	128
Running Your C Application	129
▼ How to Use the DRMAA 0.95 C Language Binding	129
C Application Examples	130
Developing with the Java Language Binding	134
Important Files for the Java Language Binding	134
Importing the DRMAA Java Classes and Packages	134
Compiling Your Java Application	134
▼ How to Use DRMAA with NetBeans 5.x	135
Running Your Java Application	136
Using the DRMAA 0.5 Java Language Binding	136
Java Application Examples	137
7 Error Messages, and Troubleshooting	141
How the Software Retrieves Error Reports	141
Consequences of Different Error or Exit Codes	142
Running Grid Engine System Programs in Debug Mode	145
Diagnosing Problems	147
Pending Jobs Not Being Dispatched	147
Job or Queue Reported in Error State E	148
Troubleshooting Common Problems	149
Typical Accounting and Reporting Console Errors	153
A Database Schemas	157
Schema Tables	157
sge_job	157
sge_job_usage	158
sge_job_request	160
sge_job_log	161

sgc_share_log	161
sgc_host	162
sgc_host_values	163
sgc_queue	163
sgc_queue_values	164
sgc_department	164
sgc_department_values	164
sgc_project	165
sgc_project_values	165
sgc_user	166
sgc_user_values	166
sgc_group	166
sgc_group_values	167
List of Predefined Views	167
view_accounting	167
view_job_times	168
view_jobs_completed	169
view_job_log	169
view_department_values	170
view_group_values	170
view_host_values	171
view_project_values	171
view_queue_values	172
view_user_values	172
List of Derived Values	173
 Glossary	 175
 Index	 179

Figures

FIGURE 1-1	Three Classes of Grids	21
FIGURE 1-2	Correlation Among Policies in a Grid Engine System	26
FIGURE 1-3	QMON Main Control Window, Defined	31
FIGURE 3-1	QMON Main Control Window	50
FIGURE 3-2	Submit Job Dialog Box	51
FIGURE 3-3	Job Control Dialog Box	52
FIGURE 3-4	Select a File Dialog Box	53
FIGURE 3-5	Extended Job Submission Example	62
FIGURE 3-6	Advanced Job Submission Example	66
FIGURE 3-7	Requested Resources Dialog Box	69
FIGURE 3-8	Interactive Submit Job Dialog Box, General Tab	74
FIGURE 3-9	Interactive Submit Job Dialog Box, Advanced Tab	75

Tables

TABLE 2-1	User Categories and Associated Command Capabilities	35
TABLE 7-1	Job-Related Error or Exit Codes	142
TABLE 7-2	Parallel-Environment-Related Error or Exit Codes	143
TABLE 7-3	Queue-Related Error or Exit Codes	143
TABLE 7-4	Checkpointing-Related Error or Exit Codes	143
TABLE 7-5	qacct -j failed Field Codes	144

Examples

EXAMPLE 2-1	Queue Property Information	39
EXAMPLE 2-2	Complex Attributes Displayed	44
EXAMPLE 3-1	Simple Shell Script	54
EXAMPLE 3-2	Using Script-Embedded Command Line Options	56
EXAMPLE 4-1	Example of qstat -f Output	96
EXAMPLE 4-2	Example of qstat Output	97
EXAMPLE 4-3	Example of a Checkpointing Job Script	107
EXAMPLE 5-1	Accounting per Department Pie Chart	120
EXAMPLE 5-2	CPU, Input/Output, and Memory Usage Over All Departments Bar Chart ...	121
EXAMPLE 5-3	Latebindings Examples	126
EXAMPLE 6-1	Compiling Your C Application Using Sun Studio Compiler	129
EXAMPLE 6-2	Starting and Stopping a Session	130
EXAMPLE 6-3	Running a Job	132
EXAMPLE 6-4	Starting and Stopping a Session	137
EXAMPLE 6-5	Running a Job	138

Preface

The *Sun N1 Grid Engine 6.1 User's Guide* includes the following:

- A description of the primary role of N1 Grid Engine 6.1 software in complex computing environments
- A description of the major components of the product, along with definitions of the functions of each
- A glossary of terms that are important to know in an N1 Grid Engine 6.1 software environment

Who Should Use This Book

This manual is for engineers and technical professionals, who need to use the N1 Grid Engine 6.1 software. Also, you should understand the concepts in this book if you are responsible for administering the system of networked computer hosts that run the N1 Grid Engine 6.1 software.

How This Book Is Organized

[Chapter 1](#) describes the concepts and major components of the N1 Grid Engine 6.1 software. This chapter also includes a summary of user commands, and introduces the QMON graphical user interface.

[Chapter 2](#) describes how to display information about components of the system of networked computer hosts that run the N1 Grid Engine 6.1 software such as users, queues, hosts, and job attributes.

[Chapter 3](#) provides information about how to submit jobs for processing.

[Chapter 4](#) provides information about how to monitor and control jobs and queues. The chapter also includes information about job checkpointing. .

[Chapter 5](#) describes how to use the accounting and reporting console.

[Chapter 6](#) explains how to automate N1 Grid Engine functions through a C or Java-based DRMAA API.

[Chapter 7](#) contains common problems and their solutions.

[Appendix A](#) describes in detail the reporting database data model

[Glossary](#) is a list of product-specific words and phrases and their definitions.

Related Books

Other books in the N1 Grid Engine 6.1 software documentation collection include:

- *Sun N1 Grid Engine 6.1 Installation Guide*
- *Sun N1 Grid Engine 6.1 Administration Guide*
- *Sun N1 Grid Engine 6.1 Release Notes*

Documentation, Support, and Training

The Sun web site provides information about the following additional resources:

- [Documentation](http://www.sun.com/documentation/) (<http://www.sun.com/documentation/>)
- [Support](http://www.sun.com/support/) (<http://www.sun.com/support/>)
- [Training](http://www.sun.com/training/) (<http://www.sun.com/training/>)

Typographic Conventions

The following table describes the typographic conventions that are used in this book.

TABLE P-1 Typographic Conventions

Typeface	Meaning	Example
AaBbCc123	The names of commands, files, and directories, and onscreen computer output	Edit your <code>.login</code> file. Use <code>ls -a</code> to list all files. <code>machine_name%</code> you have mail.
AaBbCc123	What you type, contrasted with onscreen computer output	<code>machine_name%</code> su Password:
<i>aabbcc123</i>	Placeholder: replace with a real name or value	The command to remove a file is <i>rm filename</i> .

TABLE P-1 Typographic Conventions (Continued)

Typeface	Meaning	Example
<i>AaBbCc123</i>	Book titles, new terms, and terms to be emphasized	Read Chapter 6 in the <i>User's Guide</i> . <i>A cache</i> is a copy that is stored locally. Do <i>not</i> save the file. Note: Some emphasized items appear bold online.

Shell Prompts in Command Examples

The following table shows the default UNIX® system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P-2 Shell Prompts

Shell	Prompt
C shell	machine_name%
C shell for superuser	machine_name#
Bourne shell and Korn shell	\$
Bourne shell and Korn shell for superuser	#

Introduction to the N1™ Grid Engine 6.1 Software

This chapter provides background information about the system of networked computer hosts that run the N1 Grid Engine 6.1 software (also referred to as the *grid engine system*). This chapter includes the following topics:

- “What Is Grid Computing?” on page 19
- “Managing Workload by Managing Resources and Policies” on page 22
- “How the System Operates” on page 23
- “Grid Engine System Components” on page 26
- “Client Commands” on page 29
- “QMON, the Grid Engine System's Graphical User Interface” on page 30

You can also find a good overview of grid computing and the N1 Grid Engine product on the YouTube web site: [Introduction to Grid Engine](http://www.youtube.com/watch?v=0JBsMitNnQ8) (<http://www.youtube.com/watch?v=0JBsMitNnQ8>).

What Is Grid Computing?

A *grid* is a collection of computing resources that perform tasks. In its simplest form, a grid appears to users as a large system that provides a single point of access to powerful distributed resources. In its more complex form, which is explained later in this section, a grid can provide many access points to users. In all cases, users treat the grid as a *single* computational resource. Resource management software such as N1 Grid Engine 6.1 software (*grid engine software*) accepts jobs submitted by users. The software uses resource management policies to schedule jobs to be run on appropriate systems in the grid. Users can submit millions of jobs at a time without being concerned about where the jobs run.

No two grids are alike. One size does not fit all situations. The three key classes of grids, which scale from single systems to supercomputer-class compute farms that use thousands of processors, are as follows:

- *Cluster grids* are the simplest class. Cluster grids are made up of a set of computer *hosts* that work together. A cluster grid provides a single point of access to users in a single project or a single department.
- *Campus grids* enable multiple projects or departments within an organization to share computing resources. Organizations can use campus grids to handle a variety of tasks, from cyclical business processes to rendering, data mining, and more.
- *Global grids* are a collection of campus grids that cross organizational boundaries to create very large virtual systems. Users have access to compute power that far exceeds resources that are available within their own organization.

Figure 1–1 shows the three classes of grids. In the cluster grid, a user's job is handled by only one of the systems within the cluster. However, the user's cluster grid might be part of the more complex campus grid, and the campus grid might be part of the largest global grid. In such cases, the user's job can be handled by any member *execution host* that is located anywhere in the world.

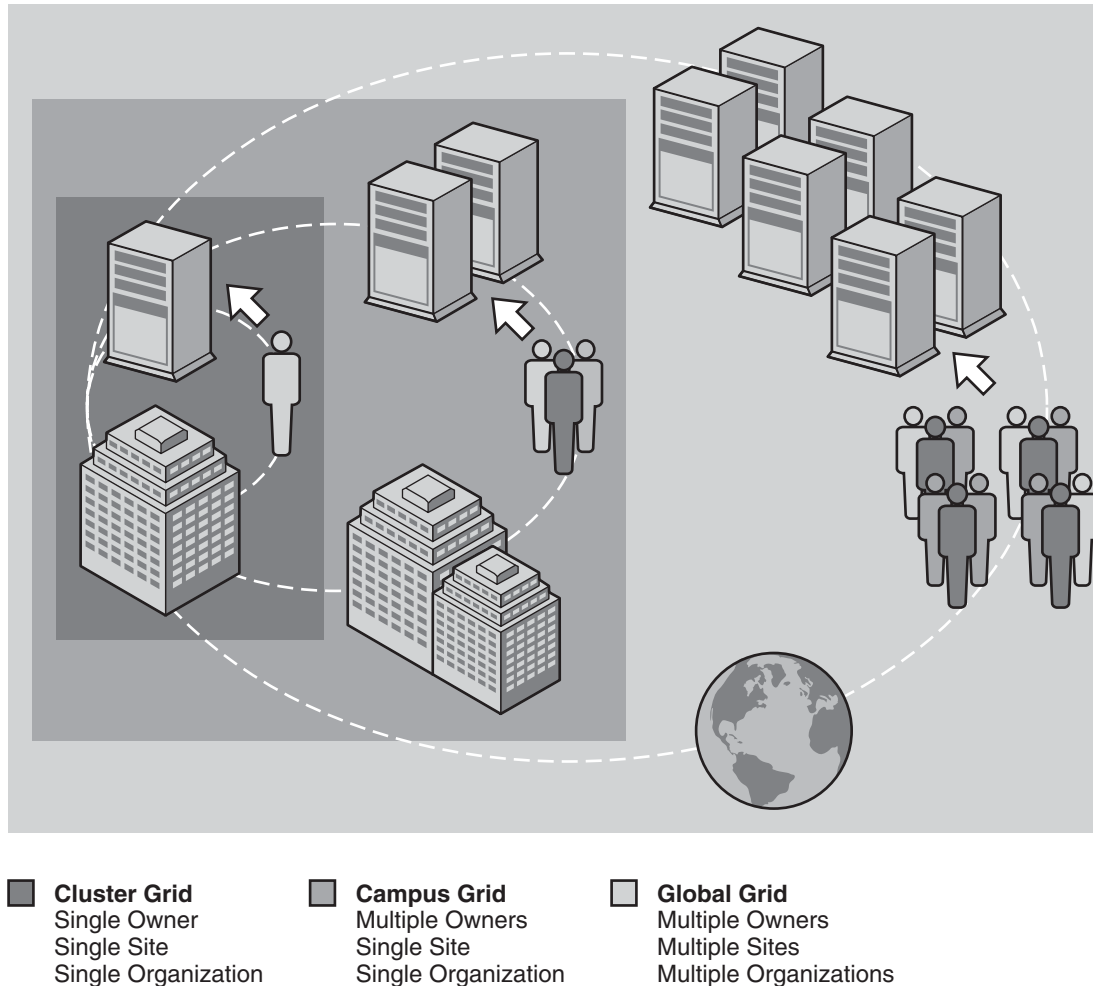


FIGURE 1-1 Three Classes of Grids

N1 Grid Engine 6.1 software provides the power and flexibility required for campus grids. The product is useful for existing cluster grids because it facilitates a smooth transition to creating a campus grid. The grid engine system effects this transition by consolidating all existing cluster grids on the campus. In addition, the grid engine system is a good start for an enterprise campus that is moving to the grid computing model for the first time.

The grid engine software orchestrates the delivery of computational power that is based on enterprise resource *policies* set by the organization's technical and management staff. The grid engine system uses these policies to examine the available computational resources within the campus grid. The system gathers these resources and then allocates and delivers resources automatically, optimizing usage across the campus grid.

To enable cooperation within the campus grid, project owners who use the grid must do the following:

- Negotiate policies
- Develop flexible policies for manual overrides for unique project requirements
- Automatically monitor and enforce the policies

The grid engine software can mediate among the entitlements of many departments and projects that are competing for computational resources.

Managing Workload by Managing Resources and Policies

The grid engine system is an advanced resource management tool for heterogeneous distributed computing environments. Workload management means that the use of shared resources is controlled to best achieve an enterprise's goals such as productivity, timeliness, level-of-service, and so forth. Workload management is accomplished through *managing resources* and *administering policies*. Sites configure the system to maximize usage and throughput, while the system supports varying levels of timeliness and importance. Job deadlines are instances of timeliness. Job priority and user share are instances of importance.

The grid engine software provides advanced resource management and policy administration for UNIX environments that are composed of multiple shared resources. The grid engine system is superior to standard *load management* tools with respect to the following major capabilities:

- Innovative dynamic scheduling and resource management that enables the grid engine software to enforce site-specific management policies.
- Dynamic collection of performance data to provide the scheduler with up-to-the-minute job-level resource consumption and system load information.
- Availability of enhanced security by way of *Certificate Security Protocol* (CSP)-based encryption. Instead of transferring messages in clear text, the messages in this more secure system are encrypted with a secret key.
- High-level policy administration for the definition and implementation of enterprise goals such as productivity, timeliness, and level-of-service.

The grid engine software provides users with the means to submit computationally demanding tasks to the grid for transparent distribution of the associated workload. Users can submit batch jobs, interactive jobs, and parallel jobs to the grid. For the administrator, the software provides comprehensive tools for monitoring and controlling jobs.

The product also supports checkpointing programs. Checkpointing jobs migrate from workstation to workstation without user intervention on load demand.

How the System Operates

The grid engine system does all of the following:

- Accepts *jobs* from the outside world. Jobs are users' requests for computer resources
- Puts jobs in a holding area until the jobs can be run
- Sends jobs from the holding area to an execution device
- Manages running jobs
- Logs the record of job execution when the jobs are finished

Matching Resources to Requests

As an analogy, imagine a large “money-center” bank in one of the world's capital cities. In the bank's lobby are dozens of customers waiting to be served. Each customer has different requirements. One customer wants to withdraw a small amount of money from his account. Arriving just after him is another customer, who has an appointment with one of the bank's investment specialists. She wants advice before she undertakes a complicated venture. Another customer in front of the first two customers wants to apply for a large loan, as do the eight customers in front of *her*.

Different customers with different needs require different types of service and different levels of service from the bank. Perhaps the bank on this particular day has many employees who can handle the one customer's simple withdrawal of money from his account. But at the same time the bank has only one or two loan officers available to help the many loan applicants. On another day, the situation might be reversed.

The effect is that customers must wait for service unnecessarily. Many of the customers could receive immediate service if only their needs were immediately recognized and then matched to available resources.

If the grid engine system were the bank manager, the service would be organized differently.

- On entering the bank lobby, customers would be asked to declare their name, their affiliations, and their service needs.
- Each customer's time of arrival would be recorded.
- Based on the information that the customers provided in the lobby, the bank would serve the following customers:
 - Customers whose needs match suitable and immediately available resources
 - Customers whose requirements have the highest priority
 - Customers who were waiting in the lobby for the longest time
- In a “grid engine system bank,” one bank employee might be able to help several customers at the same time. The grid engine system would try to assign new customers to the least-loaded and most-suitable bank employee.

- As bank manager, the grid engine system would allow the bank to define service policies. Typical service policies might be the following:
 - To provide preferential service to commercial customers because those customers generate more profit
 - To make sure a certain customer group is served well, because those customers have received bad service so far
 - To ensure that customers with an appointment get a timely response
 - To prefer a certain customer on direct demand of a bank executive
- Such policies would be implemented, monitored, and adjusted automatically by a grid engine system manager. Customers with preferential access would be served sooner. Such customers would receive more attention from employees, whose assistance those customers must share with other customers. The grid engine manager would recognize if the customers do not make progress. The manager would immediately respond by adjusting service levels in order to comply with the bank's service policies.

Jobs and Queues

In a grid engine system, *jobs* correspond to bank customers. Jobs wait in a computer holding area instead of a lobby. *Queues*, which provide services for jobs, correspond to bank employees. As in the case of bank customers, the requirements of each job, such as available memory, execution speed, available software licenses, and similar needs, can be very different. Only certain queues might be able to provide the corresponding service.

To continue the analogy, the grid engine software arbitrates available resources and job requirements in the following way:

- A user who submits a job through the grid engine system declares a requirement profile for the job. In addition, the system retrieves the identity of the user. The system also retrieves the user's affiliation with *projects* or *user groups*. The time that the user submitted the job is also stored.
- The moment that a queue is available to run a new job, the grid engine system determines what are the suitable jobs for the queue. The system immediately dispatches the job that has either the highest priority or the longest waiting time.
- Queues allow concurrent execution of many jobs. The grid engine system tries to start new jobs in the least-loaded and most-suitable queue.

Usage Policies

The administrator of a cluster can define high-level usage policies that are customized according to whatever is appropriate for the site. Four usage policies are available:

- **Urgency.** Using this policy, each job's priority is based on an urgency value. The urgency value is derived from the job's resource requirements, the job's deadline specification, and how long the job waits before it is run.
- **Functional.** Using this policy, an administrator can provide special treatment because of a user's or a job's affiliation with a certain user group, project, and so forth.
- **Share-based.** Under this policy, the level of service depends on an assigned share entitlement, the corresponding shares of other users and user groups, the past usage of resources by all users, and the current presence of users within the system.
- **Override.** This policy requires manual intervention by the cluster administrator, who modifies the automated policy implementation.

Policy management automatically controls the use of shared resources in the cluster to best achieve the goals of the administration. High-priority jobs are dispatched preferentially. Such jobs receive higher CPU entitlements if the jobs compete for resources with other jobs. The grid engine software monitors the progress of all jobs and adjusts their relative priorities correspondingly and with respect to the goals defined in the policies.

Using Tickets to Administer Policies

The functional, share-based, and override policies are defined through a grid engine system concept that is called *tickets*. You might compare tickets to shares of a public company's stock. The more stock shares that you own, the more important you are to the company. If shareholder A owns twice as many shares as shareholder B, A also has twice the votes of B. Therefore shareholder A is twice as important to the company. Similarly, the more tickets that a job has, the more important the job is. If job A has twice the tickets of job B, job A is entitled to twice the resource usage of job B.

Jobs can retrieve tickets from the functional, share-based, and override policies. The total number of tickets, as well as the number retrieved from each ticket policy, often changes over time.

The administrator controls the number of tickets that are allocated to each ticket policy in total. Just as ticket allocation does for jobs, this allocation determines the relative importance of the *ticket policies* among each other. Through the ticket pool that is assigned to particular ticket policies, the administration can run a grid engine system in different ways. For example, the system can run in a share-based mode only. Or the system can run in a combination of modes, for example, 90% share-based and 10% functional.

Using the Urgency Policy to Assign Job Priority

The urgency policy can be used in combination with two other job priority specifications:

- The number of tickets assigned by the functional, share-based, and override policies
- A priority value specified by the `qsub -p` command

A job can be assigned an urgency value, which is derived from three sources:

- The job's resource requirements
- The length of time a job must wait before the job runs
- The time at which a job must finish running, that is, the job's deadline

The administrator can separately weight the importance of each of these sources in order to arrive at a job's overall urgency value. For more information, see Chapter 5, “Managing Policies and the Scheduler,” in *Sun N1 Grid Engine 6.1 Administration Guide*.

Figure 1–2 shows the correlation among policies.

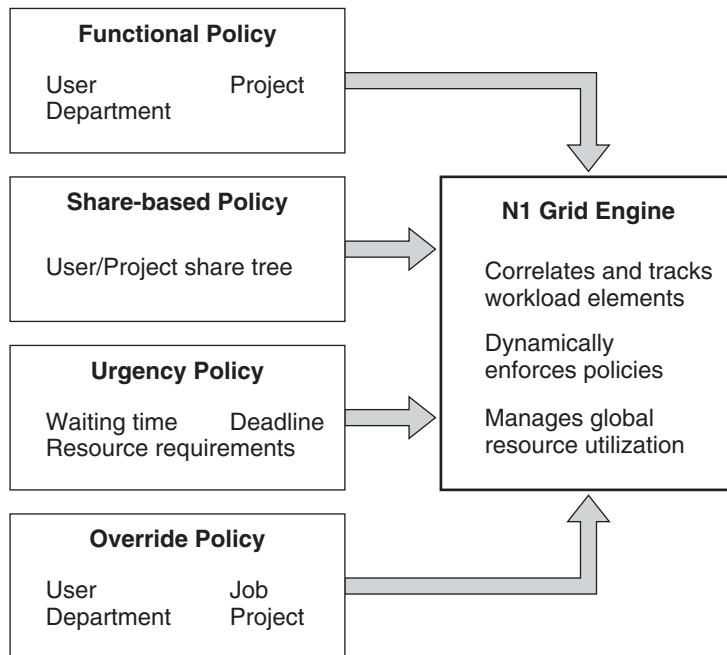


FIGURE 1–2 Correlation Among Policies in a Grid Engine System

Grid Engine System Components

The following sections explain the functions of the most important grid engine system components.

Hosts

Four types of hosts are fundamental to the grid engine system:

- Master host
- Execution hosts
- Administration hosts
- Submit hosts

Master Host

The master host is central to the overall cluster activity. The master host runs the master daemon `sge_qmaster` and the scheduler daemon `sge_schedd`. Both daemons control all grid engine system components, such as queues and jobs. The daemons maintain tables about the status of the components, user access permissions, and the like.

By default, the master host is also an administration host and a submit host.

Execution Hosts

Execution hosts are systems that have permission to execute jobs. Therefore execution hosts have queue instances attached to them. Execution hosts run the execution daemon `sge_execd`.

Administration Hosts

Administration hosts are hosts that have permission to carry out any kind of administrative activity for the grid engine system.

Submit Hosts

Submit hosts enable users to submit and control *batch jobs only*. In particular, a user who is logged in to a submit host can submit jobs with the `qsub` command, can monitor the job status with the `qstat` command, and can use the grid engine system OSF/1 Motif graphical user interface `QMON`, which is described in [“QMON, the Grid Engine System's Graphical User Interface” on page 30](#).

Note – A system can act as more than one type of host.

Daemons

Three daemons provide the functionality of the grid engine system.

sge_qmaster – The Master Daemon

The center of the cluster's management and scheduling activities, `sge_qmaster` maintains tables about hosts, queues, jobs, system load, and user permissions. `sge_qmaster` receives scheduling decisions from `sge_schedd` and requests actions from `sge_execd` on the appropriate execution hosts.

sge_schedd – The Scheduler Daemon

The scheduling daemon maintains an up-to-date view of the cluster's status with the help of `sge_qmaster`. The scheduling daemon makes the following scheduling decisions:

- Which jobs are dispatched to which queues
- How to reorder and reprioritize jobs to maintain share, priority, or deadline

The daemon then forwards these decisions to `sge_qmaster`, which initiates the required actions.

sge_execd – The Execution Daemon

The execution daemon is responsible for the queue instances on its host and for the running of jobs in these queue instances. Periodically, the execution daemon forwards information such as job status or load on its host to `sge_qmaster`.

Queues

A *queue* is a container for a class of jobs that are allowed to run on one or more hosts concurrently. A queue determines certain job attributes, for example, whether the job can be migrated. Throughout its lifetime, a running job is associated with its queue. Association with a queue affects some of the things that can happen to a job. For example, if a queue is suspended, all jobs associated with that queue are also suspended.

Jobs need not be submitted directly to a queue. You need to specify only the requirement profile of the job. A profile might include requirements such as memory, operating system, available software, and so forth. The grid engine software automatically dispatches the job to a suitable queue and a suitable host with a light execution load. If you submit a job to a specified queue, the job is bound to this queue. As a result, the grid engine system daemons are unable to select a better-suited device or a device that has a lighter load.

A queue can reside on a single host, or a queue can extend across multiple hosts. For this reason, grid engine system queues are also referred to as *cluster queues*. Cluster queues enable users and administrators to work with a cluster of execution hosts by means of a single queue configuration. Each host that is attached to a cluster queue receives its own *queue instance* from the cluster queue.

Client Commands

The command-line user interface is a set of ancillary programs (commands) that enable you to do the following tasks:

- Manage queues
- Submit and delete jobs
- Check job status
- Suspend or enable queues and jobs

The grid engine system provides the following set of ancillary programs.

- `qacct` – Extracts arbitrary accounting information from the cluster log file.
- `qalter` – Changes the attributes of submitted but pending jobs.
- `qconf` – Provides the user interface for cluster configuration and queue configuration.
- `qdel` – Provides the means for a user, operator, or manager to send signals to jobs or to subsets thereof.
- `qhold` – Holds back submitted jobs from execution.
- `ghost` – Displays status information about execution hosts.
- `qlogin` – Initiates a `telnet` or similar login session with automatic selection of a low-loaded, suitable host.
- `qmake` – A replacement for the standard UNIX `make` facility. `qmake` extends `make` by its ability to distribute independent make steps across a cluster of suitable machines.
- `qmod` – Enables the owner to suspend or enable a queue. All currently active processes that are associated with this queue are also signaled.
- `qmon` – Provides an X Windows Motif command interface and monitoring facility.
- `qresub` – Creates new jobs by copying running or pending jobs.
- `qrts` – Releases jobs from holds that were previously assigned to them, for example, through `qhold`.
- `qrsh` – Can be used for various purposes, such as the following:
 - To provide remote execution of interactive applications through the grid engine system. `qrsh` is comparable to the standard UNIX facility `rsh`.
 - To allow for the submission of batch jobs that, upon execution, support terminal I/O and terminal control. Terminal I/O includes standard output, standard error, and standard input.
 - To provide a submission client that remains active until the batch job finishes.
 - To allow for the grid engine software-controlled remote execution of the tasks of parallel jobs.

- `qselect` – Prints a list of queue names corresponding to specified selection criteria. The output of `qselect` is usually sent to other grid engine system commands to apply actions on a selected set of queues.
- `qsh` – Opens an interactive shell in an `xterm` on a lightly loaded host. Any kind of interactive jobs can be run in this shell.
- `qstat` – Provides a status listing of all jobs and queues associated with the cluster.
- `qsub` – The user interface for submitting batch jobs to the grid engine system.
- `qtcsh` – A fully compatible replacement for the widely known and used UNIX C shell (`csh`) derivative, `tcsh`. `qtcsh` provides a command shell with the extension of transparently distributing execution of designated applications to suitable and lightly loaded hosts through grid engine software.

QMON, the Grid Engine System's Graphical User Interface

You can use QMON, the graphical user interface (GUI) tool, to accomplish most grid engine system tasks. [Figure 1–3](#) shows the QMON Main Control window, which is often the starting point for both user and administrator functions. Each icon on the Main Control window is a GUI button that you click to start a variety of tasks. To see a button's name, which also describes its function, rest the pointer over the button.

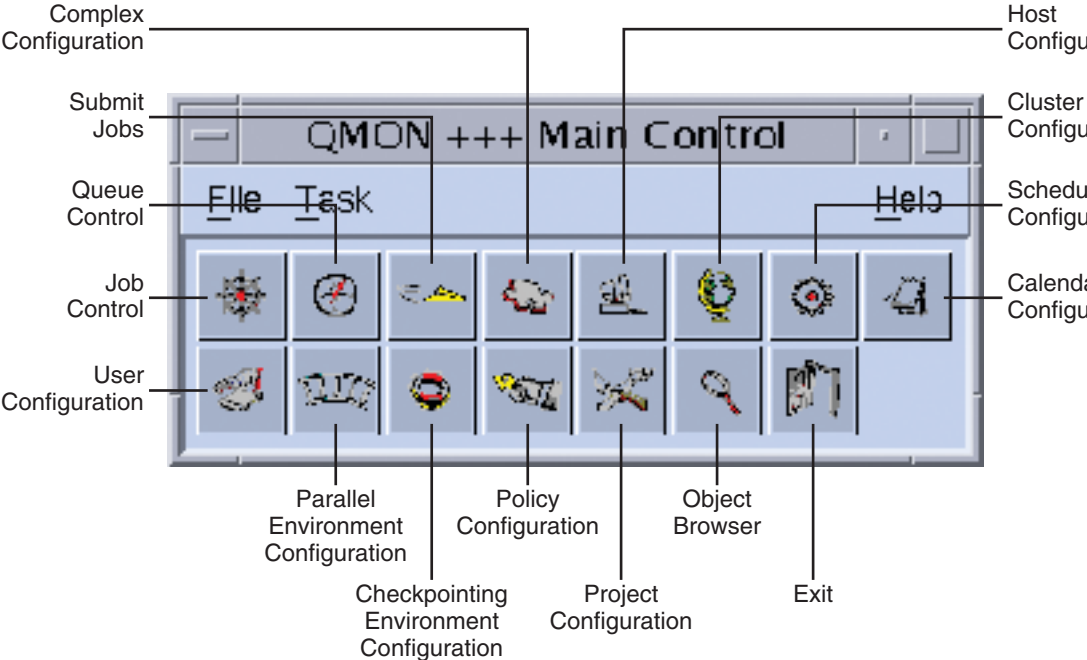


FIGURE 1-3 QMON Main Control Window, Defined

Navigating the Grid Engine System

This chapter describes how to display information about grid engine system components such as users, queues, hosts, and job attributes. The chapter also introduces some basic concepts and terminology that can help you begin to use the software. For complete background information about the product, see [Chapter 1](#).

This chapter also includes instructions for accomplishing the following tasks:

- “[Launching the QMON Main Control Window](#)” on page 33
- “[Customizing QMON](#)” on page 34
- “[Displaying a List of Queues](#)” on page 38
- “[How to Display Queue Properties With QMON](#)” on page 39
- “[Displaying Queue Properties From the Command Line](#)” on page 40
- “[Finding the Name of the Master Host](#)” on page 41
- “[Displaying a List of Execution Hosts](#)” on page 42
- “[Displaying a List of Administration Hosts](#)” on page 42
- “[Displaying a List of Submit Hosts](#)” on page 42
- “[Displaying a List of Requestable Attributes](#)” on page 44

QMON Main Control Window

The grid engine system features a graphical user interface (GUI) command tool, the QMON Main Control window. The QMON Main Control window enables users to perform most grid engine system functions, including submitting jobs, controlling jobs, and gathering important information.

Launching the QMON Main Control Window

To launch the QMON Main Control window, from the command line type the following command:

```
% qmon
```

After a message window is displayed, the QMON Main Control window appears.



See [Figure 1–3](#) to identify the meaning of the icons. The names of the icon buttons appear on screen as you rest the pointer over the buttons. The names describe the functions of the buttons.

Many instructions in this guide call for using the QMON Main Control window.

Customizing QMON

The look and feel of QMON is largely defined by a specifically designed resource file. Reasonable defaults are compiled in *sge-root/qmon/Qmon*, which also includes a sample resource file.

The cluster administration can do any of the following:

- Install site-specific defaults in standard locations such as `/usr/lib/X11/app-defaults/Qmon`
- Include QMON-specific resource definitions in the standard `.Xdefaults` or `.Xresources` files
- Put a site-specific `Qmon` file in a location referenced by standard search paths such as `XAPPLRESDIR`

Ask your administrator if any of these cases are relevant in your environment.

In addition, users can configure personal preferences. Users can modify the `Qmon` file. The `Qmon` file can be moved to the home directory or to another location pointed to by the private `XAPPLRESDIR` search path. Users can also include the necessary resource definitions in their private `.Xdefaults` or `.Xresources` files. A private `Qmon` resource file can also be installed using the `xrdb` command. The `xrdb` command can be used during operation. `xrdb` can also be used at startup of the X11 environment, for example, in a `.xinitrc` resource file.

Refer to the comment lines in the sample `Qmon` file for detailed information on the possible customizations.

You can also use the Job Customize and Queue Customize dialog boxes to customize `qmon`. These dialog boxes are shown in [“Customizing the Job Control Display” on page 90](#) and in [“Filtering Cluster Queues and Queue Instances” on page 105](#). In both dialog boxes, users can use

the Save button to store the filtering and display definitions to the file `.qmon_preferences` in their home directories. When QMON is restarted, this file is read, and QMON reactivates the previously defined behavior.

Users and User Categories

Users of the grid engine system fall into four categories. Users in each category have access to their own set of grid engine system commands.

- **Managers** – Managers have full capabilities to manipulate the grid engine system. By default, the superusers of all administration hosts have manager privileges.
- **Operators** – Operators can perform many of the same commands as managers, with the exception of making configuration changes, for example, adding, deleting, or modifying queues.
- **Owners** – Queue owners can suspend or enable the queues that they own. Queue owners can also suspend or enable the jobs within the queues they own. Queue owners have no other management permissions.
- **Users** – Users have certain access permissions, as described in [“User Access Permissions” on page 36](#). Users have no cluster management or queue management capabilities.

Table 2–1 shows the command capabilities that are available to the different user categories.

TABLE 2–1 User Categories and Associated Command Capabilities

Command	Manager	Operator	Owner	User
qacct	Full	Full	Own jobs only	Own jobs only
qalter	Full	Full	Own jobs only	Own jobs only
qconf	Full	No system setup modifications	Show only configurations and access permissions	Show only configurations and access permissions
qdel	Full	Full	Own jobs only	Own jobs only
qhold	Full	Full	Own jobs only	Own jobs only
qhost	Full	Full	Full	Full
qlogin	Full	Full	Full	Full
qmod	Full	Full	Own jobs and owned queues only	Own jobs only
qmon	Full	No system setup modifications	No configuration changes	No configuration changes

TABLE 2-1 User Categories and Associated Command Capabilities (Continued)

Command	Manager	Operator	Owner	User
qrexec	Full	Full	Full	Full
qselect	Full	Full	Full	Full
qsh	Full	Full	Full	Full
qstat	Full	Full	Full	Full
qsub	Full	Full	Full	Full

User Access Permissions

The administrator can restrict access to queues and other facilities, such as parallel environment interfaces. Access can be restricted to certain users or user groups.

Note – The grid engine software automatically takes into account the access restrictions configured by the cluster administration. The following sections are important only if you want to query your personal access permission.

For the purpose of restricting access permissions, the administrator creates and maintains access lists (ACLs). The ACLs contain user names and UNIX group names. The ACLs are then added to *access-allowed* or *access-denied* lists in the queue or in the parallel environment interface configurations. For more information, see the `queue_conf(5)` or `sge_pe(5)` man pages.

Users who belong to ACLs that are listed in *access-allowed-lists* have permission to access the queue or the parallel environment interface. Users who are members of ACLs in *access-denied-lists* cannot access the resource in question.

ACLs are also used to define *projects*, to which the corresponding users have access, that is, to which users can subordinate their jobs. The administrator can also restrict access to cluster resources on a per project basis.

The User Configuration dialog box opens when you click the User Configuration button in the QMON Main Control window. This dialog box enables you to query for the ACLs to which you have access. For details, see Chapter 4, “Managing User Access,” in *Sun N1 Grid Engine 6.1 Administration Guide*.

You can display project access by clicking the Project Configuration icon in the QMON Main Control window. Details are described in “Defining Projects” in *Sun N1 Grid Engine 6.1 Administration Guide*.

From the command line, you can get a list of the currently configured ACLs with the following command:

```
% qconf -sul
```

You can list the entries in one or more access lists with the following command:

```
% qconf -su acl-name[,...]
```

The ACLs consist of user account names and UNIX group names, with the UNIX group names identified by a prefixed @ sign. In this way, you can determine which ACLs your account belongs to.

Note – If you have permission to switch your primary UNIX group with the `newgrp` command, your access permissions might change.

You can check for those queues or parallel environment interfaces to which you have access or to which your access is denied. Query the queue or parallel environment interface configuration, as described in “[Displaying Queues and Queue Properties](#)” on [page 38](#) and “[Configuring Parallel Environments With QMON](#)” in *Sun N1 Grid Engine 6.1 Administration Guide*.

The access-allowed-lists are named `user_lists`. The access-denied-lists are named `xuser_lists`. If your user account or primary UNIX group is associated with an access-allowed-list, you are allowed to access the resource in question. If you are associated with an access-denied-list, you cannot access the queue or parallel environment interface. If both lists are empty, every user with a valid account can access the resource in question.

You can control project configurations from the command line using the following commands:

```
% qconf -sprjl  
% qconf -sprj project-name
```

These commands display a list of defined projects and a list of particular project configurations, respectively. The projects are defined through ACLs. You must query the ACL configurations, as described in the previous paragraph.

If you have access to a project, you are allowed to submit jobs that are subordinated to the project. You can submit such jobs from the command line using the following command:

```
% qsub -P project-name options
```

The cluster configurations, host configurations, and queue configurations define project access in the same way as for ACLs. These configurations use the `project_lists` and `xproject_lists` parameters for this purpose.

Managers, Operators, and Owners

Use the following command to display a list of grid engine system managers:

```
% qconf -sm
```

Use the following command to display a list of operators:

```
% qconf -so
```

Note – The superuser of an administration host is considered to be a manager by default.

Users who are owners of a certain queue are contained in the queue configuration, as described in [“Displaying Queues and Queue Properties” on page 38](#). You can display the queue configuration by typing the following command:

```
% qconf -sq {cluster-queue | queue-instance | queue-domain}
```

The queue configuration entry in question is called `owner_list`.

Displaying Queues and Queue Properties

To make the best use of the grid engine system at your site, you should be familiar with the queue structure. You should also be familiar with the properties of the queues that are configured for your grid engine system.

Displaying a List of Queues

The QMON Queue Control dialog box is shown and described in [“Monitoring and Controlling Queues With QMON” on page 98](#). This dialog box provides a quick overview of the installed queues and their current status.

To display a list of queues, from the command line, type the following command.

```
% qconf -sql
```

Displaying Queue Properties

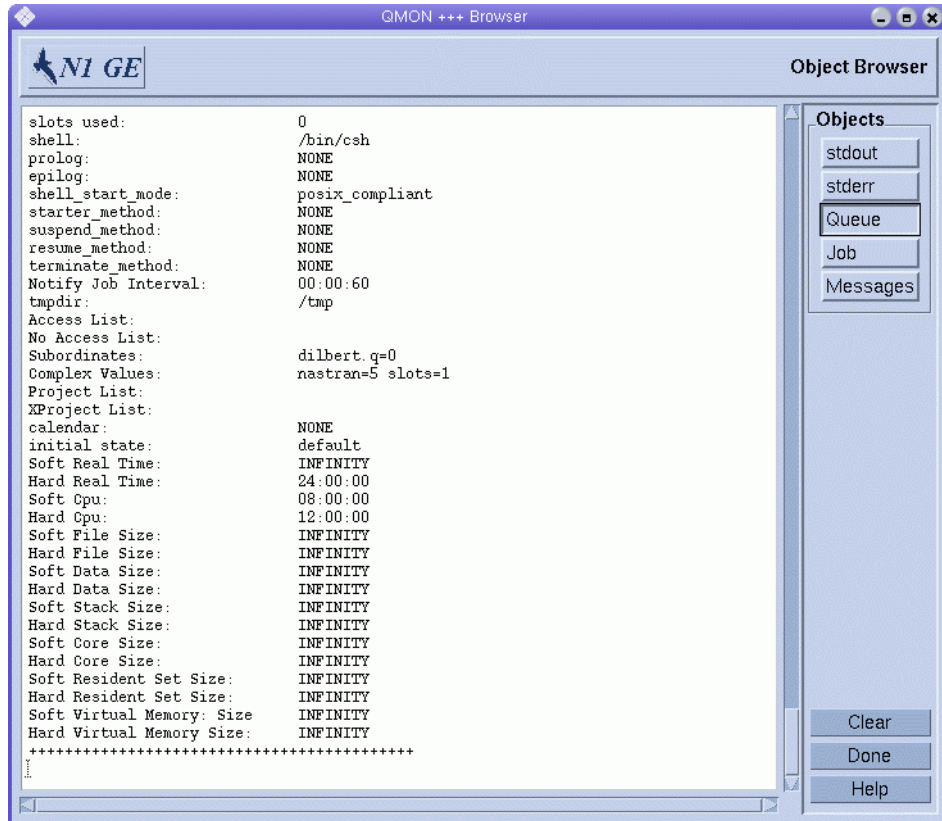
You can use either QMON or the command line to display queue properties.

▼ **How to Display Queue Properties With QMON**

- 1 Launch the QMON Main Control window.**
- 2 Click the Queue Control button.**
The Cluster Queue dialog box appears.
- 3 Select a queue, and then click Show Detached Settings.**
The Browser dialog box appears.
- 4 In the Browser dialog box, click Queue.**
- 5 In the Cluster Queue dialog box, click the Queue Instances tab.**
- 6 Select a queue instance.**
The Browser dialog box lists the queue properties for the selected queue instance.

Example 2-1 Queue Property Information

The following figure shows an example of some of the queue property information that is displayed.



Displaying Queue Properties From the Command Line

To display queue properties from the command line, type the following command:

```
% qconf -sq {queue | queue-instance | queue-domain}
```

Information like that shown in the previous figure is displayed.

Interpreting Queue Property Information

You can find a detailed description of each queue property in the `queue_conf(5)` man page.

The following is a list of some of the more important parameters:

- `qname` – The queue name as requested.
- `hostlist` – A list of hosts and host groups associated with the queue.
- `processors` – The processors of a multiprocessor system to which the queue has access.



Caution – Do not change this value unless you are certain that you need to change it.

- `qtype` – The type of job that can run in this queue. Currently, type can be either batch or interactive.
- `slots` – The number of jobs that can be executed concurrently in that queue.
- `owner_list` – The owners of the queue, which is explained in [“Managers, Operators, and Owners” on page 38](#)
- `user_lists` – The user or group identifiers in the user access lists who are listed under this parameter can access the queue. For more information, see [“User Access Permissions” on page 36](#).
- `xuser_lists` – The user or group identifiers in the user access lists who are listed under this parameter *cannot* access the queue. For more information, see [“User Access Permissions” on page 36](#).
- `project_lists` – Jobs submitted with the project identifiers that are listed under this parameter can access the queue. For more information, see “Defining Projects” in *Sun N1 Grid Engine 6.1 Administration Guide*.
- `xproject_lists` – Jobs submitted with the project identifiers that are listed under this parameter cannot access the queue. For more information, see “Defining Projects” in *Sun N1 Grid Engine 6.1 Administration Guide*.
- `complex_values` – Assigns capacities as provided for this queue for certain complex resource attributes. For more information, see [“Requestable Attributes” on page 42](#).

Hosts and Host Functionality

Clicking the Host Configuration button in the QMON Main Control window displays an overview of the functionality that is associated with the hosts in your cluster. However, without manager privileges, you cannot apply any changes to the configuration.

The host configuration dialog boxes are described in Chapter 1, “Configuring Hosts and Clusters,” in *Sun N1 Grid Engine 6.1 Administration Guide*. The following sections describe the commands used to retrieve host information from the command line.

Finding the Name of the Master Host

The location of the master host can migrate between the current master host and one of the shadow master hosts at any time. Therefore, the location of the master host should be transparent to the user.

With a text editor, open the `sge-root/cell/common/act_qmaster` file.

The name of the current master host is in the file.

Displaying a List of Execution Hosts

To display a list of hosts that are configured as execution hosts in your cluster, use the following commands:

```
% qconf -sel
% qconf -se hostname
% qhost
```

The `qconf -sel` command displays a list of the names of all hosts that are currently configured as execution hosts. The `qconf -se` command displays detailed information about the specified execution host. The `qhost` command displays status and load information about the execution hosts.

See the `host_conf(5)` man page for details on the information displayed using `qconf`. See the `qhost(1)` man page for details on its output and other options.

Displaying a List of Administration Hosts

Use the following command to display a list of hosts with administrative permission:

```
% qconf -sh
```

Displaying a List of Submit Hosts

Use the following command to display a list of submit hosts:

```
% qconf -ss
```

Requestable Attributes

When users submit a job, a requirement profile can be specified for the job. Users can specify attributes or characteristics of a host or queue that the job requires in order to run successfully. The grid engine software maps these job requirements onto the host and queue configurations of the cluster and therefore finds suitable hosts for a job.

The attributes that can be used to specify the job requirements are related to one of the following:

- The cluster, for example, space required on a network shared disk
- Individual hosts, for example, operating system architecture
- Queues, for example, permitted CPU time

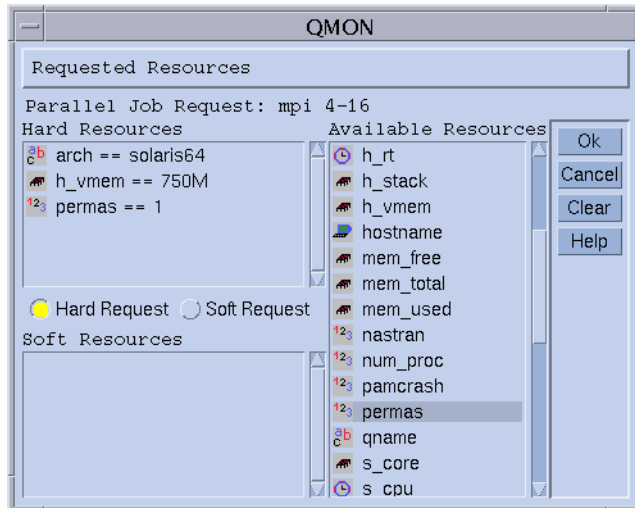
The attributes can also be derived from site policies such as the availability of installed software only on certain hosts.

The available attributes include the following:

- Queue property list – See [“Displaying Queues and Queue Properties” on page 38](#)
- List of global and host-related attributes – See “Assigning Resource Attributes to Queues, Hosts, and the Global Cluster” in *Sun N1 Grid Engine 6.1 Administration Guide*
- Administrator-defined attributes

For convenience, however, the administrator commonly chooses to define only a subset of all available attributes to be requestable.

The currently requestable attributes are displayed in the Requested Resources dialog box, which is shown in the following figure.



Use the QMON Submit Job dialog box to access the Requested Resources dialog box. Requestable attributes are listed under Available Resources.

Displaying a List of Requestable Attributes

To display the list of configured *resource attributes*, from the command line type the following command:

```
% qconf -sc
```

The grid engine system complex contains the definitions for all resource attributes. For more information about resource attributes, see Chapter 3, “Configuring Complex Resource Attributes,” in *Sun N1 Grid Engine 6.1 Administration Guide*. See also the complex format description on the `complex(5)` man page.

Sample output from the `qconf -sc` command is shown in [Example 2–2](#).

EXAMPLE 2–2 Complex Attributes Displayed

```
gimli% qconf -sc
```

#name	shortcut	type	relop	requestable	consumable	default	urgency
arch	a	RESTRING	==	YES	NO	NONE	0
calendar	c	STRING	==	YES	NO	NONE	0
cpu	cpu	DOUBLE	>=	YES	NO	0	0
h_core	h_core	MEMORY	<=	YES	NO	0	0
h_cpu	h_cpu	TIME	<=	YES	NO	0:0:0	0
h_data	h_data	MEMORY	<=	YES	NO	0	0
h_fsize	h_fsize	MEMORY	<=	YES	NO	0	0
h_rss	h_rss	MEMORY	<=	YES	NO	0	0
h_rt	h_rt	TIME	<=	YES	NO	0:0:0	0
h_stack	h_stack	MEMORY	<=	YES	NO	0	0
h_vmem	h_vmem	MEMORY	<=	YES	NO	0	0
hostname	h	HOST	==	YES	NO	NONE	0
load_avg	la	DOUBLE	>=	NO	NO	0	0
load_long	ll	DOUBLE	>=	NO	NO	0	0
load_medium	lm	DOUBLE	>=	NO	NO	0	0
load_short	ls	DOUBLE	>=	NO	NO	0	0
mem_free	mf	MEMORY	<=	YES	NO	0	0
mem_total	mt	MEMORY	<=	YES	NO	0	0
mem_used	mu	MEMORY	>=	YES	NO	0	0
min_cpu_interval	mci	TIME	<=	NO	NO	0:0:0	0
np_load_avg	nla	DOUBLE	>=	NO	NO	0	0
np_load_long	nll	DOUBLE	>=	NO	NO	0	0
np_load_medium	nlm	DOUBLE	>=	NO	NO	0	0
np_load_short	nls	DOUBLE	>=	NO	NO	0	0
num_proc	p	INT	==	YES	NO	0	0
qname	q	STRING	==	YES	NO	NONE	0
rerun	re	BOOL	==	NO	NO	0	0
s_core	s_core	MEMORY	<=	YES	NO	0	0
s_cpu	s_cpu	TIME	<=	YES	NO	0:0:0	0

EXAMPLE 2-2 Complex Attributes Displayed (Continued)

s_data	s_data	MEMORY	<=	YES	NO	0	0
s_fsize	s_fsize	MEMORY	<=	YES	NO	0	0
s_rss	s_rss	MEMORY	<=	YES	NO	0	0
s_rt	s_rt	TIME	<=	YES	NO	0:0:0	0
s_stack	s_stack	MEMORY	<=	YES	NO	0	0
s_vmem	s_vmem	MEMORY	<=	YES	NO	0	0
seq_no	seq	INT	==	NO	NO	0	0
slots	s	INT	<=	YES	YES	1	1000
swap_free	sf	MEMORY	<=	YES	NO	0	0
swap_rate	sr	MEMORY	>=	YES	NO	0	0
swap_rsvd	srsv	MEMORY	>=	YES	NO	0	0
swap_total	st	MEMORY	<=	YES	NO	0	0
swap_used	su	MEMORY	>=	YES	NO	0	0
tmpdir	tmp	STRING	==	NO	NO	NONE	0
virtual_free	vf	MEMORY	<=	YES	NO	0	0
virtual_total	vt	MEMORY	<=	YES	NO	0	0
virtual_used	vu	MEMORY	>=	YES	NO	0	0

>#< starts a comment but comments are not saved across edits -----

The column name is identical to the first column displayed by the `qconf -sq` command. The shortcut column contains administrator-definable abbreviations for the full names in the first column. The user can supply either the full name or the shortcut in the request option of a `qsub` command.

The column `requestable` tells whether the resource attribute can be used in a `qsub` command. The administrator can, for example, disallow the cluster's users to request certain machines or queues for their jobs directly. The administrator can disallow direct requests by setting the entries `qname`, `hostname`, or both, to be unrequestable. Making queues or hosts unrequestable implies that feasible user requests can be met in general by multiple queues, which enforces the load balancing capabilities of the grid engine system.

The column `relop` defines the relational operator used to compute whether a queue or a host meets a user request. The comparison that is executed is as follows:

```
User_Request      relop      Queue/Host/... -Property
```

If the result of the comparison is false, the user's job cannot be run in the queue or on the host. For example, let the queue `q1` be configured with a soft CPU time limit of 100 seconds. Let the queue `q2` be configured to provide 1000 seconds soft CPU time limit. See the `queue_conf(5)` and the `setrlimit(2)` man pages for a description of user process limits.

The columns `consumable` and `default` affect how the administrator declares consumable resources. See “Consumable Resources” in *Sun N1 Grid Engine 6.1 Administration Guide*.

The user requests consumables just like any other attribute. The grid engine system internal bookkeeping for the resources is different, however.

Assume that a user submits the following request:

```
% qsub -l s_cpu=0:5:0 nastran.sh
```

The `s_cpu=0:5:0` request asks for a queue that grants at least 5 minutes of soft limit CPU time. Therefore, only queues providing at least 5 minutes soft CPU runtime limit are set up properly to run the job. See the `qsub(1)` man page for details on the syntax.

Note – The grid engine software considers workload information in the scheduling process only if more than one queue or host can run a job.

Submitting Jobs

This chapter provides background information about submitting jobs, as well as instructions for how to submit jobs for processing. The chapter begins with an example of how to run a simple job. The chapter then continues with instructions for how to run more complex jobs.

Instructions for accomplishing the following tasks are included in this chapter.

- “How To Submit a Simple Job From the Command Line” on page 48
- “How To Submit a Simple Job With QMON” on page 49
- “Submitting Extended Jobs With QMON” on page 59
- “Submitting Extended Jobs From the Command Line” on page 63
- “Submitting Advanced Jobs With QMON” on page 63
- “Submitting Advanced Jobs From the Command Line” on page 66
- “Submitting an Array Job With QMON” on page 72
- “Submitting an Array Job From the Command Line” on page 72
- “Submitting Interactive Jobs With QMON” on page 74
- “Submitting Interactive Jobs With qsh” on page 75
- “Submitting Interactive Jobs With qlogin” on page 75

Submitting a Simple Job

Use the information and instructions in this section to become familiar with basic procedures involved in submitting jobs.

Note – If you installed the N1 Grid Engine 6.1 software under an unprivileged user account, you must log in as that user to be able to run jobs. See “Installation Accounts” in *Sun N1 Grid Engine 6.1 Installation Guide* for details.

▼ How To Submit a Simple Job From the Command Line

Before you run any grid engine system command, you must first set your executable search path and other environment conditions properly.

1 From the command line, type one of the following commands.

- If you are using `csh` or `tcsh` as your command interpreter, type the following:

```
% source sge-root/cell/common/settings.csh
```

sge-root specifies the location of the root directory of the grid engine system. This directory was specified at the beginning of the installation procedure.

- If you are using `sh`, `ksh`, or `bash` as your command interpreter, type the following:

```
# . sge-root/cell/common/settings.sh
```

Note – You can add these commands to your `.login`, `.cshrc`, or `.profile` files, whichever is appropriate. By adding these commands, you guarantee proper settings for all interactive session you start later.

2 Submit a simple job script to your cluster by typing the following command:

```
% qsub simple.sh
```

The command assumes that `simple.sh` is the name of the script file, and that the file is located in your current working directory.

You can find the following job in the file `/sge-root/examples/jobs/simple.sh`.

```
#!/bin/sh
#
#
# (c) 2004 Sun Microsystems, Inc. Use is subject to license terms.

# This is a simple example of a SGE batch script

# request Bourne shell as shell for job
#$ -S /bin/sh

#
# print date and time
date
# Sleep for 20 seconds
sleep 20
# print date and time again
date
```

If the job submits successfully, the `qsub` command responds with a message similar to the following example:

```
your job 1 ("simple.sh") has been submitted
```

3 Type the following command to retrieve status information about your job.

```
% qstat
```

You should receive a status report that provides information about all jobs currently known to the grid engine system. For each job, the status report lists the following items:

- Job ID, which is the unique number that is included in the submit confirmation
- Name of the job script
- Owner of the job
- State indicator; for example, `r` means running
- Submit or start time
- Name of the queue in which the job runs

If `qstat` produces no output, no jobs are actually known to the system. For example, your job might already have finished.

You can control the output of the finished jobs by checking their `stdout` and `stderr` redirection files. By default, these files are generated in the job owner's home directory on the host that ran the job. The names of the files are composed of the job script file name with a `.o` extension for the `stdout` file and a `.e` extension for the `stderr` file, followed by the unique job ID. The `stdout` and `stderr` files of your job can be found under the names `simple.sh.o1` and `simple.sh.e1` respectively. These names are used if your job was the first ever executed in a newly installed grid engine system.

▼ How To Submit a Simple Job With QMON

A more convenient way to submit and control jobs and of getting an overview of the grid engine system is the graphical user interface QMON. Among other facilities, QMON provides a job submission dialog box and a Job Control dialog box for the tasks of submitting and monitoring jobs.

1 Type the following command to start the QMON GUI:

```
% qmon
```

During startup, a message window appears, and then the QMON Main Control window appears.

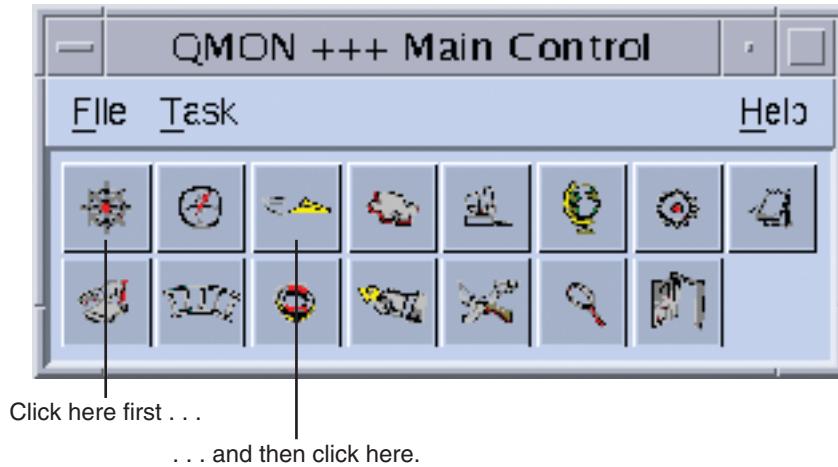


FIGURE 3-1 QMON Main Control Window

- 2 Click the Job Control button, and then click the Submit Jobs button.

Tip – The button names, such as Job Control, are displayed when you rest the mouse pointer over the buttons.

The Submit Job and the Job Control dialog boxes appear, as shown in the following figures.

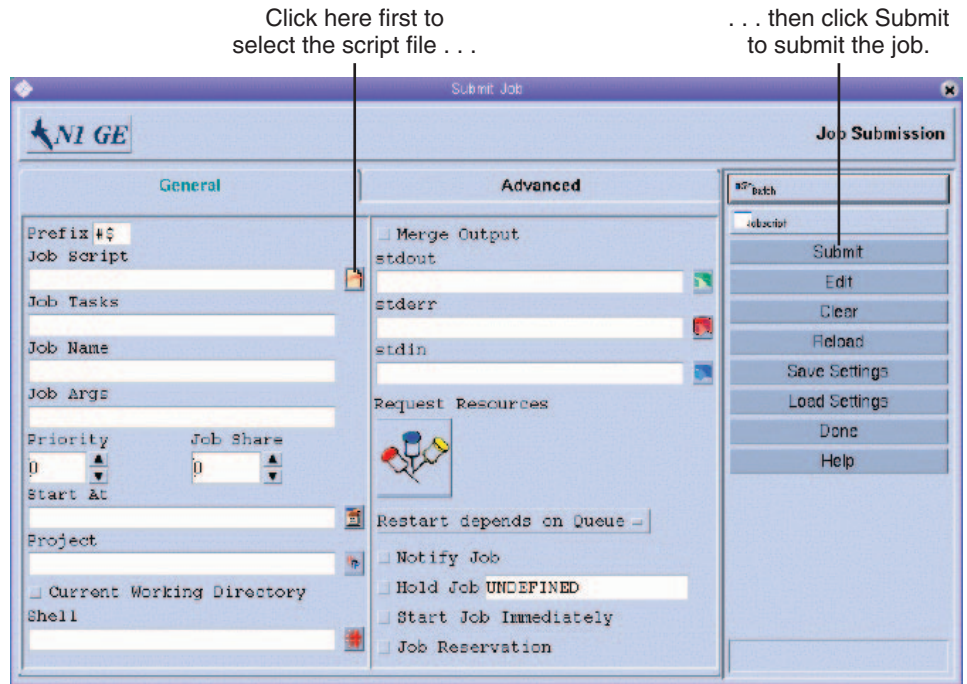


FIGURE 3-2 Submit Job Dialog Box

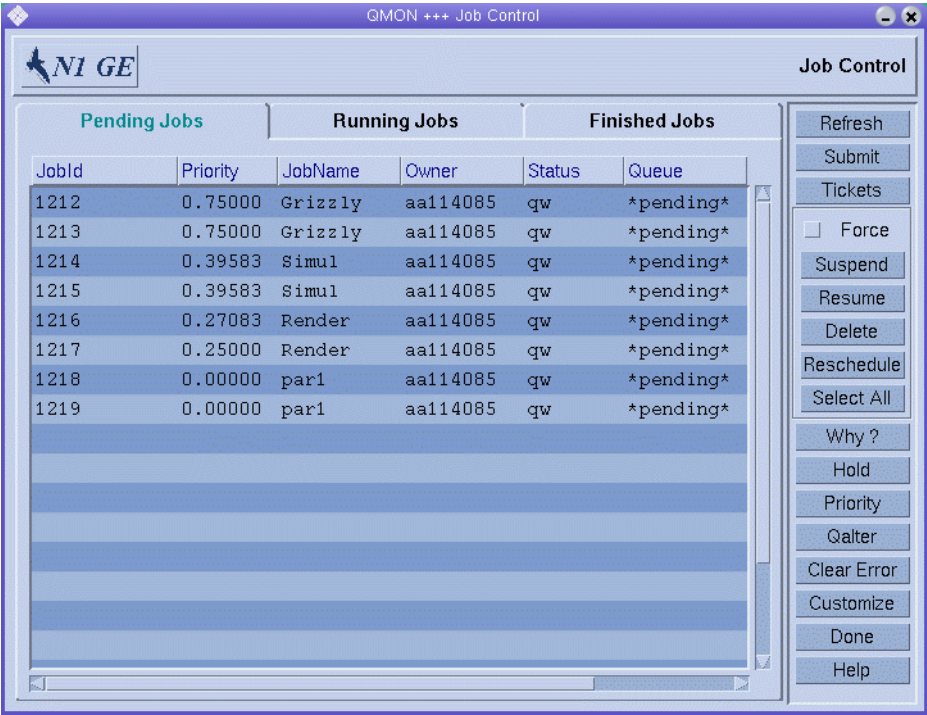


FIGURE 3-3 Job Control Dialog Box

- 3 In the Submit Job dialog box, click the icon at the right of the Job Script field.
The Select a File dialog box appears.

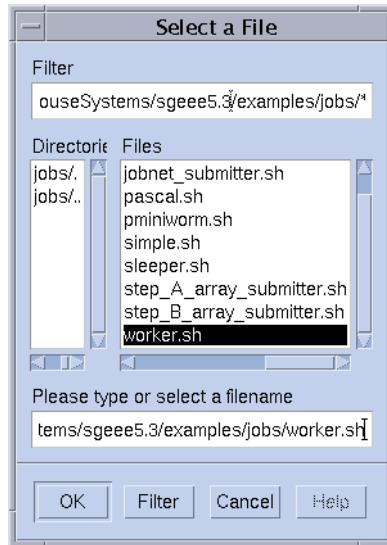


FIGURE 3-4 Select a File Dialog Box

4 Select your script file.

For example, select the file `simple.sh` that was used in the command line example.

5 Click OK to close the Select a File dialog box.

6 On the Submit Job dialog box, click Submit.

After a few seconds you should be able to monitor your job on the Job Control dialog box. You first see your job on the Pending Jobs tab. The job quickly moves to the Running Jobs tab once the job starts running.

Submitting Batch Jobs

The following sections describe how to submit more complex jobs through the grid engine system.

About Shell Scripts

Shell scripts, also called batch jobs, are a sequence of command-line instructions that are assembled in a file. Script files are made executable by the `chmod` command. If scripts are invoked, a command interpreter is started. Each instruction is interpreted as if the instruction

were typed manually by the user who is running the script. `csh`, `tcsh`, `sh`, or `ksh` are typical command interpreters. You can invoke arbitrary commands, applications, and other shell scripts from within a shell script.

The command interpreter can be invoked as login shell. To do so, the name of the command interpreter must be contained in the `login_shells` list of the grid engine system configuration that is in effect for the particular host and queue that is running the job.

Note – The grid engine system configuration might be different for the various hosts and queues configured in your cluster. You can display the effective configurations with the `-sconf` and `-sq` options of the `qconf` command. For detailed information, see the `qconf(1)` man page.

If the command interpreter is invoked as login shell, the environment of your job is the same as if you logged in and ran the script. In using `csh`, for example, `.login` and `.cshrc` are executed in addition to the system default startup resource files, such as `/etc/login`, whereas only `.cshrc` is executed if `csh` is not invoked as `login-shell`. For a description of the difference between being invoked and not being invoked as `login-shell`, see the man page of your command interpreter.

Example of a Shell Script

[Example 3–1](#) is a simple shell script. The script first compiles the application `flow` from its Fortran77 source and then runs the application.

EXAMPLE 3–1 Simple Shell Script

```
#!/bin/csh
# This is a sample script file for compiling and
# running a sample FORTRAN program under N1 Grid Engine 6
cd TEST
# Now we need to compile the program "flow.f" and
# name the executable "flow".
f77 flow.f -o flow
```

Your local system user's guide provides detailed information about building and customizing shell scripts. You might also want to look at the `sh`, `ksh`, `csh`, or `tcsh` man page. The following sections emphasize special things that you should consider when you prepare batch scripts for the grid engine system.

In general, you can submit to the grid engine system all shell scripts that you can run from your command prompt by hand. Such shell scripts must not require a terminal connection, and the scripts must not need interactive user intervention. The exceptions are the standard error and standard output devices, which are automatically redirected. Therefore, [Example 3–1](#) is ready to be submitted to the grid engine system and the script will perform the desired action.

Extensions to Regular Shell Scripts

Some extensions to regular shell scripts influence the behavior of scripts that run under grid engine system control. The following sections describe these extensions.

How a Command Interpreter Is Selected

At submit time, you can specify the command interpreter to use to process the job script file as shown in [Figure 3–5](#). However, if nothing is specified, the configuration variable `shell_start_mode` determines how the command interpreter is selected:

- If `shell_start_mode` is set to `unix_behavior`, the first line of the script file specifies the command interpreter. The first line of the script file must begin with `#!`. If the first line does not begin with `#!`, the Bourne Shell `sh` is used by default.
- For all other settings of `shell_start_mode`, the default command interpreter is determined by the `shell` parameter for the queue where the job starts. See [“Displaying Queues and Queue Properties” on page 38](#) and the `queue_conf(5)` man page.

Output Redirection

Since batch jobs do not have a terminal connection, their standard output and their standard error output must be redirected into files. The grid engine system enables the user to define the location of the files to which the output is redirected. Defaults are used if no output files are specified.

The standard location for the files is in the current working directory where the jobs run. The default standard output file name is `job-name.ojob-id`, the default standard error output is redirected to `job-name>.ejob-id`. The `job-name` can be built from the script file name, or defined by the user. See, for example, the `-N` option in the `submit(1)` man page. `job-id` is a unique identifier that is assigned to the job by the grid engine system.

For array job tasks, the task identifier is added to these filenames, separated by a dot. The resulting standard redirection paths are `job-name.ojob-id.task-id>` and `job-name.ejob-id.task-id`. For more information, see [“Submitting Array Jobs” on page 71](#).

In case the standard locations are not suitable, the user can specify output directions with `QMON`, as shown in [Figure 3–6](#). Or the user can use the `-e` and `-o` options to the `qsub` command to specify output directions. Standard output and standard error output can be merged into one file. The redirections can be specified on a per execution host basis, in which case, the location of the output redirection file depends on the host on which the job is executed. To build custom but unique redirection file paths, use dummy environment variables together with the `qsub -e` and `-o` options. A list of these variables follows.

- `HOME` – Home directory on execution machine
- `USER` – User ID of job owner
- `JOB_ID` – Current job ID

- JOB_NAME – Current job name; see the -N option
- HOSTNAME – Name of the execution host
- TASK_ID – Array job task index number

When the job runs, these variables are expanded into the actual values, and the redirection path is built with these values.

See the `qsub(1)` man page for further details.

Active Comments

Lines with a leading `#` sign are treated as comments in shell scripts. However, the grid engine system recognizes special comment lines and uses these lines in a special way. The special comment script line is treated as part of the command line argument list of the `qsub` command. The `qsub` options that are supplied within these special comment lines are also interpreted by the QMON Submit Job dialog box. The corresponding parameters are preset when a script file is selected.

By default, the special comment lines are identified by the `#$` prefix string. You can redefine the prefix string with the `qsub -C` command.

This use of special comments is called script embedding of submit arguments. The following example shows a script file that uses script-embedded command-line options.

EXAMPLE 3-2 Using Script-Embedded Command Line Options

```
#!/bin/csh

#Force csh if not Grid Engine default
#shell

#$ -S /bin/csh

# This is a sample script file for compiling and
# running a sample FORTRAN program under N1 Grid Engine 6
# We want Grid Engine to send mail
# when the job begins
# and when it ends.

#$ -M EmailAddress
#$ -m b e

# We want to name the file for the standard output
# and standard error.

#$ -o flow.out -j y
```

EXAMPLE 3-2 Using Script-Embedded Command Line Options (Continued)

```
# Change to the directory where the files are located.

cd TEST

# Now we need to compile the program "flow.f" and
# name the executable "flow".

f77 flow.f -o flow

# Once it is compiled, we can run the program.

flow
```

Environment Variables

When a job runs, several variables are preset into the job's environment.

- **ARC** – The architecture name of the node on which the job is running. The name is compiled into the `sge_execd` binary.
- **SGE_ROOT** – The root directory of the grid engine system as set for `sge_execd` before startup, or the default `/usr/SGE` directory.
- **SGE_BINARY_PATH** – The directory in which the grid engine system binaries are installed.
- **SGE_CELL** – The cell in which the job runs.
- **SGE_JOB_SPOOL_DIR** – The directory used by `sge_shepherd` to store job-related data while the job runs.
- **SGE_O_HOME** – The path to the home directory of the job owner on the host from which the job was submitted.
- **SGE_O_HOST** – The host from which the job was submitted.
- **SGE_O_LOGNAME** – The login name of the job owner on the host from which the job was submitted.
- **SGE_O_MAIL** – The content of the `MAIL` environment variable in the context of the job submission command.
- **SGE_O_PATH** – The content of the `PATH` environment variable in the context of the job submission command.
- **SGE_O_SHELL** – The content of the `SHELL` environment variable in the context of the job submission command.
- **SGE_O_TZ** – The content of the `TZ` environment variable in the context of the job submission command.
- **SGE_O_WORKDIR** – The working directory of the job submission command.

- `SGE_CHKPT_ENV` – The checkpointing environment under which a checkpointing job runs. The checkpointing environment is selected with the `qsub -ckpt` command.
- `SGE_CHKPT_DIR` – The path `ckpt_dir` of the checkpoint interface. Set only for checkpointing jobs. For more information, see the `checkpoint(5)` man page.
- `SGE_STDERR_PATH` – The path name of the file to which the standard error stream of the job is diverted. This file is commonly used for enhancing the output with error messages from prolog, epilog, parallel environment start and stop scripts, or checkpointing scripts.
- `SGE_STDOUT_PATH` – The path name of the file to which the standard output stream of the job is diverted. This file is commonly used for enhancing the output with messages from prolog, epilog, parallel environment start and stop scripts, or checkpointing scripts.
- `SGE_TASK_ID` – The task identifier in the array job represented by this task.
- `ENVIRONMENT` – Always set to `BATCH`. This variable indicates that the script is run in batch mode.
- `HOME` – The user's home directory path as taken from the `passwd` file.
- `HOSTNAME` – The host name of the node on which the job is running.
- `JOB_ID` – A unique identifier assigned by the `sges_qmaster` daemon when the job was submitted. The job ID is a decimal integer from 1 through 9,999,999.
- `JOB_NAME` – The job name, which is built from the file name provided with the `qsub` command, a period, and the digits of the job ID. You can override this default with `qsub -N`.
- `LOGNAME` – The user's login name as taken from the `passwd` file.
- `NHOSTS` – The number of hosts in use by a parallel job.
- `NQUEUES` – The number of queues that are allocated for the job. This number is always 1 for serial jobs.
- `NSLOTS` – The number of queue slots in use by a parallel job.
- `PATH` – A default shell search path of: `/usr/local/bin:/usr/ucb:/bin:/usr/bin`.
- `PE` – The parallel environment under which the job runs. This variable is for parallel jobs only.
- `PE_HOSTFILE` – The path of a file that contains the definition of the virtual parallel machine that is assigned to a parallel job by the grid engine system. This variable is used for parallel jobs only. See the description of the `$pe_hostfile` parameter in `sges_pe` for details on the format of this file.
- `QUEUE` – The name of the queue in which the job is running.
- `REQUEST` – The request name of the job. The name is either the job script file name or is explicitly assigned to the job by the `qsub -N` command.
- `RESTARTED` – Indicates whether a checkpointing job was restarted. If set to value 1, the job was interrupted at least once. The job is therefore restarted.
- `SHELL` – The user's login shell as taken from the `passwd` file.

Note – SHELL is not necessarily the shell that is used for the job.

- TMPDIR – The absolute path to the job's temporary working directory.
- TMP – The same as TMPDIR. This variable is provided for compatibility with NQS.
- TZ – The time zone variable imported from sge_execd, if set.
- USER – The user's login name as taken from the passwd file.

Submitting Extended Jobs and Advanced Jobs

Extended jobs and *advanced jobs* are more complex forms of job submission. Before attempting to submit such jobs, you should understand some important background information about the process. The following sections describe those job processes.

Submitting Extended Jobs With QMON

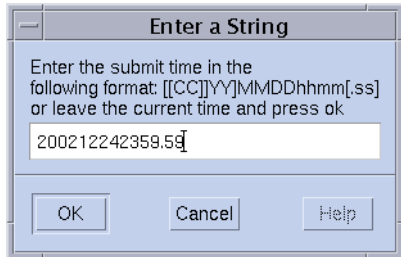
The General tab of the Submit Job dialog box enables you to configure the following parameters for an extended job. The General tab is shown in [Figure 3–2](#).

- Prefix – A prefix string that is used for script-embedded submit options. See “[Active Comments](#)” on [page 56](#) for details.
- Job Script – The job script to use. Click the icon at the right of the Job Script field to open a file selection box. The file selection box is shown in [Figure 3–4](#).
- Job Tasks – The task ID range for submitting array jobs. See “[Submitting Array Jobs](#)” on [page 71](#) for details.
- Job Name – The name of the job. A default is set after you select a job script.
- Job Args – Arguments to the job script.
- Priority – A counting box for setting the job's initial priority. This priority ranks a single user's jobs. Priority tells the scheduler how to choose among a single user's jobs when several of that user's jobs are in the system simultaneously.

Note – To enable users to set the priorities of their own jobs, the administrator must enable priorities with the `weight_priority` parameter of the scheduler configuration. For more information, see Chapter 5, “Managing Policies and the Scheduler,” in *Sun N1 Grid Engine 6.1 Administration Guide*.

- Job Share – Defines the share of the job's tickets relative to other jobs. The job share influences only the share tree policy and the functional policy.

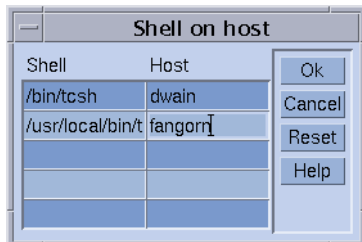
- **Start At** – The time at which the job is considered eligible for execution. Click the icon at the right of the Start At field to open a dialog box for entering the correctly formatted time:



- **Project** – The project to which the job is subordinated. Click the icon at the right of the Project field to select among the available projects:

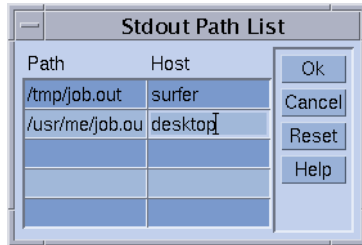


- **Current Working Directory** – A flag that indicates whether to execute the job in the current working directory. Use this flag only for identical directory hierarchies between the submit host and the potential execution hosts.
- **Shell** – The command interpreter to use to run the job script. See [“How a Command Interpreter Is Selected” on page 55](#) for details. Click the icon at the right of the Shell field to open a dialog box for entering the command interpreter specifications of the job:



- **Merge Output** – A flag indicating whether to merge the job's standard output and standard error output together into the standard output stream.

- **stdout** – The standard output redirection to use. See [“Output Redirection” on page 55](#) for details. A default is used if nothing is specified. Click the icon at the right of the **stdout** field to open a dialog box for entering the output redirection alternatives:



- **stderr** – The standard error output redirection to use, similar to the standard output redirection.
- **stdin** – The standard input file to use, similar to the standard output redirection.
- **Request Resources** – Click this button to define the resource requirement for your job. If resources are requested for a job, the button changes its color.
- **Restart depends on Queue** – Click this button to define whether the job can be restarted after being aborted by a system crash or similar events. This button also controls whether the restart behavior depends on the queue or is demanded by the job.
- **Notify Job** – A flag indicating whether the job is to be notified by SIGUSR1 or by SIGUSR2 signals if the job is about to be suspended or cancelled.
- **Hold Job** – A flag indicating that either a user hold or a job dependency is to be assigned to the job. The job is not eligible for execution as long as any type of hold is assigned to the job. See [“Monitoring and Controlling Jobs” on page 85](#) for more details. The Hold Job field enables restricting the hold only to a specific range of tasks of an array job. See [“Submitting Array Jobs” on page 71](#) for information about array jobs.
- **Start Job Immediately** – A flag that forces the job to be started immediately if possible, or to be rejected otherwise. Jobs are not queued if this flag is selected.
- **Job Reservation** – A flag specifying that resources should be reserved for this job. See [“Resource Reservation and Backfilling”](#) in *Sun N1 Grid Engine 6.1 Administration Guide* for details.

The buttons at the right side of the Submit Job dialog box enable you to start various actions:

- **Submit** – Submit the currently specified job.
- **Edit** – Edit the selected script file in an X terminal, using either `vi` or the editor defined by the `EDITOR` environment variable.
- **Clear** – Clear all settings in the Submit Job dialog box, including any specified resource requests.

- **Reload** – Reload the specified script file, parse any script-embedded options, parse default settings, and discard intermediate manual changes to these settings. For more information, see “[Active Comments](#)” on page 56 and “[Default Request Files](#)” on page 67. This action is the equivalent to a Clear action with subsequent specifications of the previous script file. The option has an effect only if a script file is already selected.
- **Save Settings** – Save the current settings to a file. Use the file selection box to select the file. The saved files can either be loaded later or be used as default requests. For more information, see **Load Settings** and “[Default Request Files](#)” on page 67.
- **Load Settings** – Load settings previously saved with the Save Settings button. The loaded settings overwrite the current settings. See **Save Settings**.
- **Done** – Closes the Submit Job dialog box.

Extended Job Example

Figure 3–5 shows the Submit Job dialog box with most of the parameters set.

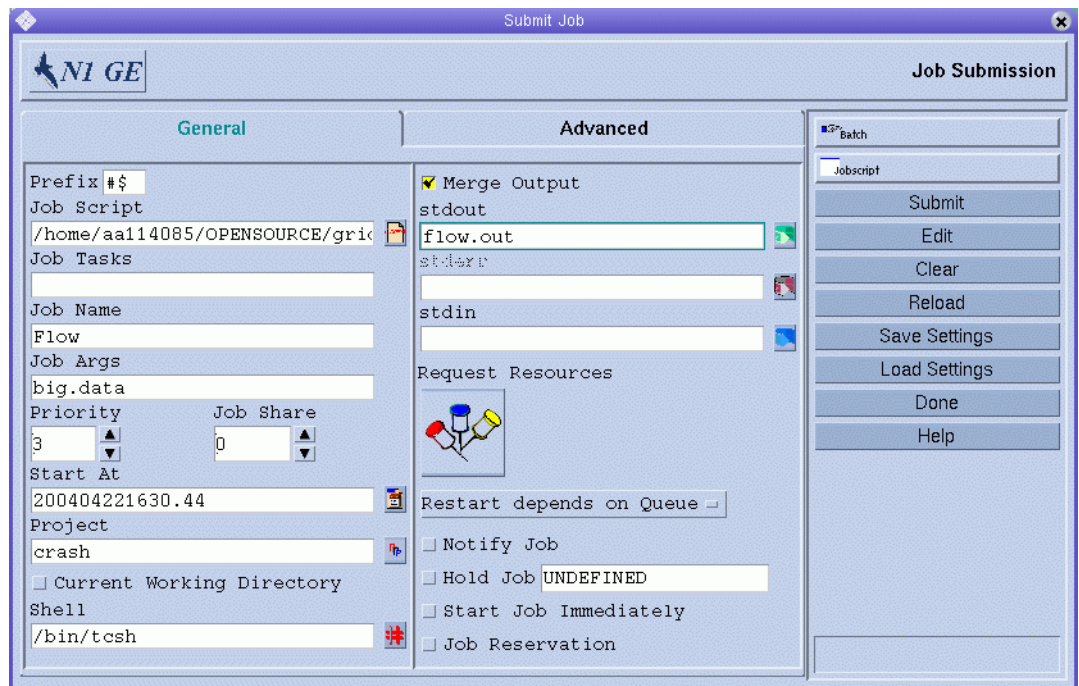


FIGURE 3–5 Extended Job Submission Example

The parameters of the job configured in the example are:

- The job has the script file `flow.sh`, which must reside in the working directory of `QMON`.
- The job is called `Flow`.

- The script file takes the single argument `big.data`.
- The job starts with priority 3.
- The job is eligible for execution not before 4:30.44 AM of the 22th of April in the year 2004.
- The project definition means that the job is subordinated to project `crash`.
- The job is executed in the submission working directory.
- The job uses the `tcsh` command interpreter.
- Standard output and standard error output are merged into the file `flow.out`, which is created in the current working directory.

Submitting Extended Jobs From the Command Line

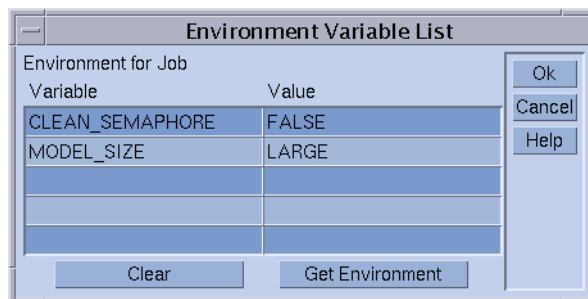
To submit the extended job request that is shown in [Figure 3–5](#) from the command line, type the following command:

```
% qsub -N Flow -p -111 -P devel -a 200404221630.44 -cwd \
      -S /bin/tcsh -o flow.out -j y flow.sh big.data
```

Submitting Advanced Jobs With QMON

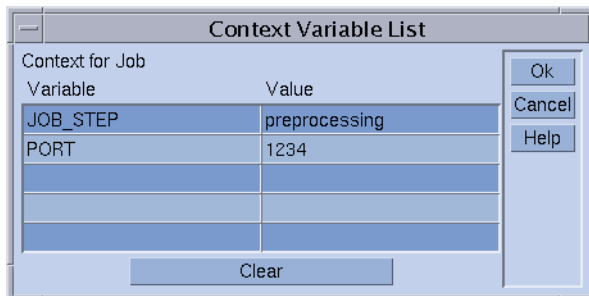
The Advanced tab of the Submit Job dialog box enables you to define the following additional parameters:

- Parallel Environment – A parallel environment interface to use
- Environment – A set of environment variables to set for the job before the job runs. Click the icon at the right of the Environment field to open a dialog box that enables you to define the environment variables to export:



Environment variables can be taken from QMON's runtime environment, or you can define your own environment variables.

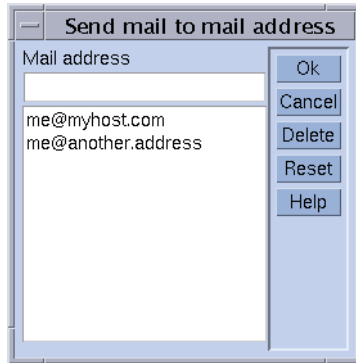
- **Context** – A list of name/value pairs that can be used to store and communicate job-related information. This information is accessible anywhere from within a cluster. You can modify context variables from the command line with the `-ac`, `-dc`, and `-sc` options to `qsub`, `qrsh`, `qsh`, `qlogin`, and `qalter`. You can retrieve context variables with the `qstat -j` command.



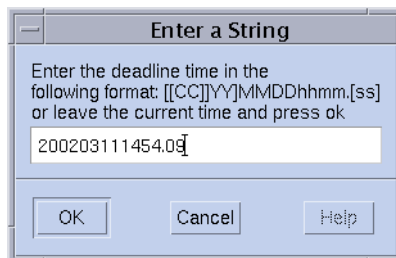
The image shows a dialog box titled "Context Variable List". It contains a table with two columns: "Variable" and "Value". The table has two rows of data: "JOB_STEP" with value "preprocessing" and "PORT" with value "1234". There are three empty rows below the data. To the right of the table are three buttons: "Ok", "Cancel", and "Help". Below the table is a "Clear" button.

Variable	Value
JOB_STEP	preprocessing
PORT	1234

- **Checkpoint Object** – The checkpointing environment to use if checkpointing the job is desirable and suitable. See [“Using Job Checkpointing” on page 106](#) for details.
- **Account** – An account string to associate with the job. The account string is added to the accounting record that is kept for the job. The accounting record can be used for later accounting analysis.
- **Verify Mode** – The Verify flag determines the consistency checking mode for your job. To check for consistency of the job request, the grid engine system assumes an empty and unloaded cluster. The system tries to find at least one queue in which the job could run. Possible checking modes are as follows:
 - **Skip** – No consistency checking at all.
 - **Warning** – Inconsistencies are reported, but the job is still accepted. Warning mode might be desirable if the cluster configuration should change after the job is submitted.
 - **Error** – Inconsistencies are reported. The job is rejected if any inconsistencies are encountered.
 - **Just verify** – The job is not submitted. An extensive report is generated about the suitability of the job for each host and queue in the cluster.
- **Mail** – The events about which the user is notified by email. The events' start, end, abort, and suspend are currently defined for jobs.
- **Mail To** – A list of email addresses to which these notifications are sent. Click the icon at the right of the Mail To field to open a dialog box for defining the mailing list.



- **Hard Queue List, Soft Queue List** – A list of queue names that are requested to be the mandatory selection for the execution of the job. The Hard Queue List and the Soft Queue List are treated identically to a corresponding resource requirement.
- **Master Queue List** – A list of queue names that are eligible as *master queue* for a parallel job. A parallel job is started in the master queue. All other queues to which the job spawns parallel tasks are called *slave queues*.
- **Job Dependencies** – A list of IDs of jobs that must finish before the submitted job can be started. The newly created job depends on completion of those jobs.
- **Deadline** – The deadline initiation time for deadline jobs. Deadline initiation defines the point in time at which a deadline job must reach maximum priority to finish before a given deadline. To determine the deadline initiation time, subtract an estimate of the running time, at maximum priority, of a deadline job from its desired deadline time. Click the icon at the right of the Deadline field to open the dialog box that enables you to set the deadline.



Note – Not all users are allowed to submit deadline jobs. Ask your system administrator if you are permitted to submit deadline jobs. Contact the cluster administrator for information about the maximum priority that is given to deadline jobs.

Advanced Job Example

Figure 3–6 shows an example of an advanced job submission.

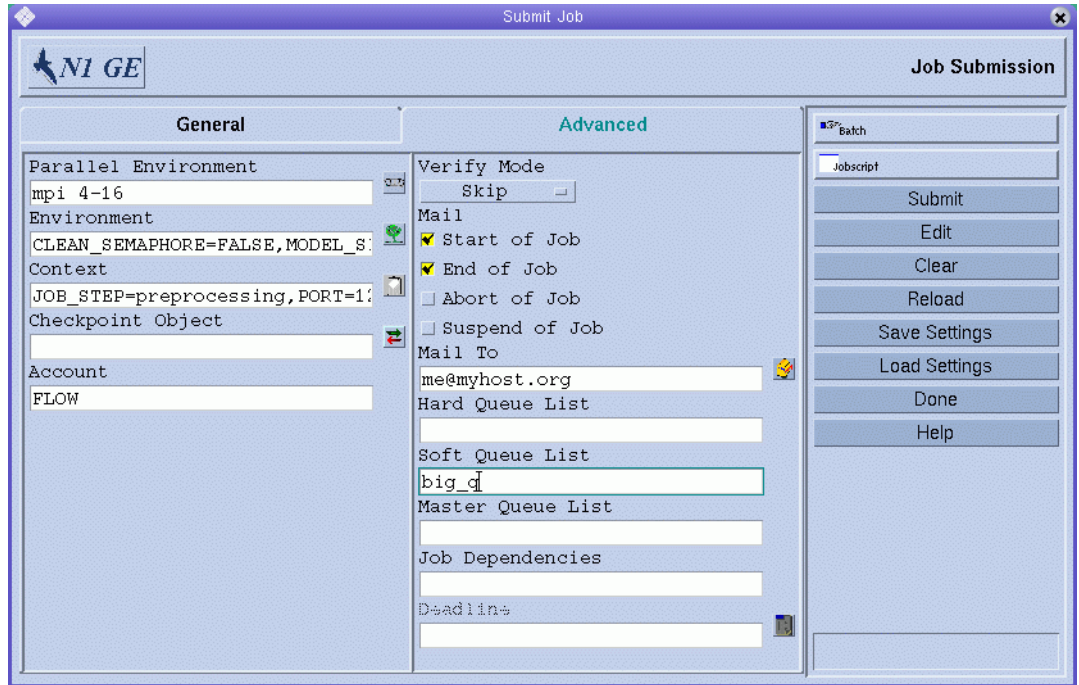


FIGURE 3-6 Advanced Job Submission Example

The job defined in “[Extended Job Example](#)” on page 62 has the following additional characteristics as compared to the job definition in “[Submitting Extended Jobs With QMON](#)” on page 59.

- The job requires the use of the parallel environment `mpi`. The job needs at least 4 parallel processes to be created. The job can use up to 16 processes if the processes are available.
- Two environment variables are set and exported for the job.
- Two context variables are set.
- The account string `FLOW` is to be added to the job accounting record.
- Mail must be sent to `me@myhost.org` as soon as the job starts and finishes.
- The job should preferably be executed in the queue `big_q`.

Submitting Advanced Jobs From the Command Line

To submit the advanced job request that is shown in [Figure 3-6](#) from the command line, type the following command:

```
% qsub -N Flow -p -111 -P devel -a 200012240000.00 -cwd \
-S /bin/tcsh -o flow.out -j y -pe mpi 4-16 \
-v SHARED_MEM=TRUE,MODEL_SIZE=LARGE \
-ac JOB_STEP=preprocessing,PORT=1234 \
-A FLOW -w w -m s,e -q big_q\
-M me@myhost.com,me@other.address \
flow.sh big.data
```

Default Request Files

The preceding command shows that advanced job requests can be rather complex and unwieldy, in particular if similar requests need to be submitted frequently. To avoid the cumbersome and error-prone task of entering such commands, users can embed qsub options in the script files, or use *default request files*. For more information, see [“Active Comments” on page 56](#).

Note – The `-binary yes|no` option when specified with the `y` argument, allows you to use `qrun` to submit executable jobs without the script wrapper. See the `qsub` man page.

The cluster administration can set up a default request file for all grid engine system users. Users, on the other hand, can create private default request files located in their home directories. Users can also create application-specific default request files that are located in their working directories.

Default request files contain the qsub options to apply by default to the jobs in one or more lines. The location of the global cluster default request file is `sge-root/cell/common/sge_request`. The private general default request file is located under `$HOME/.sge_request`. The application-specific default request files are located under `$cwd/.sge_request`.

If more than one of these files are available, the files are merged into one default request, with the following order of precedence:

1. Application-specific default request file
2. General private default request file
3. Global default request file

Script embedding and the qsub command line have higher precedence than the default request files. Therefore, script embedding overrides default request file settings. The qsub command line options can override these settings again.

To discard any previous settings, use the `qsub -clear` command in a default request file, in embedded script commands, or in the qsub command line.

Here is an example of a private default request file:

```
-A myproject -cwd -M me@myhost.com -m b e  
-r y -j y -S /bin/ksh
```

Unless overridden, for all of this user's jobs the following is true:

- The account string is `myproject`
- The jobs execute in the current working directory
- Mail notification is sent to `me@myhost.com` at the beginning and at the end of the jobs
- The standard output and standard error output are merged
- The `ksh` is used as command interpreter

Defining Resource Requirements

In the examples so far, the submit options do not express any resource requirements for the hosts on which the jobs are to be executed. The grid engine system assumes that such jobs can be run on any host. In practice, however, most jobs require that certain prerequisites be met on the executing host in order for the job to finish successfully. Such prerequisites include enough available memory, required software to be installed, or a certain operating system architecture. Also, the cluster administration usually imposes restrictions on the use of the machines in the cluster. For example, the CPU time that can be consumed by the jobs is often restricted.

The grid engine system provides users with the means to find suitable hosts for their jobs without precise knowledge of the cluster's equipment and its usage policies. Users specify the requirement of their jobs and let the grid engine system manage the task of finding a suitable and lightly loaded host.

You specify resource requirements through *requestable attributes*, which are described in [“Requestable Attributes” on page 42](#). QMON provides a convenient way to specify the requirements of a job. The Requested Resources dialog box displays only those attributes in the Available Resource list that are currently eligible. Click Request Resources in the Submit Job dialog box to open the Requested Resources dialog box. See [Figure 3–7](#) for an example.

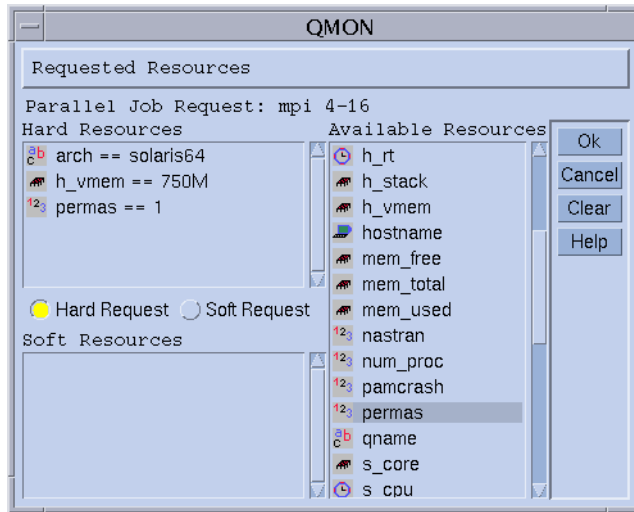


FIGURE 3-7 Requested Resources Dialog Box

When you double-click an attribute, the attribute is added to the Hard or Soft Resources list of the job. A dialog box opens to guide you in entering a value specification for the attribute in question, except for BOOLEAN attributes, which are set to True. For more information, see [“How the Grid Engine System Allocates Resources” on page 70](#).

Figure 3-7 shows a resource profile for a job that requests a `solaris64` host with an available `permas` license offering at least 750 MBytes of memory. If more than one queue that fulfills this specification is found, any defined soft resource requirements are taken into account. However, if no queue satisfying both the hard and the soft requirements is found, any queue that grants the hard requirements is considered suitable.

Note – The `queue_sort_method` parameter of the scheduler configuration determines where to start the job only if more than one queue is suitable for a job. See the `sched_conf(5)` man page for more information.

The attribute `permas`, an integer, is an administrator extension to the global resource attributes. The attribute `arch`, a string, is a host resource attribute. The attribute `h_vmem`, memory, is a queue resource attribute.

An equivalent resource requirement profile can as well be submitted from the `qsub` command line:

```
% qsub -l arch=solaris64,h_vmem=750M,permas=1 \
    permas.sh
```

The implicit `-hard` switch before the first `-l` option has been skipped.

The notation `750M` for 750 MBytes is an example of the quantity syntax of the grid engine system. For those attributes that request a memory consumption, you can specify either integer decimal, floating-point decimal, integer octal, and integer hexadecimal numbers. The following multipliers must be appended to these numbers:

- `k` – Multiplies the value by 1000
- `K` – Multiplies the value by 1024
- `m` – Multiplies the value by 1000 times 1000
- `M` – Multiplies the value by 1024 times 1024

Octal constants are specified by a leading zero and digits ranging from 0 to 7 only. To specify a hexadecimal constant, you must prefix the number with `0x`. You must also use digits ranging from 0 to 9, a through f, and A through F. If no multipliers are appended, the values are considered to count as bytes. If you are using floating-point decimals, the resulting value is truncated to an integer value.

For those attributes that impose a time limit, you can specify time values in terms of hours, minutes, or seconds, or any combination. Hours, minutes, and seconds are specified in decimal digits separated by colons. A time of `3:5:11` is translated to 11111 seconds. If zero is a specifier for hours, minutes, or seconds, you can leave it out if the colon remains. Thus a value of `:5:` is interpreted as 5 minutes. The form used in the Requested Resources dialog box that is shown in [Figure 3–7](#) is an extension, which is valid only within QMON.

How the Grid Engine System Allocates Resources

As shown in the previous section, knowing how grid engine software processes resource requests and allocates resources is important. The schematic view of grid engine software's resource allocation algorithm is as follows.

1. Read in and parse all default request files. See [“Default Request Files” on page 67](#) for details.
2. Process the script file for embedded options. See [“Active Comments” on page 56](#) for details.
3. Read all script-embedding options when the job is submitted, regardless of their position in the script file.
4. Read and parse all requests from the command line.

As soon as all `qsub` requests are collected, *hard* and *soft* requests are processed separately, the hard requests first. The requests are evaluated, according to the following order of precedence:

1. From left to right of the script or default request file
2. From top to bottom of the script or default request file
3. From left to right of the command line

In other words, you can use the command line to override the embedded flags.

The resources requested as hard are allocated. If a request is not valid, the submission is rejected. If one or more requests cannot be met at submit time, the job is spooled and

rescheduled to be run at a later time. A request might not be met, for example, if a requested queue is busy. If all hard requests can be met, the requests are allocated and the job can be run.

The resources requested as soft are checked. The job can run even if some or all of these requests cannot be met. If multiple queues that meet the hard requests provide parts of the soft resources list, the grid engine software selects the queues that offer the most soft requests.

The job is started and covers the allocated resources.

You might want to gather experience of how argument list options and embedded options or hard and soft requests influence each other. You can experiment with small test script files that execute UNIX commands such as `hostname` or `date`.

Job Dependencies

Often the most convenient way to build a complex task is to split the task into subtasks. In these cases, subtasks depend on the completion of other subtasks before the dependent subtasks can get started. An example is that a predecessor task produces an output file that must be read and processed by a dependent task.

The grid engine system supports interdependent tasks with its job dependency facility. You can configure jobs to depend on the completion of one or more other jobs. The facility is enforced by the `qsub -hold_jid` command. You can specify a list of jobs upon which the submitted job depends. The list of jobs can also contain subsets of array jobs. The submitted job is not eligible for execution unless all jobs in the dependency list have finished.

Submitting Array Jobs

Parameterized and repeated execution of the same set of operations that are contained in a job script is an ideal application for the *array job* facility of the grid engine system. Typical examples of such applications are found in the Digital Content Creation industries for tasks such as rendering. Computation of an animation is split into frames. The same rendering computation can be performed for each frame independently.

The array job facility offers a convenient way to submit, monitor, and control such applications. The grid engine system provides an efficient implementation of array jobs, handling the computations as an array of independent tasks joined into a single job. The tasks of an array job are referenced through an array index number. The indexes for all tasks span an index range for the entire array job. The index range is defined during submission of the array job by a single `qsub` command.

You can monitor and control an array job. For example, you can suspend, resume, or cancel an array job as a whole or by individual task or subset of tasks. To reference the tasks, the corresponding index numbers are suffixed to the job ID. Tasks are executed very much like

regular jobs. Tasks can use the environment variable `SGE_TASK_ID` to retrieve their own task index number and to access input data sets designated for this task identifier.

Submitting an Array Job With QMON

Follow the instructions in [“How To Submit a Simple Job With QMON” on page 49](#), additionally taking into account the following information.

The submission of array jobs from QMON works virtually identically to how the submission of a simple job is described in [“How To Submit a Simple Job With QMON” on page 49](#). The only difference is that the Job Tasks input window that is shown in [Figure 3–5](#) must contain the task range specification. The task range specification uses syntax that is identical to the `qsub -t` command. See the `qsub(1)` man page for detailed information about array index syntax.

For information about monitoring and controlling jobs in general, and about array jobs in particular, see [“Monitoring and Controlling Jobs” on page 85](#) and [“Monitoring and Controlling Jobs From the Command Line” on page 94](#). See also the man pages for `qstat(1)`, `qhold(1)`, `qrts(1)`, `qmod(1)`, and `qdel(1)`.

Array jobs offer full access to all facilities of the grid engine system that are available for regular jobs. In particular, array jobs can be parallel jobs at the same time. Array jobs also can have interdependencies with other jobs.

Note – Array tasks cannot have interdependencies with other jobs or with other array tasks.

Submitting an Array Job From the Command Line

To submit an array job from the command line, type the `qsub` command with appropriate arguments.

The following is an example of how to submit an array job:

```
% qsub -l h_cpu=0:45:0 -t 2-10:2 render.sh data.in
```

The `-t` option defines the task index range. In this case, `2-10:2` specifies that 2 is the lowest index number, and 10 is the highest index number. Only every second index, the `:2` part of the specification, is used. Thus, the array job is made up of 5 tasks with the task indices 2, 4, 6, 8, and 10. Each task requests a hard CPU time limit of 45 minutes with the `-l` option. Each task executes the job script `render.sh` once the task is dispatched and started by the grid engine system. Tasks can use `SGE_TASK_ID` to find their index number, which they can use to find their input data record in the data file `data.in`.

Submitting Interactive Jobs

The submission of interactive jobs instead of batch jobs is useful in situations where a job requires your direct input to influence the job results. Such situations are typical for X Windows applications or for tasks in which your interpretation of immediate results is required to steer further processing.

You can create interactive jobs in three ways:

- `qlogin` – A telnet-like session that is started on a host selected by grid engine software.
- `qssh` – The equivalent of the standard UNIX `rsh` facility. A command is run remotely on a host selected by the grid engine system. If no command is specified, a remote `rlogin` session is started on a remote host.
- `qsh` – An `xterm` that is displayed from the machine that is running the job. The display is set corresponding to your specification or to the setting of the `DISPLAY` environment variable. If the `DISPLAY` variable is not set, and if no display destination is defined, the grid engine system directs the `xterm` to the 0.0 screen of the X server on the host from which the job was submitted.

Note – To function correctly, all the facilities need proper configuration of cluster parameters of the grid engine system. The correct `xterm` execution paths must be defined for `qsh`. Interactive queues must be available for this type of job. Contact your system administrator to find out if your cluster is prepared for interactive job execution.

The default handling of interactive jobs differs from the handling of batch jobs. Interactive jobs are not queued if the jobs cannot be executed when they are submitted. A job's not being queued indicates immediately that not enough appropriate resources are available to dispatch an interactive job at the time the job is submitted. The user is notified in such cases that the cluster is currently too busy.

You can change this default behavior with the `-now no` option to `qsh`, `qlogin`, and `qssh`. If you use this option, interactive jobs are queued like batch jobs. When you use the `-now yes` option, batch jobs that are submitted with `qsub` can also be handled like interactive jobs. Such batch jobs are either dispatched for running immediately, or they are rejected.

Note – Interactive jobs can be run only in queues of the type `INTERACTIVE`. See “Configuring Queues” in *Sun N1 Grid Engine 6.1 Administration Guide* for details.

The following sections describe how to use the `qlogin` and `qsh` facilities. The `qssh` command is explained in a broader context in [“Transparent Remote Execution” on page 76](#).

Submitting Interactive Jobs With QMON

The only type of interactive jobs that you can submit from QMON are jobs that bring up an xterm on a host selected by the grid engine system.

At the right side of the Submit Job dialog box, click the button above the Submit button until the Interactive icon is displayed. Doing so prepares the Submit Job dialog box to submit interactive jobs. See [Figure 3–8](#) and [Figure 3–9](#).

The meaning and the use of the selection options in the dialog box is the same as that described for batch jobs in “[Submitting Batch Jobs](#)” on page 53. The difference is that several input fields are grayed out because those fields do not apply to interactive jobs

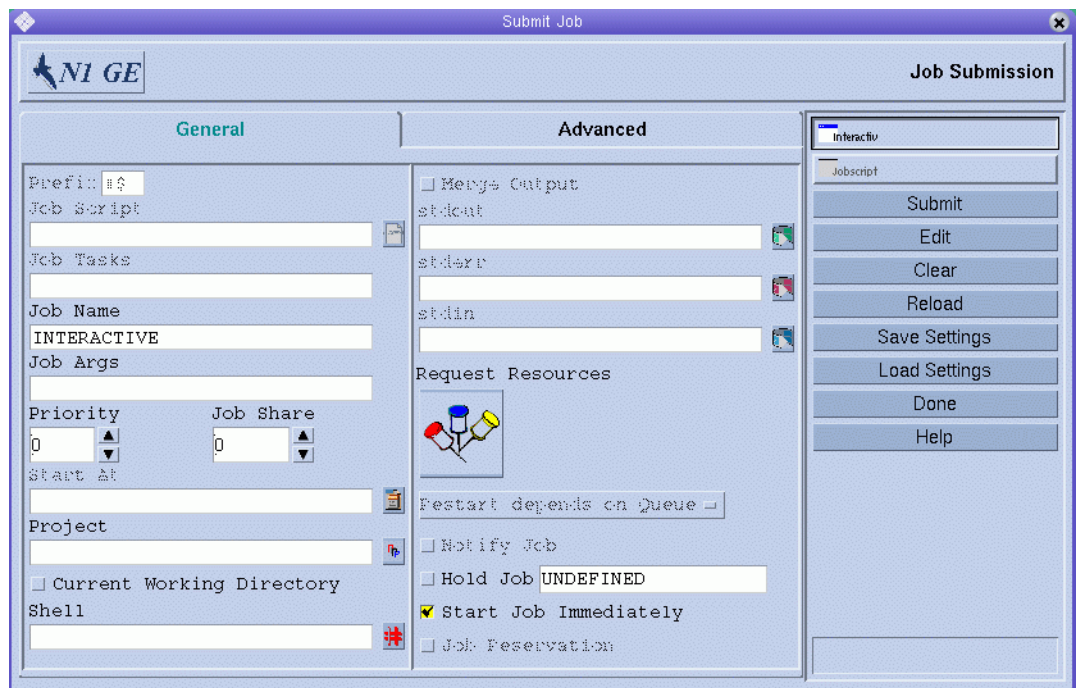


FIGURE 3–8 Interactive Submit Job Dialog Box, General Tab

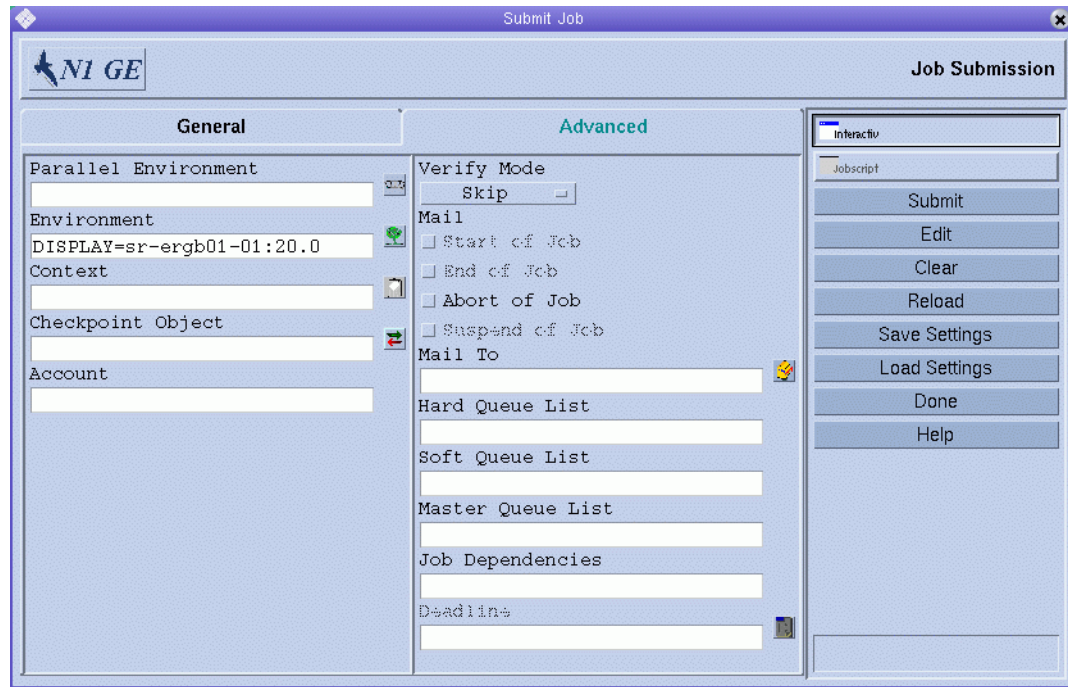


FIGURE 3-9 Interactive Submit Job Dialog Box, Advanced Tab

Submitting Interactive Jobs With `qsh`

`qsh` is very similar to `qsub`. `qsh` supports several of the `qsub` options, as well as the additional option `-display` to direct the display of the `xterm` to be invoked. See the `qsub(1)` man page for details.

To submit an interactive job with `qsh`, type a command like the following:

```
% qsh -l arch=solaris64
```

This command starts an `xterm` on any available Sun Solaris 64-bit operating system host.

Submitting Interactive Jobs With `qlogin`

Use the `qlogin` command from any terminal or terminal emulation to start an interactive session under the control of the grid engine system.

To submit an interactive job with `qlogin`, type a command like the following:

```
% qlogin -l star-cd=1,h_cpu=6:0:0
```

This command locates a low-loaded host. The host has a Star-CD license available. The host also has at least one queue that can provide a minimum of six hours hard CPU time limit.

Note – Depending on the remote login facility that is configured to be used by the grid engine system, you might have to provide your user name, your password, or both, at a login prompt.

Transparent Remote Execution

The grid engine system provides a set of closely related facilities that support the transparent remote execution of certain computational tasks. The core tool for this functionality is the `qrsh` command, which is described in [“Remote Execution With `qrsh`” on page 76](#). Two high-level facilities, `qtcsh` and `qmake`, build on top of `qrsh`. These two commands enable the grid engine system to transparently distribute implicit computational tasks, thereby enhancing the standard UNIX facilities `make` and `csh`. `qtcsh` is described in [“Transparent Job Distribution With `qtcsh`” on page 77](#). `qmake` is described in [“Parallel Makefile Processing With `qmake`” on page 79](#).

Remote Execution With `qrsh`

`qrsh` is built around the standard `rsh` facility. See the information that is provided in *sge-root/3rd_party* for details on the involvement of `rsh`. `qrsh` can be used for various purposes, including the following:

- To provide remote execution of interactive applications that use the grid engine system comparable to the standard UNIX facility `rsh`. `rsh` is also called `remsh` on HP-UX systems.
- To offer interactive login session capabilities that use the grid engine system, similar to the standard UNIX facility `rlogin`. `qlogin` is still required as a grid engine system's representation of the UNIX `telnet` facility.
- To allow for the submission of batch jobs that support terminal I/O (standard output, standard error, and standard input) and terminal control.
- To provide a way to submit a standalone program that is not embedded in a shell script.

Note – You can also submit scripts with `qrsh` by using the `-b n` option. For more information, see the `qrsh` man page.

- To provide a submission client that remains active while a batch job is pending or running and that goes away only if the job finishes or is cancelled.

- To allow for the grid engine system-controlled remote running of job tasks within the framework of the dispersed resources allocated by parallel jobs. See “Tight Integration of Parallel Environments and Grid Engine Software” in *Sun N1 Grid Engine 6.1 Administration Guide*.

By virtue of these capabilities, `qrsh` is the major enabling infrastructure for the implementation of the `qtcsh` and the `qmake` facilities. `qrsh` is also used for the tight integration of the grid engine system with parallel environments such as MPI or PVM.

Invoking Transparent Remote Execution With `qrsh`

Type the `qrsh` command, adding options and arguments according to the following syntax:

```
% qrsh [options] program|shell-script [arguments] \
      [> stdout] [>&2 stderr] [< stdin]
```

`qrsh` understands almost all options of `qsub`. `qrsh` provides the following options:

- `-now yes|no` – `-now yes` specifies that the job is scheduled immediately. The job is rejected if no appropriate resources are available. `-now yes` is the default. `-now no` specifies that the job is queued like a batch job if the job cannot be started at submission time.
- `-inherit` – `qrsh` does not go through the scheduling process to start a job-task. Instead, `qrsh` assumes that the job is embedded in a parallel job that already has allocated suitable resources on the designated remote execution host. This form of `qrsh` is commonly used in `qmake` and in a tight parallel environment integration. The default is not to inherit external job resources.
- `-binary yes|no` – When specified with the `n` option, enables you to use `qrsh` to submit script jobs.
- `-noshell` – With this option, you do not start the command line that is given to `qrsh` in a user's login shell. Instead, you execute the command without the wrapping shell. Use this option to speed up execution, as some overhead, such as the shell startup and the sourcing of shell resource files, is avoided.
- `-nostdin` – Suppresses the input stream `STDIN`. With this option set, `qrsh` passes the `-n` option to the `rsh` command. Suppression of the input stream is especially useful if multiple tasks are executed in parallel using `qrsh`, for example, in a `make` process. Which process gets the input is undefined.
- `-verbose` – This option presents output on the scheduling process. `-verbose` is mainly intended for debugging purposes and is therefore switched off by default.

Transparent Job Distribution With `qtcsh`

`qtcsh` is a fully compatible replacement for the widely known and used UNIX C shell derivative `tcsh`. `qtcsh` is built around `tcsh`. See the information that is provided in *sge-root/3rd_party* for details on the involvement of `tcsh`. `qtcsh` provides a command shell with the extension of

transparently distributing execution of designated applications to suitable and lightly loaded hosts that use the grid engine system. The `.qtask` configuration files define the applications to execute remotely and the requirements that apply to the selection of an execution host.

These applications are transparent to the user and are submitted to the grid engine system through the `qrsh` facility. `qrsh` provides standard output, error output, and standard input handling as well as terminal control connection to the remotely executing application. Three noticeable differences between running such an application remotely and running the application on the same host as the shell are:

- The remote host might be more powerful, lower-loaded, and have required hardware and software resources installed. Therefore, such a remote host would be much better suited than the local host, which might not allow running the application at all.
- A small delay is incurred by the remote startup of the jobs and by their handling through the grid engine system.
- Administrators can restrict the use of resources through interactive jobs (`qrsh`) and thus through `qtcsh`. If not enough suitable resources are available for an application to be started through `qrsh`, or if all suitable systems are overloaded, the implicit `qrsh` submission fails. A corresponding error message is returned, such as `Not enough resources . . . try later`.

In addition to the standard use, `qtcsh` is a suitable platform for third-party code and tool integration. The single-application execution form of `qtcsh` is `qtcsh -c app-name`. The use of this form of `qtcsh` inside integration environments presents a persistent interface that almost never needs to be changed. All the required application, tool, integration, site, and even user-specific configurations are contained in appropriately defined `.qtask` files. A further advantage is that this interface can be used in shell scripts of any type, in C programs, and even in Java applications.

qtcsh Usage

The invocation of `qtcsh` is exactly the same as for `tcsh`. `qtcsh` extends `tcsh` in providing support for the `.qtask` file and by offering a set of specialized shell built-in modes.

The `.qtask` file is defined as follows. Each line in the file has the following format:

```
% [!]app-name qrsh-options
```

The optional leading exclamation mark (!) defines the precedence between conflicting definitions in a global cluster `.qtask` file and the personal `.qtask` file of the `qtcsh` user. If the exclamation mark is missing in the global cluster file, a conflicting definition in the user file overrides the definition in the global cluster file. If the exclamation mark is in the global cluster file, the corresponding definition cannot be overridden.

app-name specifies the name of the application that, when typed on a command line in a `qtcsh`, is submitted to the grid engine system for remote execution.

qrsh-options specifies the options to the *qrsh* facility to use. These options define resource requirements for the application.

The application name must appear in the command line exactly as the application is defined in the *.qtask* file. If the application name is prefixed with a path name, a local binary is addressed. No remote execution is intended.

qsh aliases are expanded before a comparison with the application names is performed. The applications intended for remote execution can also appear anywhere in a *qtcsh* command line, in particular before or after standard I/O redirections.

Hence, the following examples are valid and meaningful syntax:

```
# .qtask file
netscape -v DISPLAY=myhost:0
grep -l h=filesurfer
```

Given this *.qtask* file, the following *qtcsh* command lines:

```
netscape
~/mybin/netscape
cat very_big_file | grep pattern | sort | uniq
```

implicitly result in:

```
qrsh -v DISPLAY=myhost:0 netscape
~/mybin/netscape
cat very_big_file | qrsh -l h=filesurfer grep pattern | sort | uniq
```

qtcsh can operate in different modes, influenced by switches that can be set on or off:

- Local or remote execution of commands. Remote is the default.
- Immediate or batch remote execution. Immediate is the default.
- Verbose or nonverbose output. Nonverbose is the default.

The setting of these modes can be changed using option arguments of *qtcsh* at start time or with the shell built-in command *qrshmode* at runtime. See the *qtcsh(1)* man page for more information.

Parallel Makefile Processing With *qmake*

qmake is a replacement for the standard UNIX *make* facility. *qmake* extends *make* by enabling the distribution of independent *make* steps across a cluster of suitable machines. *qmake* is built around the popular GNU-*make* facility *gmake*. See the information that is provided in *sgc-root/3rd_party* for details on the involvement of *gmake*.

To ensure that a distributed make process can run to completion, `qmake` first allocates the required resources in a way analogous to a parallel job. `qmake` then manages this set of resources without further interaction with the scheduling. `qmake` distributes make steps as resources become available, using the `qrsh` facility with the `-inherit` option.

`qrsh` provides standard output, error output, and standard input handling as well as terminal control connection to the remotely executing make step. Therefore, only three noticeable differences exist between executing a make procedure locally and using `qmake`:

- Provided that individual make steps have a certain duration and that enough independent make steps exist to process, the parallelization of the make process will speed up significantly.
- In the make steps to be started up remotely, an implied small overhead exists that is caused by `qrsh` and the remote execution.
- To take advantage of the make step distribution of `qmake`, the user must specify as a minimum the degree of parallelization. That is, the user must specify the number of concurrently executable make steps. In addition, the user can specify the resource characteristics required by the make steps, such as available software licenses, machine architecture, memory, or CPU-time requirements.

The most common use of `make` is the compilation of complex software packages. Compilation might not be the major application for `qmake`, however. Program files are often quite small as a matter of good programming practice. Therefore, compilation of a single program file, which is a single make step, often takes only a few seconds. Furthermore, compilation usually implies significant file access. Nested include files can cause this problem. File access might not be accelerated if done for multiple make steps in parallel because the file server can become a bottleneck. Such a bottleneck effectively serializes all the file access. Therefore, the compilation process sometimes cannot be accelerated in a satisfactory manner.

Other potential applications of `qmake` are more appropriate. An example is the steering of the interdependencies and the workflow of complex analysis tasks through makefiles. Each make step in such environments is typically a simulation or data analysis operation with nonnegligible resource and computation time requirements. A considerable acceleration can be achieved in such cases.

qmake Usage

The command-line syntax of `qmake` looks similar to the syntax of `qrsh`:

```
% qmake [-pe pe-name pe-range] [options] \  
-- [gnu-make-options] [target]
```

Note – The `-inherit` option is also supported by `qmake`, as described later in this section.

Pay special attention to the use of the `-pe` option and its relation to the `gmake -j` option. You can use both options to express the amount of parallelism to be achieved. The difference is that `gmake` provides no possibility with `-j` to specify something like a parallel environment to use. Therefore, `qmake` assumes that a default environment for parallel makes is configured that is called `make`. Furthermore, `gmake 's -j` allows for no specification of a range, but only for a single number. `qmake` interprets the number that is given with `-j` as a range of $1-n$. By contrast, `-pe` permits the detailed specification of all these parameters. Consequently the following command line examples are identical:

```
% qmake -- -j 10
% qmake -pe make 1-10 --
```

The following command lines cannot be expressed using the `-j` option:

```
% qmake -pe make 5-10,16 --
% qmake -pe mpi 1-99999 --
```

Apart from the syntax, `qmake` supports two modes of invocation: interactively from the command line without the `-inherit` option, or within a batch job with the `-inherit` option. These two modes start different sequences of actions:

- **Interactive** – When `qmake` is invoked on the command line, the `make` process is implicitly submitted to the grid engine system with `qsh`. The process takes the resource requirements that are specified in the `qmake` command line into account. The grid engine system then selects a *master machine* for the execution of the parallel job that is associated with the parallel `make` job. The grid engine system starts the `make` procedure there. The procedure must start there because the `make` process can be architecture-dependent. The required architecture is specified in the `qmake` command line. The `qmake` process on the master machine then delegates execution of individual `make` steps to the other hosts that are allocated for the job. The steps are passed to `qmake` through the parallel environment hosts file.
- **Batch** – In this case, `qmake` appears inside a batch script with the `-inherit` option. If the `-inherit` option is not present, a new job is spawned, as described in the first case earlier. This results in `qmake` making use of the resources already allocated to the job into which `qmake` is embedded. `qmake` uses `qsh -inherit` directly to start `make` steps. When calling `qmake` in batch mode, the specification of resource requirements, the `-pe` option and the `-j` option are ignored.

Note – Single CPU jobs also must request a parallel environment:

```
qmake -pe make 1 --
```

If no parallel execution is required, call `qmake` with `gmake` command-line syntax without grid engine system options and without `--`. This `qmake` command behaves like `gmake`.

See the `qmake(1)` man page for further details.

How Jobs Are Scheduled

The grid engine software's policy management automatically controls the use of shared resources in the cluster to best achieve the goals of the administration. High priority jobs are dispatched preferentially. Such jobs receive better access to resources. The administration of a cluster can define high-level usage policies. The following policies are available:

- **Functional** – Special treatment is given because of affiliation with a certain user group, project, and so forth.
- **Share-based** – Level of service depends on an assigned share entitlement, the corresponding shares of other users and user groups, the past usage of resources by all users, and the current presence of users in the system.
- **Urgency** – Preferential treatment is given to jobs that have greater urgency. A job's urgency is based on its resource requirements, how long the job must wait, and whether the job is submitted with a deadline requirement.
- **Override** – Manual intervention by the cluster administrator modifies the automated policy implementation.

The grid engine software can be set up to routinely use either a share-based policy, a functional policy, or both. These policies can be combined in any proportion, from giving zero weight to one policy and using only the second policy, to giving both policies equal weight.

Along with the routine policies, jobs can be submitted with an initiation deadline. See the description of the deadline submission parameter under [“Submitting Advanced Jobs With QMON” on page 63](#). Deadline jobs disturb routine scheduling. Administrators can also temporarily *override* share-based scheduling and functional scheduling. An override can be applied to an individual job, or to all jobs associated with a user, a department, or a project.

Job Priorities

In addition to the four policies for mediating among all jobs, the grid engine software sometimes lets users set priorities among their own jobs. A user who submits several jobs can specify, for example, that job 3 is the most important and that jobs 1 and 2 are equally important but less important than job 3.

Priorities for jobs are set by using the QMON Submit Job parameter Priority or by using the `qsub -p` option. A priority range of -1024 (lowest) to 1023 (highest) can be given. This priority tells the scheduler how to choose among a single user's jobs when several of that user's jobs are in the system simultaneously. The relative importance assigned to a particular job depends on the maximum and minimum priorities that are given to any of that user's jobs, and on the priority value of the specific job.

Ticket Policies

The functional policy, the share-based policy, and the override policy are all implemented with *tickets*. Each ticket policy has a ticket pool from which tickets are allocated to jobs that are entering the multimachine grid engine system. Each routine ticket policy that is in force allocates some tickets to each new job. The ticket policy can reallocate tickets to the executing job at each scheduling interval. The criteria that each ticket policy uses to allocate tickets are explained in this section.

Tickets weight the three policies. For example, if no tickets are allocated to the functional policy, that policy is not used. If an equal number of tickets are assigned to the functional ticket pool and to the share-based ticket pool, both policies have equal weight in determining a job's importance.

Grid engine managers allocate tickets to the routine ticket policies at system configuration. Managers and operators can change ticket allocations at any time. Additional tickets can be injected into the system temporarily to indicate an override. Ticket policies are combined by assignment of tickets: when tickets are allocated to multiple ticket policies, a job gets a portion of its tickets from each ticket policy in force.

The grid engine system grants tickets to jobs that are entering the system to indicate their importance under each ticket policy in force. Each running job can gain tickets, for example, from an override; lose tickets, for example, because the job is getting more than its fair share of resources; or keep the same number of tickets at each scheduling interval. The number of tickets that a job holds represents the resource share that the grid engine system tries to grant that job during each scheduling interval.

You can display the number of tickets a job holds with `QMON` or using `qstat -ext`. See [“Monitoring and Controlling Jobs With QMON” on page 85](#). The `qstat` command also displays the priority value assigned to a job, for example, using `qsub -p`. See the `qstat(1)` man page for more details.

Queue Selection

The grid engine system does not dispatch jobs that request nonspecific queues if the jobs cannot be started immediately. Such jobs are marked as spooled at the `sge_qmaster`, which tries to reschedule the jobs from time to time. The jobs are dispatched to the next suitable queue that becomes available.

As opposed to spooling jobs, jobs that are submitted to a certain queue by name go directly to the named queue, regardless of whether the jobs can be started or need to be spooled. Therefore, viewing the queues of the grid engine system as computer science *batch queues* is valid only for jobs requested by name. Jobs submitted with nonspecific requests use the spooling mechanism of `sge_qmaster` for queueing, thus using a more abstract and flexible queuing concept.

If a job is scheduled and multiple free queues meet its resource requests, the job is usually dispatched to a suitable queue belonging to the least loaded host. By setting the scheduler configuration entry `queue_sort_method` to `seq_no`, the cluster administration can change this load-dependent scheme into a fixed order algorithm. The queue configuration entry `seq_no` defines a precedence among the queues, assigning the highest priority to the queue with the lowest sequence number.

Monitoring and Controlling Jobs and Queues

After you submit jobs, you need to monitor and control them. This chapter provides background information about monitoring, and controlling jobs and queues, as well as instructions for how to do these tasks. The chapter also includes information about job checkpointing.

This chapter includes instructions for the following tasks:

- “Monitoring and Controlling Jobs With QMON” on page 85
- “Monitoring Jobs With qstat” on page 94
- “Controlling Jobs With qdel and qmod” on page 97
- “Monitoring and Controlling Queues With QMON” on page 98
- “Controlling Queues With qmod” on page 105
- “Submitting, Monitoring, or Deleting a Checkpointing Job From the Command Line” on page 108
- “Submitting a Checkpointing Job With QMON” on page 109

Monitoring and Controlling Jobs

You can monitor and control submitted jobs in three ways:

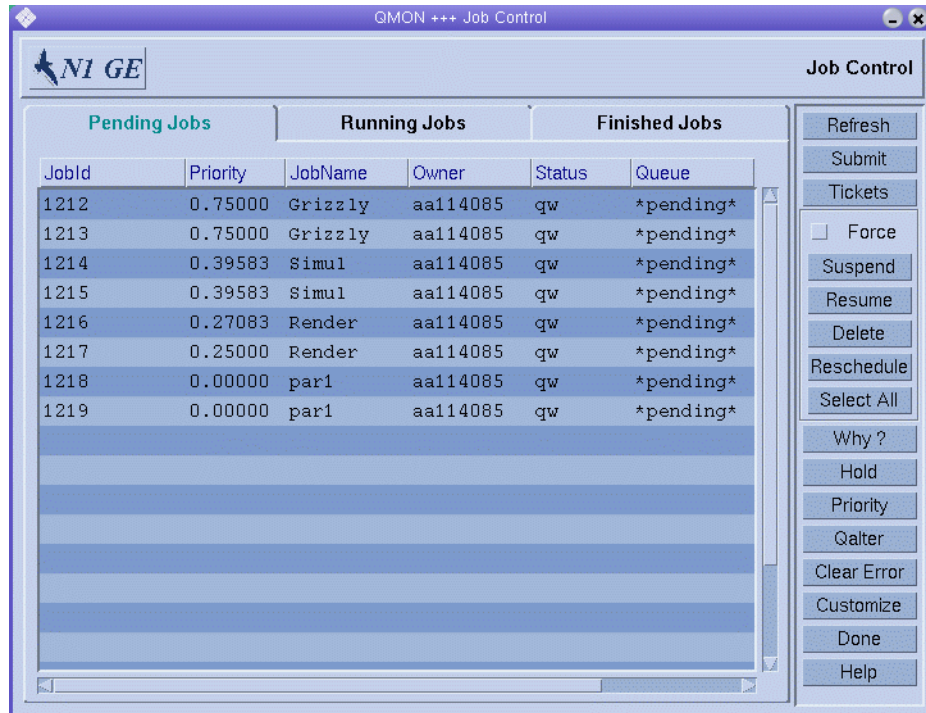
- With QMON
- From the command line with the qstat, qdel, and qmod commands
- By email

The following sections describe each of these methods.

Monitoring and Controlling Jobs With QMON

You use the QMON Job Control dialog box to control jobs.

To monitor and control your submitted jobs, in the QMON Main Control window click the Job Control button. The Job Control dialog box appears.



The Job Control dialog box has three tabs, a tab for Running Jobs, a tab for Pending Jobs that are waiting to be dispatched to an appropriate resource, and a tab for recently Finished Jobs.

The Submit button provides a link to the Submit Job dialog box.

The Job Control dialog box enables you to monitor all running, pending, and finished jobs that are known to the system. You can also use this dialog box to manage jobs. You can change a job's priority. You can also suspend, resume, and cancel jobs.

In its default format, the Job Control dialog box displays the following columns for each running and pending job:

- JobId
- Priority
- JobName
- Owner
- Status
- Queue

You can change the default display by customizing the format. See [“Customizing the Job Control Display” on page 90](#) for details.

Refreshing the Job Control Display

To keep the displayed information up-to-date, QMON uses a polling scheme to retrieve the status of the jobs from `sge_qmaster`. Click Refresh to force an update of the Job Control display.

Selecting Jobs

You can select jobs with the following mouse and key combinations:

- To select multiple noncontiguous jobs, hold down the Control key and click two or more jobs.
- To select a contiguous range of jobs, hold down the Shift key, click the first job in the range, and then click the last job in the range.
- To toggle between selecting a job and clearing the selection, click the job while holding down the Control key.

You can also use a filter to select the jobs that you want to display. See [“Filtering the Job List” on page 92](#) for details.

Managing Jobs

You can use the buttons at the right of the dialog box to manage selected jobs in the following ways:

- Suspend
- Resume (unsuspend)
- Delete
- Hold back
- Release
- Reprioritize
- Reschedule
- Modify with `qalter`

Only the job owner or grid engine managers and operators can suspend and resume jobs, delete jobs, hold back jobs, modify job priority, and modify jobs. See [“Managers, Operators, and Owners” on page 38](#). Only running jobs can be suspended or resumed. Only pending jobs can be rescheduled, held back and modified, in priority as well as in other attributes.

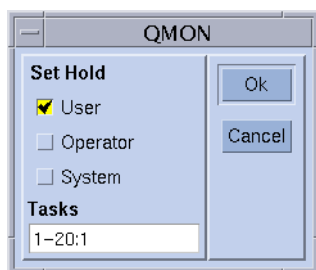
Suspension of a job sends the signal `SIGSTOP` to the process group of the job with the UNIX `kill` command. `SIGSTOP` halts the job and no longer consumes CPU time. Resumption of the job sends the signal `SIGCONT`, thereby unsuspending the job. See the `kill(1)` man page for your system for more information on signalling processes.

Note – You can force suspending, resuming, and deleting jobs. In other words, you can register these actions with `sge_qmaster` without notifying the `sge_execd` that controls the jobs. Forcing is useful when the corresponding `sge_execd` is unreachable, for example, due to network problems. Select the Force option for this purpose.

Click Reschedule to reschedule a currently running job.

Putting Jobs on Hold

To put a job on hold, select a pending job and click Hold. The Set Hold dialog box appears.



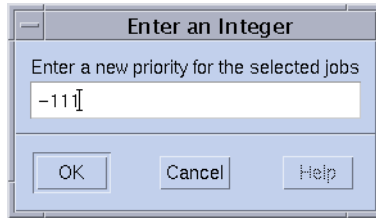
The Set Hold dialog box enables setting and resetting user, operator, and system holds. User holds can be set or reset by the job owner as well as by grid engine managers and operators. Operator holds can be set or reset by managers and operators. System holds can be set or reset by managers only. As long as any hold is assigned to a job, the job is not eligible for running. You can also set or reset holds by using the `qalter`, `qhold`, and `qrls` commands.

Putting Array Job Tasks on Hold

The Tasks field on the Set Hold dialog box applies to Array jobs. Use this button to put a hold on particular subtasks of an array job. Note the format of the text in the Tasks field. The task ID range specified in this field can be a single number, a simple range of the form *n-m*, or a range with a step size. The task ID range specified by, for example, `2-10:2` results in the task ID indexes 2, 4, 6, 8, and 10. This range represents a total of five identical tasks, with the environment variable `SGE_TASK_ID` containing one of the five index numbers. For detailed information about job holds, see the `qsub(1)` man page.

Changing Job Priority

When you click Priority on the Job Control dialog box, the following dialog box appears.



This dialog box enables you to provide the new priority of selected pending or running jobs. The priority ranks a single user's jobs among themselves. Priority tells the scheduler how to choose among a single user's jobs when several jobs are in the system simultaneously.

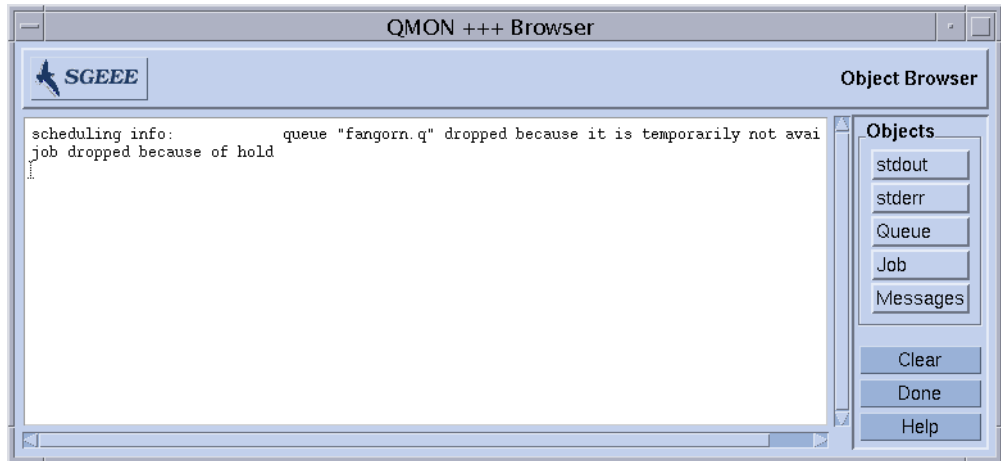
When you select a pending job and click `Qalter`, the Submit Job window appears. All the entries of the dialog box are set corresponding to the attributes of the job that were defined when the job was submitted. Entries that cannot be changed are grayed out. The other entries can be edited. The changes are registered with the grid engine system when you click `Qalter` on the Submit Job dialog box. The `Qalter` button is a substitute for the Submit button.

Verifying Job Consistency

The Verify flag on the Submit Job dialog box has a special meaning when the flag is used in the `Qalter` mode. You can check pending jobs for their consistency, and you can investigate why jobs are not yet scheduled. Select the desired consistency-checking mode for the Verify flag, and then click `Qalter`. The system displays warnings on inconsistencies, depending on the checking mode you select. See [“Submitting Advanced Jobs With QMON” on page 63](#) and the `-w` option on the `qalter(1)` man page for more information.

Using the Why? Button to Get Information About Pending Jobs

Another method for checking why jobs are still pending is to select a job and click `Why?` on the Job Control dialog box. Doing so opens the Object Browser dialog box. This dialog box displays a list of reasons that prevented the scheduler from dispatching the job in its most recent pass. An example of a Browser window that displays such a message is shown in the following figure.



The Why? button delivers meaningful output only if the scheduler configuration parameter `schedd_job_info` is set to `true`. See the `sched_conf(5)` man page. The displayed scheduler information relates to the last scheduling interval. The information might not be accurate by the time you investigate why your job was not scheduled.

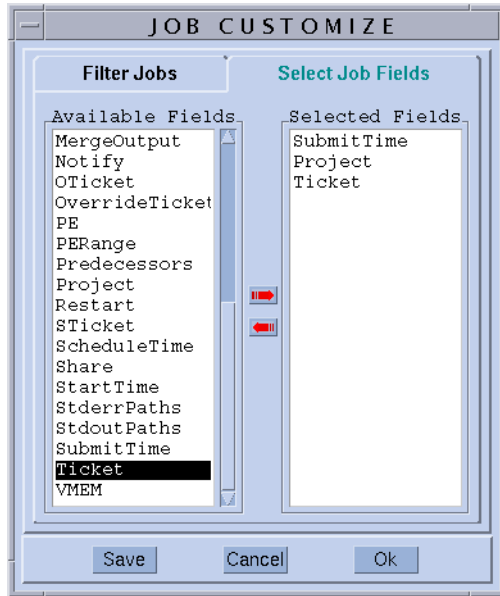
Clearing Error States

Click Clear Error to remove an error state from a pending job that failed due to a job-dependent problem. For example, the job might have insufficient permissions to write to the specified job output file.

Error states appear in red text in the pending jobs list. You should remove jobs only after you correct the error condition, for example, using `qalter`. Such error conditions are automatically reported through email if the job requests to send email when the job is aborted. For example, the job might have been aborted with the `qsub -m a` command.

Customizing the Job Control Display

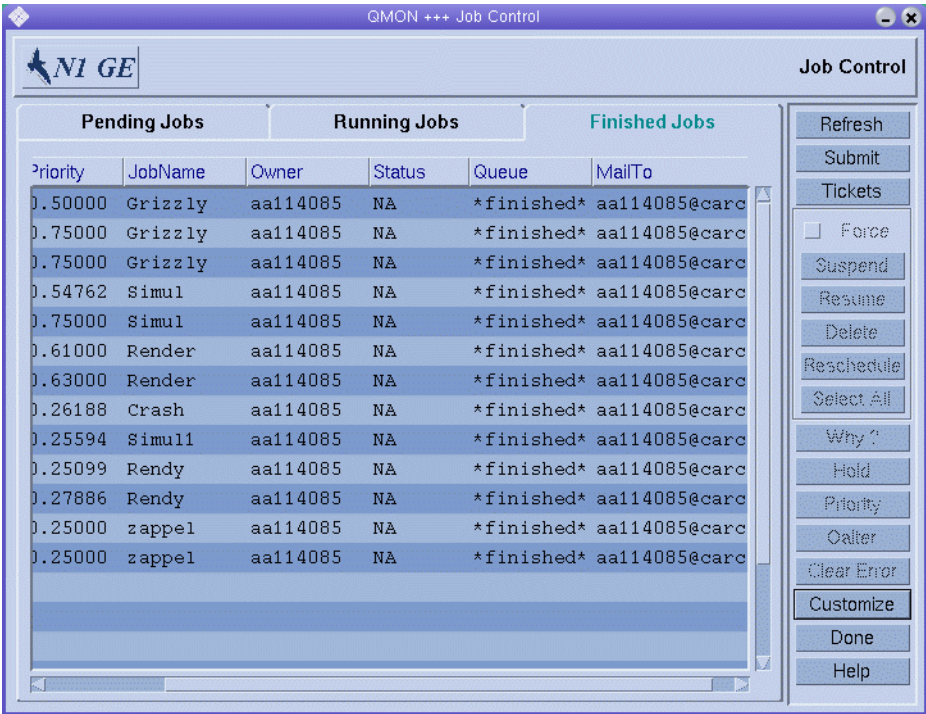
To customize the default Job Control display, click Customize. The Job Customize dialog box appears. Click the Select Job Fields tab. A sample Select Job Fields tab is shown in the following figure.



Use the Job Customize dialog box to configure the set of information to display.

With the Job Customize dialog box, you can select more entries of the job object to be displayed. You can also filter the jobs that you are interested in. The example in the preceding figure selects the additional fields Projects, Tickets, and Submit Time.

The following figure shows the enhanced look after customization is applied to the Finished Jobs list.



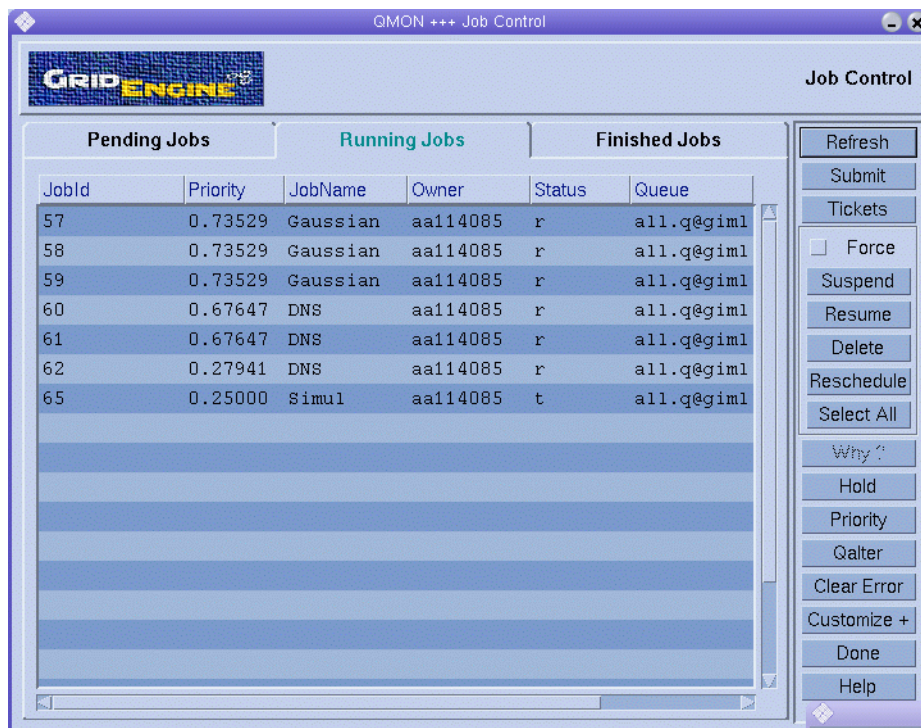
Use the Save button on the Customize Job dialog box to store the customizations in the file `.qmon_preferences`. This file is located in the user's home directory. By saving your customizations, you redefine the appearance of the Job Control dialog box.

Filtering the Job List

The following example of the filtering facility selects only those jobs owned by `aa114085` that are suitable to be run on the architecture `solaris64`.



The following figure shows the resulting Running Jobs tab of the Job Control dialog box.



The Job Control dialog box that is shown in the previous figure is also an example of how QMON displays array jobs.

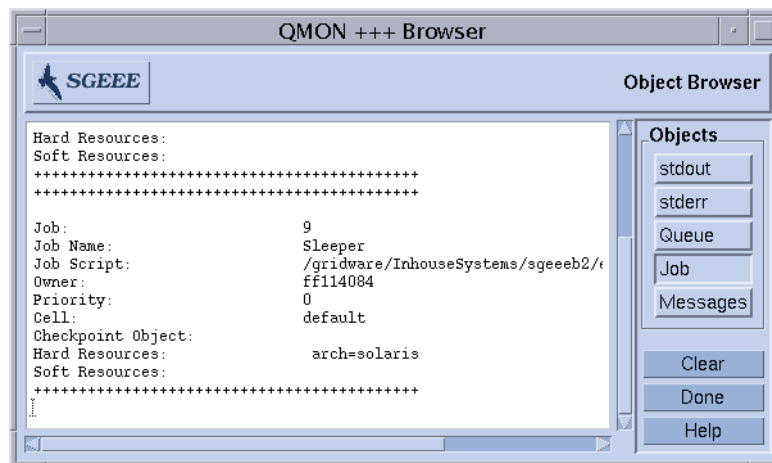
Getting Additional Information About Jobs With the QMON Object Browser

You can use the QMON Object Browser to quickly retrieve additional information about jobs without having to customize the Job Control dialog box, as explained in [“Monitoring and Controlling Jobs With QMON” on page 85](#).

You can open the Object Browser to display information about jobs in two ways:

- Click the Browser button in the QMON Main Control window, and then click Job in the Browser dialog box.
- Move the pointer over a job in the Job Control dialog box.

The following Browser window shows an example of the job information that is displayed:



Monitoring and Controlling Jobs From the Command Line

This section describes how to use the commands `qstat`, `qdel`, and `qmod` to monitor, delete, and modify jobs from the command line.

Monitoring Jobs With `qstat`

To monitor jobs, type one of the following commands, guided by information that is detailed in the following sections:

```
qstat
qstat -f
qstat -ext
```

`qstat` with no options provides an overview of submitted jobs only. `qstat -f` includes information about the currently configured queues in addition. `qstat -ext` contains details such as up-to-date job usage and tickets assigned to a job.

In the first form, a header line indicates the meaning of the columns. The purpose of most of the columns should be self-explanatory. The state column, however, contains single character codes with the following meaning: *r* for running, *s* for suspended, *q* for queued, and *w* for waiting. See the `qstat(1)` man page for a detailed explanation of the `qstat` output format.

The second form is divided into two sections. The first section displays the status of all available queues. The second section, titled `PENDING JOBS`, shows the status of the `sgl_qmaster` job spool area. The first line of the queue section defines the meaning of the columns with respect to the queues that are listed. The queues are separated by horizontal lines. If jobs run in a queue, the job names appear below the associated queue in the same format as in the `qstat` command in its first form. The pending jobs in the second output section are also listed as in `qstat`'s first form.

The columns of the queue description provide the following information:

- `qtype` – Queue type. Queue type is either *B* (batch) or *I* (interactive).
- `used/free` – Count of used and free job slots in the queue.
- `states` – State of the queue. See the `qstat(1)` man page for detailed information about queue states.

The `qstat(1)` man page contains a more detailed description of the `qstat` output format.

In the third form, the usage and ticket values assigned to a job are shown in the following columns:

- `cpu/mem/io` – Currently accumulated CPU, memory, and I/O usage.
- `tckts/ovrts/otckt/ftckt/stckt` – These values are as follows:
 - `tckts` – Total number of tickets assigned to the job
 - `ovrts` – Override tickets assigned through `qalter -ot`
 - `otckt` – Tickets assigned through the override policy
 - `ftckt` – Tickets assigned through the functional policy
 - `stckt` – Tickets assigned through the share-based policy

In addition, the deadline initiation time is displayed in the column `deadline`, if applicable. The `share` column shows the current resource share that each job has with respect to the usage generated by all jobs in the cluster. See the `qstat(1)` man page for further details.

Various additional options to the `qstat` command enhance the functionality. Use the `-r` option to display the resource requirements of submitted jobs. Furthermore, the output can be restricted to a certain user or to a specific queue. You can use the `-l` option to specify resource requirements, as described in [“Defining Resource Requirements” on page 68](#), for the `qsub` command. If resource requirements are used, only those queues, and the jobs that are running in those queues, are displayed that match the resource requirement specified by `qstat`.

Note – The `qstat` command has been enhanced so that the administrator and the user may define files that can contain useful options. See the `sge_qstat(5)` man page. A cluster-wide `sge_qstat` file may be placed under `$$xQS_NAME_Sxx_ROOT/$$xQS_NAME_Sxx_CELL/common/sge_qstat`. The user private file is processed under the location `$HOME/.sge_qstat`. The home directory request file has the highest precedence, then the cluster global file. You can use the command line to override the flags contained in a file.

[Example 4–1](#) and [Example 4–2](#) show examples of output from the `qstat` and `qstat -f` commands.

EXAMPLE 4–1 Example of `qstat -f` Output

queue name			qtype	used/free	load_avg	arch	states	
dq			BIP	0/1	99.99	sun4	au	
durin.q			BIP	2/2	0.36	sun4		
231	0	hydra	craig	r	07/13/96	20:27:15	MASTER	
232	0	compile	penny	r	07/13/96	20:30:40	MASTER	
dwain.q			BIP	3/3	0.36	sun4		
230	0	blackhole	don	r	07/13/96	20:26:10	MASTER	
233	0	mac	elaine	r	07/13/96	20:30:40	MASTER	
234	0	golf	shannon	r	07/13/96	20:31:44	MASTER	
fq			BIP	0/3	0.36	sun4		
#####								
- PENDING JOBS - PENDING JOBS - PENDING JOBS - PENDING JOBS - PENDING JOBS -								
#####								
236	5	word	elaine	qw	07/13/96	20:32:07		
235	0	andrun	penny	qw	07/13/96	20:31:43		

EXAMPLE 4-2 Example of qstat Output

job-ID	prior	name	user	state	submit/start at	queue	function
231	0	hydra	craig	r	07/13/96 20:27:15	durin.q	MASTER
232	0	compile	penny	r	07/13/96 20:30:40	durin.q	MASTER
230	0	blackhole	don	r	07/13/96 20:26:10	dwain.q	MASTER
233	0	mac	elaine	r	07/13/96 20:30:40	dwain.q	MASTER
234	0	golf	shannon	r	07/13/96 20:31:44	dwain.q	MASTER
236	5	word	elaine	qw	07/13/96 20:32:07		
235	0	andrun	penny	qw	07/13/96 20:31:43		

Controlling Jobs With qdel and qmod

To control jobs from the command line, type one of the following commands with the appropriate arguments.

```
% qdel arguments
% qmod arguments
```

Use the qdel command to cancel jobs, regardless of whether the jobs are running or are spooled. Use the qmod command to suspend and resume (unsuspend) jobs already running.

For both commands, you need to know the job identification number, which is displayed in response to a successful qsub command. If you forget the number, you can retrieve it with qstat. See [“Monitoring Jobs With qstat” on page 94](#).

The following list provides several examples of the qdel and qmod commands:

```
% qdel job-id
% qdel -f job-id1, job-id2
% qmod -s job-id
% qmod -us -f job-id1, job-id2
% qmod -s job-id.task-id-range
```

In order to delete, suspend, or resume a job, you must be the owner of the job or a grid engine manager or operator. See [“Managers, Operators, and Owners” on page 38](#).

You can use the -f (force) option with both commands to register a job status change at sge_qmaster without contacting sge_execd. You might want to use the force option in cases where sge_execd is unreachable, for example, due to network problems. The -f option is intended for use only by the administrator. In the case of qdel, however, users can force

deletion of their own jobs if the flag `ENABLE_FORCED_QDEL` in the cluster configuration `qmaster_params` entry is set. See the `sge_conf(5)` man page for more information.

Monitoring Jobs by Email

From the command line, type the following command with appropriate arguments.

```
% qsub arguments
```

The `qsub -m` command requests email to be sent to the user who submitted a job or to the email addresses specified by the `-M` flag if certain events occur. See the `qsub(1)` man page for a description of the flags. An argument to the `-m` option specifies the events. The following arguments are available:

- `b` – Send email at the beginning of the job.
- `e` – Send email at the end of the job.
- `a` – Send email when the job is rescheduled or aborted (for example, by using the `qdel` command).
- `s` – Send email when the job is suspended.
- `n` – Do not send email. `n` is the default.

Use a string made up of one or more of the letter arguments to specify several of these options with a single `-m` option. For example, `-m be` sends email at the beginning and at the end of a job.

You can also use the Submit Job dialog box to configure these mail events. See [“Submitting Advanced Jobs With QMON” on page 63](#).

Monitoring and Controlling Queues

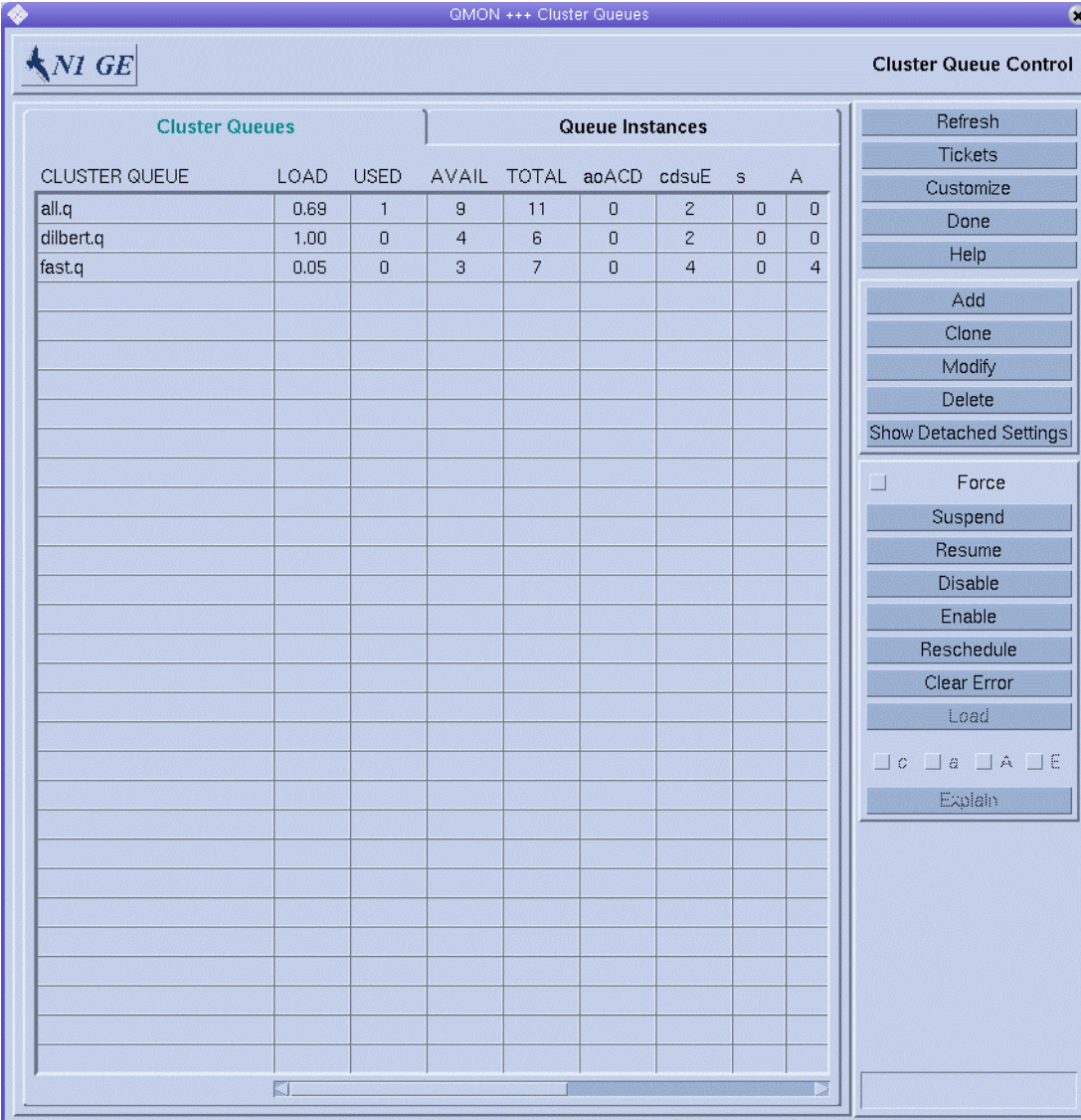
As described in [“Displaying Queues and Queue Properties” on page 38](#), the owners of queues have permission to suspend and resume queues, and to disable and enable queues. Owners might want to suspend or disable queues if certain machines are needed for important work, and those machines are strongly affected by jobs running in the background.

You can control queues in two ways:

- Using the QMON Queue Control dialog box
- Using the `qmod` command

Monitoring and Controlling Queues With QMON

In the QMON Main Control window, click the Queue Control button. The Cluster Queues dialog box appears.



Monitoring and Controlling Cluster Queues

The Cluster Queue tab provides a quick overview of all cluster queues that are defined for the cluster. The Cluster Queue tab also provides the means to suspend and resume cluster queues, to disable and enable cluster queues, as well as to configure them.

Information displayed in the Cluster Queue dialog box is updated periodically. Click Refresh to force an update. Click a cluster queue name to select the queue.

Click Delete, Suspend, Resume, Disable, or Enable to execute the corresponding operation on cluster queues that you select. The suspend/resume and disable/enable operations require notification of the corresponding `sge_execd`. If notification is not possible, you can force an `sge_qmaster` internal status change by clicking Force. For example, notification might not be possible because a host is down.

The suspend/resume and disable/enable operations require cluster queue owner permission, grid engine manager permission, or operator permission. See [“Managers, Operators, and Owners” on page 38](#) for details.

Suspended cluster queues are closed for further jobs. The jobs already running in suspended queues are also suspended, as described in [“Monitoring and Controlling Jobs With QMON” on page 85](#). The cluster queue and its jobs are unsuspended as soon as the queue is resumed.

Note – If a job in a suspended cluster queue was suspended explicitly, the job is not resumed when the queue is resumed. The job must be resumed explicitly.

Disabled cluster queues are closed. However, the jobs that are running in those queues are allowed to continue. The disabling of a cluster queue is commonly used to clear a queue. After the cluster queue is enabled, it is eligible to run jobs again. No action on currently running jobs is performed.

Error states are displayed using a red font in the queue list. Click Clear Error to remove an error state from a queue.

Click Reschedule to reschedule all jobs currently running in the selected cluster queues.

To configure cluster queues and queue instances, click Add or Modify on the Cluster Queue dialog box. See [“Configuring Queues With QMON” in *Sun N1 Grid Engine 6.1 Administration Guide*](#) for details.

Click Done to close the dialog box.

Cluster Queue Status

Each row in the cluster queue table represents one cluster queue. For each cluster queue, the table lists the following information:

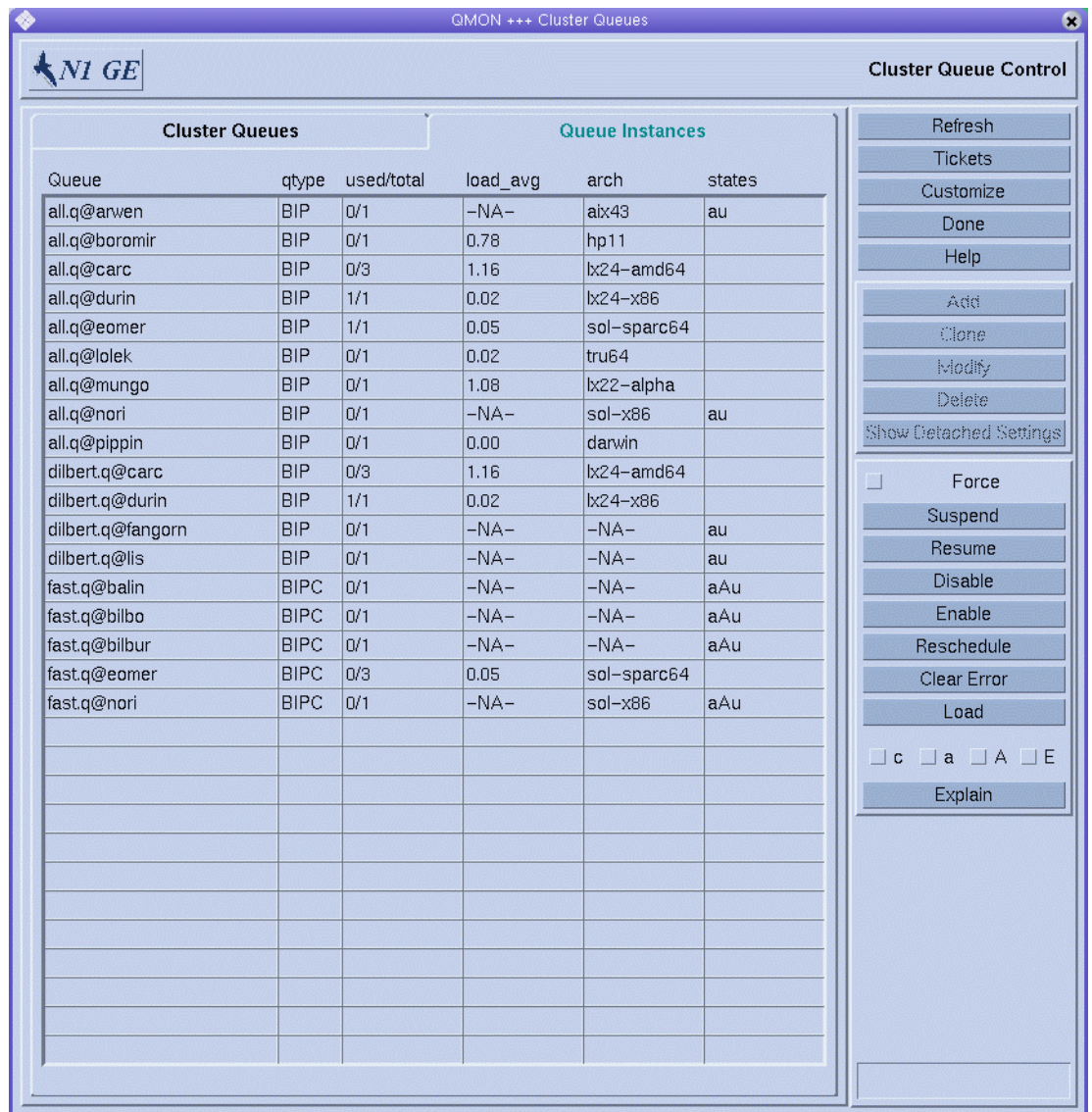
- Cluster Queue – Name of the cluster queue.
- Load – Average of the normalized load average of all cluster queue hosts. Only hosts with a load value are considered.
- Used – Number of currently used job slots.
- Avail – Number of currently available job slots.
- Total – Total number of job slots.

- **aoACD** – Number of queue instances that are in at least one of the following states:
 - **a** – Load threshold alarm
 - **o** – Orphaned
 - **A** – Suspend threshold alarm
 - **C** – Suspended by calendar
 - **D** – Disabled by calendar
- **cdsuE** – Number of queue instances that are in at least one of the following states:
 - **c** – Configuration ambiguous
 - **d** – Disabled
 - **s** – Suspended
 - **u** – Unknown
 - **E** – Error
- **s** – Number of queue instances that are in the suspended state.
- **A** – Number of queue instances where one or more suspend thresholds are currently exceeded. No more jobs
- **S** – Number of queue instances that are suspended through subordination to another queue.
- **C** – Number of queue instances that are automatically suspended by the grid engine system calendar.
- **u** – Number of queue instances that are in an unknown state.
- **a** – Number of queue instances where one or more load thresholds are currently exceeded.
- **d** – Number of queue instances that are in the disabled state.
- **D** – Number of queue instances that are automatically disabled by the grid engine system calendar.
- **c** – Number of queue instances whose configuration is ambiguous.
- **o** – Number of queue instances that are in the orphaned state.
- **E** – Number of queue instances that are in the error state.

See the `qstat(1)` man page for complete information about cluster queues and their states.

Monitoring and Controlling Queue Instances

The Queue Instances tab provides a quick overview of all queue instances that are associated with the selected cluster queue. The Queue Instance tab also provides the means to suspend, resume, disable, and enable queue instances.



Click a cluster queue name to select the queue instance.

Click Suspend, Resume, Disable, or Enable to execute the corresponding operation on queue instances that you select. The suspend/resume and disable/enable operations require notification of the corresponding `sge_execd`. If notification is not possible, for example, because the host is not reachable, you can force an `sge_qmaster` internal status change by clicking Force.

The suspend/resume and disable/enable operations require queue owner permission, manager permission, or operator permission. See [“Managers, Operators, and Owners” on page 38](#).

Suspended queue instances are closed for further jobs. The jobs already running in suspended queue instances are also suspended, as described in [“Monitoring and Controlling Jobs With QMON” on page 85](#). The queue instance and its jobs are unsuspended as soon as the queue instance is resumed.

Note – If a job in a suspended queue instance was suspended explicitly, the job is not resumed when the queue instance is resumed. The job must be resumed explicitly.

Disabled queue instances are closed. However, the jobs executing in those queue instances are allowed to continue. The disabling of a queue instance is commonly used to clear a queue instance. After the queue instance is enabled, it is eligible to run jobs again. No action on currently running jobs is performed.

Queue Instance Status

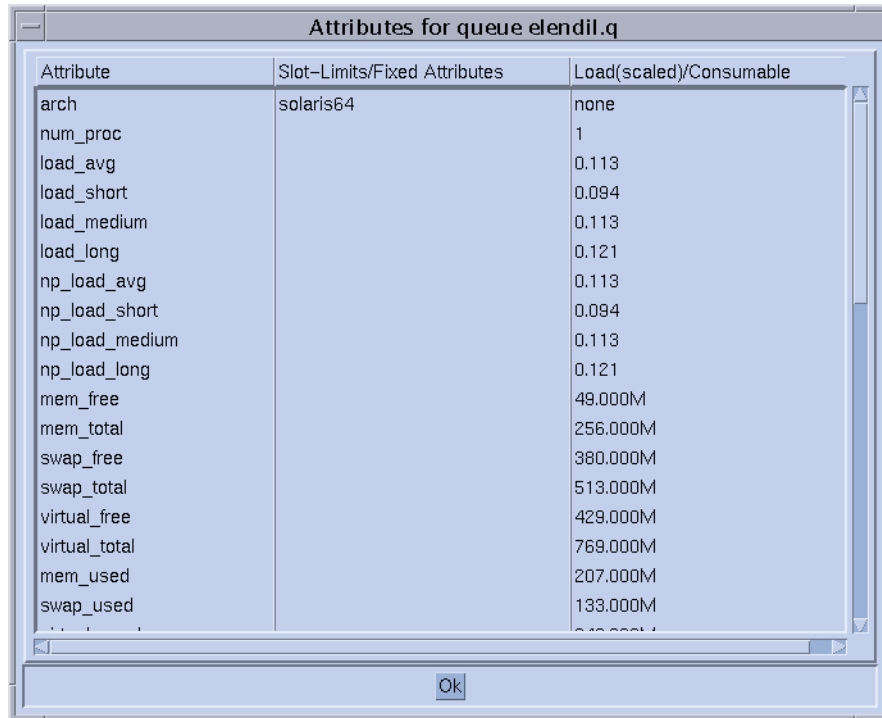
Each row in the queue instances table represents one queue instance. For each queue instance, the table lists the following information:

- Queue – Name of the queue instance
- qtype – Type of queue instance, which can be B (batch), I (interactive), or P (parallel)
- used/total – Number of used job slots and the total number of job slots
- load_avg – Load average of the queue instance host
- arch – Architecture of the queue instance host
- states – States of the queue instance

See [“Cluster Queue Status” on page 100](#) for a list of queue states. See the `qstat(1)` man page for complete information about queue instances and their states.

Displaying Queue Instance Attributes

To retrieve a queue instance's current attribute information, load information, and resource consumption information, select the queue instance, and then click Load. This information also implicitly includes information about the machine that is hosting the queue instance. The window shown in the following figure appears:



Attribute	Slot-Limits/Fixed Attributes	Load(scaled)/Consumable
arch	solaris64	none
num_proc		1
load_avg		0.113
load_short		0.094
load_medium		0.113
load_long		0.121
np_load_avg		0.113
np_load_short		0.094
np_load_medium		0.113
np_load_long		0.121
mem_free		49.000M
mem_total		256.000M
swap_free		380.000M
swap_total		513.000M
virtual_free		429.000M
virtual_total		769.000M
mem_used		207.000M
swap_used		133.000M

The Attribute column lists all attributes attached to the queue instance, including those attributes that are inherited from the host or the global cluster.

The Slot-Limits/Fixed Attributes column shows values for those attributes that are defined as per queue instance slot limits or as fixed resource attributes.

The Load(scaled)/Consumable column shows information about the reported and scaled load parameters. The column also shows information about the available resource capacities based on the consumable resources facility. See “Load Parameters” in *Sun N1 Grid Engine 6.1 Administration Guide* and “Consumable Resources” in *Sun N1 Grid Engine 6.1 Administration Guide*.

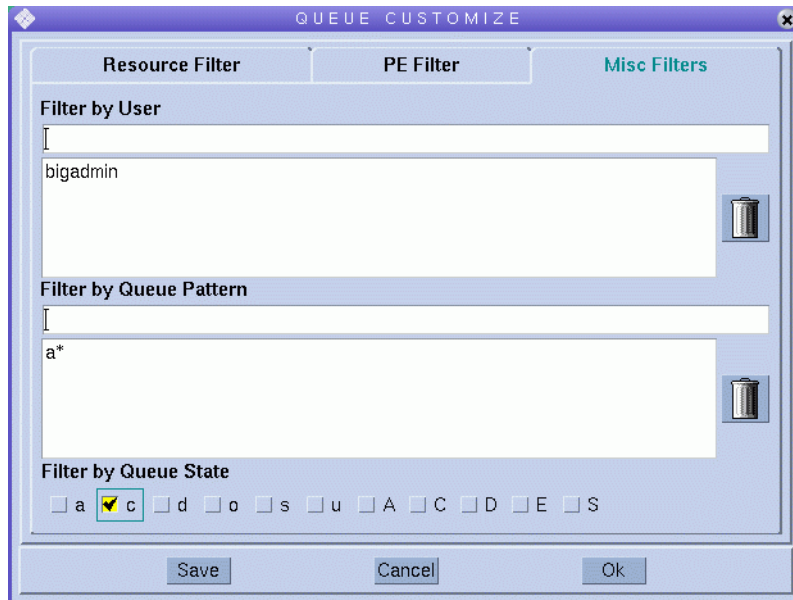
Load reports and consumable capacities can override each other if a load attribute is configured as a consumable resource. The minimum value of both, which is used in the job-dispatching algorithm, is displayed.

Note – The displayed load and consumable values currently do not take into account load adjustment corrections, as described in [“Execution Hosts” on page 27](#).

Filtering Cluster Queues and Queue Instances

The Customize button enables you to filter the cluster queues and queue instances you want to display.

The following figure shows a filtered selection of only those queue instances whose current configuration is ambiguous.



Click Save in the Queue Customize dialog box to store your settings in the file `.qmon_preferences` in your home directory for standard reactivation on later invocations of QMON.

Controlling Queues With qmod

You can use the `qmod` command to suspend and resume queues. You can also use `qmod` to disable and enable queues.

The following commands illustrate how to use `qmod`:

```
% qmod -s q-name
% qmod -us -f q-name1, q-name2
% qmod -d q-name
% qmod -e q-name1, q-name2, q-name3
```

`qmod -s` suspends a queue. `qmod -us -f` resumes (unsuspends) two queues. `qmod -d` disables a queue. `qmod -e` enables three queues.

The `-f` option forces registration of the status change in `sge_qmaster` when the corresponding `sge_execd` is not reachable, for example, due to network problems.

Suspending and resuming queues as well as disabling and enabling queues requires queue owner permission, manager permission, or operator permission. See “[Managers, Operators, and Owners](#)” on page 38.

Note – You can use `qmod` commands with `crontab` or `at` jobs.

Using Job Checkpointing

This section explores two different kinds of job checkpointing: user-level and kernel-level.

User-Level Checkpointing

Many application programs, especially programs that consume considerable CPU time, use checkpointing and restart mechanisms to increase fault tolerance. Status information and important parts of the processed data are repeatedly written to one or more files at certain stages of the algorithm. If the application is aborted, these restart files can be processed and restarted at a later time. The files reach a consistent state that is comparable to the situation just before the checkpoint. Because the user mostly has to move the restart files to a proper location, this kind of checkpointing is called *user-level* checkpointing.

Application programs that do not have integrated user-level checkpointing can use a checkpointing library. A checkpointing library can be provided by some hardware vendors or by the public domain. The Condor project of the University of Wisconsin is an example. By relinking an application with such a library, a checkpointing mechanism is installed in the application without requiring source code changes.

Kernel-Level Checkpointing

Some operating systems provide checkpointing support inside the operating system kernel. No preparations in the application programs and no relinking of the application is necessary in this case. Kernel-level checkpointing usually applies to single processes as well as to complete process hierarchies. A hierarchy of interdependent processes can be checkpointed and restarted at any time. Usually both a user command and a C library interface are available to initiate a checkpoint.

The grid engine system supports operating system checkpointing if available. See the release notes for the N1 Grid Engine 6.1 software for information about the currently supported kernel-level checkpointing facilities.

Migrating Checkpointing Jobs

Checkpointing jobs are interruptible at any time because their restart capability ensures that very little work that is already done must be repeated. This ability is used to build migration and dynamic load balancing mechanism in the grid engine system. If requested, checkpointing jobs are stopped on demand. The jobs are migrated to other machines in the grid engine system, thus averaging the load in the cluster dynamically. Checkpointing jobs are stopped and migrated for the following reasons:

- The executing queue or the job is suspended explicitly by a `qmod` or a `QMON` command.
- The job or the queue where the job runs is suspended automatically because a suspend threshold for the queue is exceeded. The checkpoint occasion specification for the job includes the suspension case. For more information, see “Configuring Load and Suspend Thresholds” in *Sun N1 Grid Engine 6.1 Administration Guide* and “[Submitting, Monitoring, or Deleting a Checkpointing Job From the Command Line](#)” on page 108.

A migrating job moves back to `sgc_qmaster`. The job is subsequently dispatched to another suitable queue if such a queue is available. In such a case, the `qstat` output shows `R` as the status.

Composing a Checkpointing Job Script

Shell scripts for kernel-level checkpointing are the same as regular shell scripts.

Shell scripts for user-level checkpointing jobs differ from regular batch scripts only in their ability to properly handle the restart process. The environment variable `RESTARTED` is set for checkpointing jobs that are restarted. Use this variable to skip sections of the job script that need to be executed only during the initial invocation.

[Example 4–3](#) shows a sample transparently checkpointing job script.

EXAMPLE 4–3 Example of a Checkpointing Job Script

```
#!/bin/sh
#Force /bin/sh in Grid Engine
#$ -S /bin/sh

# Test if restarted/migrated
if [ $RESTARTED = 0 ]; then
    # 0 = not restarted
    # Parts to be executed only during the first
    # start go in here
    set_up_grid
fi

# Start the checkpointing executable
```

EXAMPLE 4-3 Example of a Checkpointing Job Script (Continued)

```
fem
#End of scriptfile
```

The job script restarts from the beginning if a user-level checkpointing job is migrated. The user is responsible for directing the program flow of the shell script to the location where the job was interrupted. Doing so skips those lines in the script that must be executed more than once.

Note – Kernel-level checkpointing jobs are interruptible at any time. The embracing shell script is restarted exactly from the point where the last checkpoint occurred. Therefore, the `RESTARTED` environment variable is not relevant for kernel-level checkpointing jobs.

Submitting, Monitoring, or Deleting a Checkpointing Job From the Command Line

Type the following command with the appropriate options:

```
# qsub options arguments
```

The *submission* of a checkpointing job works in the same way as for regular batch scripts, except for the `qsub -ckpt` and `qsub -c` commands. These commands request a checkpointing mechanism. The commands also define the occasions at which checkpoints must be generated for the job.

The `-ckpt` option takes one argument, which is the name of the checkpointing environment to use. See “Configuring Checkpointing Environments” in *Sun N1 Grid Engine 6.1 Administration Guide*.

The `-c` option is not required. `-c` also takes one argument. Use the `-c` option to override the definitions of the `when` parameter in the checkpointing environment configuration. See the `checkpoint(5)` man page for details.

The argument to the `-c` option can be one of the following one-letter selections, or any combination. The argument can also be a time value.

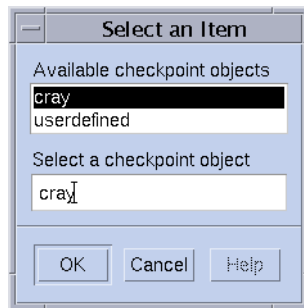
- `n` – No checkpoint is performed. `n` has the highest precedence.
- `s` – A checkpoint is generated only if the `sge_execd` on the jobs host is shut down.
- `m` – Generate the checkpoint at the minimum CPU interval defined in the corresponding queue configuration. See the `min_cpu_interval` parameter in the `queue_conf(5)` man page.
- `x` – A checkpoint is generated if the job is suspended.
- `interval` – Generate the checkpoint in the given interval but not more frequently than defined by `min_cpu_interval`. The time value must be specified as `hh:mm:ss`. This format specifies two digit hours, minutes, and seconds, separated by colons.

The *monitoring* of checkpointing jobs differs from monitoring regular jobs. Checkpointing jobs can migrate from time to time. Checkpointing jobs are therefore not bound to a single queue. However, the unique job identification number and the job name stay the same.

The *deletion* of checkpointing jobs works in the same way as described in [“Monitoring and Controlling Jobs From the Command Line” on page 94](#).

Submitting a Checkpointing Job With QMON

The submission of checkpointing jobs with QMON is identical to submitting regular batch jobs, with the addition of specifying an appropriate checkpointing environment. As explained in [“Submitting Advanced Jobs With QMON” on page 63](#), the Submit Job dialog box provides a field for the checkpointing environment that is associated with a job. Click the button next to that field to open the following Selection dialog box.



You can select a suitable checkpoint environment from the list of available checkpoint objects. Ask your system administrator for information about the properties of the checkpointing environments that are installed at your site. For more information, refer to “Configuring Checkpointing Environments” in *Sun N1 Grid Engine 6.1 Administration Guide*.

File System Requirements for Checkpointing

When a user-level checkpoint or a kernel-level checkpoint that is based on a checkpointing library is written, a complete image of the virtual memory covered by the process or job to be checkpointed must be saved. Sufficient disk space must be available for this purpose. If the checkpointing environment configuration parameter `ckpt_dir` is set, the checkpoint information is saved to a job private location under `ckpt_dir`. If `ckpt_dir` is set to `NONE`, the directory where the checkpointing job started is used. See the `checkpoint(5)` man page for detailed information about the checkpointing environment configuration.

Note – You should start a checkpointing job with the `qsub -cwd` script if `ckpt_dir` is set to `NONE`.

Checkpointing files and restart files must be visible on all machines in order to successfully migrate and restart jobs. Because file visibility is necessary for the way file systems must be organized, NFS or a similar file system is required. Ask your cluster administration if your site meets this requirement.

If your site does not run NFS, you can transfer the restart files explicitly at the beginning of your shell script. For example, you can use `rcp` or `ftp` in the case of user-level checkpointing jobs.

Accounting and Reporting

This chapter covers the following topics:

- “Starting the Accounting and Reporting Console” on page 111
- “Creating and Running Simple Queries” on page 113
- “Creating and Running Advanced Queries” on page 124
- “Latebindings for Advanced Queries” on page 126

Starting the Accounting and Reporting Console

The accounting and reporting console is installed separately from the N1 Grid Engine 6.1 software. For details on the installation process, see Chapter 8, “Installing the Accounting and Reporting Console,” in *Sun N1 Grid Engine 6.1 Installation Guide*. In addition, you must enable your grid engine system to collect reporting information. For details about how to enable the collection of reporting data, see “Report Statistics (ARCo)” in *Sun N1 Grid Engine 6.1 Administration Guide*.

▼ How to Start the Accounting and Reporting Console

1 Start a web browser.

2 Type the URL to connect to the Sun Java Web Console.

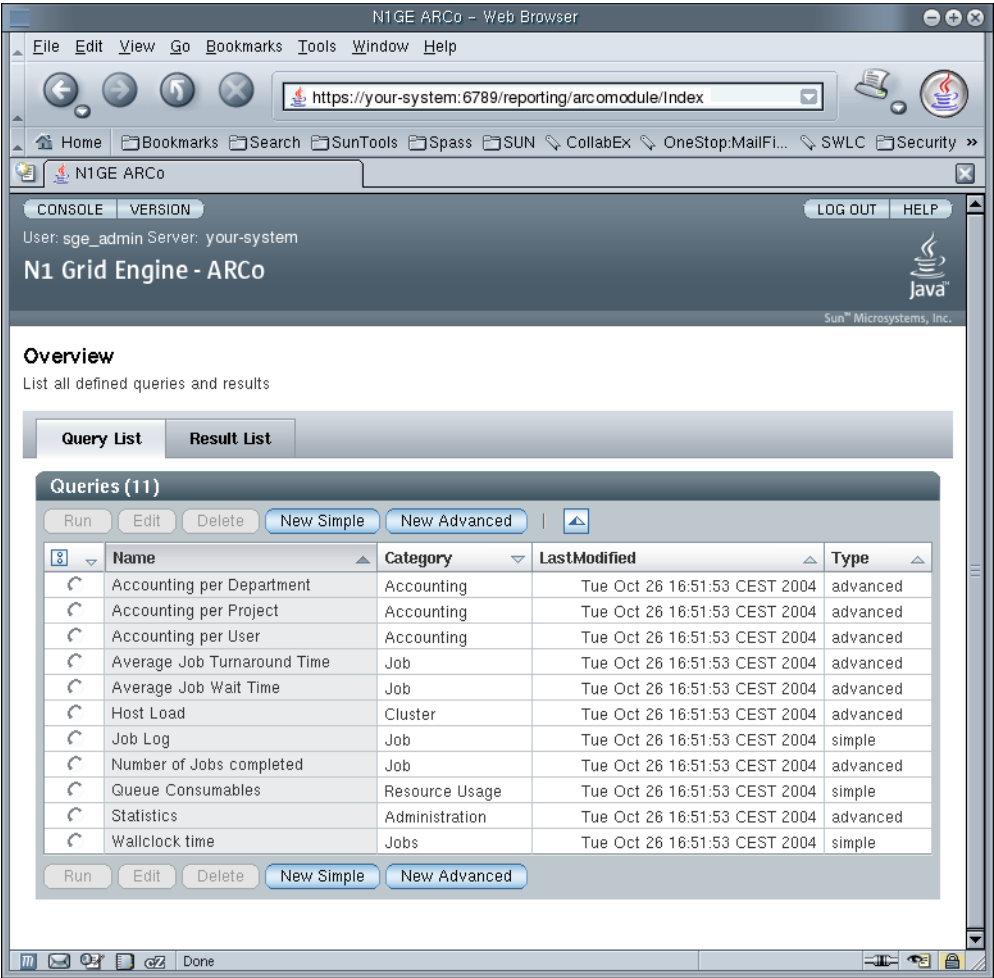
In the following example, *hostname* is the host on which the accounting and reporting software has been installed.

```
https://hostname:6789
```

3 Log in to your UNIX account.

4 Select the N1 Grid Engine 6 ARCo application.

You are redirected to an Overview page that shows you a list of predefined ARCo queries.



Tip – The direct link to the ARCo application is
https://hostname:6789/console/login/Login?redirect_url=%22/reporting/arcomodule/Index%22)

If you press the tab labeled Result List, you see any stored Query Results. Clicking on Query List brings you back to the Query List overview.

CONSOLE


VERSION

LOG OUT

HELP

User: sge_admin Server: your-system

N1 Grid Engine - ARCo



Sun® Microsystems, Inc.

Overview

List all defined queries and results

Query List

Result List

Results (1)

View

Delete

▲

	Name ▲	Category ▼	LastModified ▲	Type ▲
↺	Accounting per Department (2004)	Accounting	Fri Feb 18 15:27:26 CET 2005	

View

Delete

Creating and Running Simple Queries

The query defines the data set that you want to retrieve. You can create simple queries for which the system formulates the SQL query string. If you know SQL and you want to write the query yourself, you can create advanced queries.

▼ How to Create a Simple Query

- 1 Go to Query List and press the New Simple button.

The following screen appears with three tabs showing common information, such as the query category and description. This information is optional. To define the query, go to the Simple Query tab. To define the configuration how to display the results of the query, go to the View tab.

[Overview](#) > Simple Query

Simple Query

Definition of the ARCo query

[Save](#)
[Save as...](#)
[Reset](#)
[Run](#)
[To Advanced..](#)

Common	Simple Query	View
--------	---------------------	------

Common Query Properties

Category:

Description:

[Save](#)
[Save as...](#)
[Reset](#)
[Run](#)
[To Advanced..](#)

Click the Simple Query tab to access the Query definition page. The page provides the following features:

- A Table/View dropdown menu you use to choose a database table or view to predefine your query
- The Field List where all the fields are listed as a row
- A Filter List for defining filter conditions for your query
- The Row Limit field to restrict the number of result entries for your query

[Overview](#) > Simple Query**Simple Query**

Definition of the ARCo query

Common	Simple Query	View
--------	--------------	------

Simple Query Definition

Table/View: view_job_times

Field List (2)

	Function	Name	Parameter	Username	Sort
<input checked="" type="checkbox"/>	Count	job_number		Job Count	ASC
<input type="checkbox"/>	Value	department		Department	ASC

Filter List (0)

And/Or	Field	Condition	Parameter	Late Binding	Active
No items found.					

Row Limit: 0

The single steps how to construct a simple query are outlined as follows.

- 2 Select a table from the table list.**
- 3 Define the fields that you want to see.**

The Field Function describes the functionality used for the field. The following list shows the supported values of Field Function.

VALUE	Use the current value of the field
SUM	Accumulate the values of the field
COUNT	Count the number of values of the field
MIN	Get the minimum value of the field
MAX	Get the maximum value of the field
AVG	Get the average value of the field

- The Field Name is a field in the selected table.
- The User Defined Name allows the results to display a more meaningful name.
- Sort allows to define the sorting order for every field if needed.

4 (Optional) Define Filters.

You must specify at least one field before you can define filters.

- AND/OR is needed for any filter except the first. This setting provides the logical connection to the previous filter condition.
- The Field Name is the name of the field to be filtered. If a field has a user-defined name, that name is shown in the selection list. Otherwise, a generated name is shown.
- The Condition field specifies the operators that are used to filter the values from the database. The following table lists the supported operators.

Filter	Symbol	Description	Number of Requirements
Equal	=	The value must equal the Requirement	1
Not Equal	<>, !=	The value must not equal the Requirement	1
Less Than	<	The value must be less than the Requirement	1
Less Than or Equal	<=, ≤	The value must be less than or equal to the Requirement	1
Greater Than	>	The value must be greater than the Requirement	1
Greater Than or Equal	>=, ≥	The value must be greater than or equal to the Requirement	1
Null		The value must be null	0
Not Null		The value must not be null	0
Between		The value must be in a specified interval	2
In		The value must be equal to an element of a specified list	1 or more
Like		The value must contain the given Requirement	1

The Requirement field contains a value that is used for filtering the values returned by the query. The following list contains some examples of things that might go into the Requirement field.

1 AND 100	For a between condition
d%	For a like condition
%d%	For a like condition
%d%e%	For a like condition
Wert-1', Wert-2', ..., 'Wert- <i>n</i>	For an in condition

5 (Optional) Limit the number of data sets.

To limit the number of data sets, select the Limit Query To First option. Then type the number of data sets you want returned.

6 Click Save to save the query.

The following figure displays the Save this Query As screen. Type a name for the query in the Query Name field, and then click Ok.

After you save your query, you return to a modified version of the Simple Query screen.

▼ How to Create a View Configuration

1 To change the view configuration for a query, click the View tab.

To create a view for a saved query:

- Choose the query from the Query List on the Overview page.
- Click the Edit button.
- Click the View tab.

The queries current view configuration displays.

2 Declare how you want to view the results of your query.

You can add three different sections to the view configuration, decide if additional information about the query is shown, and in which order it is shown.

Common Simple Query View

View Configuration Database Table

View Configuration

Add Pivot Add Graphic

Hide Description: ☐

Hide Filter Conditions: ☐

Hide SQL: ☐

[Back to top](#)

Use the links at the top of the page to move to the corresponding section. The possible sections are Database Table, Pivot Table and Graphic. The View Configuration section is always visible and enables you to display the query description that has been entered on the common tab, the filter conditions from the filter list, and the resulting SQL statement of the query definition or the content of the SQL tab for advanced queries.

Selecting Add Database, Add Pivot, or Add Graphic adds the corresponding section.

For some queries, only a subset of the possible view selections are meaningful. For example, if you have only two columns to select from, pivot makes no sense.

For the Database Table add and choose the columns that you need to display under Name and adjust their Type and Format. The order in which the columns are added will be the order in which the columns are presented. The selections that you make for this report do not affect the filters applied to the data.

Database Table

Remove Table

Selected Columns (2)

Add

Delete

<input checked="" type="checkbox"/>	<input type="text"/>	Name	Type	Format
<input type="checkbox"/>		Job Count	Number	###,###.##
<input type="checkbox"/>		Department	Text	yyyy/MM/dd - hh:mm:ss z

Add

Delete

[Back to top](#)

For the Pivot Table, add the pivot column, row, and data entries. Then choose the column Name, Type, and Format. To shift an entry to a different pivot type, select it under Pivot Type.

Pivot Table

Remove Pivot

Move Down

Selected Columns, Data and Rows (3)

Add Column

Add Row

Add Data

Delete

<input checked="" type="checkbox"/>	<input type="text"/>	Name	Type	Format	Pivot Type
<input type="checkbox"/>		time	Date-Time	yyyy/MM/dd - hh:mm:ss z	Column
<input type="checkbox"/>		department	Text	yyyy/MM/dd - hh:mm:ss z	Row
<input type="checkbox"/>		cpu	Number	#0.00	Data

Add Column

Add Row

Add Data

Delete

[Back to top](#)

For the Graphic section, you can attach the query data to different chart diagram types. The following chart types are available from the Diagram Type menu:

- Bar Chart
- Bar Chart (3D)
- Bar Chart Stacked
- Bar Chart Stacked (3d)
- Pie Chart, Pie Chart 3D

- Line Chart
- Line Chart Stacked Line

Three different diagram types are available:

- Bar
- Pie
- Line

Bar and Pie types can be display with a 3D effect. Bar and Line diagrams can be drawn as stacked diagrams with values on the y-axis summarized.

Graphical Presentation

Remove Graphic Move Up Move Down

Diagram Type: Pie Chart

X Axis: time

Series From Columns

Available:		Selected:
time	Add >	
department	Add All >>	
cpu	< Remove	
mem	<< Remove All	
io		

Series From Row

Label: department

Value: cpu

[Back to top](#)

- 3 Click **Save** or **Save As** to save your View configuration to the query.
- 4 Click **Run** to run your query.

Defining Data Series for Diagrams

Two ways to define the data series for a diagram are:

- Series from columns: All column values are added to a series. The name of the series is the column header
- Series from rows: All column values define the series. The names of the series is defined by the values of the label column. The values of the series are defined by the value column.

EXAMPLE 5-1 Accounting per Department Pie Chart

The query “Accounting per Department” results in a table with the columns: time, department, and cpu.

Database Table (7)		
time	department	cpu
2005.01.01	defaultdepartment	1523.62
2005.01.01	dep1	1153.35
2005.01.01	dep2	24.95
2005.02.01	dep1	29.66
2005.02.01	dep2	222.09
2005.03.01	dep1	922.03
2005.03.01	dep2	1732.70

To display the result in a pie chart, select the following configuration:

Graphical Presentation

Remove Graphic

Move Up

Move Down

Diagram Type:

Pie Chart (3D)

X Axis:

time

Series From Columns

Available:

time

department

cpu

mem

io

Add >

Add All >>

< Remove

<< Remove All

Selected:

Series From Row

Label:

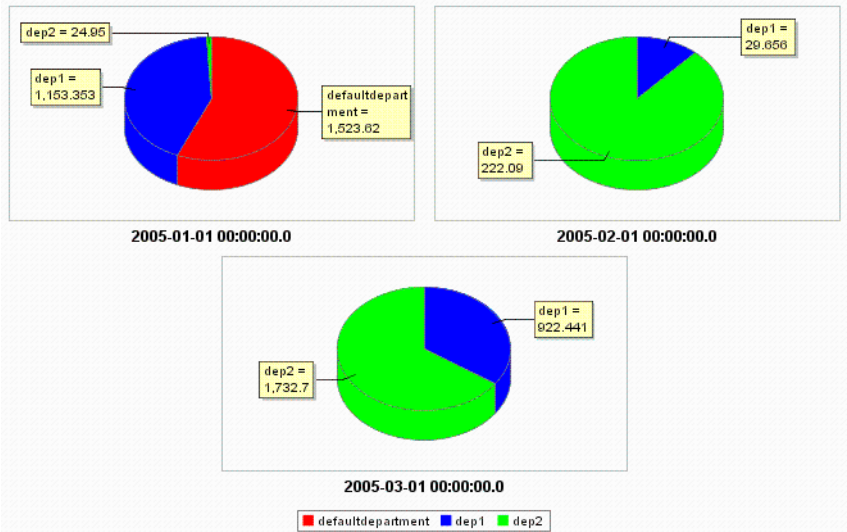
department

Value:

cpu

Show legend: ☒

The result will be multiple pie charts



EXAMPLE 5-2 CPU, Input/Output, and Memory Usage Over All Departments Bar Chart

A query summarizes CPU, IO, and Mem usage over all departments:

Database Table (3)			
department	cpu	mem	io
defaultdepartment	1523.62	27.00	0.03
dep1	2106.44	8.05	0.07
dep2	1979.74	411.05	0.00

To display the results in a bar chart, select the following configuration

Graphical Presentation

Remove Graphic

Move Down

Diagram Type:

Bar Chart (3D)

X Axis:

department

Series From Columns

Available:

department

Add >

Add All >>

< Remove

<< Remove All

Selected:

cpu

mem

io

Series From Row

Label:

department

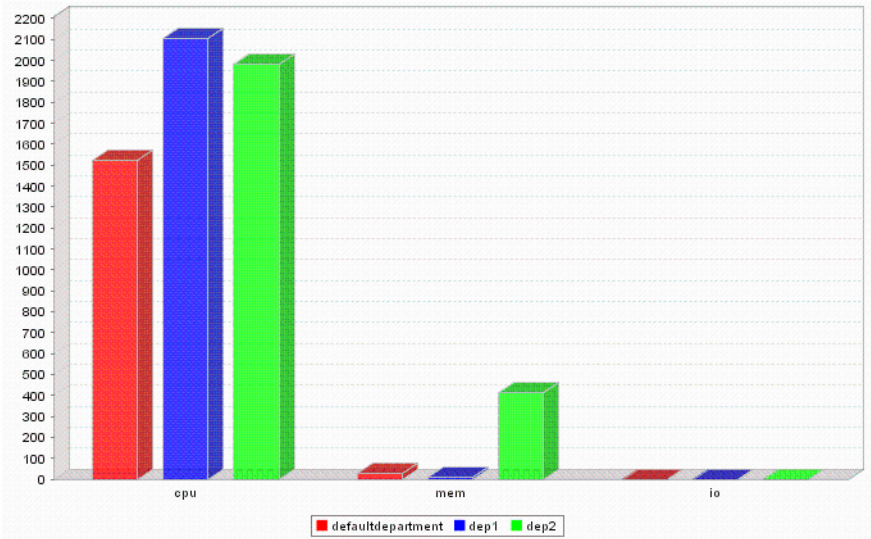
Value:

cpu

Show legend: ☒

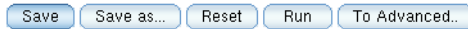
EXAMPLE 5-2 CPU, Input/Output, and Memory Usage Over All Departments Bar Chart (Continued)

The results will be a bar chart with three bars for each department:

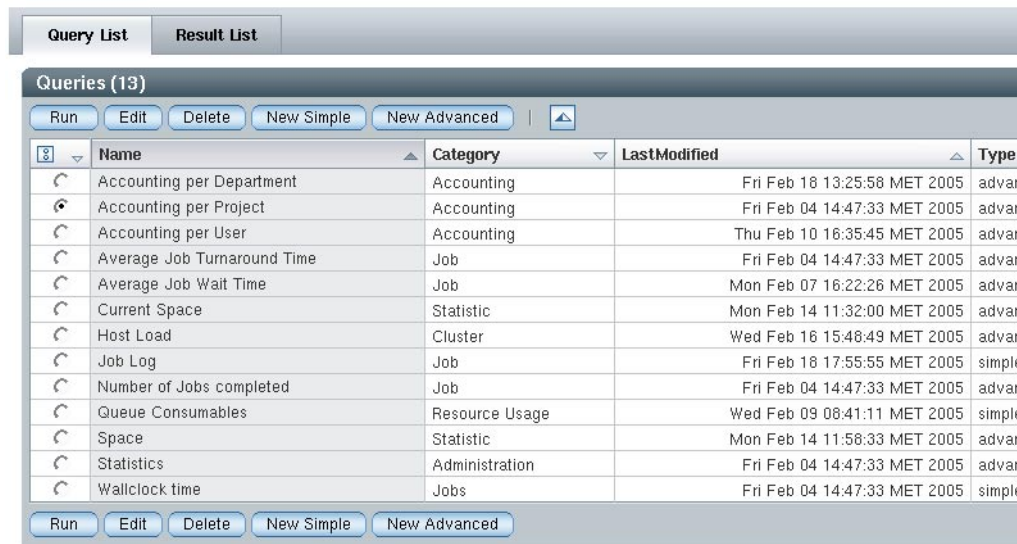


▼ How to Run a Simple Query

- Run the query.
 - To run a query that you just created, click Run on the Simple Query screen.



- To run a query that you previously saved, select the query from the Query List screen and click Run.



▼ How to Edit a Simple Query

- 1 Select a query from the list on the Query List screen
- 2 Click Edit.

The selected Simple Query screen displays.
- 3 Make changes to the Simple Query screen by navigating through the tabs and making your changes as you would when creating a simple query .
- 4 Save or run your changed query.

Creating and Running Advanced Queries

You must have previous experience writing SQL queries to use this feature of the accounting and reporting console.

▼ How to Create an Advanced Query

- 1 Click New Advanced Query on the Query List screen.
- 2 Type your SQL query in the field.

[Overview](#) > Advanced Query

Accounting per Department

Definition of the ARCo query

Save

Save as...

Reset

Run

To Advanced...

Common

SQL

View

* Indicates required field

Advanced Query Definition

* Sql Statement:

```
SELECT time, department, SUM(cpu) as cpu, SUM(mem) as mem, SUM(io) as io
FROM (
  SELECT trunc( cast(start_time as date), 'month') AS time,
         department, cpu, mem, io
  FROM view_accounting
  WHERE start_time > (SYSDATE - INTERVAL '1' YEAR)
)
GROUP BY time, department
```

Save

Save as...

Reset

Run

To Advanced...

- 3 Save or run your query.
 - To save your query, click Save.
 - To run your query, click Run.

▼ How to Run an Advanced Query

- Run the query.
 - To run a query that you just created, click Run on the Advanced Query screen.
 - To run a query that you previously saved, select the query from the Query List screen and click Run.

Query List

Result List

Queries (13)

Run

Edit

Delete

New Simple

New Advanced

	Name	Category	LastModified	Type
	Accounting per Department	Accounting	Fri Feb 18 13:25:58 MET 2005	advanced
	Accounting per Project	Accounting	Fri Feb 04 14:47:33 MET 2005	advanced
	Accounting per User	Accounting	Thu Feb 10 16:35:45 MET 2005	advanced
	Average Job Turnaround Time	Job	Fri Feb 04 14:47:33 MET 2005	advanced
	Average Job Wait Time	Job	Mon Feb 07 16:22:26 MET 2005	advanced
	Current Space	Statistic	Mon Feb 14 11:32:00 MET 2005	advanced
	Host Load	Cluster	Wed Feb 16 15:48:49 MET 2005	advanced
	Job Log	Job	Fri Feb 18 17:55:55 MET 2005	simple
	Number of Jobs completed	Job	Fri Feb 04 14:47:33 MET 2005	advanced
	Queue Consumables	Resource Usage	Wed Feb 09 08:41:11 MET 2005	simple
	Space	Statistic	Mon Feb 14 11:58:33 MET 2005	advanced
	Statistics	Administration	Fri Feb 04 14:47:33 MET 2005	advanced
	Wallclock time	Jobs	Fri Feb 04 14:47:33 MET 2005	simple

Run

Edit

Delete

New Simple

New Advanced

▼ How to Edit an Advanced Query

- 1 Select a query from the list on the Query List screen
- 2 Click Edit.

A completed version of the Advanced Query screen displays.
- 3 Make changes to the SQL query.
- 4 Save or run your changed query.
 - To save your changed query, click Save.
 - To run your changed query, click Run.

Latebindings for Advanced Queries

The syntax for the latebindings in advanced queries is:

```
LATEBINDING{ <column>;<operator>;<default value> }
```

```
<column>    name if the latebinding  
<operator>  a SQL operator (e.g. = < > in .. )  
<value>     default value (e.g. 'localhost' )
```

EXAMPLE 5-3 Latebindings Examples

```
select hostname from sge_host where LATEBINDING{hostname, like, 'a%'}  
select hostname from sge_host where LATEBINDING{hostname, in, ('localhost', 'foo.bar')}
```

Automating Grid Engine Functions Through the Distributed Resource Management Application API

You can automate N1 Grid Engine functions by writing scripts that run N1 Grid Engine commands and parse the results. However, for more consistent and efficient results, you can use the C or Java™ language and the Distributed Resource Management Application API. This chapter introduces the DRMAA concept and explains how to use it with the C and Java languages.

The chapter includes the following information:

- [“Introduction to Distributed Resource Management Application API \(DRMAA\)” on page 127](#)
- [“Developing with the C Language Binding” on page 128](#)
- [“Developing with the Java Language Binding” on page 134](#)

Introduction to Distributed Resource Management Application API (DRMAA)

The Distributed Resource Management Application API (DRMAA, which is pronounced like drama) is an Open Grid Forum specification to standardize job submission, monitoring, and control in Distributed Resource Management Systems (DRMS). The objective of the DRMAA Working Group was to produce an API that would be easy to learn, easy to implement, and that would enable useful application integrations with DRMS in a standard way.

The DRMAA specification is language, platform, and DRMS agnostic. A wide variety of different systems should be able to implement the DRMAA specification. To provide additional guidance for DRMAA implementations in specific languages, the DRMAA Working Group also produced several DRMAA language binding specifications. These specifications define what a DRMAA implementation should resemble in a given language.

The DRMAA specification is currently at version 1.0. The DRMAA Java Language Binding Specification is also at version 1.0, as is the DRMAA C Language Binding Specification. N1 Grid Engine 6.1 provides implementations of both the 1.0 Java language binding and the 1.0 C

language binding as well as older versions of each for backward compatibility. For more information about the DRMAA 1.0 specification and the language-specific binding specifications, see the [Open Grid Forum DRMAA Working Group web site](http://drmaa.org/wiki/?sfProjectId=proj1076) (<http://drmaa.org/wiki/?sfProjectId=proj1076>).

Developing with the C Language Binding

Important Files for the C Language Binding

To use the DRMAA C language binding implementation included with N1 Grid Engine 6.1, you need to know where to find the important files. The most important file is the DRMAA header file that you include from your C application to make the DRMAA functions available to your application. The DRMAA header file resides in *sge-root/include/drmaa.h*, where *sge-root* defaults to */usr/SGE*. For detailed reference information about the DRMAA functions, see section 5 of the N1 Grid Engine man pages, located in the *sge-root/man* directory. To compile and link your application, use the DRMAA shared library at *sge-root/lib/arch/libdrmaa.so*.

Including the DRMAA Header File

To use the DRMAA functions in your application, every source file that uses a DRMAA function must include the DRMAA header file. To include the DRMAA header file in your source file, add the following line to your source code, usually near the top:

```
#include "drmaa.h"
```

Compiling Your C Application

When you compile your DRMAA application, you need to include some additional compiler directives to direct the compiler and linker to use DRMAA. The following directions apply for the Sun Studio Compiler Collection and for gcc. These instructions might not apply for other compilers and linkers. Consult the documentation for your specific compiler and linker products.

You must include two directives:

- Tell the compiler to include the DRMAA header file by adding the following statement to the compiler command line:

```
-I<sge-root>/include
```

- Tell the linker to include the DRMAA library by adding the following statement to the compiler and/or linker command line:

```
-ldrmaa
```

You also need to verify that the *sge-root/lib/arch* directory is included in your library search path (`LD_LIBRARY_PATH` on the Solaris Operating Environment and Linux). The *sge-root/lib/arch* directory is *not* included automatically when you set your environment using the `settings.sh` or `settings.csh` files.

EXAMPLE 6-1 Compiling Your C Application Using Sun Studio Compiler

The following example shows how you would compile your DRMAA application using the Sun Studio Compiler. The following assumptions apply:

- You are using the `csh` shell on a Solaris host.
- N1 Grid Engine is installed in `/sge`
- The DRMAA application is stored in `app.c`.

Sample commands would look like the following

```
% source /sge/default/common/settings.csh
% cc -I/sge/include -ldrmaa app.c
```

Running Your C Application

To run your compiled DRMAA application, verify the following:

The *sge-root/lib/arch* directory must be included in the library search path (`LD_LIBRARY_PATH` on the Solaris Operating Environment and Linux). The *sge-root/lib/arch* directory is *not* included automatically when you set your environment using the `settings.sh` or `settings.csh` files.

You must be logged into a machine that is an N1 Grid Engine submit host. If the machine is not an N1 Grid Engine submit host, all DRMAA function calls will fail, returning `DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE`.

▼ How to Use the DRMAA 0.95 C Language Binding

The DRMAA shared library, which is enabled by default, supports version 1.0 of the DRMAA C Language Binding Specification. For reasons of backward compatibility, however, Grid Engine also includes an implementation of the 0.95 version of the DRMAA C Language Binding Specification. You should develop all new applications with the 1.0 shared library, but you might occasionally discover an application that requires the 0.95 implementation.

To enable the 0.95 version of the shared library, follow these steps:

1 Log in as a user that has permissions to modify the Grid Engine installation.

```
% su -
```

2 Change to the *sge-root/lib/arch* directory.

```
% cd /sge/lib/sol-sparc64
```

3 Remove the *libdrmaa.so* symbolic link.

```
% rm libdrmaa.so
```

4 Create a new symbolic link to the 0.95 library.

```
% ln -s libdrmaa.so.0.95 libdrmaa.so
```

On the Solaris and Linux platforms, the shared library is tagged with a version number. Applications compiled and linked against the 1.0 version will fail claiming that the library could not be found if the 0.95 version of the shared library is enabled, and vice versa. On other platforms, a 1.0 application will load the 0.95 shared library successfully but might fail due to unknown symbols. A 0.95 application will load the 1.0 shared library successfully but will likely fail due to DRMAA functions returning unexpected error codes.

- **To restore the 1.0 version of the shared library, perform steps 1 through 3 and create a new symbolic link to the 1.0 library.**

```
% ln -s libdrmaa.so.1.0 libdrmaa.so
```

C Application Examples

The following examples illustrate some application interactions that use the C language bindings. You can find additional examples on the [“How To” section of the Grid Engine Community Site](#).

EXAMPLE 6-2 Starting and Stopping a Session

The following code segment shows the most basic DRMAA C binding program.

Every call to a DRMAA function returns an error code. If everything goes well, that code is `DRMAA_ERRNO_SUCCESS`. If an error occurs, an appropriate error code is returned. Every DRMAA function also takes at least two parameters. These two parameters are a string to populate with a error message in case of an error and an integer representing the maximum length of the error string.

On line 8, the example calls `drmaa_init()`. This function sets up the DRMAA session and must be called before most other DRMAA functions. Some functions, like `drmaa_get_contact()`, can be called before `drmaa_init()`, but these functions only provide general information. Any function that performs an action, such as `drmaa_run_job()` or `drmaa_wait()` must be called after `drmaa_init()` returns. If such a function is called before `drmaa_init()` returns, it will return the error code `DRMAA_ERRNO_NO_ACTIVE_SESSION`.

EXAMPLE 6-2 Starting and Stopping a Session (Continued)

The `drmaa_init()` function creates a session and starts an event client listener thread. The session is used for organizing jobs submitted through DRMAA, and the thread is used to receive updates from the queue master about the state of jobs and the system in general. Once `drmaa_init()` has been called successfully, the calling application must also call `drmaa_exit()` before terminating. If an application does not call `drmaa_exit()` before terminating, the queue master might be left with a dead event client handle, which can decrease queue master performance.

At the end of the program, on line 17, `drmaa_exit()` cleans up the session and stops the event client listener thread. Most other DRMAA functions must be called before `drmaa_exit()`. Some functions, like `drmaa_get_contact()`, can be called after `drmaa_exit()`, but these functions only provide general information. Any function that performs an action, such as `drmaa_run_job()` or `drmaa_wait()` must be called before `drmaa_exit()` is called. If such a function is called after `drmaa_exit()` is called, it will return the error code `DRMAA_ERRNO_NO_ACTIVE_SESSION`.

```

01: #include
02: #include "drmaa.h"
03:
04: int main(int argc, char **argv) {
05:     char error[DRMAA_ERROR_STRING_BUFFER];
06:     int errnum = 0;
07:
08:     errnum = drmaa_init(NULL, error, DRMAA_ERROR_STRING_BUFFER);
09:
10:     if (errnum != DRMAA_ERRNO_SUCCESS) {
11:         fprintf(stderr, "Could not initialize the DRMAA library: %s\n", error);
12:         return 1;
13:     }
14:
15:     printf("DRMAA library was started successfully\n");
16:
17:     errnum = drmaa_exit(error, DRMAA_ERROR_STRING_BUFFER);
18:
19:     if (errnum != DRMAA_ERRNO_SUCCESS) {
20:         fprintf(stderr, "Could not shut down the DRMAA library: %s\n", error);
21:         return 1;
22:     }
23:
24:     return 0;
25: }

```

EXAMPLE 6-3 Running a Job

The following code segment shows how to use the DRMAA C binding to submit a job to N1 Grid Engine. The beginning and end of this program are the same as in [Example 6-2](#). The differences are on lines 16-59. On line 16, DRMAA allocates a job template. A job template is a structure used to store information about a job to be submitted. The same template can be reused for multiple calls to `drmaa_run_job()` or `drmaa_run_bulk_job()`.

On line 22, the `DRMAA_REMOTE_COMMAND` attribute is set. This attribute tells DRMAA where to find the program to run. Its value is the path to the executable. The path can be relative or absolute. If relative, the path is relative to the `DRMAA_WD` attribute, which defaults to the user's home directory. For more information on DRMAA attributes, see the `drmaa_attributes` man page. For this program to work, the script `sleep.sh` must be in your default path.

On line 32, the `DRMAA_V_ARGV` attribute is set. This attribute tells DRMAA what arguments to pass to the executable. For more information on DRMAA attributes, refer to the `drmaa_attributes` man page.

On line 43, `drmaa_run_job()` submits the job. DRMAA places the id assigned to the job into the character array that is passed to `drmaa_run_job()`. The job is now running as though submitted by `qsub`. At this point, calling `drmaa_exit()` or terminating the program will have no effect on the job.

To clean things up, the job template is deleted on line 54. This frees the memory DRMAA set aside for the job template, but has no effect on submitted jobs.

Finally, on line 61, call `drmaa_exit()` is called. The call to `drmaa_exit()` is outside of the *if* structure started on line 18 because once `drmaa_init()` is called `drmaa_exit()` must be called before terminating, regardless of whether the other commands succeed.

```
01: #include
02: #include "drmaa.h"
03:
04: int main(int argc, char **argv) {
05:     char error[DRMAA_ERROR_STRING_BUFFER];
06:     int errnum = 0;
07:     drmaa_job_template_t *jt = NULL;
08:
09:     errnum = drmaa_init(NULL, error, DRMAA_ERROR_STRING_BUFFER);
10:
11:     if (errnum != DRMAA_ERRNO_SUCCESS) {
12:         fprintf(stderr, "Could not initialize the DRMAA library: %s\n", error);
13:         return 1;
14:     }
15:
16:     errnum = drmaa_allocate_job_template(&jt, error, DRMAA_ERROR_STRING_BUFFER);
17:
```

EXAMPLE 6-3 Running a Job (Continued)

```

18:   if (errno != DRMAA_ERRNO_SUCCESS) {
19:       fprintf(stderr, "Could not create job template: %s\n", error);
20:   }
21:   else {
22:       errno = drmaa_set_attribute(jt, DRMAA_REMOTE_COMMAND, "sleeper.sh",
23:                                   error, DRMAA_ERROR_STRING_BUFFER);
24:
25:       if (errno != DRMAA_ERRNO_SUCCESS) {
26:           fprintf(stderr, "Could not set attribute \"%s\": %s\n",
27:                   DRMAA_REMOTE_COMMAND, error);
28:       }
29:       else {
30:           const char *args[2] = {"5", NULL};
31:
32:           errno = drmaa_set_vector_attribute(jt, DRMAA_V_ARGV, args, error,
33:                                             DRMAA_ERROR_STRING_BUFFER);
34:       }
35:
36:       if (errno != DRMAA_ERRNO_SUCCESS) {
37:           fprintf(stderr, "Could not set attribute \"%s\": %s\n",
38:                   DRMAA_REMOTE_COMMAND, error);
39:       }
40:       else {
41:           char jobid[DRMAA_JOBNAME_BUFFER];
42:
43:           errno = drmaa_run_job(jobid, DRMAA_JOBNAME_BUFFER, jt, error,
44:                                DRMAA_ERROR_STRING_BUFFER);
45:
46:           if (errno != DRMAA_ERRNO_SUCCESS) {
47:               fprintf(stderr, "Could not submit job: %s\n", error);
48:           }
49:           else {
50:               printf("Your job has been submitted with id %s\n", jobid);
51:           }
52:       } /* else */
53:
54:       errno = drmaa_delete_job_template(jt, error, DRMAA_ERROR_STRING_BUFFER);
55:
56:       if (errno != DRMAA_ERRNO_SUCCESS) {
57:           fprintf(stderr, "Could not delete job template: %s\n", error);
58:       }
59:   } /* else */
60:
61:   errno = drmaa_exit(error, DRMAA_ERROR_STRING_BUFFER);
62:
63:   if (errno != DRMAA_ERRNO_SUCCESS) {

```

EXAMPLE 6-3 Running a Job (Continued)

```
64:      fprintf(stderr, "Could not shut down the DRMAA library: %s\n", error);
65:      return 1;
66:  }
67:
68:      return 0;
69: }
```

Developing with the Java Language Binding

Important Files for the Java Language Binding

To use the DRMAA Java language binding implementation included with N1 Grid Engine 6.1, you need to know where to find the important files. The most important file is the DRMAA JAR file *sge-root/lib/drmaa.jar*. To compile your DRMAA application, you must include the DRMAA JAR file in your CLASSPATH. The DRMAA classes are documented in the DRMAA Javadoc™, located in the *sge-root/doc/javadocs* directory. To access the Javadocs, open the file *sge-root/doc/javadocs/index.html* in your browser. When you are ready to run your application, you also need the DRMAA shared library, *sge-root/lib/arch/libdrmaa.so*, which provides the required native routines.

Importing the DRMAA Java Classes and Packages

To use the DRMAA classes in your application, your classes should import the DRMAA classes or packages. In most cases, only the classes in the `org.ggf.drmaa` package will be used. You can import these packages individually or using a wildcard package import. In some rare cases, you might need to reference the N1 Grid Engine DRMAA implementation classes found in the `com.sun.grid.drmaa` package. In those cases, you can import the classes individually or you can import all the classes in a given package. The names of the `com.sun.grid.drmaa` classes do not overlap with the `org.ggf.drmaa` classes, so you can import both packages without creating a namespace collision.

Compiling Your Java Application

To compile your DRMAA application, you must include the *sge-root/lib/drmaa.jar* file in your CLASSPATH. The *drmaa.jar* file will *not* be included automatically when you set your environment using the `settings.sh` or `settings.csh` files.

▼ How to Use DRMAA with NetBeans 5.x

To use the DRMAA classes with your NetBeans 5.0 or 5.5 project, follow these steps:

- 1 Click mouse button 3 on the project node and select Properties.
- 2 Determine whether your project generates a build file or uses an existing file.
 - If your project uses a generated build file:
 - a. Select Libraries in the left column.
 - b. Click Add Library.
 - c. Click Manage Libraries in the Libraries dialog box.
 - d. Click New Library in the Library Management dialog box.
 - e. Type DRMAA in the Library Name field in the New Library dialog box.
 - f. Click OK to dismiss the New Library dialog box.
 - g. Click Add JAR/Folder.
 - h. Browse to the *sge-root/lib* directory in the file chooser dialog box and select the *drmaa.jar* file.
 - i. Click Add JAR/Folder to dismiss the file chooser dialog box.
 - j. Click OK to dismiss the Library Management dialog box.
 - k. Select the DRMAA library and click Add Library to dismiss the Libraries dialog box.
 - If your project uses an existing build file:
 - a. Select Java Sources Classpath in the left column.
 - b. Click Add JAR/Folder.
 - c. Browse to the *sge-root/lib* directory in the file chooser dialog box and select the *drmaa.jar* file.
 - d. Click Choose to dismiss the file chooser dialog box.

3 Click OK to dismiss the properties dialog box.

4 Verify that the DRMAA shared library is in the library search path.

To run your application from NetBeans, the DRMAA shared library file `sge-root/lib/arch/libdrmaa.so` must be included in the library search path (`LD_LIBRARY_PATH` on the Solaris Operating Environment and Linux). The `sge-root/lib/arch` directory is *not* included automatically when you set your environment using the `settings.sh` or `settings.csh` files. To set up the path for the shared library, perform one of the following:

- **Set up your environment in the shell before launching NetBeans.**
- **Add to the `netbeans-root/etc/netbeans.conf` file to set up the environment, such as:**

```
# Setup environment for SGE
. <sge-root>/<sge_cell>/common/settings.sh
ARCH='$SGE_ROOT/util/arch'
LD_LIBRARY_PATH=$SGE_ROOT/lib/$ARCH; export LD_LIBRARY_PATH
```

Running Your Java Application

To run your compiled DRMAA application, verify the following:

- The `sge-root/lib/arch` directory must be included in the library search path (`LD_LIBRARY_PATH` on the Solaris Operating Environment and Linux). The `sge-root/lib/arch` directory is *not* included automatically when you set your environment using the `settings.sh` or `settings.csh` files.
- You must be logged into a machine that is an N1 Grid Engine submit host. If the machine is not an N1 Grid Engine submit host, all DRMAA method calls will fail, throwing a `DrmCommunicationException`.

Using the DRMAA 0.5 Java Language Binding

The DRMAA shared library, which is used by default, supports version 1.0 of the DRMAA Java Language Binding Specification. For reasons of backward compatibility, however, N1 Grid Engine also includes an implementation of the 0.5 version of the DRMAA Java Language Binding Specification. You should develop all new applications with the 1.0 shared library, but you might occasionally discover an application that requires the 0.5 implementation.

To use the 0.5 version of the `drmaa.jar` file, you should include the `sge-root/lib/drmaa-0.5.jar` file in your `CLASSPATH` either before or instead of the usual `sge-root/lib/drmaa.jar` file. In addition, the use of the 0.5 Java language binding requires enabling the 0.95 C language binding. See [“How to Use the DRMAA 0.95 C Language Binding” on page 129](#).

Java Application Examples

The following examples illustrate some application interactions that use the Java language bindings. You can find additional examples on the [“How To” section of the Grid Engine Community Site](#).

EXAMPLE 6-4 Starting and Stopping a Session

The following code segment shows the most basic DRMAA Java language binding program.

Everything that you as a programmer do with DRMAA, you do through a `Session` object. You get the `Session` object from a `SessionFactory`. You get the `SessionFactory` from the static `SessionFactory.getFactory()` method. The reason for this chain is that the `org.ggf.drmaa.*` classes should be considered an immutable package to be used by every DRMAA Java language binding implementation. Because the package is immutable, to load a specific implementation, the `SessionFactory` uses a system property to find the implementation's session factory, which it then loads. That session factory is then responsible for creating the session in whatever way it sees fit. It should be noted that even though there is a session factory, only one session may exist at a time.

On line 9, `SessionFactory.getFactory()` gets a session factory instance. On line 10, `SessionFactory.getSession()` gets the session instance. On line 13, `Session.init()` initializes the session. `" "` is passed in as the contact string to create a new session because no initialization arguments are needed.

`Session.init()` creates a session and starts an event client listener thread. The session is used for organizing jobs submitted through DRMAA, and the thread is used to receive updates from the queue master about the state of jobs and the system in general. Once `Session.init()` has been called successfully, the calling application must also call `Session.exit()` before terminating. If an application does not call `Session.exit()` before terminating, the queue master might be left with a dead event client handle, which can decrease queue master performance. Use the `Runtime.addShutdownHook()` method to make sure `Session.exit()` gets called.

At the end of the program, on line 14, `Session.exit()` cleans up the session and stops the event client listener thread. Most other DRMAA methods must be called before `Session.exit()`. Some functions, like `Session.getContact()`, can be called after `Session.exit()`, but these functions only provide general information. Any function that performs an action, such as `Session.runJob()` or `Session.wait()` must be called before `Session.exit()` is called. If such a function is called after `Session.exit()` is called, it will throw a `NoActiveSessionException`.

```
01: package com.sun.grid.drmaa.howto;
02:
03: import org.ggf.drmaa.DrmaaException;
04: import org.ggf.drmaa.Session;
05: import org.ggf.drmaa.SessionFactory;
```

EXAMPLE 6-4 Starting and Stopping a Session (Continued)

```
06:
07: public class Howto1 {
08:     public static void main(String[] args) {
09:         SessionFactory factory = SessionFactory.getFactory();
10:         Session session = factory.getSession();
11:
12:         try {
13:             session.init("");
14:             session.exit();
15:         } catch (DrmaaException e) {
16:             System.out.println("Error: " + e.getMessage());
17:         }
18:     }
19: }
```

EXAMPLE 6-5 Running a Job

The following code segment shows how to use the DRMAA Java language binding to submit a job to N1 Grid Engine. The beginning and end of this program are the same as [Example 6-4](#). The differences are on lines 16-24.

On line 16, DRMAA allocates a `JobTemplate`. A `JobTemplate` is an object that is used to store information about a job to be submitted. The same template can be reused for multiple calls to `Session.runJob()` or `Session.runBulkJobs()`.

On line 17, the `remoteCommand` attribute is set. This attribute tells DRMAA where to find the program to run. Its value is the path to the executable. The path can be relative or absolute. If relative, the path is relative to the `workingDirectory` attribute, which defaults to the user's home directory. For more information on DRMAA attributes, see the DRMAA Javadoc or the `drmaa_attributes` man page. For this program to work, the script `sleep.sh` must be in your default path.

On line 18, the `args` attribute is set. This attribute tells DRMAA what arguments to pass to the executable. For more information on DRMAA attributes, see the DRMAA Javadoc or the `drmaa_attributes` man page.

On line 20, `Session.runJob()` submits the job. This method returns the ID assigned to the job by the queue master. The job is now running as though submitted by `qsub`. At this point, calling `Session.exit()` or terminating the program will have no effect on the job.

To clean things up, the job template is deleted on line 24. This action frees the memory DRMAA set aside for the job template, but has no effect on submitted jobs.

```
01: package com.sun.grid.drmaa.howto;
02:
03: import java.util.Collections;
```

EXAMPLE 6-5 Running a Job *(Continued)*

```

04: import org.ggf.drmaa.DrmaaException;
05: import org.ggf.drmaa.JobTemplate;
06: import org.ggf.drmaa.Session;
07: import org.ggf.drmaa.SessionFactory;
08:
09: public class Howto2 {
10:     public static void main(String[] args) {
11:         SessionFactory factory = SessionFactory.getFactory();
12:         Session session = factory.getSession();
13:
14:         try {
15:             session.init("");
16:             JobTemplate jt = session.createJobTemplate();
17:             jt.setRemoteCommand("sleeper.sh");
18:             jt.setArgs(Collections.singletonList("5"));
19:
20:             String id = session.runJob(jt);
21:
22:             System.out.println("Your job has been submitted with id " + id);
23:
24:             session.deleteJobTemplate(jt);
25:             session.exit();
26:         } catch (DrmaaException e) {
27:             System.out.println("Error: " + e.getMessage());
28:         }
29:     }
30: }

```


Error Messages, and Troubleshooting

This chapter describes the error messaging procedures of the grid engine system and offers tips on how to resolve various common problems.

- [“How the Software Retrieves Error Reports” on page 141](#)
- [“Diagnosing Problems” on page 147](#)
- [“Troubleshooting Common Problems” on page 149](#)

How the Software Retrieves Error Reports

The grid engine software reports errors and warnings by logging messages into certain files or by sending email, or both. The log files include message files and job STDERR output.

As soon as a job is started, the standard error (STDERR) output of the job script is redirected to a file. The default file name and location are used, or you can specify the filename and the location with certain options of the qsub command. See the grid engine system man pages for detailed information.

Separate messages files exist for the `sge_qmaster`, the `sge_schedd`, and the `sge_execds`. The files have the same file name: `messages`. The `sge_qmaster` log file resides in the master spool directory. The `sge_schedd` message file resides in the scheduler spool directory. The execution daemons' log files reside in the spool directories of the execution daemons. See “Spool Directories Under the Root Directory” in *Sun N1 Grid Engine 6.1 Installation Guide* for more information about the spool directories.

Each message takes up a single line in the files. Each message is subdivided into five components separated by the vertical bar sign (`|`).

The components of a message are as follows:

1. The first component is a time stamp for the message.
2. The second component specifies the grid engine system daemon that generates the message.

3. The third component is the name of the host where the daemon runs.
4. The fourth is a message type. The message type is one of the following:
 - N for notice – for informational purposes
 - I for info – for informational purposes
 - W for warning
 - E for error – an error condition has been detected
 - C for critical – can lead to a program abort

Use the `loglevel` parameter in the cluster configuration to specify on a global basis or a local basis what message types you want to log.

5. The fifth component is the message text.

Note – If an error log file is not accessible for some reason, the grid engine system tries to log the error message to the files `/tmp/sgc_qmaster_messages`, `/tmp/sgc_schedd_messages`, or `/tmp/sgc_execd_messages` on the corresponding host.

In some circumstances, the grid engine system notifies users, administrators, or both, about error events by email. The email messages sent by the grid engine system do not contain a message body. The message text is fully contained in the mail subject field.

Consequences of Different Error or Exit Codes

The following table lists the consequences of different job-related error codes or exit codes. These codes are valid for every type of job.

TABLE 7-1 Job-Related Error or Exit Codes

Script/Method	Exit or Error Code	Consequence
Job script	0	Success
	99	Requeue
	Rest	Success: exit code in accounting file
prolog/epilog	0	Success
	99	Requeue
	Rest	Queue error state, job requeued

The following table lists the consequences of error codes or exit codes of jobs related to parallel environment (PE) configuration.

TABLE 7-2 Parallel-Environment-Related Error or Exit Codes

Script/Method	Exit or Error Code	Consequence
pe_start	0	Success
	Rest	Queue set to error state, job queued
pe_stop	0	Success
	Rest	Queue set to error state, job not queued

The following table lists the consequences of error codes or exit codes of jobs related to queue configuration. These codes are valid only if corresponding methods were overwritten.

TABLE 7-3 Queue-Related Error or Exit Codes

Script/Method	Exit or Error Code	Consequence
Job starter	0	Success
	Rest	Success, no other special meaning
Suspend	0	Success
	Rest	Success, no other special meaning
Resume	0	Success
	Rest	Success, no other special meaning
Terminate	0	Success
	Rest	Success, no other special meaning

The following table lists the consequences of error or exit codes of jobs related to checkpointing.

TABLE 7-4 Checkpointing-Related Error or Exit Codes

Script/Method	Exit or Error Code	Consequence
Checkpoint	0	Success
	Rest	Success. For kernel checkpoint, however, this means that the checkpoint was not successful.
Migrate	0	Success
	Rest	Success. For kernel checkpoint, however, this means that the checkpoint was not successful. Migration will occur.
Restart	0	Success

TABLE 7-4 Checkpointing-Related Error or Exit Codes (Continued)

Script/Method	Exit or Error Code	Consequence
Clean	Rest	Success, no other special meaning
	0	Success
	Rest	Success, no other special meaning

For jobs that run successfully, the `qacct -j` command output shows a value of 0 in the `failed` field, and the output shows the exit status of the job in the `exit_status` field. However, the shepherd might not be able to run a job successfully. For example, the epilog script might fail, or the shepherd might not be able to start the job. In such cases, the `failed` field displays one of the code values listed in the following table.

TABLE 7-5 `qacct -j failed` Field Codes

Code	Description	acctvalid	Meaning for Job
0	No failure	t	Job ran, exited normally
1	Presumably before job	f	Job could not be started
3	Before writing config	f	Job could not be started
4	Before writing PID	f	Job could not be started
5	On reading config file	f	Job could not be started
6	Setting processor set	f	Job could not be started
7	Before prolog	f	Job could not be started
8	In prolog	f	Job could not be started
9	Before pestart	f	Job could not be started
10	In pestart	f	Job could not be started
11	Before job	f	Job could not be started
12	Before pestop	t	Job ran, failed before calling PE stop procedure
13	In pestop	t	Job ran, PE stop procedure failed
14	Before epilog	t	Job ran, failed before calling epilog script
15	In epilog	t	Job ran, failed in epilog script
16	Releasing processor set	t	Job ran, processor set could not be released
24	Migrating (checkpointing jobs)	t	Job ran, job will be migrated

TABLE 7-5 qacct -j failed Field Codes (Continued)

Code	Description	acctvalid	Meaning for Job
25	Rescheduling	t	Job ran, job will be rescheduled
26	Opening output file	f	Job could not be started, stderr/stdout file could not be opened
27	Searching requested shell	f	Job could not be started, shell not found
28	Changing to working directory	f	Job could not be started, error changing to start directory
100	Assumedly after job	t	Job ran, job killed by a signal

The Code column lists the value of the failed field. The Description column lists the text that appears in the `qacct -j` output. If `acctvalid` is set to `t`, the job accounting values are valid. If `acctvalid` is set to `f`, the resource usage values of the accounting record are not valid. The Meaning for Job column indicates whether the job ran or not.

Running Grid Engine System Programs in Debug Mode

For some severe error conditions, the error-logging mechanism might not yield sufficient information to identify the problems. Therefore, the grid engine system offers the ability to run almost all ancillary programs and the daemons in *debug* mode. Different debug levels vary in the extent and depth of information that is provided. The debug levels range from zero through 10, with 10 being the level delivering the most detailed information and zero turning off debugging.

To set a debug level, an extension to your `.cshrc` or `.profile` resource files is provided with the distribution of the grid engine system. For `csh` or `tcsh` users, the file `sge-root/util/dl.csh` is included. For `sh` or `ksh` users, the corresponding file is named `sge-root/util/dl.sh`. The files must be sourced into your standard resource file. As `csh` or `tcsh` user, include the following line in your `.cshrc` file:

```
source sge-root/util/dl.csh
```

As `sh` or `ksh` user, include the following line in your `.profile` file:

```
. sge-root/util/dl.sh
```

As soon as you log out and log in again, you can use the following command to set a debug level:

```
% dl level
```

If *level* is greater than 0, starting a grid engine system command forces the command to write trace output to `STDOUT`. The trace output can contain warning messages, status messages, and

error messages, as well as the names of the program modules that are called internally. The messages also include line number information, which is helpful for error reporting, depending on the debug level you specify.

Note – To watch a debug trace, you should use a window with a large scroll-line buffer. For example, you might use a scroll-line buffer of 1000 lines.

Note – If your window is an xterm, you might want to use the xterm logging mechanism to examine the trace output later on.

If you run one of the grid engine system daemons in debug mode, the daemons keep their terminal connection to write the trace output. You can abort the terminal connections by typing the interrupt character of the terminal emulation you use. For example, you might use Control-C.

To switch off debug mode, set the debug level back to 0.

Setting the dbwriter Debug Level

The `sgepdbwriter` script starts the `dbwriter` program. The script is located in `sge_root/dbwriter/bin/sgepdbwriter`. The `sgepdbwriter` script reads the `dbwriter` configuration file, `dbwriter.conf`. This configuration file is located in `sge_root/cell/common/dbwriter.conf`. This configuration file sets the debug level of `dbwriter`. For example:

```
#
# Debug level
# Valid values: WARNING, INFO, CONFIG, FINE, FINER, FINEST, ALL
#
DBWRITER_DEBUG=INFO
```

You can use the `-debug` option of the `dbwriter` command to change the number of messages that the `dbwriter` produces. In general, you should use the default debug level, which is `info`. If you use a more verbose debug level, you substantially increase the amount of data output by `dbwriter`.

You can specify the following debug levels:

<code>warning</code>	Displays only severe errors and warnings.
<code>info</code>	Adds a number of informational messages. <code>info</code> is the default debug level.
<code>config</code>	Gives additional information that is related to <code>dbwriter</code> configuration, for example, about the processing of rules.

<code>fine</code>	Produces more information. If you choose this debug level, all SQL statements run by <code>dbwriter</code> are output.
<code>finer</code>	For debugging.
<code>finest</code>	For debugging.
<code>all</code>	Displays information for all levels. For debugging.

Diagnosing Problems

The grid engine system offers several reporting methods to help you diagnose problems. The following sections outline their uses.

Pending Jobs Not Being Dispatched

Sometimes a pending job is obviously capable of being run, but the job does not get dispatched. To diagnose the reason, the grid engine system offers a pair of utilities and options, `qstat -j job-id` and `qalter -w v job-id`.

- `qstat -j job-id`

When enabled, `qstat -j job-id` provides a list of reasons why a certain job was not dispatched in the last scheduling run. This monitoring can be enabled or disabled. You might want to disable monitoring because it can cause undesired communication overhead between the `schedd` daemon and `qmaster`. See `schedd_job_info` in the `sched_conf(5)` man page. The following example shows output for a job with the ID 242059:

```
% qstat -j 242059
scheduling info: queue "fangorn.q" dropped because it is temporarily not available
queue "lolek.q" dropped because it is temporarily not available
queue "balrog.q" dropped because it is temporarily not available
queue "saruman.q" dropped because it is full
cannot run in queue "bilbur.q" because it is not contained in its hard queue list (-q)

cannot run in queue "dwain.q" because it is not contained in its hard queue list (-q)
has no permission for host "ori"
```

This information is generated directly by the `schedd` daemon. The generating of this information takes the current usage of the cluster into account. Sometimes this information does not provide what you are looking for. For example, if all queue slots are already occupied by jobs of other users, no detailed message is generated for the job you are interested in.

- `qalter -w v job-id`

This command lists the reasons why a job is not dispatchable in principle. For this purpose, a dry scheduling run is performed. All consumable resources, as well as all slots, are considered to be fully available for this job. Similarly, all load values are ignored because these values vary.

Job or Queue Reported in Error State E

Job or queue errors are indicated by an uppercase E in the `qstat` output.

A job enters the error state when the grid engine system tries to run a job but fails for a reason that is specific to the job.

A queue enters the error state when the grid engine system tries to run a job but fails for a reason that is specific to the queue.

The grid engine system offers a set of possibilities for users and administrators to gather diagnosis information in case of job execution errors. Both the queue and the job error states result from a failed job execution. Therefore the diagnosis possibilities are applicable to both types of error states.

- **User abort mail.** If jobs are submitted with the `qsub -m a` command, abort mail is sent to the address specified with the `-M user[@host]` option. The abort mail contains diagnosis information about job errors. Abort mail is the recommended source of information for users.
- **qacct accounting.** If no abort mail is available, the user can run the `qacct -j` command. This command gets information about the job error from the grid engine system's job accounting function.
- **Administrator abort mail.** An administrator can order administrator mails about job execution problems by specifying an appropriate email address. See under `administrator_mail` on the `sge_conf(5)` man page. Administrator mail contains more detailed diagnosis information than user abort mail. Administrator mail is the recommended method in case of frequent job execution errors.
- **Messages files.** If no administrator mail is available, you should investigate the `qmaster` messages file first. You can find entries that are related to a certain job by searching for the appropriate job ID. In the default installation, the `qmaster` messages file is `sge-root/cell/spool/qmaster/messages`.

You can sometimes find additional information in the messages of the `execd` daemon from which the job was started. Use `qacct -j job-id` to discover the host from which the job was started, and search in `sge-root/cell/spool/host/messages` for the job ID.

Troubleshooting Common Problems

This section provides information to help you diagnose and respond to the cause of common problems.

- **Problem** — The output file for your job says, `Warning: no access to tty; thus no job control in this shell...`
 - **Possible cause** — One or more of your login files contain an `stty` command. These commands are useful only if a terminal is present.
 - **Possible solution** — No terminal is associated with batch jobs. You must remove all `stty` commands from your login files, or you must bracket such commands with an `if` statement. The `if` statement should check for a terminal before processing. The following example shows an `if` statement:

```
/bin/csh:
stty -g          # checks terminal status
if ($status == 0) # succeeds if a
terminal is present
<put all stty commands in here>
endif
```

- **Problem** — The job standard error log file says `'tty': Ambiguous`. However, no reference to `tty` exists in the user's shell that is called in the job script.
 - **Possible cause** — `shell_start_mode` is, by default, `posix_compliant`. Therefore all job scripts run with the shell that is specified in the queue definition. The scripts do not run with the shell that is specified on the first line of the job script.
 - **Possible solution** — Use the `-S` flag to the `qsub` command, or change `shell_start_mode` to `unix_behavior`.
- **Problem** — You can run your job script from the command line, but the job script fails when you run it using the `qsub` command.
 - **Possible cause** — Process limits might be being set for your job. To test whether limits are being set, write a test script that performs `limit` and `limit -h` functions. Run both functions interactively, at the shell prompt and using the `qsub` command, to compare the results.
 - **Possible solution** — Remove any commands in configuration files that sets limits in your shell.
- **Problem** — Execution hosts report a load of 99.99.
 1. **Possible cause** — The `execd` daemon is not running on the host.
 - **Possible solution** — As root, start up the `execd` daemon on the execution host by running the `$SGE_ROOT/default/common/'rcsge'` script.
 2. **Possible cause** — A default domain is incorrectly specified.

Possible solution — As the grid engine system administrator, run the `qconf -mconf` command and change the `default_domain` variable to none.

3. **Possible cause** — The `qmaster` host sees the name of the execution host as different from the name that the execution host sees for itself.

Possible solution — If you are using DNS to resolve the host names of your compute cluster, configure `/etc/hosts` and NIS to return the fully qualified domain name (FQDN) as the primary host name. Of course, you can still define and use the short alias name, for example, `168.0.0.1 myhost.dom.com myhost`.

If you are *not* using DNS, make sure that all of your `/etc/hosts` files and your NIS table are consistent, for example, `168.0.0.1 myhost.corp myhost` or `168.0.0.1 myhost`

- **Problem** — Every 30 seconds a warning that is similar to the following message is printed to `cell/spool/host/messages`:

```
Tue Jan 23 21:20:46 2001|execd|meta|W|local
configuration meta not defined - using global configuration
```

But `cell/common/local_conf` contains a file for each host, with FQDN.

- **Possible cause** — The host name resolving at your machine `meta` returns the short name, but at your master machine, `meta` with FQDN is returned.
- **Possible solution** — Make sure that all of your `/etc/hosts` files and your NIS table are consistent in this respect. In this example, a line such as the following text could erroneously be included in the `/etc/hosts` file of the host `meta`:

```
168.0.0.1 meta meta.your.domain
```

The line should instead be:

```
168.0.0.1 meta.your.domain meta.
```

- **Problem** — Occasionally you see CHECKSUM ERROR, WRITE ERROR, or READ ERROR messages in the messages files of the daemons.
- **Possible cause** — As long as these messages do not appear in a one-second interval, you need not do anything. These messages typically can appear between 1 and 30 times a day.
- **Problem** — Jobs finish on a particular queue and return the following message in `qmaster/messages`:

```
Wed Mar 28 10:57:15 2001|qmaster|masterhost|I|job 490.1
finished on host execest
```

Then you see the following error messages in the execution host's `execest/messages` file:

```
Wed Mar 28 10:57:15 2001|execd|execest|E|can't find directory
"active_jobs/490.1" for reaping job 490.1
```

```
Wed Mar 28 10:57:15 2001|execd|exechost|E|can't remove directory
"active_jobs/490.1": opendir(active_jobs/490.1) failed:
Input/output error
```

- **Possible cause** — The \$SGE_ROOT directory, which is automounted, is being unmounted, causing the sge_execd daemon to lose its current working directory.
- **Possible solution** — Use a local spool directory for your execd host. Set the parameter `execd_spool_dir`, using `qmon` or the `qconf` command.
- **Problem** — When submitting interactive jobs with the `qrsh` utility, you get the following error message:

```
% qrsh -l mem_free=1G error: error: no suitable queues
```

However, queues are available for submitting batch jobs with the `qsub` command. These queues can be queried using `qhost -l mem_free=1G` and `qstat -f -l mem_free=1G`.

- **Possible cause** — The message `error: no suitable queues` results from the `-w e` submit option, which is active by default for interactive jobs such as `qrsh`. Look for `-w e` on the `qrsh(1)` man page. This option causes the submit command to fail if the `qmaster` does not know for sure that the job is dispatchable according to the current cluster configuration. The intention of this mechanism is to decline job requests in advance, in case the requests can't be granted.
- **Possible solution** — In this case, `mem_free` is configured to be a consumable resource, but you have not specified the amount of memory that is to be available at each host. The memory load values are deliberately not considered for this check because memory load values vary. Thus they can't be seen as part of the cluster configuration. You can do one of the following:
 - Omit this check generally by explicitly overriding the `qrsh` default option `-w e` with the `-w n` option. You can also put this command into `sge-root/cell/common/cod_request`.
 - If you intend to manage `mem_free` as a consumable resource, specify the `mem_free` capacity for your hosts in `complex_values` of `host_conf` by using `qconf -me hostname`.
 - If you do *not* intend to manage `mem_free` as a consumable resource, make it a nonconsumable resource again in the consumable column of `complex(5)` by using `qconf -mc hostname`.
- **Problem** — `qrsh` won't dispatch to the same node it is on. From a `qsh` shell you get a message such as the following:

```
host2 [49]% qrsh -inherit host2 hostname
error: executing task of job 1 failed:
```

```
host2 [50]% qrsh -inherit host4 hostname
host4
```

- **Possible cause** — `gid_range` is not sufficient. `gid_range` should be defined as a range, not as a single number. The grid engine system assigns each job on a host a distinct `gid`.
- **Possible solution** — Adjust the `gid_range` with the `qconf -mconf` command or with `QMON`. The suggested range is as follows:

```
gid_range                20000-20100
```

- **Problem** — `qrsh -inherit -V` does not work when used inside a parallel job. You get the following message:

```
cannot get connection to "qlogin_starter"
```

- **Possible cause** — This problem occurs with nested `qrsh` calls. The problem is caused by the `-V` option. The first `qrsh -inherit` call sets the environment variable `TASK_ID`. `TASK_ID` is the ID of the tightly integrated task within the parallel job. The second `qrsh -inherit` call uses this environment variable for registering its task. The command fails as it tries to start a task with the same ID as the already-running first task.
- **Possible solution** — You can either unset `TASK_ID` before calling `qrsh -inherit`, or choose to use the `-v` option instead of `-V`. This option exports only the environment variables that you really need.
- **Problem** — `qrsh` does not seem to work at all. Messages like the following are generated:

```
host2$ qrsh -verbose hostname
local configuration host2 not defined - using global configuration
waiting for interactive job to be scheduled ...
Your interactive job 88 has been successfully scheduled.
Establishing /share/gridware/utilbin/solaris64/rsh session
to host exehost ...
rcmd: socket: Permission denied
/share/gridware/utilbin/solaris64/rsh exited with exit code 1
reading exit code from shepherd ...
error: error waiting on socket for client to connect:
Interrupted system call
error: error reading return code of remote command
cleaning up after abnormal exit of
/share/gridware/utilbin/solaris64/rsh
host2$
```

- **Possible cause** — Permissions for `qrsh` are not set properly.
- **Possible solution** — Check the permissions of the following files, which are located in `$SGE_ROOT/utilbin/`. (Note that `rlogin` and `rsh` must be setuid and owned by root.)


```
-r-s--x--x 1 root root 28856 Sep 18 06:00 rlogin*
-r-s--x--x 1 root root 19808 Sep 18 06:00 rsh*
```

```
-rwxr-xr-x 1 sgeadmin adm 128160 Sep 18 06:00 rshd*
```

Note – The *sge-root* directory also needs to be NFS-mounted with the *setuid* option. If *sge-root* is mounted with *nosuid* from your submit client, *qrsh* and associated commands will not work.

- **Problem** – When you try to start a distributed make, *qmake* exits with the following error message:

```
qrsh_starter: executing child process
qmake failed: No such file or directory
```

- **Possible cause** — The grid engine system starts an instance of *qmake* on the execution host. If the grid engine system environment, especially the *PATH* variable, is not set up in the user's shell resource file (*.profile* or *.cshrc*), this *qmake* call fails.
- **Possible solution** — Use the *-v* option to export the *PATH* environment variable to the *qmake* job. A typical *qmake* call is as follows:

```
qmake -v PATH -cwd -pe make 2-10 --
```

- **Problem** — When using the *qmake* utility, you get the following error message:

```
waiting for interactive job to be scheduled ...timeout (4 s)
expired while waiting on socket fd 5
```

Your "qrsh" request could not be scheduled, try again later.

- **Possible cause** — The *ARCH* environment variable could be set incorrectly in the shell from which *qmake* was called.
- **Possible solution** – Set the *ARCH* variable correctly to a supported value that matches an available host in your cluster, or else specify the correct value at submit time, for example, *qmake -v ARCH=solaris64 ...*

Typical Accounting and Reporting Console Errors

Problem: The installation of the Sun Web console Version 2.0.3 fails with the follow error message:

```
# ./inst_reporting
...
Register the N1 SGE reporting module in the webconsole

Registering com.sun.grid.arco_6u3.
```

Starting Sun(TM) Web Console Version 2.0.3...

Ambiguous output redirect.

Solution: . This Sun Web Console Version can only be installed by the user noaccess who has /bin/sh as their login shell. The user must be added with the following command:

```
# useradd -u 60002 -g 60002 -d /tmp -s /bin/sh -c "No Access User" noaccess
```

Problem: The table/view dropdown menu of a simple query definition does not contain any entry, but the tables are defined in the database.

Solution: The problem normally occurs if Oracle is used as the database. During the installation of the reporting module the wrong database schema name has been specified. For Oracle, the database schema name is equal to the name of the database user which is used by dbwriter (the default name is arco_write). For Postgres, the database schema name should be public.

Problem: Connection refused.

Solution: The smcwebserver might be down. Start or restart the smcwebserver.

Problem: The list of queries or the list of results is empty.

Solution: The cause can be any of the following:

- The database is down. Start or restart the database.
- No more database connections are available. Increase the number of allowable connections to the database.
- An error exists in the configuration file of the application. Check the configuration for wrong database users, wrong user passwords, or wrong type of database, and then restart the application.
- No queries are available. If the query directory /var/spool/arco/queries is not empty, the following errors might have occurred:
 - Queries in the XML files are syntactically incorrect. Check the log file for error messages from the XML parser.
 - User noaccess has no read or write permissions on the query directory.

Problem: The list of available database tables is empty.

Solution: The cause can be any of the following:

- The database is down. Start or restart the database.
- No more database connections are available. Increase the number of allowable connections to the database.

- An error exists in the configuration file of the application. Check the configuration for wrong database users, wrong user passwords, or wrong type of database, and then restart the application.

Problem: The list of selectable fields is empty.

Solution: No table is selected. Select a table from the list.

Problem: The list of filters is empty.

Solution: No fields are selected. Define at least one field.

Problem: The sort list is empty.

Solution: No fields are selected. Define at least one field.

Problem: A defined filter is not used.

Solution: The filter may be inactive. Modify the unused filter and make it active.

Problem: The late binding in the advanced query is ignored, but the execution runs into an error.

Solution: The late binding macro has a syntactical error. The correct syntax for the late binding macro in the advanced query is as follows:

```
latebinding{attribute;operator}
latebinding{attribute;operator;defaultvalue}
```

Problem: The breadcrumb is used to move back, but the login screen is shown.

Solution: The session timed out. Log in again, or raise the session time in the app.xml.

Problem: The view configuration is defined, but the default configuration is shown.

Solution: The defined view configuration is not set to be visible. Open the view configuration and define the view configuration to be used.

Problem: The view configuration is defined, but the last configuration is shown.

Solution: The defined view configuration is not set to be visible. Open the view configuration and define the view configuration to be used.

Problem: The execution of a query takes a very long time.

Solution: The results coming from the database are very large. Set a limit for the results, or extend the filter conditions.

Database Schemas

This appendix contains database schema information in a series of tables. The topics include:

Schema Tables

sgc_job

The `sgc_job` table contains one record for each array task (one record for non array jobs with the array task number 1) and for each parallel task started in a tightly integrated parallel job.

For N1GE 6.0 systems, a record is created as soon as a job, an array task, or a parallel task is scheduled. It is updated during the job's runtime.

A short description of N1GE jobs, array jobs, parallel jobs and their differences can be found in this *Sun Grid Engine User's Guide*. The Glossary may be especially useful as an introduction.

Column	Type	Description
j_id	Integer	Unique record identifier
j_job_number	integer	JOB_ID
j_task_number	integer	Array task id.
j_pe_taskid	text	ID of a task of a tightly integrated parallel task.
j_job_name	text	job name (script name or value set with the submit option -N)

Column	Type	Description
j_group	text	UNIX group name of the primary group the job was executed in. References the group table.
j_owner	text	UNIX user account the job was running in. References the user table.
j_account	text	Account string set with the submit option -A.
j_priority	integer	Priority set with the submit option -p or assigned from the queue configuration.
j_submission_time	timestamp	Time of the job submission.
j_project	text	Project (only in Sun Grid Engine, Enterprise Edition) References the project table.
j_department	text	Department (only in Sun Grid Engine, Enterprise Edition) References the department table.

sge_job_usage

The sge_job_usage table holds the job's resource usage over time.

For N1GE 5.3 systems, only one record exists per finished job, array task and parallel task. The ju_curr_time column holds the job's end time (j_end_time in sge_job).

For N1GE 6.0 systems, the online usage is stored as well; this results in multiple records for one job, array task, and parallel task stored in sge_job. The resource usage of a job can be monitored over time (ju_curr_time), the last record per job, array task, or parallel task holds the total usage that can be used in accounting, ju_curr_time for this record will equal j_end_time from sge_job.

Column	Type	Description
ju_id	Integer	Unique record identifier
ju_parent	Integer	Reference to sge_job table

Column	Type	Description
ju_curr_time	Integer	current time for usage
ju_qname	text	Name of the queue the job was running in. In N1GE 6.0 systems this will be the cluster queue name. References to queues in the queue table.
ju_hostname	text	Name of the host the job was running on. References to hosts in the host table.
ju_start_time	timestamp	Time when the job was started.
ju_end_time	timestamp	Time when the job finished.
ju_failed	integer	if != 0 indicates a problem
ju_exit_status	integer	exit status of the job
ju_granted_pe	text	The parallel environment which was selected for that job.
ju_slots	integer	The number of slots which were dispatched to the job.
ju_state	text	job state
ju_ru_wallclock	integer	end_time – start_time
ju_ru_utime	double	user time used
ju_ru_stime	double	system time used
ju_ru_maxrss	integer	maximum resident set size
ju_ru_ixrss	integer	currently 0
ju_ru_ismrss	integer	
ju_ru_idrss	integer	integral resident set size
ju_ru_isrss	integer	currently 0
ju_ru_minflt	integer	page faults not requiring physical I/O
ju_ru_majflt	integer	page faults requiring physical I/O
ju_ru_nswap	integer	swaps

Column	Type	Description
ju_ru_inblock	integer	block input operations
ju_ru_oublock	integer	block output operations
ju_ru_msgsnd	integer	messages sent
ju_ru_msgrcv	integer	messages received
ju_ru_nsignals	integer	signals received
ju_ru_nvcsw	integer	voluntary context switches
ju_ru_nivcsw	integer	involuntary context switches
ju_cpu	double	The cpu time usage in seconds.
ju_mem	double	The integral memory usage in Gbytes seconds.
ju_io	double	The amount of data transferred in input/output operations.
ju_iow	double	The io wait time in seconds.
ju_maxvmem	double	The maximum vmem size in bytes.

sge_job_request

Stores resources a job's requests.

Two types of requests (qsub options) are currently handled:

1. -l resource requests, e.g. -l arch=solaris,mem_total=100M

For each request one record is created.

2. -q queue request, e.g. -q balrog.q

One record is created containing “queue” as variable and the request contents as variable.

Column	Type	Description
jr_id	Integer	Unique record identifier
jr_parent	Integer	reference to the sge_job table
jr_variable	text	name of the requested complex variable
jr_value	text	requested value

sgc_job_log

The sgc_job_log table contains job logging information.

Column	Type	Description
jl_id	Integer	Unique record identifier
jl_parent	integer	Reference to sgc_job table
jl_time	unix timestamp	Time when the job login entry was generated.
jl_event	text	
jl_job_number	integer	
jl_task_number	integer	
jl_pe_task_id	text	
jl_state	text	job state after the reported event
jl_user	text	user who initiated action for the event
jl_host	text	host on which the event action was initiated
jl_state_time	unix timestamp	describes, how long the job was in a certain state, see description below
jl_message	text	a message explaining what happened

sgc_share_log

The sgc_share_log table contains information about the N1GE(EE) sharetree configuration and usage.

Further information can be found in the N1GE manual [sharetree\(5\)](#).

Column	Type	Description
sl_id	Integer	Unique identifier for share log record
sl_curr_time	timestamp	Current time

Column	Type	Description
sl_usage_time	timestamp	Usage time
sl_node	text	Node name in the sharetree
sl_user	text	Name of the user (job owner) References the user table.
sl_project	text	Name of the project References the project table.
sl_shares	integer	shares configured in sharetree
sl_job_count	integer	number of jobs that are considered for share tree policy
sl_level	double	share in % within this tree level
sl_total	double	total share in % within whole sharetree
sl_long_target_share	double	targeted long term share in %
sl_short_target_share	double	targeted short term share in %
sl_actual_share	double	actual share in %
sl_usage	double	combined usage, weight of cpu, mem and io can be configured
sl_cpu	double	cpu usage in seconds
sl_mem	double	integral memory usage in Gbyte seconds
sl_io	double	The amount of data transferred in input/output operations.
sl_ltcpu	double	long term cpu
sl_ltmem	double	long term mem
sl_ltio	double	long term io

sgc_host

The sgc_host table lists all hosts in the Cluster.

Column	Type	Description
h_id	Integer	Unique host identifier
h_hostname	text	The hostname.

sgc_host_values

The sgc_host_values table stores the values of host variables that are subject to change, e.g. the load average.

In addition, derived host values will be stored, e.g. hourly averages, sums etc.

Column	Type	Description
hv_hostname	text	References the host table.
hv_time_start	timestamp	Start time for the validity of a value.
hv_time_end	timestamp	End time for the validity of a value.
hv_variable	text	Variable name, e.g. load_avg.
hv_value	text	Variable value, e.g. 0.34.
hv_dvalue	double precision	Variable value as number.
hv_dconfig	double precision	In case of consumables: Consumable maximum available value (configured value).

sgc_queue

The sgc_queue table lists all queues configured in the cluster.

Column	Type	Description
q_id	Integer	Unique queue identifier
q_qname	text	Name of the queue
q_hostname	text	Name of host

sgc_queue_values

The sgc_queue_values table stores the values of queue variables that are subject to change, e.g. the number of free slots.

In addition, derived queue values will be stored, e.g. hourly averages, sums etc.

Column	Type	Description
qv_parent	integer	References q_id in the sgc_queue table.
qv_time_start	timestamp	Start time for the validity of a value.
qv_time_end	timestamp	End time for the validity of a value.
qv_variable	text	Variable name, e.g. slots.
qv_value	text	Variable value, e.g. 5.
qv_dvalue	double precision	Variable value as number.
qv_dconfig	double precision	In case of consumables: Consumable maximum available value (configured value).

sgc_department

Lists all departments referenced in the database.

Column	Type	Description
d_id	Integer	Unique department identifier
d_department	text	Name of the department.

Table 9: The sgc_department Table

sgc_department_values

The sgc_department_values table stores the values of department related variables that are subject to change. Currently these are derived values, e.g. hourly averages, sums etc.

Column	Type	Description
dv_parent	integer	References d_id in the sge_department table.
dv_time_start	timestamp	Start time for the validity of a value.
dv_time_end	timestamp	End time for the validity of a value.
dv_variable	text	Variable name, e.g. h_sum_jobs
dv_value	text	Variable value, e.g. 5.
dv_dvalue	double precision	Variable value as number.
dv_dconfig	double precision	In case of consumables: Consumable maximum available value (configured value).

sge_project

Lists all projects referenced in the database.

Column	Type	Description
p_id	Integer	Unique project identifier
p_project	text	Name of the project.

sge_project_values

The sge_project_values table stores the values of project related variables that are subject to change. Currently these values are derived values, e.g. hourly averages, sums etc.

Column	Type	Description
pv_parent	integer	References q_id in the sge_queue table.
pv_time_start	timestamp	Start time for the validity of a value.
pv_time_end	timestamp	End time for the validity of a value.
pv_variable	text	Variable name, e.g. h_avg_cpu
pv_value	text	Variable value, e.g. 345.5
pv_dvalue	double precision	Variable value as number.

Column	Type	Description
pv_dconfig	double precision	In case of consumables: Consumable maximum available value (configured value).

sge_user

Lists all users referenced in the database.

Column	Type	Description
u_id	Integer	Unique user id
u_user	text	Name of the user.

sge_user_values

The sge_user_values table stores the values of user related variables that are subject to change. These values are currently derived queue values, e.g. hourly averages, sums etc.

Column	Type	Description
uv_parent	integer	References q_id in the sge_queue table.
uv_time_start	timestamp	Start time for the validity of a value.
uv_time_end	timestamp	End time for the validity of a value.
uv_variable	text	Variable name, e.g. h_sum_cpu
uv_value	text	Variable value, e.g. 23.2
uv_dvalue	double precision	Variable value as number.
uv_dconfig	double precision	In case of consumables: Consumable maximum available value (configured value).

sge_group

Lists all user groups referenced in the database.

Column	Type	Description
g_id	Integer	Unique group id
g_group	text	Name of the group.

Table 15: The sge_group Table

sge_group_values

The sge_group_values table stores the values of group related variables that are subject to change. These are currently derived values, e.g. hourly averages, sums etc.

Column	Type	Description
gv_parent	integer	References q_id in the sge_queue table.
gv_time_start	timestamp	Start time for the validity of a value.
gv_time_end	timestamp	End time for the validity of a value.
gv_variable	text	Variable name, e.g. h_sum_jobs.
gv_value	text	Variable value, e.g. 53
gv_dvalue	double precision	Variable value as number.
gv_dconfig	double precision	In case of consumables: Consumable maximum available value (configured value).

List of Predefined Views

view_accounting

Accounting records for jobs, array tasks, and tightly integrated tasks. Contains only finished jobs.

Column	Type	Description
job_number	integer	Job number

Column	Type	Description
task_number	integer	Array task id
pe_taskid	text	ID of a tightly integrated parallel task
name	text	job name (script name or value set with the submit option -N)
groupname	text	UNIX group name of the primary group the job was executed in. References the group table.
username	text	UNIX user account the job was running in. References the user table.
account	text	Account string set with the submit option -A
project	text	Project, References the project table
department	text	Department, References the department table
submission_time	timestamp	Time of the job submission
start_time	timestamp	Time when the job was started
end_time	timestamp	Time when the job finished
wallclock_time	integer	end_time - start_time
cpu	double	The CPU time usage in seconds
io	double	The amount of data transferred in input/output operations
iow	double	The io wait time in seconds
maxvmem	double	The maximum vmem size in bytes
wait_time	integer	start_time - submission_time
turnaround_time	integer	end_time - submission_time

view_job_times

This is the same as view_accounting, but no tasks of tightly integrated parallel jobs are listed.

view_jobs_completed

Finished jobs per hour, one record per hour.

Column	Type	Description
completed	integer	Completed jobs
time	timestamp	Time when the jobs finished

view_job_log

Job logging (e.g. Submission, state changes, job finish).

Column	Type	Description
job_number	integer	Job number
task_number	integer	Array task id
pe_taskid	text	ID of a tightly integrated parallel task
name	text	job name (script name or value set with the submit option -N)
username	text	UNIX group name of the primary group the job was executed in. References the group table
account	text	UNIX user account the job was running in. References the user table
project	text	Project. References the project table
department	text	Department. References the department table
time	timestamp	Time when the job logging entry was generated
event	text	Event being recorded
state	text	Job state after the reported event
initiator	text	User who initiated action for the event

Column	Type	Description
host	text	Host on with the event action was initiated
message	text	A message explaining what happened

view_department_values

Department specific variables.

Column	Type	Description
department	text	Name of the department
time_start	timestamp	Start time for the validity of a value
time_end	timestamp	End time for the validity of a value
variable	text	Variable name, e.g. h_sum_jobs
str_value	text	Variable value, e.g. 5
num_value	double precision	Variable value as number
num_config	double precision	In case of consumables: Consumable maximum available value (configured value)

view_group_values

Group specific variables

Column	Type	Description
groupname	text	Name of the group
time_start	timestamp	Start time for the validity of a value
time_end	timestamp	End time for the validity of a value
variable	text	Variable name, e.g. h_sum_jobs
str_value	text	Variable value, e.g. 53
num_value	double precision	Variable value as number

Column	Type	Description
num_config	double precision	In case of consumables: Consumable maximum available value (configured value)

view_host_values

Host specific variables

Column	Type	Description
hostname	text	The hostname
time_start	timestamp	Start time for the validity of a value
time_end	timestamp	End time for the validity of a value
variable	text	Variable name, e.g. load_avg
str_value	text	Variable value, e.g. 0.34
num_value	double precision	Variable value as number
num_config	double precision	In case of consumables: Consumable maximum available value (configured value)

view_project_values

Project specific variables

Column	Type	Description
project	text	Name of the project
time_start	timestamp	Start time for the validity of a value
time_end	timestamp	End time for the validity of a value
variable	text	Variable name, e.g. h_avg_cpu
str_value	text	Variable value, e.g. 345.5
num_value	double precision	Variable value as number

Column	Type	Description
num_config	double precision	In case of consumables: Consumable maximum available value (configured value)

view_queue_values

Queue specific variables

Column	Type	Description
qname	text	Name of the queue
hostname	text	Name of host
time_start	timestamp	Start time for the validity of a value
time_end	timestamp	End time for the validity of a value
variable	text	Variable name, e.g. slots
str_value	text	Variable value, e.g. 5
num_value	double precision	Variable value as number
num_config	double precision	In case of consumables: Consumable maximum available value (configured value)

view_user_values

User specific variables.

Column	Type	Description
username	text	Name of the user
time_start	timestamp	Start time for the validity of a value
time_end	timestamp	End time for the validity of a value
variable	text	Variable name, e.g. h_sum_cpu
str_value	text	Variable value, e.g. 23.2
num_value	double precision	Variable value as number

Column	Type	Description
num_config	double precision	In case of consumables: Consumable maximum available value (configured value)

List of Derived Values

Derived values stored in the database can highly reduce query processing time. The reporting database contains aggregated values (sum, average, min, max) on an hourly basis. After some time period (e.g. one year), these values can even be further compressed to daily, weekly or monthly values.

The following derived values are delivered:

table	variable	description
sge_host_values	h_sum_cpu, d_sum_cpu, m_sum_cpu	cpu usage per host and hour, day, month
sge_user_values	h_sum_cpu, d_sum_cpu, m_sum_cpu	cpu usage per user and hour, day, month
sge_group_values	h_sum_cpu, d_sum_cpu, m_sum_cpu	cpu usage per group and hour, day, month
sge_department_values	h_sum_cpu, d_sum_cpu, m_sum_cpu	cpu usage per department and hour, day, month
sge_project_values	h_sum_cpu, d_sum_cpu, m_sum_cpu	cpu usage per project and hour, day, month
sge_host_values	h_avg_load	average host load
sge_host_values	h_max_load	maximum host load

Rules for the generation of any derived value can be specified in a configuration file.

Glossary

access list	A list of users and UNIX groups who are permitted or denied access to a resource such as a queue or a host. Users and groups can belong to multiple access lists, and the same access lists can be used in various contexts.
administration host	Administration hosts are hosts that have permission to carry out administrative activity for the grid engine system.
array job	A job made up of a range of independent identical tasks. Each task is similar to a separate job. Array job tasks differ among themselves only by having unique task identifiers, which are integer numbers.
batch job	A batch job is a UNIX shell script that can be run without user intervention and does not require access to a terminal.
campus grid	A grid that enables multiple projects or departments within an organization to share computing resources.
cell	A separate cluster with a separate configuration and a separate master machine. Cells can be used to loosely couple separate administrative units.
checkpointing	A procedure that saves the execution status of a job into a <i>checkpoint</i> , thereby allowing the job to be aborted and resumed later without loss of information and already completed work. The process is called <i>migration</i> if the checkpoint is moved to another host before execution resumes.
checkpointing environment	A grid engine system configuration entity that defines events, interfaces, and actions that are associated with a certain method of checkpointing.
cluster	A collection of machines, called hosts, on which grid engine system functions occur.
cluster grid	The simplest form of a grid, consisting of computer <i>hosts</i> working together to provide a single point of access to users in a single project or department.
cluster queue	A container for a class of jobs that are allowed to run concurrently. A queue determines certain job attributes, for example, whether it can be migrated. Throughout its lifetime, a running job is

associated with its queue. Association with a queue affects some of the things that can happen to a job. For example, if a queue is suspended, all jobs associated with that queue are also suspended.

complex A set of resource attribute definitions that can be associated with a queue, a host, or the entire cluster.

department A list of users and groups who are treated alike in the functional and override scheduling policies of the grid engine system. Users and groups can belong to only one department.

entitlement The same as *share*. The amount of resources that are planned to be consumed by a certain job, user, user group, or project.

execution host Systems that have permission to run grid engine system jobs. These systems host queue instances, and run the execution daemon `sge_execd`.

functional policy A policy that assigns specific levels of importance to jobs, users, user groups, and projects. For instance, through the functional policy, a high-priority project and all its jobs can receive a higher resource share than a low-priority project.

global grid A collection of campus grids that cross organizational boundaries to create very large virtual systems.

grid A collection of computing resources that perform tasks. Users treat the grid as a *single* computational resource.

group A UNIX group.

hard resource requirements The resources that must be allocated before a job can be started. Contrast with *soft resource requirements*.

host A system on which grid engine system functions occur.

interactive job An interactive job is a session started with the commands `qrsh`, `qsh`, or `qlogin`, which open an *xterm* window for user interaction or provide the equivalent of a remote login session.

job A request from a user for computational resources from the grid.

job class A set of jobs that are equivalent in some sense and treated similarly. A job class is defined by the identical requirements of the corresponding jobs and by the characteristics of the queues that are suitable for those jobs.

manager	A user who can manipulate all aspects of the grid engine software. The superusers of the master host and of any other machine that is declared to be an administration host have manager privileges. Manager privileges can be assigned to nonroot user accounts as well.
master host	The master host is central to the overall cluster activity. It runs the master daemon <code>sge_qmaster</code> and the scheduler daemon <code>sge_schedd</code> . By default, the master host is also an administration host and a submit host.
migration	The process of moving a checkpointing job from one host to another before execution of the job resumes.
operator	Users who can perform the same commands as managers except that they cannot change the configuration. Operators are supposed to maintain operation.
override policy	A policy commonly used to override the automated resource entitlement management of the functional and share-based policies. The cluster administrator can modify the automated policy implementation to assign override to jobs, users, user groups, and projects.
owner	Users who can suspend or resume, and disable or enable, the queues they own. Typically, users are owners of the queue instances that reside on their workstations.
parallel environment	A grid engine system configuration that defines the necessary interfaces for the grid engine software to correctly handle parallel jobs.
parallel job	A job that is made up of more than one closely correlated task. Tasks can be distributed across multiple hosts. Parallel jobs usually use communication tools such as shared memory or message passing (MPI, PVM) to synchronize and correlate tasks.
policy	A set of rules and configurations that the administrator can use to define the behavior of the grid engine system. Policies are implemented automatically by the system.
priority	The relative level of importance of a job compared to others.
project	A grid engine system project.
resource	A computational device consumed or occupied by running jobs. Typical examples are memory, CPU, I/O bandwidth, file space, software licenses, and so forth.
share	The same as <i>entitlement</i> . The amount of resources that are planned to be consumed by a certain job, user, or project.
share-based policy	A policy that allows definition of the entitlements of user and projects and arbitrary groups thereof in a hierarchical fashion. An enterprise, for instance, can be subdivided into divisions, departments, projects active in the departments, user groups working on those projects, and

users in those user groups. The share-based hierarchy is called a share-tree, and once a share-tree is defined, its entitlement distribution is automatically implemented by the grid engine software.

share-tree The hierarchical definition of a share-based policy.

soft resource requirements Resources that a job needs but that do not have to be allocated before a job can be started. Allocated to a job on an as-available basis. Contrast with *hard resource requirements*.

submit host Submit hosts allow for submitting and controlling *batch jobs only*. In particular, a user who is logged in to a submit host can submit jobs using `qsub`, can control the job status using `qstat`, and can use the grid engine system's OSF/1 Motif graphical user interface `QMON`.

suspension The process of holding a running job but keeping it on the execution host (in contrast to checkpointing, where the job is aborted). A suspended job still consumes some resources, such as swap memory or file space.

ticket A generic unit for resource share definition. The more ticket shares that a job, user, project, or other component has, the more important it is. If a job has twice as many tickets as another job, for example, that job is entitled to twice the resource consumption.

usage Another term for "resources consumed." Usage is determined by an administrator-configurable weighted sum of CPU time consumed, memory occupied over time, and amount of I/O performed.

users People who can submit jobs to the grid and run them if they have a valid login ID on at least one submit host and one execution host.

userset Either an *access list* or a *department*.

Index

Numbers and Symbols

\$pe_hostfile, 58
3rd_party file, 76, 77, 79

A

access-allowed-list, 36
access-denied-list, 36
access list, 175
access lists, 36
accounting and reporting console, 111-112
 advanced query
 create, 124
 run, 125
 creating queries
 simple, 113-117
 editing queries
 simple, 123
 query
 advanced, 124, 125
 simple, 113-117, 123
 running queries
 advanced, 124, 125
 simple, 123
 simple query
 create, 113-117
 edit, 123
 run, 123
 starting, 111-112
ACLs, 36
act_qmaster file, 41

administration host, 175
administration hosts, 27
 listing, 42
advanced jobs
 example, 65-66
 submitting, 63-66, 66-68
app-defaults directory, 34
ARC, 57
arguments in scripts, 56
array, job, 175
array jobs, 55, 71-72
 index, 71
 tasks, 71
at jobs, 106
attributes, *See* resource attributes, requestable
 attributes, consumable resources

B

-b qrsh option, 67, 77
batch jobs, submitting, 53-59
batch qmake, 81
batch queues, 83

C

C, critical message, 142
C program integration, 78
-C qsub option, 56
-c qsub option, 108
-c qtcsh option, 78

- campus grid, 20, 175
- cell, 57, 175
- Certificate Security Protocol (CSP), 22
- checkpointing, 106-110, 175
 - and restarting, 58
 - disk space requirements, 109
 - file system requirements, 110
 - kernel-level, 106
 - memory requirements, 109
 - migrating jobs, 107
 - process hierarchies, 106
 - user-level, 106
- checkpointing directory, 109
- checkpointing environment, 175
- ckpt_dir, 109
- clear qsub option, 67
- cluster, 175
- cluster grid, 20, 175
- cluster queues, 28
 - configuring, 100
 - disabling, 100
 - enabling, 100
 - resuming, 100
 - suspending, 100
- command line interface, 29
- comment lines, 56-57
- complex, 176
- configuring queues, 100
- consistency checking, 89
- consumable flag, 45
- consumable resources, 103, 104
- critical message, 142
- crontab jobs, 106
- csh shell, 53
- .cshrc file, 54
- customizing
 - QMON, 30, 34-35, 92, 105

D

- d qmod option, 105
- daemons
 - execution, 27, 28
 - master, 27, 28

- daemons (*Continued*)
 - scheduler, 27, 28
- debug mode, 145
 - trace output, 145
- debugging with dl, 145
- default request files, 67-68
- department, 176
- dependencies
 - job, 61, 71
- disabling
 - cluster queues, 100
 - queue instances, 102
 - queues with qmod, 105
- disk space, requirements for checkpointing, 109
- dispatching jobs, with named queue requests, 83
- DISPLAY variable, 73
- displaying, queues, 38-41
- dl, 145
- dynamic load balancing, 107

E

- E, error message, 142
- e qmod option, 105
- email, 98, 141
 - format of error mail, 142
 - sent at beginning of job, 98
 - sent at end of job, 98
 - sent when job is aborted, 98
 - sent when job is suspended, 98
- embedding of qsub arguments, 56
- enabling
 - cluster queues, 100
 - queue instances, 102
- entitlement, 176
- ENVIRONMENT, 58
- environment
 - checkpointing, 175
 - parallel, 177
- environment variables, 57-59
- error message, 142
- error reporting, 145
- errors
 - job state, 90

errors (*Continued*)

- queue state, 100
- /etc/login file, 54
- execution daemon, 27, 28
- execution host, 176
- execution hosts, 27
 - listing, 42
- ext qstat option, 83
- extended jobs
 - example, 62
 - submitting, 59-63, 63

F

- f qdel option, 97
- f qmod option, 97, 106
- fault tolerance, 106
- file system, requirements for checkpointing, 110
- fixed resource attributes, 104
- format, messages file, 141
- functional policy, 25, 59, 82, 176

G

- global grid, 20, 176
- gmake, 79
 - j, 81
- grid, 176
 - campus, 20, 175
 - classes of, 20
 - cluster, 20, 175
 - defined, 19-22
 - global, 20, 176
- group, 176

H

- hard requests, 70
- hard resource requirements, 176
- hold, user, 61
- hold_jid qsub option, 71
- holding back jobs, 87

- HOME, 55, 58
- home directory path, 58
- host, 176
 - administration, 175
 - execution, 176
 - master, 177
 - submit, 178
- HOSTNAME, 56, 58
- hosts, 41-42
 - administration, 27
 - execution, 27
 - listing administration hosts, 42
 - listing execution hosts, 42
 - listing submit hosts, 42
 - master, 27, 41
 - submit, 27
 - types of, 27

I

- I, info message, 142
- index, of array jobs, 71
- info message, 142
- inherit qmake option, 80
- inherit qrsh option, 77, 80
- instances, *See* queue instances
- integration
 - of C programs, 78
 - of Java programs, 78
- interactive job, 175, 176
- interactive jobs
 - handling, 73
 - submitting, 73-76
- interactive qmake, 81
- interface, command line, 29

J

- j gmake option, 81
- j qmake option, 81
- Java program integration, 78
- job, 176
 - batch, 175

job (Continued)

- cancelling with `qdel`, 97
- class, 176
- monitoring with `qmod`, 97
- parallel, 177

job array, 175

job class, 176

`JOB_ID`, 55, 58

`JOB_NAME`, 56, 58

job slots, 103

jobs

- advanced job example, 65-66
- and queues, 24
- array, 55, 71-72
- array index, 71
- array tasks, 71
- at, 106
- crontab, 106
- dependencies, 61, 71
- dispatching with named queue requests, 83
- error state, 90
- extended job example, 62
- handling interactive jobs, 73
- holding back, 87
- modifying pending, 87
- notifying, 61
- pending, 95
- priority, 59, 82
- releasing, 87
- rescheduling, 88, 100
- spooling, 83
- submitting advanced jobs, 63-66, 66-68
- submitting batch jobs, 53-59
- submitting extended jobs, 59-63, 63
- submitting from the command line, 48-49
- submitting interactive jobs, 73-76
- submitting with `QMON`, 49
- subtasks, 71

K

kernel-level checkpointing, 106

- scripts for, 107

ksh shell, 53

L

- l `qstat` option, 96
- limits, per queue slot, 104
- listing
 - administration hosts, 42
 - execution hosts, 42
 - managers, 38
 - operators, 38
 - owners, 38
 - queue properties, 38-40
 - queues, 38
 - requestable attributes, 44-46
 - submit hosts, 42
 - users, 38
- load adjustments, 104
- load balancing, dynamic, 107
- load management, 22
- load parameters, 104
- log file, messages, 141
- .login file, 54
- login shell, 54
- `login_shells`, 54
- `LOGNAME`, 58

M

- M `qsub` option, 98
- m a `qsub` option, 90
- m `qsub` option, 98
- `MAIL`, 57
- Main Control window, 33-35
- make, 79
- makefile, parallel processing, 79-82
- manager, 177
- managers, 35
 - listing, 38
- master daemon, 27, 28
- master host, 27, 41, 177
- memory, requirements for checkpointing, 109
- messages, log file, 141
- messages file, format, 141
- migrating checkpointing jobs, 107
- migration, 177
- modifying pending jobs, 87

N

N, notice message, 142
 newgrp, 37
 NFS Network File System, 110
 NHOSTS, 58
 -noshell qrsh option, 77
 -nostdin qrsh option, 77
 notice message, 142
 -now no qlogin option, 73
 -now no qrsh option, 73
 -now no qsh option, 73
 -now no qsub option, 73
 -now qrsh option, 77
 NQUEUES, 58
 NSLOTS, 58

O

operator, 177
 operators, 35
 listing, 38
 -ot qalter option, 95
 output redirection, 55-56
 override policy, 25, 82, 177
 owner, 177
 owners, 35
 listing, 38
 queues, 38

P

-p qsub option, 25, 82
 parallel environment, 177
 parallel job, 177
 parallel jobs, 58
 parallel makefile processing, 79-82
 PATH, 57, 58
 path, default shell search, 58
 PE, 58
 PE_HOSTFILE, 58
 -pe qmake option, 81
 pending jobs, 95
 checking consistency, 89

permissions, user access, 36-37
 policies, 21, 82-84
 administering, 22
 administering ticket policies, 25
 functional, 25, 59, 82
 override, 25, 82
 share-based, 25, 82
 ticket-based, 83
 types of, 24
 urgency, 25, 82
 policy, 177
 functional, 176
 override, 177
 share-based, 177
 preferences
 QMON, 34, 92, 105
 priority, 82, 177
 job, 59, 82
 range, 82
 process hierarchies, checkpointing, 106
 project, 177
 project_lists, 37
 projects, 24
 access permissions, 36
 job submission, 60

Q

qacct, 29
 qalter, 29, 87
 consistency checking, 89
 -ot, 95
 qconf, 29
 -sc, 44
 -se, 42
 -sel, 42
 -sh, 42
 -sm, 38
 -so, 38
 -sq, 38, 40
 -sql, 38
 -ss, 42
 -su, 37
 -sul, 36

qdel, 29
 cancelling jobs with, 97
 -f, 97
qhold, 29
qhost, 29, 42
qlogin, 29, 73, 75-76
 -now no, 73
qmake, 29, 79
 batch usage, 81
 -inherit, 80
 interactive usage, 81
 -j, 81
 -pe, 81
qmod, 29
 -d, 105
 disabling queues with, 105
 -e, 105
 -f, 97, 106
 monitoring jobs with, 97
 -s, 97, 105
 suspending queues with, 105
 -us, 97, 105
 with crontab or at, 106
qmon, 29
QMON
 and embedded script arguments, 56
 customizing, 30, 34-35, 92, 105
 Main Control window, 33-35
 preferences, 34, 92, 105
Qmon file, 34
.qmon_preferences file, 34, 92, 105
QMON resource file, 34
qresub, 29
qr!s, 29
qrsh, 29, 73, 76-77
 -b, 67, 77
 -inherit, 77, 80
 -noshell, 77
 -nostdin, 77
 -now, 77
 -now no, 73
 -verbose, 77
qrshmode, 79
qselect, 30
qsh, 30, 73, 75
 -now no, 73
qstat, 27, 30
 -ext, 83
 -l, 96
 -r, 96
 resource requirements, 96
qsub, 27, 30
 arguments in scripts, 56
 -C, 56
 -c, 108
 -clear, 67
 -cwd for checkpointing jobs, 109
 -hold_jid, 71
 -M, 98
 -m, 98
 -m a, 90
 -now no, 73
 -p, 25, 82
 -t, 72
.qtask file, 77, 78
qtcsh, 30, 77-79
 -c, 78
QUEUE, 58
queue, 175
queue instances, 28, 101-103
 configuring, 100
 disabling, 102
 enabling, 102
 resuming, 102
 suspending, 102
queue_sort_method, 84
queues, 28
 and jobs, 24
 batch, 83
 configuring, 100
 consumable resources, 103, 104
 disabling, 100, 102
 disabling with qmod, 105
 displaying, 38-41
 enabling, 100, 102
 error state, 100
 listing, 38
 listing properties, 38-40

queues (Continued)

- load parameters, 104
- owners, 35, 38
- resource attributes, 103, 104
- resuming, 100, 102
- selection of, 83-84
- sequence number, 84
- shell parameter, 55
- slot limits, 104
- suspending, 100, 102
- suspending with `qmod`, 105
- system load, 103

R

- r `qstat`, option, 96
- relational operator, 45
- releasing jobs, 87
- remsh, 76
- REQUEST, 58
- requestable attributes, 42-46, 68
 - listing, 44-46
- requestable flag, 45
- requests
 - hard, 70
 - soft, 70
- requirements
 - hard, 176
 - soft, 178
- rescheduling jobs, 88, 100
- resource, 177
- resource attributes, 42-46, 103
 - attached to queue, 104
 - consumable flag, 45
 - fixed, 104
 - relational operator, 45
 - requestable flag, 45
- resource capacity, 104
- resource requirements, 68-71
 - hard, 176
 - soft, 178
 - with `qstat`, 96
- resources
 - allocation algorithm, 70-71

resources (Continued)

- managing, 22
- restart mechanism, 106
- RESTARTED, 58
- restarting checkpointed jobs, 58
- resuming
 - cluster queues, 100
 - queue instances, 102
- rlogin, 73, 76
- rsh, 73, 76-77

S

- s `qmod` option, 97, 105
- sc `qconf` option, 44
- `schedd_job_info`, 90
- scheduler daemon, 27, 28
- scheduling
 - policies, 82-84
 - tickets, 83
- script embedding, 56
- se `qconf` option, 42
- sel `qconf` option, 42
- `seq_no`, 84
- sequence number, 84
- SGE_BINARY_PATH, 57
- SGE_CELL, 57
- SGE_CKPT_DIR, 58
- SGE_CKPT_ENV, 58
- `sge_execd`, 27, 28
- SGE_JOB_SPOOL_DIR, 57
- SGE_O_HOME, 57
- SGE_O_HOST, 57
- SGE_O_LOGNAME, 57
- SGE_O_MAIL, 57
- SGE_O_PATH, 57
- SGE_O_SHEL, 57
- SGE_O_TZ, 57
- SGE_O_WORKDIR, 57
- `sge_qmaster`, 27, 28
- `.sge_request` file, 67
- `sge_request` file, 67
- SGE_ROOT, 57
- `sge_schedd`, 27

- SGE_STDERR_PATH, 58
- SGE_STDOUT_PATH, 58
- SGE_TASK_ID, 58, 71
- sh qconf option, 42
- sh shell, 53
- share, 177
- share-based policy, 25, 82, 177
- share-tree, 178
- SHELL, 57, 58
- shell queue parameter, 55
- shell scripts, 53-54
 - example, 54
 - grid engine system extensions, 55-59
- shell_start_mode, 55
- slots, 103
- sm qconf option, 38
- so qconf option, 38
- soft requests, 70
- soft resource requirements, 178
- spooling jobs, 83
- sq qconf option, 38, 40
- sql qconf option, 38
- ss qconf option, 42
- standard error, 54, 55
- standard output, 54, 55
- stderr redirection
 - redirection
 - stderr, 141
- su qconf option, 37
- submit host, 178
- submit hosts, 27
 - listing, 42
- submitting
 - advanced jobs, 63-66, 66-68
 - batch jobs, 53-59
 - extended jobs, 59-63, 63
 - interactive jobs, 73-76
 - jobs from the command line, 48-49
 - jobs with QMON, 49
- subtasks, job, 71
- sul qconf option, 36
- suspending
 - cluster queues, 100
 - queue instances, 102

- suspending (*Continued*)
 - queues with qmod, 105
- suspension, 178
- system load, 103

T

- t qsub option, 72
- TASK_ID, 56
- tasks, 71
- tcsh shell, 77
- tcsh shell, 53
- telnet, 73, 76
- ticket, 178
- tickets, 25, 83
- time zone, 59
- TMP, 59
- TMPDIR, 59
- trace output, debug mode, 145
- TZ, 59
 - time zone, 57

U

- unix_behavior, 55
- urgency policy, 25, 82
- us qmod option, 97, 105
- usage, 178
- usage policies, 82-84
- USER, 55, 59
- user, 178
- user access
 - permissions, 36-37
 - projects, 36
- user groups, 24
- user hold, 61
- user-level checkpointing, 106
 - scripts for, 107
- user_lists, 37
- users, 35-38
 - categories of, 35
 - listing, 38
- userset, 178

V

variables, environment, 57-59
-verbose qrsh option, 77
Verify flag, 89
verifying job consistency, 89

W

W, warning messages, 142
warning messages, 142

X

XAPPLRESDIR, 34
.Xdefaults file, 34
.xinitrc file, 34
xproject_lists, 37
xrdb, 34
.Xresources file, 34
xterm, 73
xuser_lists, 37

