CHAPTER 1

Introduction to Sun Grid Engine 5.3

This chapter provides background information about the Sun Grid Engine 5.3 system that is useful to users and administrators alike. In addition to a description of the product's role in managing what could otherwise be a chaotic world of clustered computers, this chapter includes the following topics.

- A brief description of grid computing
- An overview of `QMON`, the Sun Grid Engine 5.3 graphical user interface
- An explanation of each of the important components of the product
- A detailed list of client commands that are available to users and administrators
- A complete glossary of Sun Grid Engine 5.3 terminology

What Is Grid Computing?

Conceptually, a grid is quite simple. It is a collection of computing resources that perform tasks. It appears to users as a large system, providing a single point of access to powerful distributed resources. Users treat the grid as a single computational resource. Resource management software, such as Sun Grid Engine, accepts jobs submitted by users and schedules them for execution on appropriate systems in the grid based upon resource management policies. Users can literally submit thousands of jobs at a time without being concerned about where they run.

No two grids are alike; one size does not fit all situations. There are three key classes of grids, which scale from single systems to supercomputer-class compute farms that utilize thousands of processors. *Cluster grids*, consisting of a many computational resources working together to provide a single point of access to users in a single project or department, are what the Sun Grid Engine 5.3 system helps you to create and manage.

(Two other types of more complex grids—*campus grids* and g*lobal grids*—are created and managed by a related product from Sun, Sun Grid Engine, Enterprise Edition.)

Sun Grid Engine 5.3 software orchestrates the delivery of computational power based upon enterprise resource *policies* set by the organization's technical and management staff. The Sun Grid Engine system uses these policies to examine the available computational resources within the grid, gathers these resources, and then allocates and delivers them automatically in a way that optimizes usage across the grid.

Managing Workload by Managing Resources and Policies

Sun Grid Engine software provides the user with the means to submit computationally demanding tasks to the Sun Grid Engine system for transparent distribution of the associated workload. The user can submit batch jobs, interactive jobs, and parallel jobs to the Sun Grid Engine system.

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

For the administrator, the software provides comprehensive tools for monitoring and controlling Sun Grid Engine jobs.

How the System Operates

The Sun Grid Engine system accepts jobs—users' requests for computer resources—from the outside world, puts them in a holding area until they can be executed, sends them from the holding area to an execution device, manages them during execution, and logs the record of their execution when they are finished.

As an analogy, imagine a large "money-center" bank in one of the world's capitol cities.

Matching Resources to Requests

In the bank building's lobby are dozens upon dozens of customers, each with different requirements, who are waiting to be served. One customer merely 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 is seeking advice before undertaking a complicated venture. In front of both of them in the long line is another customer who intends to apply for a large loan—as do the eight customers in front of *her*.

Different customers and different intentions require different types and levels of the bank's resources. Perhaps, on this particular day, the bank has many employees who have sufficient time available to handle the one customer's simple withdrawal of money from his account. But on that day, only one or two loan officers are on hand to help the many loan applicants. On another day, the situation may be reversed.

The effect, of course, is that customers must wait for service—even though many of them could probably receive immediate service if only their requirements were immediately discerned and matched to available resources.

If the Sun Grid Engine system were the bank manager, it would organize the service differently.

- Upon entering the bank lobby, customers would be asked to declare their name, their affiliations (such as representing a company), and their requirements.
- The customers' time of arrival would be recorded.
- Based on the information that the customers provided in the lobby, those whose requirements match suitable and immediately available resources, those whose requirements have the highest priority, and those who have been waiting in the lobby for the longest time would be served.
- Of course, in a "Sun Grid Engine bank," one bank employee may be able to provide assistance to several customers at the same time. The Sun Grid Engine system would try to assign new customers to the least loaded and most suitable bank employee.

Jobs and Queues: The Sun Grid Engine World

In a Sun Grid Engine system, *jobs* correspond to bank customers, jobs wait in a computer holding area instead of a lobby, and *queues* located on computer servers take the place of bank employees, providing services for jobs. As in the case of bank customers in the analogy, the requirements of each of the jobs—which typically consist of available memory, execution speed, available software licenses, and similar needs—may be very different and only certain queues may be able to provide the corresponding service.

Corresponding to the analogy, Sun Grid Engine software arbitrates available resources and job requirements in the following fashion.

- A user who submits a job through the Sun Grid Engine system declares a requirement profile for the job. In addition, the identity of the user and his or her affiliation with *projects* or *user groups* is retrieved by the system. The time that the user submitted the job is also stored.
- The moment, literally, that a queue becomes available for execution of a new job, the Sun Grid Engine system determines suitable jobs for the queue and immediately dispatches the job with the highest priority or longest waiting time.
- Sun Grid Engine queues may allow concurrent execution of many jobs. The Sun Grid Engine system will try to start new jobs in the least loaded and suitable queue.

Sun Grid Engine 5.3 Components

Figure FIGURE 1-1 displays the most important Sun Grid Engine components and their interaction in the system. The following sections explain the functions of the components.

Hosts

Four types of hosts are fundamental to the Sun Grid Engine 5.3 system.

- Master
- Execution
- Administration
- Submit

Master Host

The master host is central for the overall cluster activity. It runs the master daemon, `sge_qmaster`, and the scheduler daemon, `sge_schedd`. Both daemons control all Sun Grid Engine components, such as queues and jobs, and maintain tables about the stttus of the components, about user access permissions, and the like.

By default, the master host is also an administration host and submit host. See the sections relating to those hosts.

Execution Host

Execution hosts are nodes that have permission to execute Sun Grid Engine jobs. Therefore, they are hosting Sun Grid Engine queues and run the Sun Grid Engine execution daemon, `sge_execd`.

Administration Host

Permission can be given to hosts to carry out any kind of administrative activity for the Sun Grid Engine system.

Submit Host

Submit hosts allow for submitting and controlling *batch jobs only*. In particular, a user who is logged into a submit host can submit jobs via `qsub`, can control the job status via `qstat`, and can use the Sun Grid Engine OSF/1 Motif graphical user's interface, `QMON`, which is described in the section, “`QMON`, the Sun Grid Engine Graphical User Interface” on page 9.

Note – A host may belong to more than one of the above described classes.

Daemons

Four daemons provide the functionality of the Sun Grid Engine 5.3 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. It 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`. It makes the following scheduling decision:

- Which jobs are dispatched to which queues

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

`sge_execd` – the Execution Daemon

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

`sge_commd` – the Communication Daemon

The communication daemon communicates over a well-known TCP port. It is used for all communication among Sun Grid Engine components.

Queues

A Sun Grid Engine queue is a container for a class of jobs allowed to execute on a particular host concurrently. A queue determines certain job attributes; for example, whether it may be migrated. Throughout their lifetimes, running jobs are associated with their queue. Association with a queue affects some of the things that can happen to a job. For example, if a queue is suspended, all the jobs associated with that queue are also suspended.

In the Sun Grid Engine system, there is no need to submit jobs directly to a queue. You only need to specify the requirement profile of the job (e.g., memory, operating system, available software, etc.) and Sun Grid Engine software will dispatch the job to a suitable queue on a low-loaded host automatically. If a job is submitted to a particular queue, the job will be bound to this queue and to its host, and thus Sun Grid Engine daemons will be unable to select a lower-loaded or better-suited device.

Client Commands

Sun Grid Engine's command line user interface is a set of ancillary programs (commands) that enable you to manage queues, submit and delete jobs, check job status, and suspend/enable queues and jobs. The Sun Grid Engine system makes use of the following set of ancillary programs.

- `qacct` – This command extracts arbitrary accounting information from the cluster logfile.
- `qalter` – This command changes the attributes of submitted, but pending, jobs.

- `qconf` – This command provides the user interface for cluster and queue configuration.
- `qdel` – This command provides the means for a user, operator, or manager to send signals to jobs or subsets thereof.
- `qhold` – This command holds back submitted jobs from execution.
- `qhost` – This command displays status information about Sun Grid Engine execution hosts.
- `qlogin` – This command initiates a `telnet` or similar login session with automatic selection of a low-loaded and suitable host.
- `qmake` – This command is a replacement for the standard UNIX `make` facility. It extends `make` by its ability to distribute independent `make` steps across a cluster of suitable machines.
- `qmod` – This command enables the owner to suspend or enable a queue (all currently active processes associated with this queue are also signaled).
- `qmon` – This command provides an X-windows Motif command interface and monitoring facility.
- `qresub` – This command creates new jobs by copying running or pending jobs.
- `qrls` – This command releases jobs from holds previously assigned to them; e.g., via `qhold` (see above).
- `qrsh` – This command can be used for various purposes, such as the following.
  - To provide remote execution of interactive applications via the Sun Grid Engine system—comparable to the standard UNIX facility, `rsh`
  - To allow for the submission of batch jobs which, upon execution, support terminal I/O (standard/error output and standard input) and terminal control
  - To provide a batch job submission client which remains active until the job has been finished
  - To allow for the Sun Grid Engine software-controlled remote execution of the tasks of parallel jobs
- `qselect` – This command prints a list of queue names corresponding to specified selection criteria. The output of `qselect` is usually fed into other Sun Grid Engine commands to apply actions on a selected set of queues.
- `qsh` – This command opens an interactive shell (in an `xterm`) on a low-loaded host. Any kind of interactive jobs can be run in this shell.
- `qstat` – This command provides a status listing of all jobs and queues associated with the cluster.
- `qsub` – This command is the user interface for submitting a job to the Sun Grid Engine system.

- `qtcsh` – This command is a fully compatible replacement for the widely known and used Unix C-Shell (`csh`) derivative, `tcsh`. It provides a command shell with the extension of transparently distributing execution of designated applications to suitable and lightly loaded hosts via Sun Grid Engine software.

All programs communicate with `sge_qmaster` via `sge_commd`. This is reflected in the schematic view of the component interaction in the Sun Grid Engine system, depicted in FIGURE 1-1.

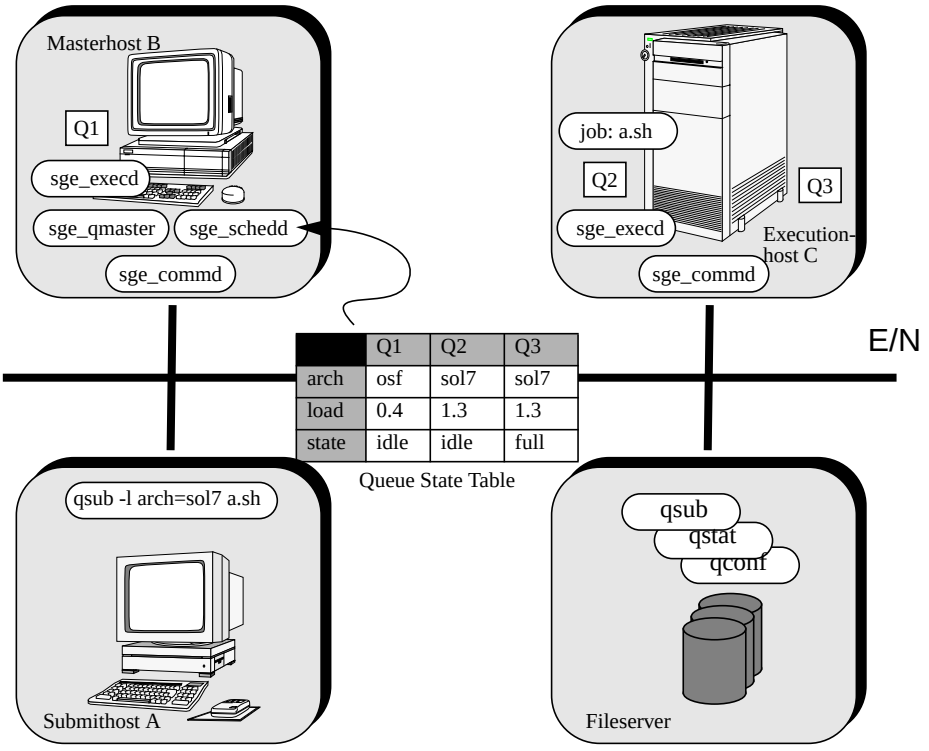


FIGURE 1-1 Component Interaction in the Sun Grid Engine System

QMON, the Sun Grid Engine Graphical User Interface

Using QMON, the graphical user interface (GUI) tool, you can accomplish most—if not all—Sun Grid Engine 5.3 tasks. FIGURE 1-2 shows the QMON Main menu, which is often the starting point for both user and administrator functions. Each icon on the Main menu is a GUI button that you press to initiate a variety of tasks. The name of each button, which appears as text on the screen when you pass the mouse pointer over it, is also descriptive of its function.

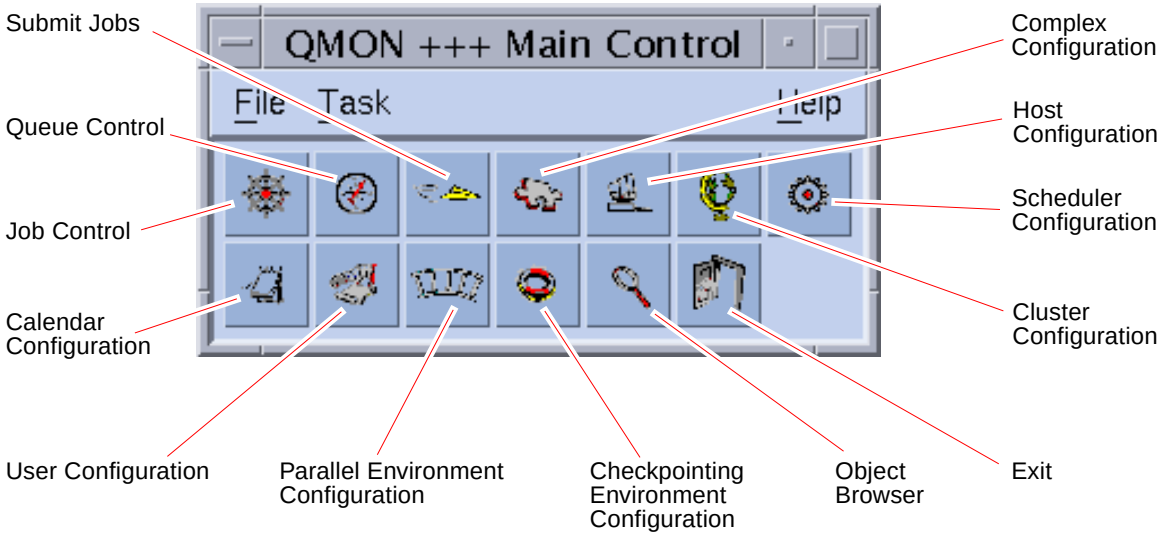


FIGURE 1-2 QMON Main Menu, Defined

Customizing QMON

The look and feel of qmon is largely defined by a specifically designed resource file. Reasonable defaults are compiled in and a sample resource file is available under <sge_*root*>/qmon/Qmon.

The cluster administration may install site specific defaults in standard locations such as `/usr/lib/X11/app-defaults/Qmon`, by including `qmon` specific resource definitions into the standard `.Xdefaults` or `.Xresources` files or by putting a site specific `Qmon` file to a location referenced by standard search paths such as `XAPPLRESDIR`. Ask your administrator if any of the above is relevant in your case,

In addition, the user can configure personal preferences by either copying and modifying the `Qmon` file into the home directory (or to another location pointed to by the private `XAPPLRESDIR` search path) or by including the necessary resource definitions into the user's private `.Xdefaults` or `.Xresources` files. A private `Qmon` resource file may also by installed via the `xrdb` command during operation or at start-up of the X11 environment, e.g. in a `.xinitrc` resource file.

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

Another means of customizing `qmon` has been explained for the Job Control and Queue Control customization dialogue boxes shown in FIGURE 5-3 and in FIGURE 5-13. In both dialogue boxes, you can use the Save button to store the filtering and display definitions configured with the customization dialogue boxes to the file, `.qmon_preferences`, in the user's home directory. Upon being restarted, `qmon` reads this file and reactivates the previously defined behavior.

Glossary of Sun Grid Engine Terms

The glossary provides a short overview on frequently used terms in the context of Sun Grid Engine and resource management in general. Many of the terms have not been used so far, but will appear in other parts of the Sun Grid Engine documentation.

access list A list of users and UNIX groups who are permitted, or denied, access to a resource such as a queue or a certain host. Users and groups may belong to multiple access lists and the same access lists can be used in various contexts.

Array job A job consisting of a range of independent identical tasks. Each task is very similar to a separate job. Job array tasks only differ by a unique task identifier (an integer number).

cell A separate Sun Grid Engine cluster with a separate configuration and master machine. Cells can be used to loosely couple separate administrative units.

checkpointing A procedure which saves the execution status of a job into a so called *checkpoint* thereby allowing for the job to be aborted and resumed later without loss of information and already completed work. The process is called *migration*, if the checkpoint is moved to another host before execution resumes.

checkpointing environment A Sun Grid Engine configuration entity, which defines events, interfaces and actions being associated with a certain method of checkpointing.

cluster A collection of machines, called hosts, on which Sun Grid Engine functions occur.

complex A set of attributes that can be associated with a queue, a host, or the entire cluster.

group A UNIX group.

hard resource requirements The resources which must be allocated before a job may be started. Contrasts with *soft resource requirements*.

host A machine on which Sun Grid Engine functions occur.

job A batch job is a UNIX shell script that can be executed without user intervention and does not require access to a terminal.

An interactive job is a session started with the Sun Grid Engine commands `qsh` or `qlogin` that will open an *xterm* window for user interaction or provide the equivalent of a remote login session, respectively.

job class A set of jobs that are equivalent in some sense and treated similarly. In Sun Grid Engine a job class is defined by the identical requirements of the corresponding jobs and the characteristics of the queues being suitable for those jobs.

manager A user who can manipulate all aspects of Sun Grid Engine. The superusers of the master host and of any other machine being declared as an administrative host have manager privileges. Manager privileges can be assigned to non-root user accounts as well.

migration The process of moving a checkpoint 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 but rather are supposed to maintain operation.

owner Users who may suspend/unsuspend and disable/enable the queues they own. Typically users are owners of the queues that reside on their workstations.

parallel environment A Sun Grid Engine configuration entity, which defines the necessary interfaces for Sun Grid Engine to correctly handle parallel jobs.

parallel job A job which consists of more than one closely correlated task. Tasks may 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 which the Sun Grid Engine administrator can use define the behavior of Sun Grid Engine. Policies will be implemented automatically by Sun Grid Engine.

priority The relative level of importance of a Sun Grid Engine job compared to others.

queue A container for a certain class and number of jobs being allowed to execute on a Sun Grid Engine execution host concurrently.

resource A computational device consumed or occupied by running jobs. Typical examples are memory, CPU, I/O bandwidth, file space, software licenses, etc.

soft resource requirements Resources which a job needs but which do not have to be allocated before a job may be started. Allocated to a job on an as available basis. Contrast with *hard resource requirements*.

suspension The process of holding a running job but keeping it on the execution machine (in contrast to checkpointing, where the job is aborted). A suspended job still consumes some resources, such as swap memory or file space.

user May submit jobs to and execute jobs with Sun Grid Engine if he or she has a valid login on at least one submit host and an execution host.

userset An access list (see above).

CHAPTER 3

Navigating Through the Sun Grid Engine 5.3 Program

This chapter introduces you to some basic Sun Grid Engine 5.3 concepts and terminology that will help you begin to use the software. For complete background information about the product, including a comprehensive glossary, see Chapter 1, “Introduction to Sun Grid Engine 5.3” on page 1.

This chapter also includes instructions for accomplishing the following tasks.

Sun Grid Engine User Types and Operations

User types are divided into four categories in Sun Grid Engine.

- **Managers** – Managers have full capabilities to manipulate Sun Grid Engine. By default, the superusers of any machine hosting a queue have manager privileges.
- **Operators** – The operators can perform many of the same commands as the manager, with the exception of making configuration changes by adding, deleting, or modifying queues, for example.

- **Owners** – The queue owners are allowed to suspend or enable the owned queues or jobs within them, but have no further management permissions.
- **Users** – Users have certain access permissions, as described in "User Access Permissions" on page 66, but no cluster or queue management capabilities.

TABLE 3-1 shows the Sun Grid Engine 5.3 command capabilities that are available to the different user categories.

TABLE 3-1 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
`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

Queues and Queue Properties

In order to be able to optimally utilize the Sun Grid Engine system at your site, you should become familiar with the queue structure and the properties of the queues that are configured for your Sun Grid Engine system.

The QMON Browser

Sun Grid Engine features a graphical user interface (GUI) command tool, the QMON browser. The QMON browser provides a myriad of Sun Grid Engine functions, including job submission, job control, and important information gathering.

▼ How To Launch the QMON Browser

- **From the command line, enter the following command.**

```
% qmon
```

After a message window is displayed, the QMON main control panel appears, similar to the following (see FIGURE 1-2 to identify the meaning of the icons).

FIGURE 3-1 QMON Main Control Menu

Many instructions in this manual call for using the QMON browser. The names of the icon buttons, which are descriptive of their functions, appear on screen as you pass the mouse pointer over them.

(For instructions on how to customize the QMON browser, see “Customizing QMON” on page 9.)

The Queue Control QMON Dialogue Box

The QMON Queue Control dialogue box displayed and described in the section, “How To Control Queues with QMON” on page 132 provides a quick overview on the installed queues and their current status.

▼ How To Display a List of Queues

- **Enter the following command.**

```
% qconf -sql
```

▼ How To Display Queue Properties

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

Using the QMON Browser

1. **From the main QMON menu, click the Browser icon.**
2. **Click the Queue button.**
3. **In the Queue Control dialog, move the mouse pointer over the icon for the appropriate queue.**

   FIGURE 3-2 is a partial example of the Queue property information that is displayed.

   FIGURE 3-2 QMON Browser Display of Queue Properties

From the Command Line

- **Enter the following command.**

```
% qconf -sq queue_name
```

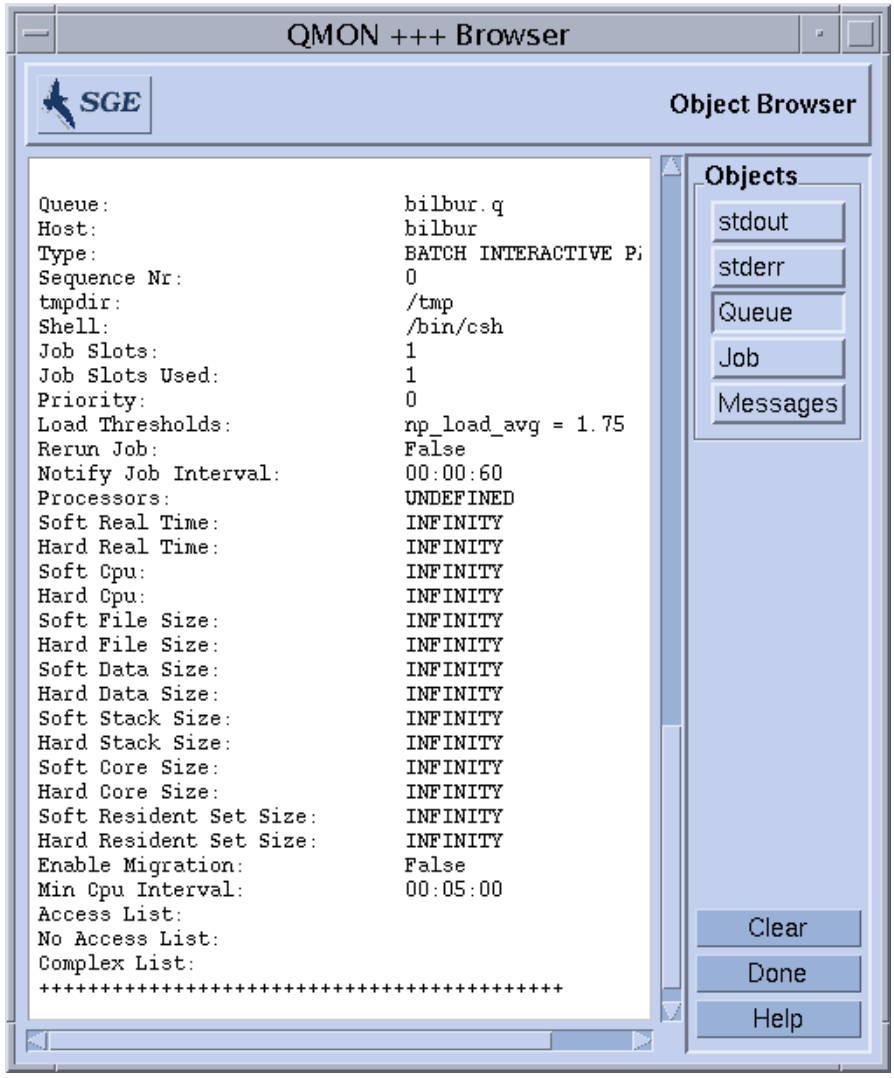
Information similar to that shown in FIGURE 3-2 is displayed.

Interpreting Queue Property Information

You can find a detailed description of each queue property in the queue_conf manual page and in the queue_conf section of the *Sun Grid Engine 5.3 and Sun Grid Engine, Enterprise Edition 5.3 Reference Manual.*

Following is a list of some of the most important parameters.

- qname – The queue name as requested.



- hostname – The host of the queue.
- processors – The processors of a multi processor system, to which the queue has access.
- qtype – The type of job which is allowed to run in this queue. Currently, this is either batch, interactive, checkpointing, parallel or any combination thereof or transfer alternatively
- slots – The number of jobs which may be executed concurrently in that queue.
- owner_list – The owners of the queue as explained in the section, “Managers, Operators and Owners” on page 67

- user_lists – The user or group identifiers in the user access lists (see “User Access Permissions” on page 66) enlisted under this parameter may access the queue.
- xuser_lists – The user or group identifiers in the user access lists (see “User Access Permissions” on page 66) enlisted under this parameter may *not* access the queue.
- complex_list – The complexes enlisted under this parameter are associated with the queue and the attributes contained in these complexes contribute to the set of requestable attributes for the queue (see “Requestable Attributes” on page 62).
- complex_values – Assigns capacities as provided for this queue for certain complex attributes (see “Requestable Attributes” on page 62).

Host Functionality

Clicking the Host Configuration button in the `QMON` Main menu displays an overview of the functionality that is associated with the hosts in your Sun Grid Engine cluster. However, without Sun Grid Engine manager privileges, you may not apply any changes to the presented configuration.

The host configuration dialogues are described in the section, “About Daemons and Hosts” on page 143. The following sections provide the commands to retrieve this kind of information from the command line.

▼ How To Find the Name of the Master Host

The location of the master host should be transparent for the user as the master host may migrate between the current master host and one of the shadow master hosts at any time.

- **Using a text editor, open the** *<sge_root>/<cell>*`/common/act_qmaster` **file.**

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

▼ How To Display a List of Execution Hosts

To display a list of hosts being configured as execution hosts in your cluster please use the commands:

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

The first command displays a list of the names of all hosts being currently configured as execution hosts. The second command displays detailed information about the specified execution host. The third command displays status and load information about the execution hosts. Please refer to the `host_conf` manual page for details on the information displayed via `qconf` and to the `qhost` manual page for details on its output and further options.

▼ How To Display a List of Administration Hosts

The list of hosts with administrative permission can be displayed with the following command:.

```
% qconf -sh
```

▼ How To Display a List of Submit Hosts

The list of submit host can be displayed with the following command.

```
% qconf -ss
```

Requestable Attributes

When submitting a Sun Grid Engine job a requirement profile of the job can be specified. The user can specify attributes or characteristics of a host or queue which the job requires to run successfully. Sun Grid Engine will map these job requirements onto the host and queue configurations of the Sun Grid Engine cluster and will, therefore, find the suitable hosts for a job.

The attributes that can be used to specify the job requirements are either related to the Sun Grid Engine cluster (e.g., space required on a network shared disk), to the hosts (e.g., operating system architecture), or to the queues (e.g., permitted CPU time), or the attributes are derived from site policies such as the availability of installed software only on some hosts.

The available attributes include the queue property list (see “Queues and Queue Properties” on page 56), the list of global and host-related attributes (see “Complex Types” on page 186), as well as administrator-defined attributes. For convenience, however, the Sun Grid Engine administrator commonly chooses to define only a subset of all available attributes to be requestable.

The attributes being currently requestable are displayed in the Requested Resources sub-dialogue (see FIGURE 3-3) to the QMON Submit dialogue box (refer to the section, “Submitting Batch Jobs” on page 75 for detailed information on how to submit jobs). They are enlisted in the Available Resources selection list.

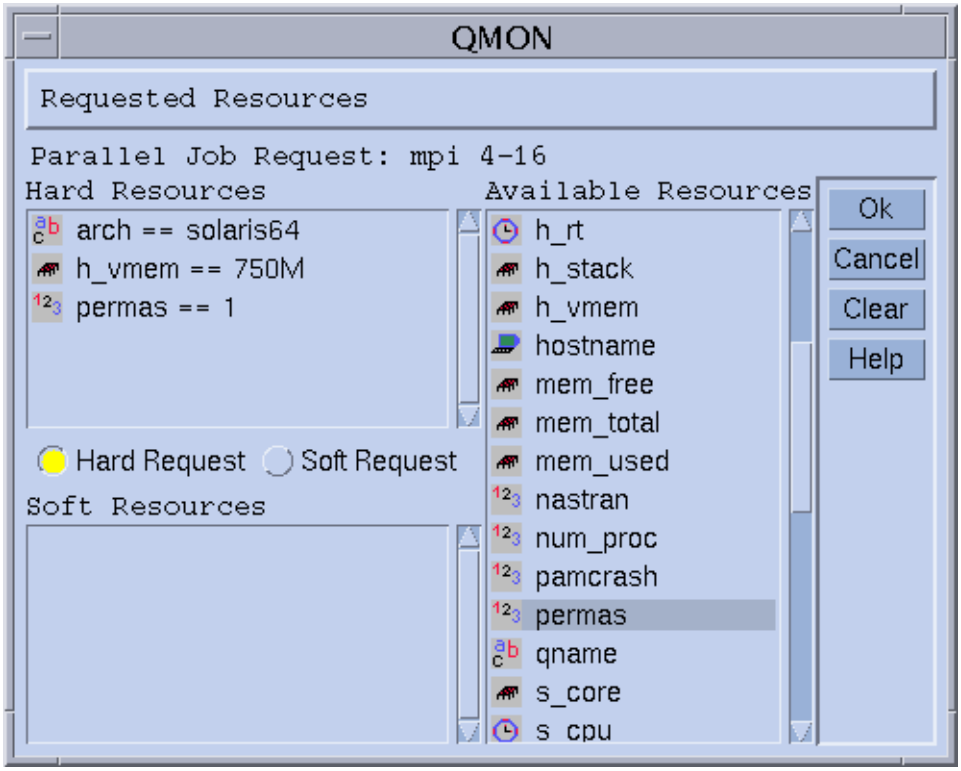


FIGURE 3-3 Requested Resources Dialogue Box

▼ How To Display a List of Requestable Attributes

1. **From the command line, display a list of configured *complexes* by entering the following command:**

```
% qconf -scl
```

A complex contains the definition for a set of attributes. There are three standard complexes:

- `global`– For the cluster global attributes (optional)
- `host` – For the host-specific attributes
- `queue` – For the queue property attributes

Any further complex names printed as a result of the above command refers to an administrator-defined complex (see Chapter 8, “The Complexes Concept” on page 183 or the complex format description in the *Sun Grid Engine 5.3 and Sun Grid Engine, Enterprise Edition 5.3 Reference Manual* for more information on complexes).

2. **Display the attributes of a particular complex by entering the following command:**

```
% qconf -sc complex_name[,...]
```

The output for the queue complex might for example look as shown in TABLE 3-2.

TABLE 3-2 `queue` Complex Attributes Displayed

#Name	Shortcut	Type	Value	Relop	Requestable	Consumable	Default
qname	q	STRING	NONE	==	YES	NO	NONE
hostname	h	HOST	unknown	==	YES	NO	NONE
tmpdir	tmp	STRING	NONE	==	NO	NO	NONE
calendar	c	STRING	NONE	==	YES	NO	NONE
priority	pr	INT	0	>=	NO	NO	0
seq_no	seq	INT	0	==	NO	NO	0
rerun	re	INT	0	==	NO	NO	0
s_rt	s_rt	TIME	0:0:0	<=	NO	NO	0:0:0
h_rt	h_rt	TIME	0:0:0	<=	YES	NO	0:0:0
s_cpu	s_cpu	TIME	0:0:0	<=	NO	NO	0:0:0
h_cpu	h_cpu	TIME	0:0:0	<=	YES	NO	0:0:0
s_data	s_data	MEMORY	0	<=	NO	NO	0
h_data	h_data	MEMORY	0	<=	YES	NO	0
s_stack	s_stack	MEMORY	0	<=	NO	NO	0
h_stack	h_stack	MEMORY	0	<=	NO	NO	0
s_core	s_core	MEMORY	0	<=	NO	NO	0
h_core	h_core	MEMORY	0	<=	NO	NO	0
s_rss	s_rss	MEMORY	0	<=	NO	NO	0
h_rss	h_rss	MEMORY	0	<=	YES	NO	0
min_cpu_interval	mci	TIME	0:0:0	<=	NO	NO	0:0:0
max_migr_time	mmt	TIME	0:0:0	<=	NO	NO	0:0:0

TABLE 3-2 queue Complex Attributes Displayed *(Continued)*

#Name	Shortcut	Type	Value	Relop	Requestable	Consumable	Default
qname	q	STRING	NONE	==	YES	NO	NONE
hostname	h	HOST	unknown	==	YES	NO	NONE
tmpdir	tmp	STRING	NONE	==	NO	NO	NONE
calendar	c	STRING	NONE	==	YES	NO	NONE
priority	pr	INT	0	>=	NO	NO	0
seq_no	seq	INT	0	==	NO	NO	0
max_no_migr	mnm	TIME	0:0:0	<=	NO	NO	0:0:0

The column `name` is basically identical to the first column displayed by the `qconf -sq` command. The queue attributes cover most of the Sun Grid Engine queue properties. The `shortcut` column contains administrator definable abbreviations for the full names in the first column. Either the full name or the shortcut can be supplied in the request option of a `qsub` command by the user.

The column `requestable` tells whether the Corresponding entry may be used in `qsub` or not. Thus the administrator can, for example, disallow the cluster's users to request certain machines / queues for their jobs directly, simply by setting the entries `qname` and / or `qhostname` to be not requestable. Doing this, implies that feasible user requests can be met in general by multiple queues, which enforces the load balancing capabilities of Sun Grid Engine.

The column `relop` defines the relation operation used in order to compute whether a queue meets a user request or not. The comparison executed is:

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

If the result of the comparison is false, the user's job cannot be run in the considered queue. Let, as an example, the queue *q1* be configured with a soft cpu time limit (see the `queue_conf` and the `setrlimit` manual pages for a description of user process limits) of 100 seconds while the queue *q2* is configured to provide 1000 seconds soft cpu time limit.

The columns `consumables` and `default` are meaningful for the administrator to declare so called consumable resources (see the section, "Consumable Resources" on page 194). The user requests consumables just like any other attribute. The Sun Grid Engine 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 (see the `qsub` manual page for details on the syntax) asks for a queue which at least grants for 5 minutes of soft limit cpu time. Therefore, only queues providing at least 5 minutes soft CPU runtime limit are setup properly to run the job.

Note – Sun Grid Engine will only consider workload information in the scheduling process if more than one queue is able to run a job.

User Access Permissions

Access to queues and other Sun Grid Engine facilities (e.g., parallel environment interfaces; see “Parallel Jobs” on page 101) can be restricted for certain users or user groups by the Sun Grid Engine administrator.

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

For the purpose of restricting access permissions, the administrator creates and maintains so called access lists (or in short *ACLs*). The ACLs contain arbitrary user 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 (see `queue_conf` or `sge_pe` in *Sun Grid Engine 5.3 and Sun Grid Engine, Enterprise Edition 5.3 Reference Manual* section 5, respectively).

User‘s belonging to ACLs which are enlisted in access-allowed-lists have permission to access the queue or the parallel environment interface. User‘s being members of ACLs in access-denied-lists may not access the concerning resource.

The Userset Configuration dialogue box opened via the User Configuration icon button in the `QMON` Main menu allows you to query for the ACLs you have access to via the Userset Configuration dialogue box. Refer to Chapter 9, “Managing User Access and Policies” on page 213 for details.

From the command line a list of the currently configured ACLs can be obtained by the command:

```
% qconf -sul
```

The entries in one or multiple access lists are printed with the command:

```
% qconf -su acl_name[,...]
```

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

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

You can now check for those queues or parallel environment interfaces to which you have access or to which access is denied for you. Please query the queue or parallel environment interface configuration as described in "Queues and Queue Properties" on page 56 and "How To Configure PEs with QMON" on page 246. The access-allowed-lists are named `user_lists`. The access-denied-list have the names `xuser_lists`. If your user account or primary UNIX group is associated with a access-allowed-list you are allowed to access the concerning resource. If you are associated with a access-denied-list you may not access the queue or parallel environment interface. If both lists are empty every user with a valid account can access the concerning resource.

Managers, Operators and Owners

A list of Sun Grid Engine managers can be obtained by:

```
% qconf -sm
```

and a list of operators by:

```
% qconf -so
```

Note – The superuser of a Sun Grid Engine administration host is considered as manager by default.

The users, which are owners to a certain queue are contained in the queue configuration database as described in section “Queues and Queue Properties” on page 56. This database can be retrieved by executing:

```
% qconf -sq queue_name
```

The concerning queue configuration entry is called `owners`.

CHAPTER 4

Submitting Jobs

This chapter provides background information about, and instructions for, using Sun Grid Engine 5.3 to submit jobs for processing. The chapter begins with an example of running a simple job, and then continues with instructions for running more complex jobs.

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

Running a Simple Job

Use the information and instructions in this section to become familiar with basic procedures involved in submitting Sun Grid Engine 5.3 jobs.

Note – If you have installed the Sun Grid Engine program under an unprivileged account, you must log in as that particular user to be able to run jobs (see "Prerequisite Tasks" on page 24 for details).

▼ How To Run a Simple Job from the Command Line

Prior to executing any Sun Grid Engine command, you must first set your executable search path and other environmental conditions properly.

1. **Enter either of the the following commands, depending on your command interpreter.**

   a. If you are using either `csh` or `tcsh` as your command interpreter:

   ```
   % source sge_root_dir/default/common/settings.csh
   ```

   sge_root_dir specifies the location of the Sun Grid Engine root directory that was selected at the beginning of the installation procedure.

   b. If you are using `sh`, `ksh`, or `bash` as your command interpreter:

   ```
   # . sge_root_dir/default/common/settings.sh
   ```

   Note – You can add the above commands into your `.login`, `.cshrc`, or `.profile` files (whichever is appropriate) to guarantee proper Sun Grid Engine settings for all interactive session you will start later.

   ---

2. **Submit the following simple job script to your Sun Grid Engine cluster.**

   You can find the following job in the file, `examples/jobs/simple.sh` in your Sun Grid Engine root directory.

   ```
   #!/bin/sh
   #This is a simple example of a Sun Grid Engine batch script
   #
   # Print date and time
   date
   # Sleep for 20 seconds
   sleep 20
   # Print date and time again
   date
   # End of script file
   ```

Enter the following command, which assumes that `simple.sh` is the name of the script file in which the above script is stored, and the file is located in your current working directory.

```
% qsub simple.sh
```

The `qsub` command should confirm the successful job submission as follows.

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

3. **Enter the following command to retrieve status information on your job.**

```
% qstat
```

You should receive a status report containing information about all jobs currently known to the Sun Grid Engine system and for each of them the so called *job ID* (the unique number being included in the submit confirmation), the name of the job script, the owner of the job, a state information (`r` means running), the submit or start time and eventually the name of the queue in which the job executes.

If no output is produced by the `qstat` command, no jobs are actually known to the system. For example, your job may 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 which has executed the job. The names of the files are composed of the job script file name, an appended dot sign followed by an "`o`" for the stdout file and an "`e`" for the stderr file and finally the unique job ID. Thus the `stdout` and `stderr` files of your job can be found under the names `simple.sh.o1` and `simple.sh.e1` respectively, if that job was the first ever executed in a newly installed Sun Grid Engine system.

▼ How To Submit Jobs From the Graphical User Interface, QMON

A more convenient method of submitting and controlling Sun Grid Engine jobs and of getting an overview of the Sun Grid Engine system is the graphical user interface, `QMON`. Among other facilities, `QMON` provides a job submission menu and a Job Control dialogue box for the tasks of submitting and monitoring jobs.

From the command line prompt, type the following command.

```
% qmon
```

During startup, a message window is displayed and then the QMON Main menu appears.

4. **Click the Job Control button and then the Submit Jobs button.**

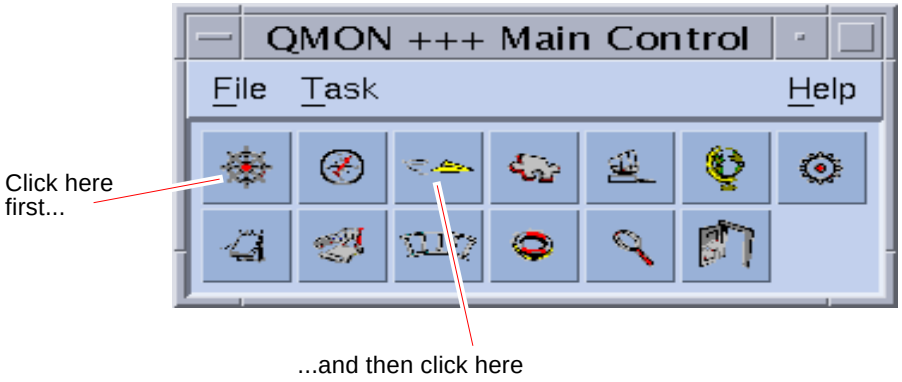


FIGURE 4-1 QMON Main Menu

The Job Submission and the Job Control dialogue boxes appear (see FIGURE 4-2 and FIGURE 4-3 respectively). The button names (such as Job Control) are displayed when you move the mouse pointer over the buttons.

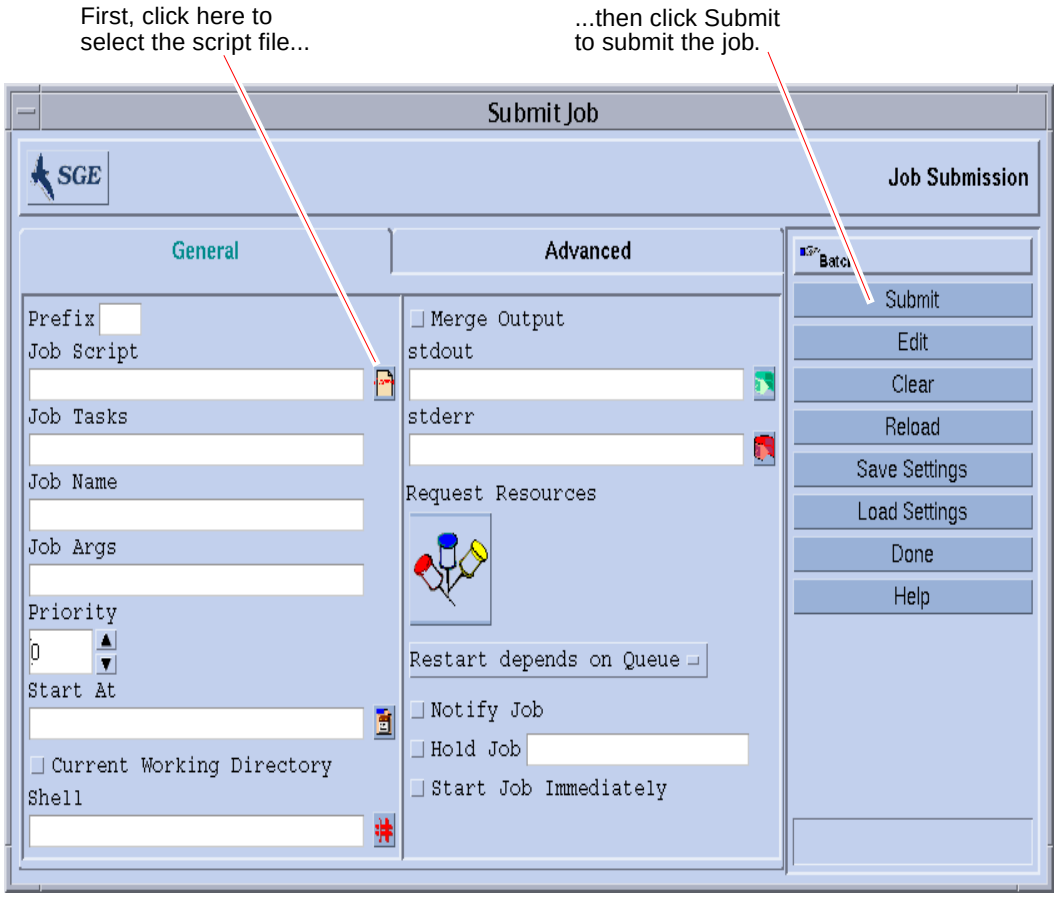


FIGURE 4-2 QMON Job Submission Dialogue Box

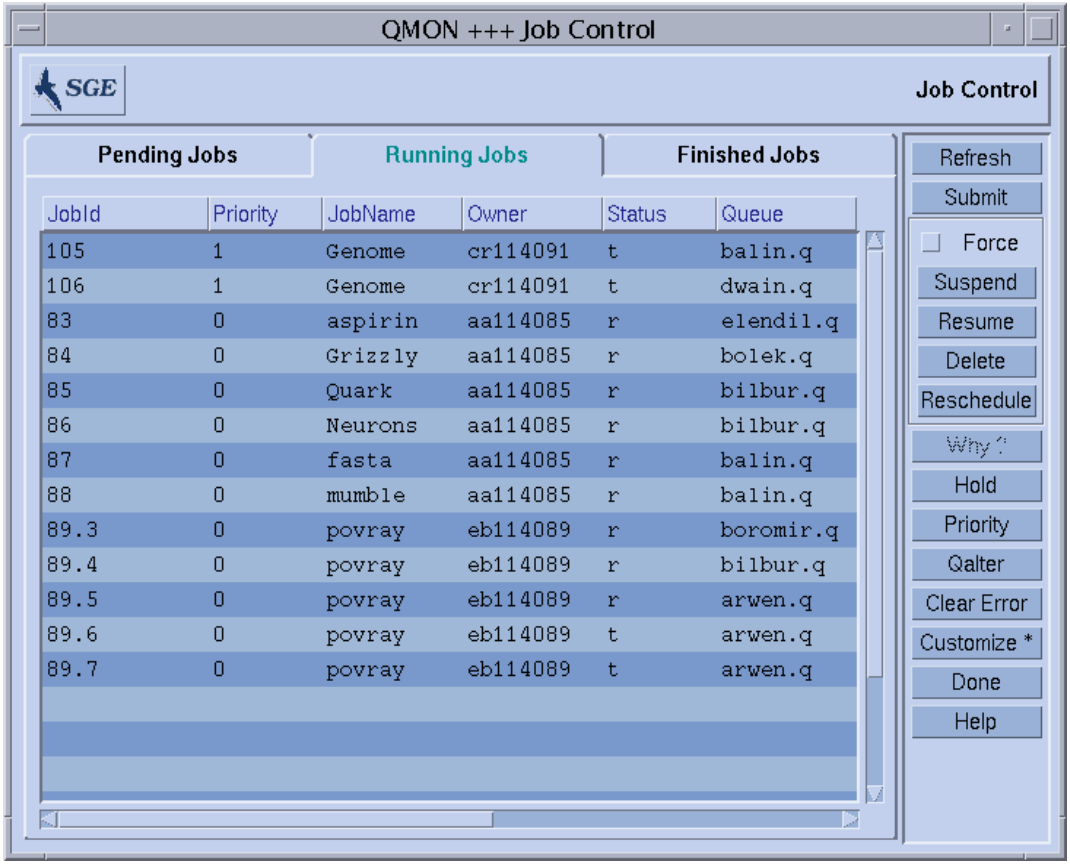


FIGURE 4-3 QMON Job Control Dialogue Box

5. **In the Job Submission menu, click the Job Script file selection icon to open a file selection box.**

6. **Click the appropriate file name to select your script file (e.g., the file *simple.sh* from the command line example).**

7. **Click the Submit button at the bottom of the Job Submission menu.**

   After a couple of seconds, you should be able to monitor your job in the Job Control panel. You will first see it under Pending Jobs, and it will quickly move to Running Jobs once it gets started.

Submitting Batch Jobs

The following sections describe how to submit more complex jobs through the Sun Grid Engine 5.3 program.

About Shell Scripts

Shell scripts, also called batch jobs, are in principal a sequence of command-line instructions assembled in a file. Script files are made executable by the `chmod` command. If scripts are invoked, a proper command interpreter is started (e.g., `csh, tcsh, sh`, or `ksh`) and each instruction is interpreted as typed in manually by the user executing the script. You can invoke arbitrary commands, applications, and other shell scripts from within a shell script.

The appropriate command interpreter is either invoked as `login-shell` or not, depending whether its name (`csh, tcsh, sh, ksh,`...) is contained in the value list of the `login_shells` entry of the Sun Grid Engine configuration in effect for the particular host and queue executing the job.

Note – The Sun Grid Engine configuration may be different for the various hosts and queues configured in your cluster. You can display the effective configurations via the `-sconf` and `-sq` options of the `qconf` command (refer to the *Sun Grid Engine 5.3 and Sun Grid Engine, Enterprise Edition 5.3 Reference Manual* for detailed information).

If the command interpreter is invoked as `login-shell`, the environment of your job will be exactly the same as if you just have logged in and executed the script. In using `csh`, for example, `.login` and `.cshrc` will be executed in addition to the system default startup resource files (e.g., something like `/etc/login`) while only `.cshrc` will be executed if `csh` is not invoked as `login-shell`. Refer to the manual page of the command interpreter of your choice for a description of the difference between being invoked as `login-shell` or not.

Example of a Script File

CODE EXAMPLE 4-1 is an example of a simple shell script, which first compiles the application, `flow`, from its Fortran77 source and then executes it:.

```
#!/bin/csh

# This is a sample script file for compiling and
# running a sample FORTRAN program under Sun Grid Engine.

cd TEST

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

f77 flow.f -o flow
```

CODE EXAMPLE 4-1 Simple Shell Script

Your local system user's guide will provide detailed information about building and customizing shell scripts (you might also want to look at the `sh`, `ksh`, `csh` or `tcsh` manual page). In the following sections, the emphasis is on specialities that are to be considered in order to prepare batch scripts for Sun Grid Engine.

In general, you can submit to Sun Grid Engine all shell scripts that you can execute from your command prompt by hand, as long as they do not require a terminal connection (except for the standard error and output devices, which are automatically redirected) and as long as they do not need interactive user intervention. Therefore, CODE EXAMPLE 4-1 is ready to be submitted to Sun Grid Engine and will perform the desired action.

Submitting Extended and Advanced Jobs with QMON

Before attempting a more complex form of job submission—*extended* or *advanced*—it is useful to understand some important background information about the process. The following sections provide that information.

Extended Example

The standard form of the Job Submission dialogue box (see FIGURE 4-2) provides the means to configure the following parameters for an extended job:

- A prefix string which is used for script-embedded Sun Grid Engine submit options (see the section, “Active Sun Grid Engine Comments” on page 90 for detailed information)
- The job script to be used

  If the associated file button is pushed, a file selection box is opened (see FIGURE 4-3)
- The task ID range for submitting array jobs (see “Array Jobs” on page 95)
- The name of the job (a default is set after a job script is selected)
- Arguments to the job script
- The job’s initial priority value

  Users without manager or operator permission may only lower their initial priority value.
- The time at which the job is to be considered eligible for execution

  If the associated file button is pushed, a helper dialogue box becomes available for entering the correctly formatted time is opened (see FIGURE 4-4)
- A flag indicating whether the job is to be executed in the current working directory (for identical directory hierarchies between the submit and the potential execution hosts only)
- The command interpreter to be used to execute the job script (see “How a Command Interpreter Is Selected” on page 89)

  If the associated button is pushed, a helper dialogue box becomes available for entering the command interpreter specifications of the job is opened (see FIGURE 4-5).
- A flag indicating whether the job’s standard output and standard error output are to be merged together into the standard output stream
- The standard output redirection to be used (see “Output Redirection” on page 89)

  A default is used if nothing is specified. If the associated file button is pushed, a helper dialogue box becomes available for entering the output redirection alternatives (“Output Redirection” on page 89).
- The standard error output redirection to be used—very similar to the standard output redirection
- The resource requirements of the job

  To define resource needs for your job, press the corresponding icon button. If resources have been requested for a job, the icon button changes its color.

- A selection list button defining whether the job can be restarted after being aborted by a system crash or similar events and whether the restart behavior depends on the queue or is demanded by the job
- A flag indicating whether the job is to be notified by `SIGUSR1` or `SIGUSR2` signals respectively if it is about to be suspended or cancelled
- 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 it (see the section, “Monitoring and Controlling Sun Grid Engine Jobs” on page 117 for more information concerning holds). The input field attached to the Hold flag allows restricting the hold to only a specific range of task of an array job (see “Array Jobs” on page 95).
- A flag forcing the job to be either started immediately if possible or being rejected

  Jobs are not queued if this flag is selected.

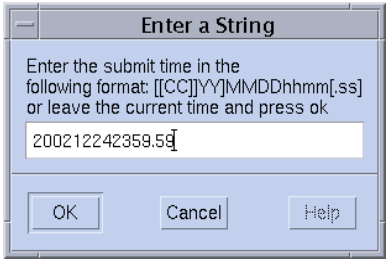


FIGURE 4-4 At Time Input Box

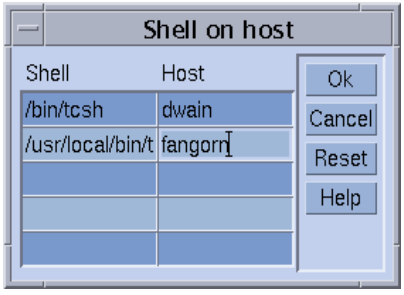


FIGURE 4-5 Shell Selection Box

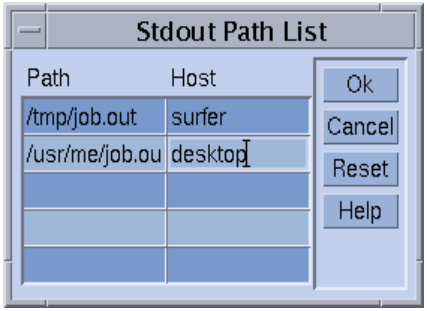


FIGURE 4-6 Output Redirection Box

The buttons at the right side of the Job Submission screen enable you to initiate various actions:

- **Submit** – Submit the job as specified in the dialogue box.
- **Edit** – Edit the selected script file in an X-terminal, either using `vi` or the editor as defined in the `$EDITOR` environment variable.
- **Clear** – Clear all settings in the Job Submission dialogue box, including any specified resource requests.
- **Reload** – Reload the specified script file, parse any script-embedded options (see the section, “Active Sun Grid Engine Comments” on page 90), parse default settings (see the section, “Default Requests” on page 94) and discard intermediate manual changes to these settings. This action is the equivalent to a Clear action with subsequent specifications of the previous script file. The option will only show an effect if a script file is already selected.
- **Save Settings** – Save the current settings to a file. A file selection box is opened to select the file. The saved files may either explicitly be loaded later (see below) or may be used as default requests (see the section, “Default Requests” on page 94).
- **Load Settings** – Load settings previously saved with the Save Settings button (see above). The loaded settings overwrite the current settings.
- **Done** – Closes the Job Submission dialogue box.
- **Help** – Display dialogue box-specific help.

FIGURE 4-7 shows the Job Submssion dialogue box with most of the parameters set.

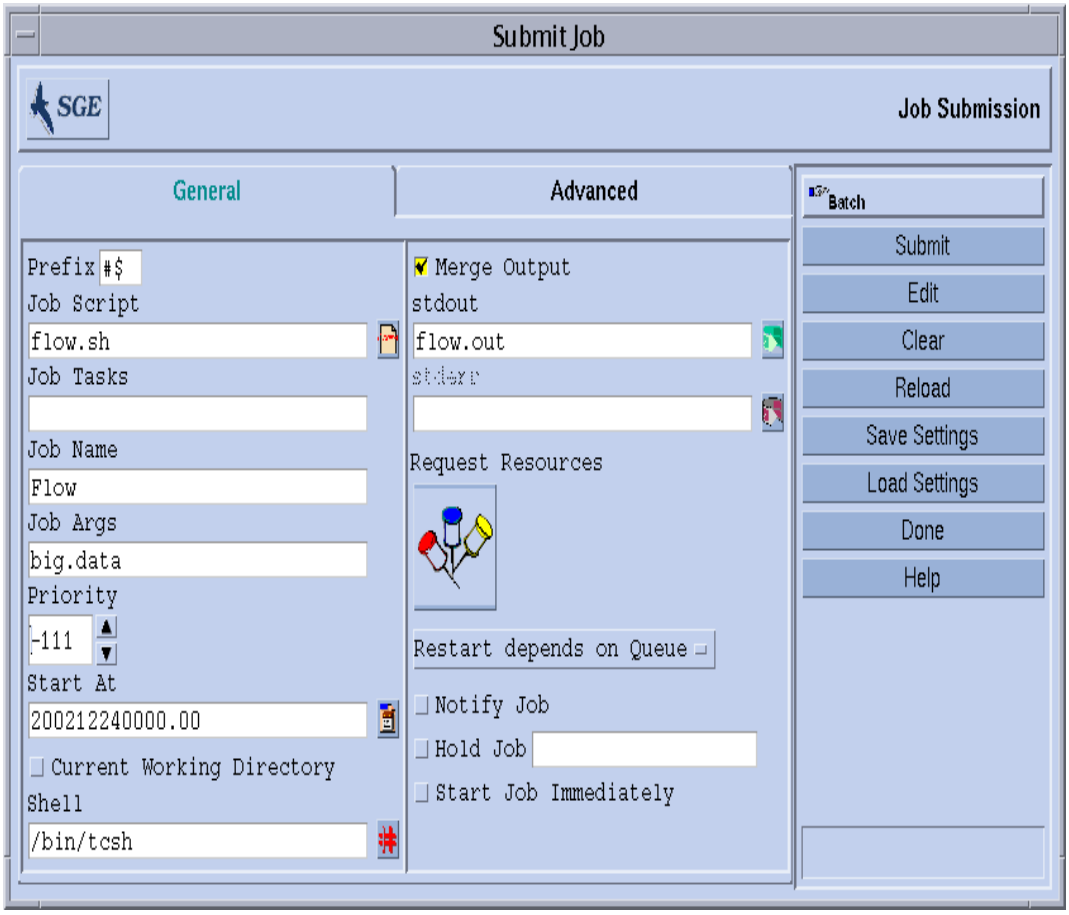


FIGURE 4-7 Extended Job Submission Example

The job configured in the example has the script file, `flow.sh`, which has to reside in the working directory of `QMON`. The job is called `Flow` and the script file takes the single argument, `big.data`. The job will be started with priority `-111` and is eligible for execution not before midnight of the 24th of December in the year 2002. The job will be executed in the submission working directory and will use the `tcsh` command interpreter. Finally, standard output and standard error output will be merged into the file, `flow.out`, which will be created in the current working directory also.

Advanced Example

The Advanced submission screen allows definition of the following additional parameters:

- A parallel environment interface to be used and the range of processes which is required (see the section, “Parallel Jobs” on page 101)
- A set of environment variables which are to be set for the job before it is executed

  If the associated icon button is pushed, a helper dialogue box becomes available for the definition of the environment variables to be exported (see FIGURE 4-8). Environment variables can be taken from `QMON`‘s runtime environment or arbitrary environment variable can be defined.
- A list of name/value pairs called Context (see FIGURE 4-9), which can be used to store and communicate job related information accessible anywhere from within a Sun Grid Engine cluster

  Context variables can be modified from the command line via the `-ac/-dc/-sc` options to `qsub,qrsh, qsh, qlogin`, or `qalter` and can be retrieved via `qstat -j`.
- The checkpointing environment to be used in case of a job for which checkpointing is desirable and suitable (see the section, “About Checkpointing Jobs” on page 111)
- An account string to be associated with the job

  The account string will be added to the accounting record kept for the job and can be used for later accounting analysis.
- The Verify flag, which determines the consistency checking mode for your job

  To check for consistency of the job request, Sun Grid Engine assumes an empty and unloaded cluster and tries to find at least one queue in which the job could run. Possible checking modes are:

  - **Skip** - No consistency checking at all.
  - **Warning** - Inconsistencies are reported, but the job is still accepted (may be desired if the cluster configuration is supposed to change after submission of the job).
  - **Error** - Inconsistencies are reported and the job will be rejected if any are encountered.
  - **Just verify** - The job will not be submitted, but an extensive report is generated about the suitability of the job for each host and queue in the cluster.
- The events about which the user is notified via electronic mail

  The events start/end/abortion/suspension of job are currently defined.
- A list of electronic mail addresses to which these notification mails are sent

If the associated button is pushed, a helper dialogue becomes available to define the mailing list (see FIGURE 4-10).

- A list of queue names which 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 as described in the bulleted list item, “The resource requirements of the job” on page 77.

- A list of queue names which 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.*

- An ID-list of jobs which need to be finished successfully before the job to be submitted can be started

  The newly created job *depends* on successful completion of those jobs.

FIGURE 4-8 Job Environment Definition

FIGURE 4-9 Job Context Definition

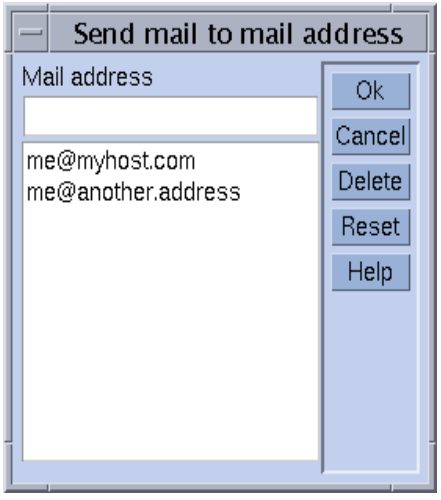


FIGURE 4-10 Mail Address Specification

The job defined in FIGURE 4-11 has the following additional characteristics as compared to the job definition from the section, “Extended Example” on page 77.

- The job requires the use of the parallel environment `mpi`. It needs at least 4 parallel processes to be created and can utilize up to 16 processes if 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.
- The job is to be restarted if it fails in case of a system crash.
- Warnings should be printed if inconsistencies between the job request and the cluster configuration are detected
- Mail has to be sent to a list of two e-mail addresses as soon as the job starts and finishes.
- Preferably, the job should be executed in the queue `big_q`.

FIGURE 4-11 shows an example of an advanced job submission.

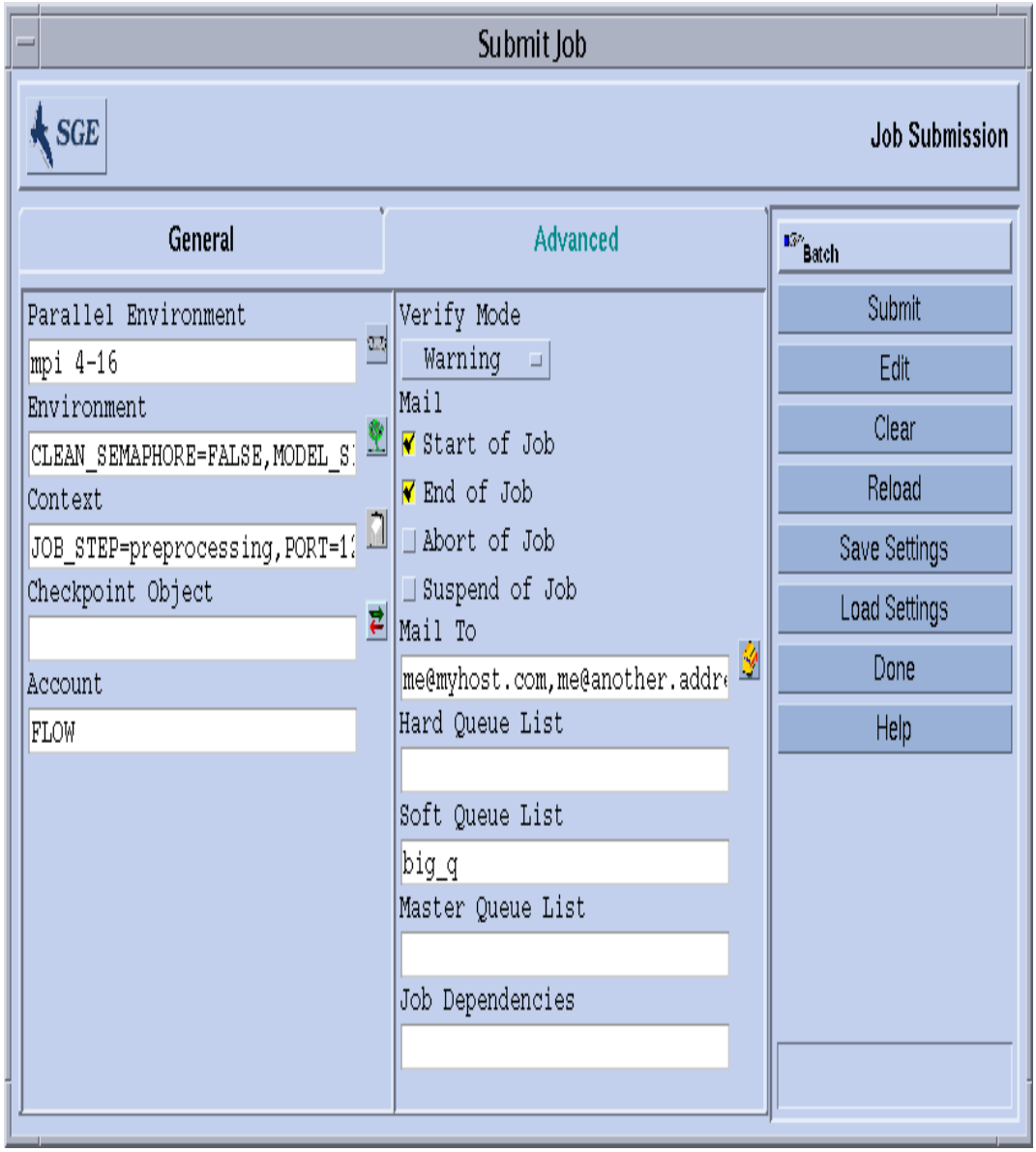


FIGURE 4-11 Advanced Job Submission Example

Resource Requirement Definition

In the examples so far the submit options used did not express any requirements for the hosts on which the jobs were to be executed. Sun Grid Engine assumes that such jobs can be run on any host. In practice, however, most jobs require certain prerequisites to be satisfied on the executing host in order to be able to complete successfully. Such prerequisites are enough available memory, required software to be installed or a certain operating system architecture. Also, the cluster administration usually imposes restrictions on the usage of the machines in the cluster. The CPU time allowed to be consumed by the jobs is often restricted, for example.

Sun Grid Engine provides the user with the means to find a suitable host for the user‘s job without a concise knowledge of the cluster‘s equipment and its utilization policies. All the user has to do is to specify the requirement of the user‘s jobs and let Sun Grid Engine manage the task of finding a suitable and lightly loaded host.

Resource requirements are specified via the *requestable attributes* explained in the section, “Requestable Attributes” on page 62. A very convenient way of specifying the requirements of a job is provided by `QMON`. The Requested Resources dialogue box, which is opened upon pressing the Requested Resources button in the Job Submission dialogue box (see FIGURE 4-12 for an example) only displays those attributes in the Available Resource selection list which currently are eligible. By double-clicking an attribute, the attribute is added to the Hard or Soft (see below) Resources list of the job and (except for `BOOLEAN` type attributes, which are just set to `True`) a helper dialogue box is opened to guide you in entering a value specification for the concerning attribute.

The example Requested Resources dialogue box displayed in FIGURE 4-12 shows a resource profile for a job in which a `solaris64` host with an available `permas` license offering at least 750 megabytes of memory is requested. If more than one queue fulfilling this specification is found, any defined soft resource requirements are taken into account (none in the example). However, if no queue satisfying both the hard and the soft requirements is found, any queue granting the hard requirements is considered to be suitable.

Note – Only if more than one queue is suitable for a job, load criteria determine where to start the job.

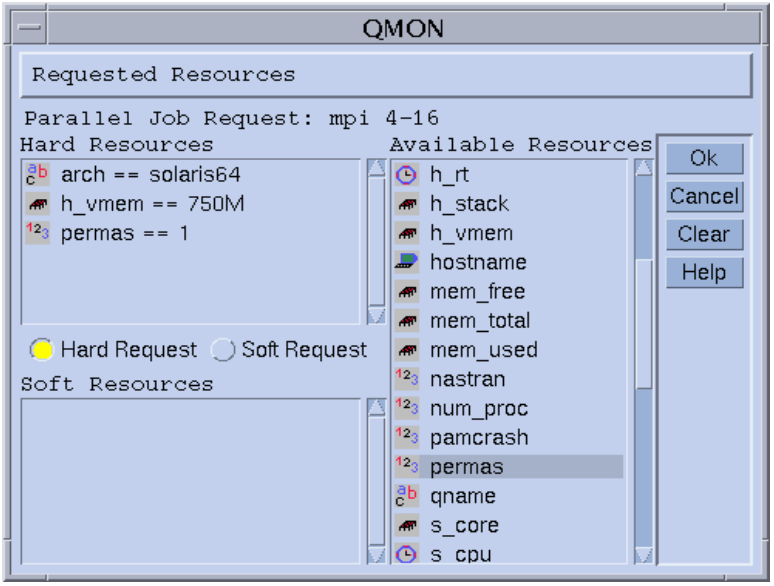


FIGURE 4-12 Requested Resources Dialogue Box

Note – The INTEGER attribute `permas` is introduced via an administrator extension to the “global” complex, the STRING attribute `arch` is imported from the “host” complex, while the MEMORY attribute `h_vmem` is imported from the “queue” complex.

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
```

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

The notation *750M* for 750 Megabytes is an example for the Sun Grid Engine quantity syntax. For those attributes requesting a memory consumption you can specify either integer decimal, floating point decimal, integer octal and integer hexadecimal numbers appended by the so called multipliers:

- 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 0 (zero) and digits ranging from 0 to 7 only. Specifying a hexadecimal constant requires to prepend the number by 0x and to use digits ranging from 0 to 9, a to f and A to F. If no multipliers are appended the values are considered to count as bytes. If using floating point decimals, the resulting value will be truncated to an integer value.

For those attributes imposing a time limit one can specify the time values in terms of hours, minutes or seconds and any combination. The hours, minutes and seconds are specified in decimal digits separated by colons. A time of `3:5:11` is translated to 11111 seconds. If a specifier for hours, minutes or seconds is 0 it can be left out if the colon remains. Thus a value of :5: is interpreted as 5 minutes. The form used in the Requested Resources dialogue box above is an extension, which is only valid within `QMON`.

How the Sun Grid Engine System Allocates Resources

As shown in the last section, it is important for you to know how Sun Grid Engine software processes resource requests and how it allocates resources. The following provides a schematic view of Sun Grid Engine software's resource allocation algorithm.

1. Read in and parse all default request files (see the section, "Default Requests" on page 94).
2. Process the script file for embedded options (see the section, "Active Sun Grid Engine Comments" on page 90).
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 first). The requests are evaluated, corresponding to the following order of precedence:

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

In other words, the command line can be used to override the embedded flags.

The resources requested as hard are allocated. If a request is not valid, the submit is rejected. If one or more requests cannot be met at submit time (e.g., a requested queue is busy) the job is spooled and will be rescheduled at a later time. If all hard requests can be met, they 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 (already meeting the hard requests) provide parts of the soft resources list (overlapping or different parts) Sun Grid Engine software will select the queues offering the most soft requests.

The job will be started and will cover the allocated resources.

It is useful to gather some experience on how argument list options and embedded options or hard and soft requests influence each other by experimenting with small test script files executing UNIX commands such as `hostname` or `date`.

Extensions to Regular Shell Scripts

There are some extensions to regular shell scripts that will influence the behavior of the script if running under Sun Grid Engine control. The following sections describe these extensions.

How a Command Interpreter Is Selected

The command interpreter to be used to process the job script file can be specified at submit time (see, for example, FIGURE 4-7). 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—if starting with a „#!“ sequence—is evaluated to determine the command interpreter. If the first line has no “`#!`“ sequence, the Bourne Shell `sh` is used by default.
- For all other settings of `shell_start_mode`, the default command interpreter as configured with the `shell` parameter for the queue in which the job is started is used (see the section, “Queues and Queue Properties” on page 56 and the `queue_conf` manual page).

Output Redirection

Since batch jobs do not have a terminal connection their standard output and their standard error output has to be redirected into files. Sun Grid Engine allows the user to define the location of the files to which the output is redirected, but uses defaults if nothing is specified.

The standard location for the files is in the current working directory where the jobs execute. The default standard output file name is <*Job_name*>.o<*Job_id*>, the default standard error output is redirected to <*Job_name*>.e<*Job_id*>. <*Job_name*> is either built from the script file name or can be defined by the user (see for example the `-N` option in the `qsub` manual page). <*Job_id*> is a unique identifier assigned to the job by Sun Grid Engine.

In case of array job tasks (see the section, "Array Jobs" on page 95), the task identifier is added to these filenames separated by a dot sign. Hence the resulting standard redirection paths are <*Job_name*>.o<*Job_id*>.<*Task_id*> and <*Job_name*>.e<*Job_id*>.<*Task_id*>.

In case the standard locations are not suitable, the user can specify output directions with `QMON` as shown in FIGURE 4-11 and FIGURE 4-6 or with the `-e` and `-o qsub` options. Standard output and standard error output can be merged into one file and the redirections can be specified on a per execution host basis. I.e., depending on the host on which the job is executed, the location of the output redirection files becomes different. To build custom but unique redirection file paths, pseudo environment variables are available which can be used together with the `qsub -e` and `-o` option. 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 `-N` option)
- `$HOSTNAME` – Name of the execution host
- `$TASK_ID` – Array job task index number

These variables are expanded during runtime of the job into the actual values and the redirection path is built with them.

See the `qsub` entry in the *Sun Grid Engine 5.3 and Sun Grid Engine, Enterprise Edition 5.3 Reference Manual* for further details.

Active Sun Grid Engine Comments

Lines with a leading "#" sign are treated as comments in shell scripts. Sun Grid Engine, however, recognizes special comment lines and uses them in a special way: the rest of such a script line will be treated as if it were part of the command line argument list of the Sun Grid Engine submit command `qsub`. The `qsub` options supplied within these special comment lines are also interpreted by the `QMON` Job Submission dialogue box and the corresponding parameters are preset when a script file is selected.

The special comment lines per default are identified by the "#$" prefix string. The prefix string can be redefined with the `qsub -C` option.

The described mechanism is called script embedding of submit arguments. The following is an example of a script file that makes use of script-embedded command line options.

```
#!/bin/csh
#Force csh if not Sun Grid Engine default shell
#$ -S /bin/csh
# This is a sample script file for compiling and
# running a sample FORTRAN program under Sun Grid Engine.
# We want Sun 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
# 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
```

CODE EXAMPLE 4-2 Using Script-Embedded Command Line Options

Environment Variables

When a Sun Grid Engine job is run, a number of variables are preset into the job's environment, as listed below.

- ARC – The Sun Grid Engine architecture name of the node on which the job is running; the name is compiled-in into the sge_execd binary
- COMMD_PORT – Specifies the TCP port on which sge_commd(8) is expected to listen for communication requests
- SGE_ROOT – The Sun Grid Engine root directory as set for sge_execd before start-up or the default /usr/SGE
- SGE_CELL – The Sun Grid Engine cell in which the job executes
- SGE_JOB_SPOOL_DIR – The directory used by sge_shepherd(8) to store job-related data during job execution

- SGE_O_HOME – The home directory path 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_CKPT_ENV – Specifies the checkpointing environment (as selected with the qsub -ckpt option) under which a checkpointing job executes
- SGE_CKPT_DIR – Only set for checkpointing jobs; contains path ckpt_dir (see the checkpoint manual page) of the checkpoint interface
- SGE_STDERR_PATH – The path name of the file to which the standard error stream of the job is diverted; commonly used for enhancing the output with error messages from prolog, epilog, parallel environment start/stop or checkpointing scripts
- SGE_STDOUT_PATH – The path name of the file to which the standard output stream of the job is diverted; commonly used for enhancing the output with messages from prolog, epilog, parallel environment start/stop 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 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 sge_qmaster when the job was submitted; the job ID is a decimal integer in the range to 99999
- JOB_NAME – The job name, built from the qsub *script filename*, a period, and the digits of the job ID; this default may be overwritten by qsub -N
- LOGNAME – The user's login name from the passwd file
- NHOSTS – The number of hosts in use by a parallel job
- NQUEUES – The number of queues allocated for the job (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 executes (for parallel jobs only)
- PE_HOSTFILE – The path of a file containing the definition of the virtual parallel machine assigned to a parallel job by Sun Grid Engine

  See the description of the $pe_hostfile parameter in sge_pe for details on the format of this file. The environment variable is only available for parallel jobs.
- QUEUE – The name of the queue in which the job is running
- REQUEST – The request name of the job, which is either the job script file name or is explicitly assigned to the job via the qsub -N option
- RESTARTED – Indicates, whether a checkpointing job has been restarted; if set (to value 1), the job has been interrupted at least once and is thus restarted
- SHELL – The user's login shell from the passwd file

Note – This is not necessarily the shell in use for the job.

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

▼ How To Submit Jobs from the Command Line

- **Enter the** qsub **command, along with appropriate arguments.**

For example, the simple job using the script file name, flow.sh—as described in the section, "How To Run a Simple Job from the Command Line" on page 70—could be submitted with the command:

```
% qsub flow.sh
```

To yield the equivalent result of the extended QMON job submission, however—as it is shown in FIGURE 4-7—would look as follows:

```
% qsub -N Flow -p -111 -a 200012240000.00 -cwd \
-S /bin/tcsh -o flow.out -j y flow.sh big.data
```

Further command line options can be added to constitute more complex requests. The advanced job request shown in FIGURE 4-11, for example, would look as follows:

```
% qsub -N Flow -p -111 -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 -r y -m s,e -q big_q\
     -M me@myhost.com,me@other.address \
     flow.sh big.data
```

Default Requests

The last example in the above section demonstrates that advanced job requests may become rather complex and unhandy, in particular if similar requests need to be submitted frequently. To avoid the cumbersome and error prone task of entering such command-lines, the user can either embed `qsub` options in the script files (see “Active Sun Grid Engine Comments” on page 90) or can utilize so called *default requests*.

The cluster administration may setup a default request file for all Sun Grid Engine users. The user, on the other hand, can create a private default request file located in the user‘s home directory as well as application specific default request files located in the working directories.

Default request files simply contain the `qsub` options to be applied by default to the Sun Grid Engine jobs in a single or multiple lines. The location of the cluster global default request file is <`sge_`*root*>/<*cell*>/`common/sge_request`. The private general default request file is located under `$HOME/.sge_request`, while the application specific default request files are expected under `$cwd/.sge_request`.

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

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

Note – Script embedding and the `qsub` command line has higher precedence than the default request files. Thus, script embedding overwrites default request file settings, and the `qsub` command line options my overwrite these settings again.

Note – The `qsub -clear` option can be used at any time in a default request file, in embedded script commands and in the `qsub` command line to discard any previous settings.

An example of a private default request file is presented below.

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

Unless overwritten, for all jobs of the given user the account string would be *myproject*, the jobs would execute in the current working directory, mail notification would be sent at the beginning and end of the jobs to *me@myhost.com*, the jobs are to be restarted after system crashes, the standard output and standard error output are to be merged and the `ksh` is to be used as command interpreter.

Array Jobs

Parametrized and repeated execution of the same set of operations (contained in a job script) is an ideal application for the Sun Grid Engine *array job* facility. Typical examples for such applications are found in the Digital Content Creation industries for tasks such as rendering. Computation of an animation is split into frames, in this example, and 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. Sun Grid Engine, on the other hand, 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 indices for all tasks span an index range for the entire array job which is defined during submission of the array job by a single `qsub` command.

An array job can be monitored and controlled (e.g., suspended, resumed, or cancelled) as a total or by individual task or subset of tasks, in which case the corresponding index numbers are suffixed to the job ID to reference the tasks. As tasks are executed (very much like regular jobs), they 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.

▼ How To Submit an Array Job from the Command Line

- **Enter the `qsub` command with appropriate arguments.**

The following is an example of submitting 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 and *10* is the highest index number while only every second index (the *:2* part of the specification) is used. Thus the array job consists of 5 tasks with the task indices 2, 4, 6, 8, and 10. Each task requests a hard CPU time limit of *45* minutes (the `-l` option) and will execute the job script *render.sh* once being dispatched and started by Sun Grid Engine. The tasks can use `$SGE_TASK_ID` to find out whether they are task 2, 4, 6, 8, or 10 and they can use their index number to find their input data record in the data file *data.in*.

▼ How To Submit an Array Job with `QMON`

- **Follow the instructions in "How To Submit Jobs From the Graphical User Interface, `QMON`" on page 71, additionally taking into account the following notes.**

Note – The submission of array jobs from `QMON` works virtually identically to how it was described in "How To Submit Jobs From the Graphical User Interface, `QMON`" on page 71. The only difference is that the Job Tasks input window shown in FIGURE 4-7 needs to contain the task range specification with the identical syntax as for the `qsub -t` option. Please refer to the `qsub` entry in the *Sun Grid Engine 5.3 and Sun Grid Engine, Enterprise Edition 5.3 Reference Manual* for detailed information on the array index syntax.

The sections "Monitoring and Controlling Sun Grid Engine Jobs" on page 117 and "Controlling Sun Grid Engine Jobs from the Command Line" on page 130, as well as the *Sun Grid Engine 5.3 and Sun Grid Engine, Enterprise Edition 5.3 Reference Manual* sections about `qstat`, `qhold`, `qrls`, `qmod`, and `qdel`, contain the pertinent information about monitoring and controlling Sun Grid Engine jobs in general and array jobs in particular.

Note – Array jobs offer full access to all Sun Grid Engine facilities known for regular jobs. In particular they can be parallel jobs at the same time or can have interdependencies with other jobs.

Submitting Interactive Jobs

Submitting interactive jobs instead of batch jobs is useful in situations where your job requires your direct input to influence the results of the job. This is typically the case for X-windows applications, which are interactive by definition, or for tasks in which your interpretation of immediate results is required to steer the further computation.

Three methods exist in Sun Grid Engine system to create interactive job.

- `qlogin` – This is a telnet-like session that is started on a host selected by Sun Grid Engine software.
- `qrsh` – This is the equivalent of the standard UNIX `rsh` facility. Either a command is executed remotely on a host selected by the Sun Grid Engine system, or a remote rlogin (`rlogin`) session is started on a remote host if no command was specified for execution.
- `qsh` – This is an `xterm` that is brought up from the machine executing the job with the display set corresponding to your specification or the setting of the `DISPLAY` environment variable. If the `DISPLAY` variable is not set and if no display destination was defined specifically, Sun Grid Engine directs the `xterm` to the 0.0 screen of the X server on the host from which the interactive job was submitted.

Note – To function correctly, all the facilities need proper configuration of Sun Grid Engine cluster parameters. The correct `xterm` execution paths have to be defined for qsh and interactive queues have to be available for this type of jobs. Contact your system administrator whether your cluster is prepared for interactive job execution.

The default handling of interactive jobs differs from the handling of batch jobs in that interactive jobs are not queued if they cannot be executed by the time of submission. This is to indicate immediately, that not enough appropriate resources are available to dispatch an interactive job right after it was submitted. The user is notified in such cases that the Sun Grid Engine cluster is too busy currently.

This default behavior can be changed with the `-now no` option to `qsh`, `qlogin` and `qrsh`. If this option is given, interactive jobs are queued like batch jobs. Using `-now yes`, batch jobs submitted with `qsub` also can be handled like interactive jobs and are either dispatched for execution immediately or are rejected.

Note – Interactive jobs can only be executed in queues of the type INTERACTIVE (refer to “About Configuring Queues” on page 163 for details).

The subsequent sections outline the usage of the `qlogin` and `qsh` facilities. The `qrsh` command is explained in a broader context in the section, “Transparent Remote Execution” on page 103.

Submitting Interactive Jobs with QMON

The only type of interactive jobs that can be submitted from `QMON` are those bringing up an `xterm` on a host selected by Sun Grid Engine.

▼ How To Submit Interactive Jobs with QMON

- **Click the icon on top of the button column at the right side of the Job Submission dialogue box until the Interactive icon is displayed.**

  This prepares the Job Submission dialogue box to submit interactive jobs (see FIGURE 4-13 and FIGURE 4-14).

  The meaning and the usage of the selection options in the dialogue box is the same as explained for batch jobs in the section, “Submitting Batch Jobs” on page 75. The basic difference is that several input fields are set insensitive because they do not apply for interactive jobs

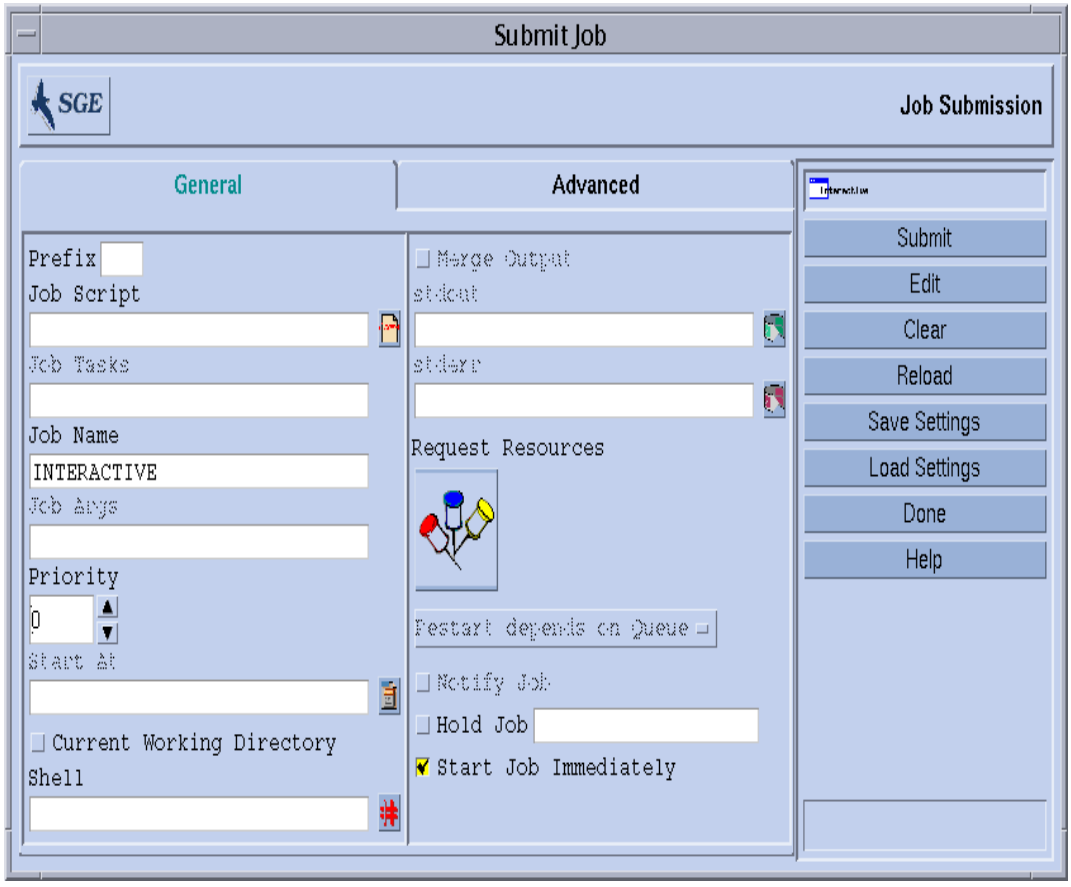


FIGURE 4-13 Interactive Job Submission Dialogue Box, General

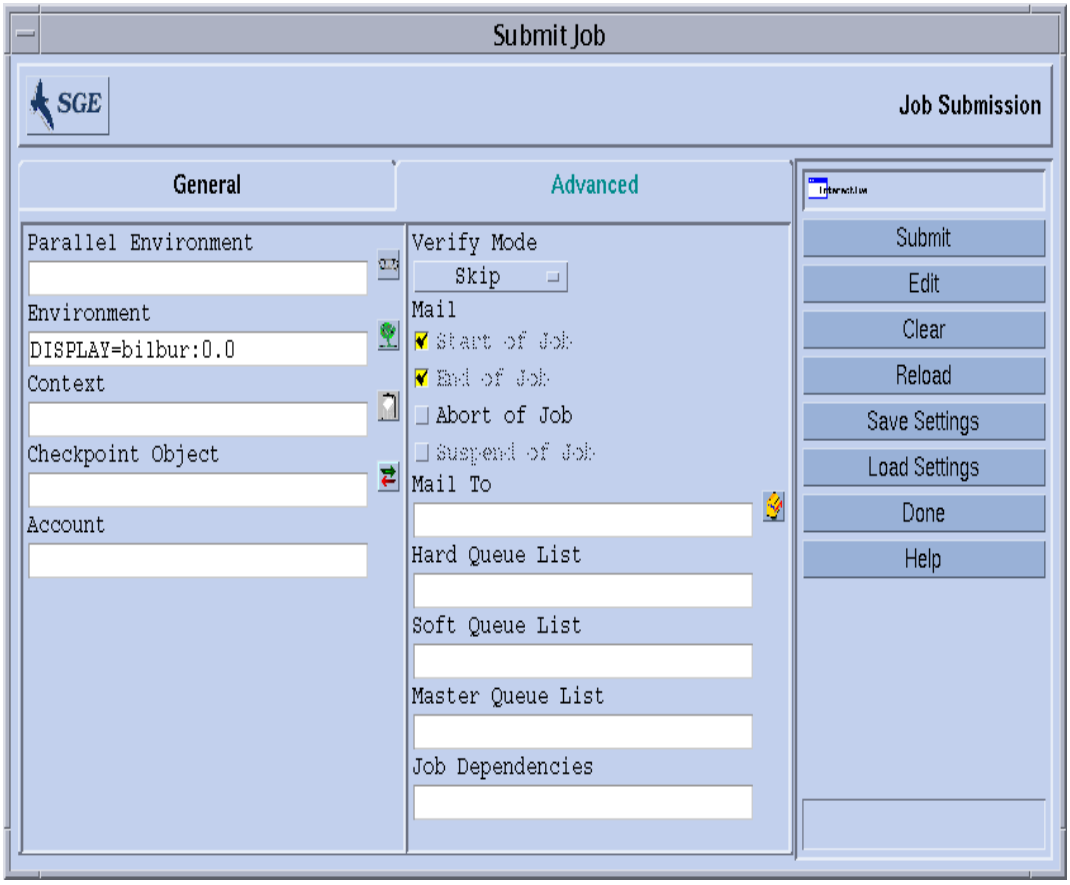


FIGURE 4-14 Interactive Job Submission Dialogue Box, Advanced

Submitting Interactive Jobs with `qsh`

`Qsh` is very similar to `qsub` and supports several of the `qsub` options, as well as the additional switch `-display` to direct the display of the `xterm` to be invoked (refer to

the qsh entry in the *Sun Grid Engine 5.3 and Sun Grid Engine, Enterprise Edition 5.3 Reference Manual* for details).

▼ How To Submit Interactive Jobs With qsh

- **Enter the following command to start an xterm on any available Sun Solaris 64bit operating system host.**

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

Submitting Interactive Jobs with qlogin

The qlogin command can be used from any terminal or terminal emulation to initiate an interactive session under the control of Sun Grid Engine.

▼ How To Submit Interactive Jobs With qlogin

- **Enter the following command to locate a low-loaded host with Star-CD license available and with at least one queue providing a minimum of 6 hours hard CPU time limit.**

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

Note – Depending on the remote login facility configured to be used by the Sun Grid Engine system, you may have to enter your user name, your password, or both at a login prompt.

Parallel Jobs

Sun Grid Engine provides means to execute parallel jobs using arbitrary message passing environments such as PVM or MPI (see the *PVM User's Guide* and the *MPI User's Guide* for details) or shared memory parallel programs on multiple slots in single queues or distributed across multiple queues and (for distributed memory

parallel jobs) across machines. An arbitrary number of different *parallel environment* (PE) interfaces may be configured concurrently at the same time. See Chapter 10, “Managing Parallel Environments” on page 245 for details about PEs.

How Sun Grid Engine Jobs Are Scheduled

Essentially, Sun Grid Engine 5.3 software uses two set of criteria to schedule jobs:

- Job Priorities
- Equal-share

Job Priorities

Concerning the order of scheduling precedence of different jobs, a *first-in-first-out* (fifo) rule is applied by default. All *pending* (not yet scheduled) jobs are inserted in a list, with the first submitted job being the head of the list, followed by the second submitted job, and so on. The job submitted first will be attempted to be scheduled first. If at least one suitable queue is available, the job will be scheduled. Sun Grid Engine software will try to schedule the second job afterwards no matter whether the first has been dispatched or not.

This order of precedence among the pending jobs may be overruled by the cluster administration via a *priority value* being assigned to the jobs. The actual priority value can be displayed by using the `qstat` command (the priority value is contained in the last column of the pending jobs display entitled `P`; refer to the section, “How To Monitor Jobs with `qstat`” on page 127 for details). The default priority value assigned to the jobs at submit time is 0. The priority values are positive and negative integers and the pending jobs list is sorted Correspondingly in the order of descending priority values. By assigning a relatively high priority value to a job, the job is moved to the top of the pending jobs list. Jobs with negative priority values are inserted even after jobs just submitted. If there are several jobs with the same priority value, the fifo rule is applied within that priority value category.

Equal-Share-Scheduling

The fifo rule sometimes leads to problems, especially if user’s tend to submit a series of jobs almost at the same time (e.g., via shell-script issuing one submit after the other). All jobs being submitted afterwards and being designated to the same group of queues will have to wait a very long time. *Equal-share-scheduling* avoids this problem by sorting jobs of users already owning a running job to the end of the precedence list. The sorting is performed only among jobs within the same priority

value category. Equal-share-scheduling is activated if the Sun Grid Engine scheduler configuration entry `user_sort` (refer to the `sched_conf` manual page for details) is set to `TRUE`.

Queue Selection

The Sun Grid Engine system does not dispatch jobs requesting nonspecific queues if they cannot be started immediately. Such jobs will be marked as spooled at the `sge_qmaster`, which will try to re-schedule them from time to time. Thus, such jobs are dispatched to the next suitable queue that becomes available.

As opposed to this, jobs that are requested by name to a certain queue will go directly to this queue, regardless of whether they can be started or they have to be spooled. Therefore, viewing Sun Grid Engine queues as computer science *batch queues* is only valid for jobs requested by name. Jobs submitted with nonspecific requests use the spooling mechanism of `sge_qmaster` for queueing, thus utilizing 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 the queue (among the suitable) belonging to the least loaded host. By setting the Sun Grid Engine scheduler configuration entry `queue_sort_method` to `seq_no`, the cluster administration may change this load dependent scheme into a fixed order algorithm: the queue configuration entry `seq_no` is used to define a precedence among the queues assigning the highest priority to the queue with the lowest sequence number.

Transparent Remote Execution

Sun Grid Engine provides a set of closely related facilities supporting transparent remote execution of certain computational tasks. The core tool for this functionality is the `qrsh` command described in section “Remote Execution with `qrsh`” on page 104. Building on top of `qrsh`, two high level facilities—`qtcsh` and `qmake`—allow the transparent distribution of implicit computational tasks via Sun Grid Engine, thereby enhancing the standard UNIX facilities `make` and `csh`. `qtcsh` is explained in the section, “Transparent Job Distribution with `qtcsh`” on page 105 and `qmake` is described in the section, “Parallel Makefile Processing with `qmake`” on page 107.

Remote Execution with qrsh

Qrsh is built around the standard rsh facility (see the information provided in <sge_*root*>/3rd_party for details on the involvement of rsh) and can be used for various purposes.

- To provide remote execution of interactive applications via Sun Grid Engine comparable to the standard UNIX facility, rsh (also called remsh for HP-UX).
- To offer interactive login session capabilities via Sun Grid Engine similar to the standard UNIX facility, rlogin (note that qlogin is still required as a Sun Grid Engine representation of the UNIX telnet facility).
- To allow for the submission of batch jobs which, upon execution, support terminal I/O (standard/error output and standard input) and terminal control.
- To offer a means for submitting a standalone program not embedded in a shell-script.
- To provide a batch job submission client which remains active while the job is pending or executing and which only finishes if the job has completed or has been cancelled.
- To allow for the Sun Grid Engine system-controlled remote execution of job tasks (such as the concurrent tasks of a parallel job) within the framework of the dispersed resources allocated by parallel jobs (see the section, "Tight Integration of PEs and Sun Grid Engine Software" on page 255).

By virtue of all these capabilities, qrsh is the major enabling infrastructure for the implementation of the qtcsh and the qmake facilities as well as for the so called tight integration of Sun Grid Engine with parallel environments such as MPI or PVM.

qrsh Usage

The general form of the qrsh command is:

```
% qrsh [options] program|shell-script [arguments] \
       [> stdout_file] [>&2 stderr_file] [< stdin_file]
```

qrsh understands almost all options of qsub and provides only a few additional ones.

- -now yes|no – This option controls whether the job is scheduled immediately and rejected if no appropriate resources are available, as usually desired for an interactive job—hence it is the default—or whether the job is queued like a batch job, if it cannot be started at submission time.

- `-inherit` – `qrsh` does not go through the Sun Grid Engine scheduling process to start a job-task, but it assumes that it is embedded inside the context of a parallel job which already has allocated suitable resources on the designated remote execution host. This form of `qrsh` commonly is used within `qmake` and within a tight parallel environment integration. The default is not to inherit external job resources.
- `-verbose` – This option presents output on the scheduling process. It is mainly intended for debugging purposes and therefore switched off per default.

Transparent Job Distribution with `qtcsh`

`qtcsh` is a fully compatible replacement for the widely known and used UNIX C-Shell (`csh`) derivative `tcsh` (`qmake` is built around `tcsh` - see the information provided in <`sge_`*root*>/`3rd_party` for details on the involvement of `tcsh`). It provides a command-shell with the extension of transparently distributing execution of designated applications to suitable and lightly loaded hosts via Sun Grid Engine. Which applications are to be executed remotely and which requirements apply for the selection of an execution host is defined in configuration files called `.qtask`.

Transparent to the user, such applications are submitted for execution to Sun Grid Engine via the `qrsh` facility. Since `qrsh` provides standard output, error output and standard input handling as well as terminal control connection to the remotely executing application, there are only three noticeable differences between executing such an application remotely as opposed to executing it on the same host as the shell.

- The remote host may be much better suited (more powerful, lower loaded, required hard/software resources installed) than the local host, which may not allow execution of the application at all. This is a desired difference, of course.
- There will be a small delay incurred by the remote startup of the jobs and by their handling through Sun Grid Engine.
- Administrators can restrict the usage of resources through interactive jobs (`qrsh`) and thus through `qtcsh`. If not enough suitable resources are available for an application to be started via the `qrsh` facility or if all suitable systems are overloaded, the implicit `qrsh` submission will fail and a corresponding error message will be returned (`Not enough resources ... try later`).

In addition to the *standard* use, `qtcsh` is a suitable platform for third party code and tool integration. Using `qtcsh` in its single-application execution form `qtcsh -c` *appl_name* inside integration environments presents a persistent interface that almost never has 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 from within shell scripts of any type, C programs and even Java applications.

qtcsh Usage

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:

```
% [!]appl_name qrsh_options
```

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

The rest of the line specifies the name of the application which, when typed on a command line in a qtcsh, will be submitted to Sun Grid Engine for remote execution, and the options to the qrsh facility, which will be used and which define resource requirements for the application.

Note – The application name must appear in the command line exactly like defined in the .qtask file. If it is prefixed with an absolute or relative directory specification it is assumed that a local binary is addressed and no remote execution is intended.

Note – Csh aliases, however, 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
```

will 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 where each of them can be on or off:

- Local or remote execution of commands (remote is default)
- Immediate or batch remote execution (immediate is default)
- Verbose or non-verbose output (non-verbose is default)

The setting of these modes can be changed using option arguments of `qtcsh` at start time or with the shell builtin command `qrshmode` at runtime. See the `qtcsh` entry in the *Sun Grid Engine 5.3 and Sun Grid Engine, Enterprise Edition 5.3 Reference Manual* for more information.

Parallel Makefile Processing with `qmake`

`qmake` is a replacement for the standard UNIX `make` facility. It extends `make` by its ability to distribute independent `make` steps across a cluster of suitable machines. `qmake` is built around the popular GNU-make facility, `gmake`. See the information provided in <`sge_`*root*>/`3rd_party` for details on the involvement of `gmake`.

To ensure that a complex distributed `make` process can run to completion, `qmake` first allocates the required resources in an analogous form like a parallel job. `Qmake` then manages this set of resources without further interaction with the Sun Grid Engine scheduling. It distributes `make` steps as resources are or become available via the `qrsh` facility with the `-inherit` option enabled.

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

- Provided that the individual `make` steps have a certain duration and that there are enough independent `make` steps to be processed, the parallelization of the `make` process will be sped up significantly. This is a desired difference, of course.
- In the `make` steps to be started up remotely, there will be an implied small overhead caused by `qrsh` and the remote execution as such.
- To take advantage of the `make` step distribution of `qmake`, the user has to specify as a minimum the degree of parallelization; i.e., 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 in general of `make` certainly is the compilation of complex software packages. This may not be the major application for `qmake`, however. Program files are often quite small (as a matter of good programming practice) and hence compilation of a single program file, which is a single `make` step, often only takes a few seconds. Furthermore, compilation usually implies a lot of file access (nested include files) which may not be accelerated if done for multiple `make` steps in parallel, because the file server can become the bottleneck effectively serializing all the file access. So a satisfactory speed-up of the compilation process sometimes cannot be expected.

Other potential applications of `qmake` are more appropriate. An example is the steering of the interdependencies and the workflow of complex analysis tasks through make-files. This is common in some areas, such as EDA, and each `make` step in such environments typically is a simulation or data analysis operation with non-negligible resource and computation time requirements. A considerable speed-up can be achieved in such cases.

`qmake` Usage

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

```
% qmake [-pe pe_name pe_range][further options] \
        -- [gnu-make-options][target]
```

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

Specific attention has to be paid on the usage of the `-pe` option and its relation to the `gmake -j` option. Both options can be used 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. Hence, `qmake` makes the assumption,

that a default environment for parallel makes is configured which is called `make`. Furthermore, `gmake´s -j` allows no specification of a range, but only for a single number. `Qmake` will interpret the number given with `-j` as a range of `1-`*<given_number>*. As opposed to this, `-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 --
```

While the following command lines cannot be expressed via 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 `-inherit`) or within a batch job (with `-inherit`). These two modes initiate a different sequence of actions:

- **Interactive** – When `qmake` is invoked on the command-line, the `make` process as such is implicitly submitted to Sun Grid Engine via `qrsh` taking the resource requirements specified in the `qmake` command-line into account. Sun Grid Engine then selects a *master machine* for the execution of the parallel job associated with the parallel `make` job and starts the `make` procedure there. This is necessary, because the `make` process can be architecture dependent and the required architecure 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 which have been allocated by Sun Grid Engine for the job and which are passed to `qmake` via the parallel environment hosts file.
- **Batch** – In this case, `qmake` appears inside a batch script with the `-inherit` option (if the `-inherit` option was not present, a new job would be spawned as described for the first case above). This results in `qmake` making use of the resources already allocated to the job into which `qmake` is embedded. It will use `qrsh -inherit` directly to start `make` steps. When calling `qmake` in batch mode, the specification of resource requirements or `-pe` and `-j` options is ignored.

Note – Also single CPU jobs have to request a parallel environment (`qmake -pe make 1 --`). If no parallel execution is required, call `qmake` with `gmake` command-line syntax (without Sun Grid Engine options and "`--`"), it will behave like `gmake`.

Refer to the `qmake` entry in the *Sun Grid Engine 5.3 and Sun Grid Engine, Enterprise Edition 5.3 Reference Manual* for further detail on `qmake`.

CHAPTER 5

Checkpointing, Monitoring, and Controlling Jobs

After you have submitted jobs by way of the Sun Grid Engine 5.3 system, you need to be able to monitor and control them. This chapter provides both background information about, and instructions for, accomplishing these tasks.

Included in this chapter are instructions for the following specific taks.

About Checkpointing Jobs

This section explores two different types of job checkpointing.

- *User-level*
- *Kernel-level*

User-Level Checkpointing

Many application programs, especially those that normally consume considerable CPU time, have implemented 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. These files (called restart files) can be processed if the application is aborted and restarted at a later time and a consistent state can be reached, comparable to the situation just before the checkpoint. As the user mostly has to deal with the restart files in order to move them to a proper location, this kind of checkpointing is called *user-level* checkpointing.

For application programs that do not have an integrated (user-level) checkpointing, an alternative can be to use a so-called *checkpointing library* which can be provided by the public domain (see the *Condor* project of the University of Wisconsin, for example) or by some hardware vendors. Relinking an application with such a library installs a checkpointing mechanism 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 re-linking of the application is necessary in this case. Kernel-level checkpointing is usually applicable for single processes as well as for complete process hierarchies. I.e., 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.

Sun Grid Engine supports operating system checkpointing if available. Please refer to the Sun Grid Engine Release Notes for information on the currently supported kernel-level checkpointing facilities.

Migration of Checkpointing Jobs

Checkpointing jobs are interruptible at any time, since their restart capability ensures that only few work already done must be repeated. This ability is used to build Sun Grid Engine's migration and dynamic load balancing mechanism. If requested, checkpointing Sun Grid Engine jobs are aborted on demand and migrated to other machines in the Sun Grid Engine pool thus averaging the load in the cluster in a dynamic fashion. Checkpointing jobs are aborted and migrated for the following reasons.

- The executing queue or the job is suspended explicitly by a `qmod` or `qmon` command.

- The executing queue or the job is suspended automatically because a suspend threshold for the queue has been exceeded (see the section, “How To Configure Load and Suspend Thresholds” on page 197) and the checkpoint occasion specification for the job includes the suspension case (see the section, “How To Submit, Monitor, or Delete a Checkpointing Job from the Command Line” on page 114).

You can identify a job that is about to migrate by the state `m` for migrating in the `qstat` output. A migrating job moves back to `sge_qmaster` and is subsequently dispatched to another suitable queue if any is available.

Composing a Checkpointing Job Script

Shell scripts for kernel-level checkpointing show no difference from regular shell scripts.

Shell scripts for user-level checkpointing jobs differ from regular Sun Grid Engine batch scripts only in their ability to properly handle the case if they get restarted. The environment variable, `RESTARTED` is set for checkpointing jobs which are restarted. It can be used to skip over sections of the job script which should be executed during the initial invocation only.

Thus, a transparently checkpointing job script may look similar to CODE EXAMPLE 5-1.

```
#!/bin/sh
#Force /bin/sh in Sun 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
fem
#End of scriptfile
```

CODE EXAMPLE 5-1 Example of Checkpointing Job Script

It is important to note that the job script is restarted 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 and thus skipping those lines in the script which are critical to be executed more than once.

Note – Kernel-level checkpointing jobs are interruptible at any point of time and also the embracing shell script is restarted exactly from the point where the last checkpoint occurred. Therefore, the `RESTARTED` environment variable is of no relevance for kernel-level checkpointing jobs.

▼ How To Submit, Monitor, or Delete a Checkpointing Job from the Command Line

Enter the following command with the appropriate switches.

```
#qsub options arguments
```

Submitting a checkpointing job works the same way as for regular batch scripts, except for the `qsub -ckpt` and `-c` switches, which request a checkpointing mechanism and define the occasions at which checkpoints have to be generated for the job. The `-ckpt` option takes one argument which is the name of the checkpointing environment ("About Checkpointing Support" on page 238) to be used. The `-c` option is not mandatory and also takes one argument. It can be used to overwrite the definitions of the `when` parameter in the checkpointing environment configuration (see the `checkpoint` entry in the *Sun Grid Engine 5.3 and Sun Grid Engine, Enterprise Edition 5.3 Reference Manual* for details).

The argument to the `-c` option can be one of the following one-letter selection (or any combination thereof) or a time value alternatively:

- `n` – No checkpoint is performed. This has highest precedence
- `s` – A checkpoint is only generated if the `sge_execd` on the jobs host is shut down.
- `m` – Generate checkpoint at minimum CPU interval defined in the corresponding queue configuration (see the `min_cpu_interval` parameter in the `queue_conf` manual page).
- `x` – A checkpoint is generated if the job gets suspended.

- `interval` – Generate checkpoint in the given interval but not more frequently than defined by `min_cpu_interval` (see above). The time value has to be specified as hh:mm:ss (two digit hours, minutes and seconds separated by colon signs).

The *monitoring* of checkpointing jobs just differs from regular jobs by the fact, that these jobs may migrate from time to time (signified by state `m` for migrating in the output of `qstat`, see above) and, therefore, are not bound to a single queue. However, the unique job identification number stays the same as well as the job name.

Deleting checkpointing jobs works just the same way as described in section "Controlling Sun Grid Engine Jobs from the Command Line" on page 130.

▼ How To Submit a Checkpointing Job with QMON

- **Follow the instructions in "Advanced Example" on page 81, taking note of the following additional information.**

Submission of checkpointing jobs via `QMON` is identical to the submission of regular batch jobs with the addition of specifying an appropriate checkpointing environment. As explained in the procedure, "Advanced Example" on page 81, the Job Submission dialogue box provides an input window for the checkpointing environment associated with a job. Aside to the input window there is an icon button, which opens the Selection dialogue box displayed in FIGURE 5-1. You can select a suitable checkpoint environment from the list of available ones with it. Ask your system administrator for information about the properties of the checkpointing environments installed at your site, or refer to the section, "About Checkpointing Support" on page 238.

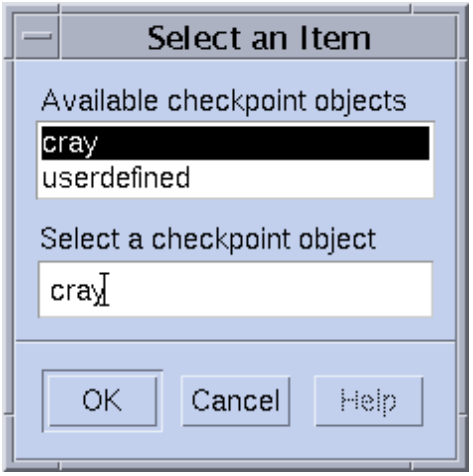


FIGURE 5-1 Checkpoint Object Selection

File System Requirements

When a checkpointing library based user-level or kernel-level checkpoint is written, a complete image of the virtual memory the process or job to be checkpointed covers needs to be dumped. Sufficient disk space must be available for this purpose. If the checkpointing environment configuration parameter `ckpt_dir` is set the checkpoint information is dumped to a job private location under `ckpt_dir`. If `ckpt_dir` is set to `NONE`, the directory in which the checkpointing job was started is used. Refer to the `checkpoint` entry in the *Sun Grid Engine 5.3 and Sun Grid Engine, Enterprise Edition 5.3 Reference Manual* 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`.

An additional requirement concerning the way how the file systems are organized is caused by the fact, that the checkpointing files and the restart files must be visible on all machines in order to successfully migrate and restart jobs. Thus NFS or a similar file system is required. Ask your cluster administration, if this requirement is met for your site.

If your site does not run NFS or if it is not desirable to use it for some reason, you should be able to transfer the restart files explicitly at the beginning of your shell script (e.g. via `rcp` or `ftp`) in the case of user-level checkpointing jobs.

Monitoring and Controlling Sun Grid Engine Jobs

In principle, there are three ways to monitor submitted jobs.

- With the Sun Grid Engine graphical user's interface, `QMON`
- From the command line with the `qstat` command
- By electronic mail

▼ How To Monitor and Control Jobs with `QMON`

The Sun Grid Engine graphical user's interface, `QMON`, provides a dialogue box specifically designed for controlling jobs.

- **In the `QMON` Main menu, press the Job Control button, then proceed according to the additional information detailed in the following sections.**

The general purpose of this dialogue box is to provide the means to monitor all running, pending and a configurable number of finished jobs known to the system or parts thereof. The dialogue box can also be used to manipulate jobs, i.e. to change their priority, to suspend, resume and to cancel them. Three list environments are displayed, one for the running jobs, another for the pending jobs waiting to be dispatched to an appropriate resource and the third for recently finished jobs. You can select between the three list environments via clicking to the corresponding tab labels at the top of the screen.

In its default form (see FIGURE 5-2) it displays the columns JobId, Priority, JobName and Queue for each running and pending job.

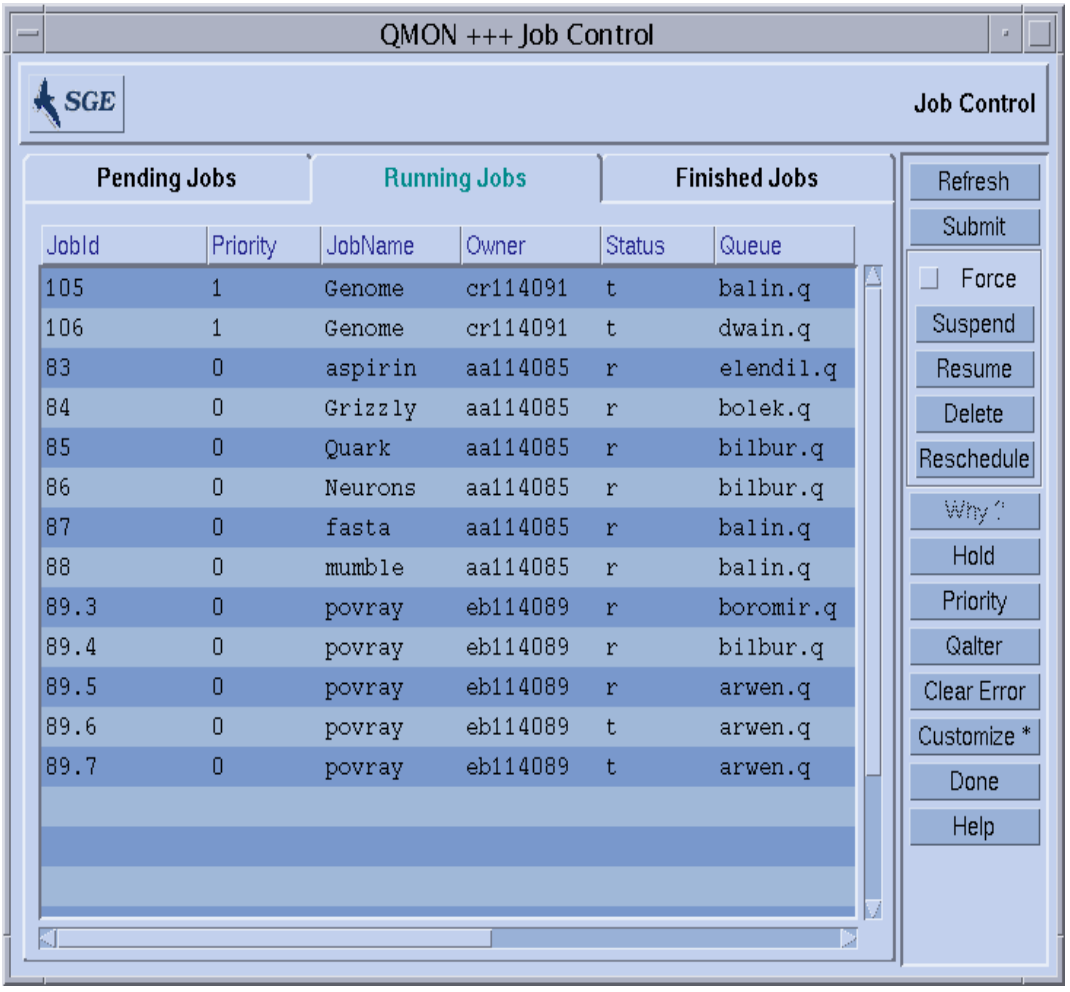


FIGURE 5-2 Job Control Dialogue Box—Standard Form

You can configure the set of information displayed with a Customization dialogue box, (see FIGURE 5-3), which is opened upon pushing the Customize button in the Job Control dialogue box.

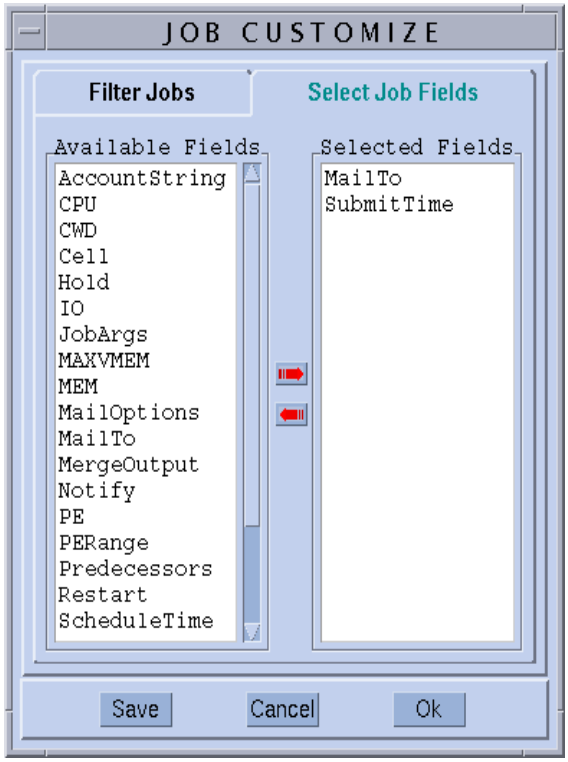


FIGURE 5-3 Job Control Customization Dialogue Box

With the Customization dialogue box it is possible to select further entries of the Sun Grid Engine job object to be displayed and to filter the jobs of interest. The example in FIGURE 5-3 selects the additional fields, MailTo and Submit Time.

The Job Control dialogue box displayed in FIGURE 5-4 depicts the enhanced look after the customization has been applied in case of the Finished Jobs list.

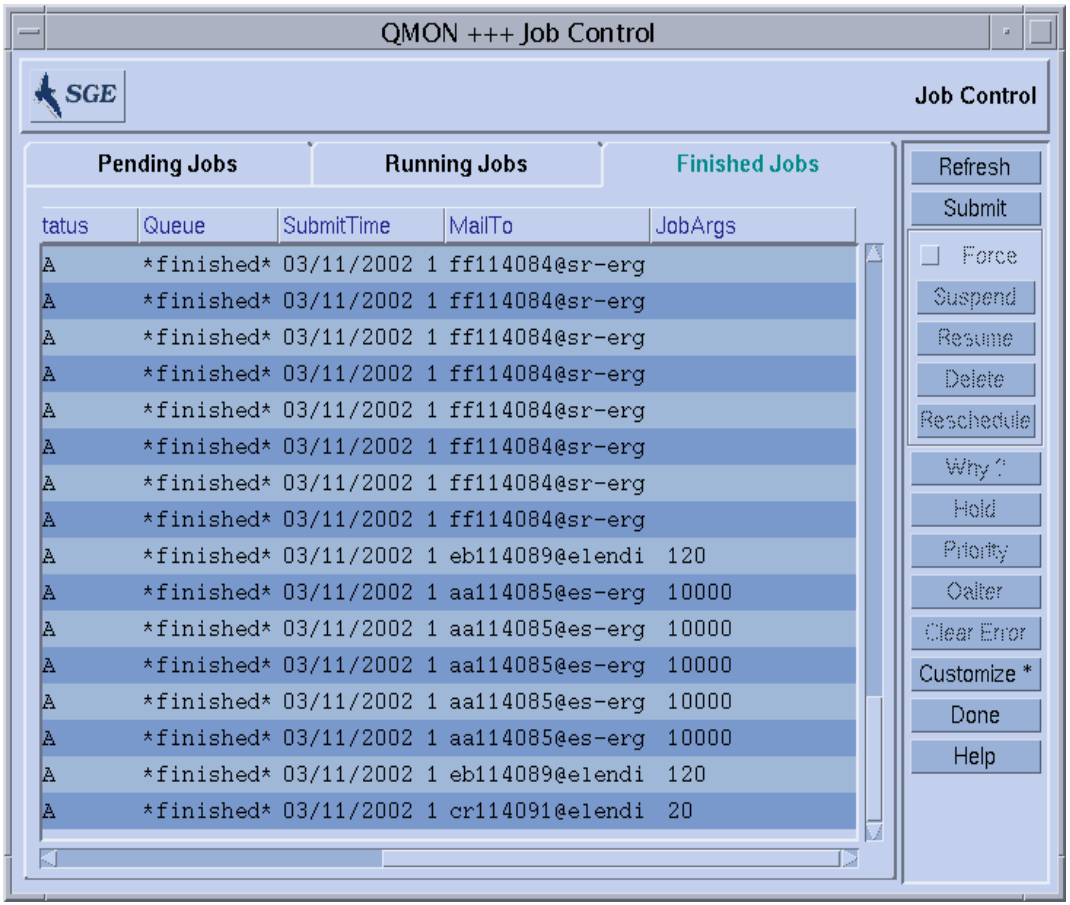


FIGURE 5-4 Job Control Dialogue Box Finished Jobs—Enhanced

The example of the filtering facility in FIGURE 5-5 selects only those jobs owned by `ferstl` which run or are suitable for architecture solaris64.

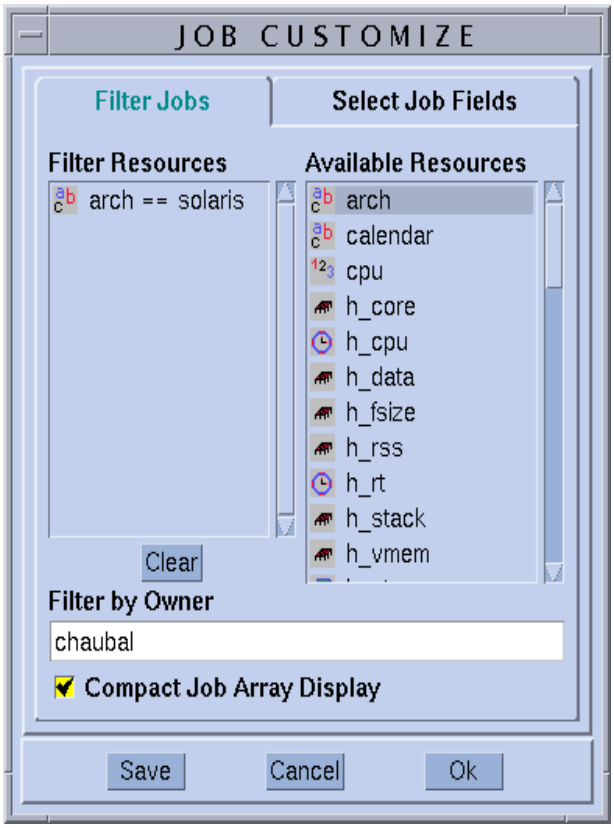


FIGURE 5-5 Job Control Filtering

The resulting Job Control dialogue box showing Pending Jobs is displayed in FIGURE 5-6.

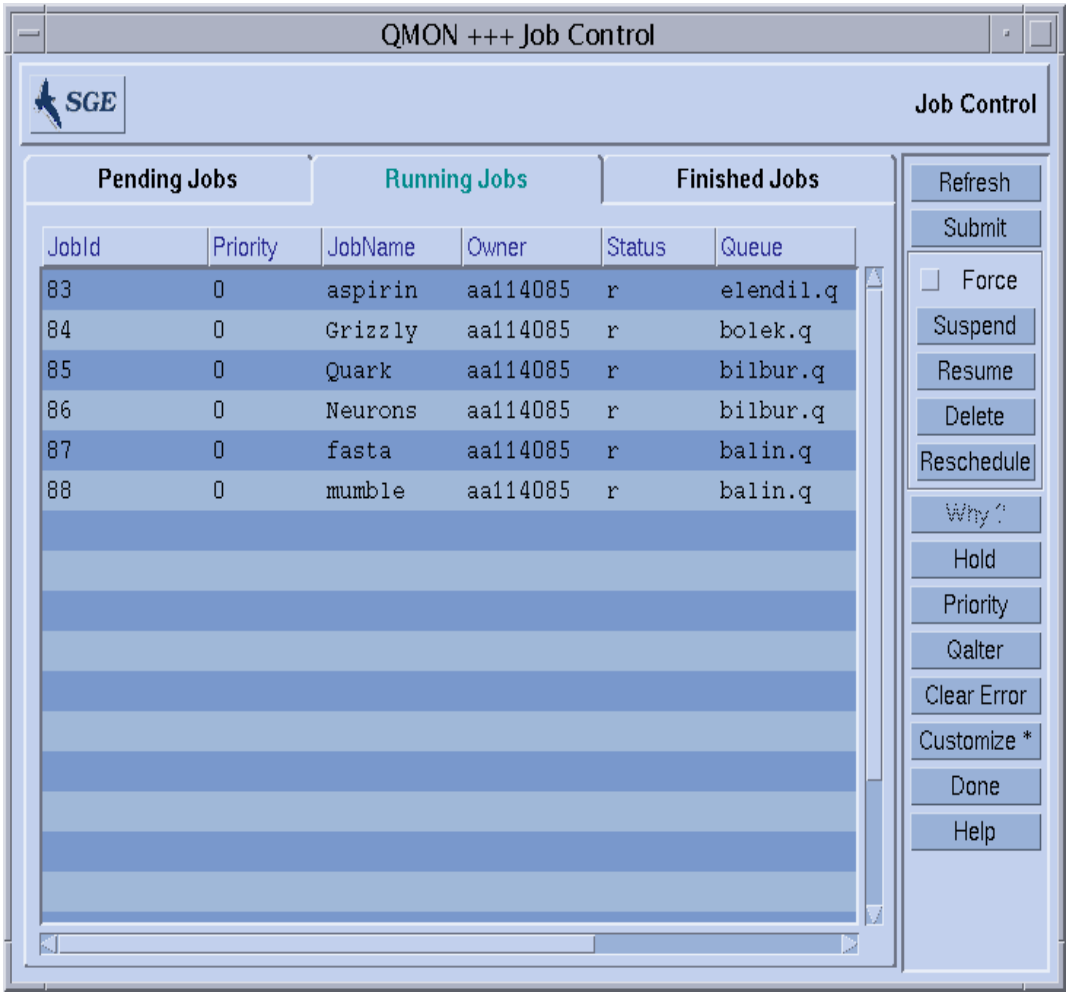


FIGURE 5-6 Job Control Dialogue Box—After Filtering

Note – The Save button displayed in the Customization dialogue box in FIGURE 5-3, for example, stores the customizations into the file `.qmon_preferences` in the user's home directory and thus redefines the default appearance of the Job Control dialogue box.

The Job Control dialogue box in FIGURE 5-6 is also an example of how array jobs are displayed in `QMON`.

Jobs can be selected (for later operation) with the following mouse/key combinations:

- Clicking on a job with the left mouse button while the Control key is pressed starts a selection of multiple jobs.
- Clicking on another job with the left mouse button while the Shift key is pressed selects all jobs in between and including the job at the selection start and the current job.
- Clicking on a job with the left mouse button while the Control key is pressed toggles the selection state of a single job.

The selected jobs can be suspended, resumed (unsuspended), deleted, held back (and released), re-prioritized and modified (`Qalter`) through the corresponding buttons at the right side of the screen.

The actions suspend, unsuspend, delete, hold, modify priority and modify job may only be applied to a job by the job owner or by Sun Grid Engine managers and operators (see “Managers, Operators and Owners” on page 67). Only running jobs can be suspended/resumed and only pending jobs can be held back and modified (in priority as well as in other attributes).

Suspending a job means the equivalent to sending the signal, `SIGSTOP`, to the process group of the job with the UNIX `kill` command, which halts the job and no longer consumes CPU time. Unsuspending the job sends the signal, SIGCONT, thereby resuming the job (see the `kill` manual page of your system for more information on signalling processes).

Note – Suspension, unsuspension and deletion can be forced; i.e., registered with `sge_qmaster` without notification of the `sge_execd` controlling the job(s), in case the corresponding `sge_execd` is unreachable—for example, due to network problems. Use the `Force` flag for this purpose.

If using the `Hold` button on a selected pending job, the Set Hold sub-dialogue box is opened (see FIGURE 5-7).

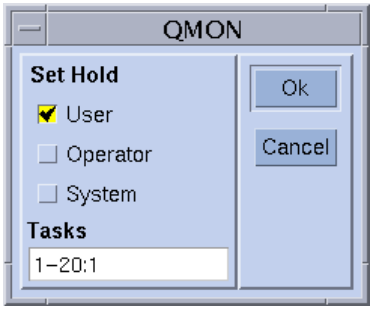


FIGURE 5-7 Job Control Holds

The Set Hold sub-dialogue box enables setting and resetting user, system, and operator holds. User holds can be set/reset by the job owner as well as Sun Grid Engine operators and managers. Operator holds can be set/reset by managers and operator and manager holds can be set/reset by managers only. As long as any hold is assigned to a job it is not eligible for execution. Alternate ways to set/reset holds are the `qalter, qhold` and `qrls` commands (see the corresponding entries in the *Sun Grid Engine 5.3 and Sun Grid Engine, Enterprise Edition 5.3 Reference Manual*).

If the Priority button is pressed, another sub-dialogue box is opened (FIGURE 5-8), which enables entering the new priority of the selected pending jobs. In Sun Grid Engine, the priority determines the order of the jobs in the pending jobs list and the order in which the pending jobs are displayed by the `Job Control` dialogue. Users can only set the priority in the range between 0 and -1024. Sun Grid Engine operators and managers can also increase the priority level up to the maximum of 1023 (see the section, “Job Priorities” on page 102 for details about job priorities).

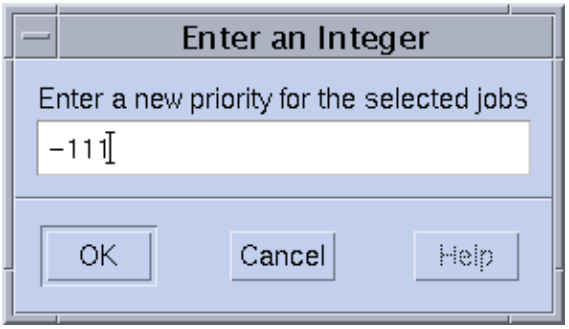


FIGURE 5-8 Job Control Priority Definition

The Qalter button, when pressed for a pending job, opens the Job Submission screen described in “How To Submit Jobs From the Graphical User Interface, QMON” on page 71 with all the entries of the dialogue box set corresponding to the attributes of the job as defined during submission. Those entries, which cannot be changed, are set insensitive. The others may be edited and the changes are registered with Sun Grid Engine by pushing the Qalter button (a replacement for the Submit button) in the Job Submission dialogue box.

The Verify flag in the Job Submission screen has a special meaning when used in the Qalter mode. You can check pending jobs for their consistency and investigate why they have not been scheduled yet. You just have to select the desired consistency checking mode for the Verify flag and push the Qalter button. The system will display warnings on inconsistencies depending on the selected checking mode. Refer to the section, “Advanced Example” on page 81 and the `-w` option in the `qalter` manual page for further information.

Another method for checking why jobs are still pending is to select a job and click on the Why? button of the Job Control dialogue box. This will open the Object Browser dialogue box and display a list of reasons which prevented the Sun Grid Engine scheduler from dispatching the job in its most recent pass. An example browser screen displaying such a message is shown in FIGURE 5-9.

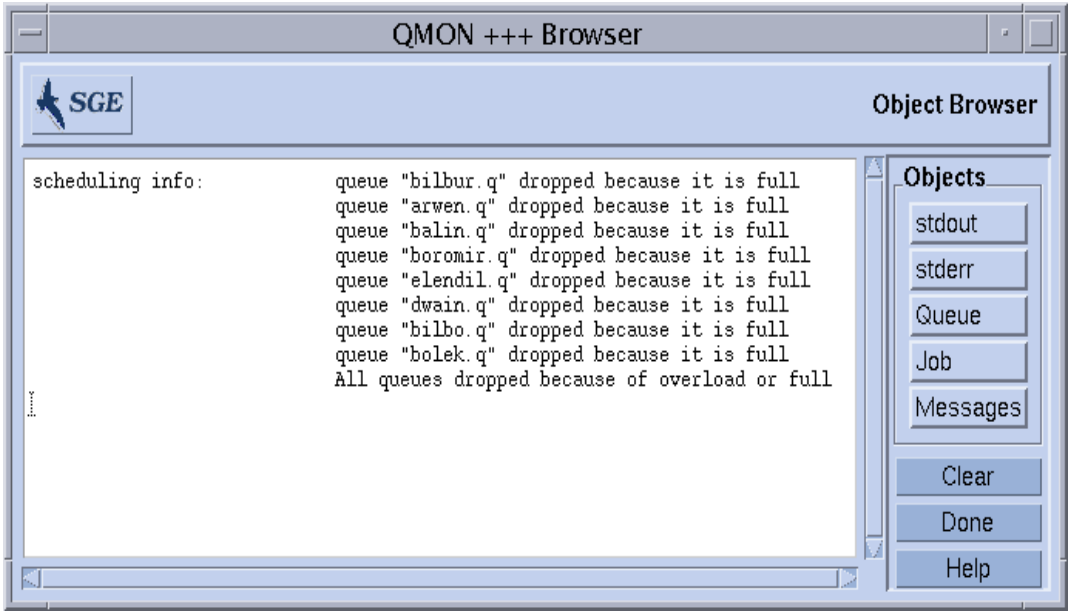


FIGURE 5-9 Browser Displaying Scheduling Information

Note – The Why? button only delivers meaningful output if the scheduler configuration parameter `schedd_job_info` is set to `true` (see `sched_conf` in the *Sun Grid Engine 5.3 and Sun Grid Engine, Enterprise Edition 5.3 Reference Manual*). The displayed scheduler information relates to the last scheduling interval. It may not be accurate anymore by the time you investigate for reasons why your job has not been scheduled.

The Clear Error button can be used to remove an error state from a selected pending job, which had been started in an earlier attempt, but failed due to a job dependent problem (e.g., insufficient permissions to write to the specified job output file).

Note – Error states are displayed using a red font in the pending jobs list and should only be removed after correcting the error condition; e.g., via `qalter`. Such error conditions are automatically reported via electronic mail, if the job requests to send e-mail in case it is aborted (e.g., via the `qsub -m a` option).

To keep the information being displayed up-to-date, QMON uses a polling scheme to retrieve the status of the jobs from sge_qmaster. An update can be forced by pressing the Refresh button.

Finally, the button provides a link to the QMON Job Submission dialogue box (see FIGURE 5-10, for example).

Additional Information with the QMON Object Browser

The QMON Object Browser can be used to quickly retrieve additional information on Sun Grid Engine jobs without a need to customize the Job Control dialogue box, as explained in section “How To Monitor and Control Jobs with QMON” on page 117.

The Object Browser is opened upon pushing the Browser icon button in the QMON main menu. The browser displays information about Sun Grid Engine jobs if the Job button in the browser is selected and if the mouse pointer is moved over a job‘s line in the Job Control dialogue box (see FIGURE 5-2 for example).

The browser screen in FIGURE 5-10 gives an example of the information displayed in such a situation.

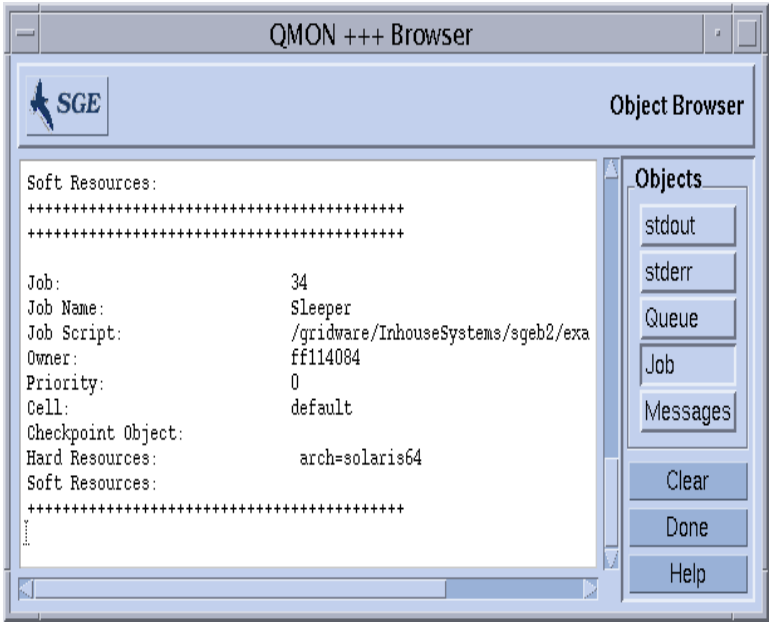


FIGURE 5-10 Object Browser—Job

▼ How To Monitor Jobs with qstat

- **From the command line, use one of the following commands, guided by information detailed in the following sections.**

```
% qstat
% qstat -f
```

The first form provides an overview of the submitted jobs only (see TABLE 5-1). The second form includes information on the currently configured queues in addition (see TABLE 5-2)).

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` entry in the *Sun Grid Engine 5.3 and Sun Grid Engine, Enterprise Edition 5.3 Reference Manual* for a detailed explanation of the `qstat` output format).

The second form is divided into two sections, the first displaying the status of all available queues, the second (entitled with the `- PENDING JOBS - ...` separator) shows the status of the `sge_qmaster` job spool area. The first line of the queue section defines the meaning of the columns with respect to the enlisted queues. The queues are separated by horizontal rules. If jobs run in a queue they are printed 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 printed as in qstat's first form.

The following columns of the queue description require some more explanation.

- `qtype` – This is the queue type—one of `B`(atch), `I`(nteractive), `P`(arallel) and `C`(heckpointing) or combinations thereof.
- `used/free` – This is the count of used / free job slots in the queue.
- `states` – This is the state of the queue—one of `u`(nknown), `a`(laram), `s`(uspended), `d`(isabled), `E`(rror), or combinations thereof.

Again, the `qstat` manual page contains a more detailed description of the `qstat` output format.

Various additional options to the `qstat` command enhance the functionality in both versions. The `-r` option can be used to display the resource requirements of submitted jobs. Furthermore the output may be restricted to a certain user, to a specific queue and the `-l` option may be used to specify resource requirements as described in the section, "Resource Requirement Definition" on page 86 for the `qsub`

command. If resource requirements are used, only those queues (and the jobs running in these queues) are displayed that match the resource requirement specification in the `qstat` command line.

TABLE 5-1 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		

TABLE 5-2 Example of `qstat -f` Output

queuename			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	

▼ How To Monitor Jobs by Electronic Mail

- **From the command line, enter the following command with appropriate arguments, guided by information detailed in the following sections.**

```
% qsub arguments
```

The `qsub -m` switch requests electronic mail to be sent to the user submitting a job or to the email address(es) specified by the `-M` flag if certain events occur (see the `qsub` manual page for a description of the flags). An argument to the `-m` option specifies the events. The following selections are available:

- `b` – Mail is sent at the beginning of the job.
- `e` – Mail is sent at the end of the job.
- `a` – Mail is sent when the job is aborted (e.g. by a `qdel` command).
- `s` – Mail is sent when the job is suspended.
- `n` – No mail is sent (the default).

Multiple of these options may be selected with a single `-m` option in a comma-separated list.

The same mail events can be configured by help of the QMON Job Submission dialogue box. See the section, “Advanced Example” on page 81.

Controlling Sun Grid Engine Jobs from the Command Line

The section “How To Monitor and Control Jobs with `QMON`” on page 117 explains how Sun Grid Engine jobs can be deleted, suspended and resumed with the Sun Grid Engine graphical user‘s interface, `QMON`.

Equivalent functionality is also available from the command line, described in this section.

▼ How To Control Jobs from the Command Line

- **From the command line, enter one of the following commands and appropriate arguments, guided by information detailed in the following sections.**

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

You use the `qdel` command to cancel Sun Grid Engine jobs, regardless whether they are running or spooled. The `qmod` command provides the means to suspend and unsuspend (resume) jobs already running.

For both commands, you will need to know the job identification number, which is displayed in response to a successful `qsub` command. If you forget the number it can be retrieved via `qstat` (see the section, "How To Monitor Jobs with `qstat`" on page 127).

Included below are several examples for both 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 unsuspend a job you must be either the owner of the job, a Sun Grid Engine manager or operator (see "Managers, Operators and Owners" on page 67).

For both commands, the `-f` force option can be used to register a status change for the job(s) at `sge_qmaster` without contacting `sge_execd` in case `sge_execd` is unreachable, e.g. due to network problems. The `-f` option is intended for usage by the administrator. In case of `qdel`, however, users can be enabled to 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` manual page in the *Sun Grid Engine 5.3 and Sun Grid Engine, Enterprise Edition 5.3 Reference Manual* for more information).

Job Dependencies

The most convenient way to build a complex task often is to split the task into sub-tasks. In these cases sub-tasks depend on the successful completion of other sub-tasks before they can get started. An example is that a predecessor task produces an output file which has to be read and processed by a successor task.

Sun Grid Engine supports interdependent tasks with its job dependency facility. Jobs can be configured to depend on the successful completion of one or multiple other jobs. The facility is enforced by the `qsub -hold_jid` option. A list of jobs can be specified upon which the submitted job depends. The list of jobs can also contain subsets of array jobs. The submitted job will not be eligible for execution unless all jobs in the dependency list have completed successfully.

Controlling Queues

As described in the section, “Queues and Queue Properties” on page 56, the owners of queues have permission to suspend / unsuspend or disable / enable queues. This is desirable, if these users need certain machines from time to time for important work and if they are affected strongly by Sun Grid Engine jobs running in the background.

There are two ways to suspend or enable queues.

- The `QMON` Queue Control dialogue box
- The `qmod` command

▼ How To Control Queues with `QMON`

- **In the `QMON` Main menu, click the Queue Control button.**

The Queue Control dialogue box, similar to that shown in FIGURE 5-11, is displayed.

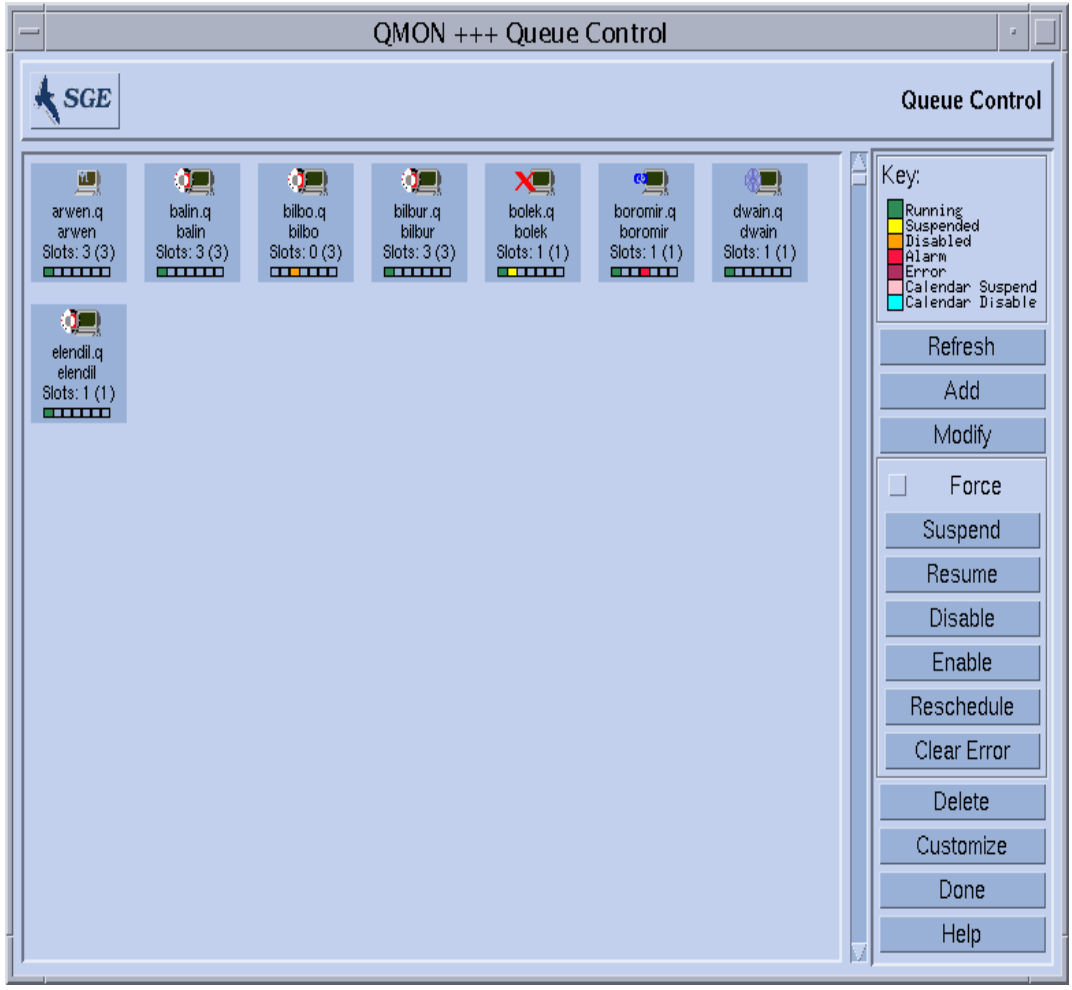


FIGURE 5-11 Queue Control Dialogue Box

The purpose of the Queue Control dialogue box is to provide a quick overview on the resources being available and on the activity in the cluster. It also provides the means to suspend / unsuspend and to disable / enable queues as well as to configure queues. Each icon being displayed represents a queue. If the main display area is empty, no queues are configured. Each queue icon is labelled with the queue name, the name of the host on which the queue resides and the number of job slots being occupied. If a `sge_execd` is running on the queue host and has already registered with `sge_qmaster` a picture on the queue icon indicates the queue host's operating

system architecture and a color bar at the bottom of the icon informs about the status of the queue. A legend on the right side of the Queue Control dialogue box displays the meaning of the colors.

For those queues, the user can retrieve the current attribute, load and resource consumption information for the queue and implicitly of the machine which hosts a queue by clicking to the queue icon with the left mouse button while the Shift key on the keyboard is pressed. This will pop-up an information screen similar to the one displayed in FIGURE 5-12.

Queues are selected by clicking with the left mouse on the button or into a rectangular area surrounding the queue icon buttons. The Delete, Suspend / Unsuspend or Disable / Enable buttons can be used to execute the corresponding operation on the selected queues. The suspend / unsuspend and disable / enable operation require notification of the corresponding `sge_execd`. If this is not possible (e.g., because the host is down) a `sge_qmaster` internal status change can be forced if the Force toggle button is switched on.

If a queue is suspended, the queue is closed for further jobs and the jobs already executing in the queue are suspended as explained in the section, “How To Monitor and Control Jobs with `QMON`” on page 117. The queue and its jobs are resumed as soon as the queue is unsuspended.

Note – If a job in a suspended queue has been suspended explicitly in addition, it will not be resumed if the queue is unsuspended. It needs to be unsuspended explicitly again.

Queues which are disabled are closed, however, the jobs executing in those queues are allowed to continue. To disable a queue is commonly used to “drain“ a queue. After the queue is enabled, it is eligible for job execution again. No action on still executing jobs is performed.

The suspend / unsuspend and disable / enable operations require queue owner or Sun Grid Engine manager or operator permission (see the section, “Managers, Operators and Owners” on page 67).

The information displayed in the Queue Control dialogue box is update periodically. An update can be forced by pressing the Refresh button. The Done button closes the dialogue box.

The Customize button enables you to select the queues to be displayed via a filter operation. The sample screen in FIGURE 5-13 shows the selection of only those queues that run on hosts belonging to architecture `osf4` (i.e, Compaq UNIX version 4). The Save button in the Customization dialogue box allows you to store your settings in the file, `.qmon_preferences` in your home directory for standard reactivation on later invocations of `QMON`.

For the purpose of configuring queues, a sub-dialogue box is opened when you press the Add or Modify button on the right side of the Queue Control screen (see the section, “How To Configure Queues with QMON” on page 164 for details).

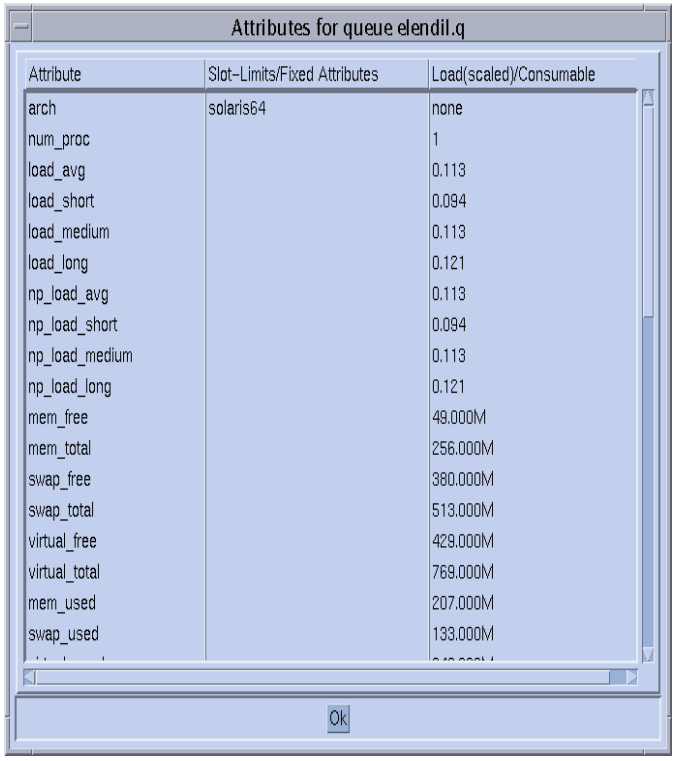


FIGURE 5-12 Queue Attribute Display

All attributes attached to the queue (including those being inherited from the host or cluster) are listed in the Attribute column. The Slot-Limits/Fixed Attributes column shows values for those attributes being defined as per queue slot limits or as fixed complex attributes. The Load(scaled)/Consumable column informs about the reported (and if configured scaled) load parameters (see the section, “Load Parameters” on page 206) and about available resource capacities based on the Sun Grid Engine consumable resources facility (see the section, “Consumable Resources” on page 194).

Note – Load reports and consumable capacities may overwrite 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 the section, “Execution Hosts” on page 29.

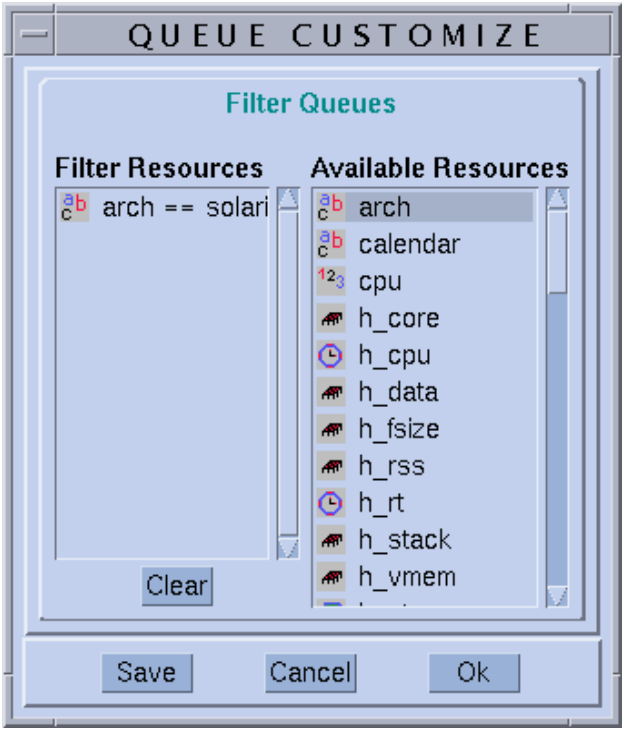


FIGURE 5-13 Queue Control Customization

▼ How To Control Queues with qmod

The section, “How To Control Jobs from the Command Line” on page 130 explained how the Sun Grid Engine command, qmod, can be used to suspend/unsuspend Sun Grid Engine jobs. However, the qmod command additionally provides the user with the means to suspend/unsuspend or disable/enable queues.

- **Enter the following command with appropriate arguments, guided by information detailed in the following sections.**

```
% qmod arguments
```

The following commands are examples how qmod is to be used for this purpose:

```
% qmod -s q_name
% qmod -us -f  q_name1, q_name2
% qmod -d q_name
% qmod -e  q_name1, q_name2, q_name3
```

The first two commands suspend or unsuspend queues, while the third and fourth command disable and enable queues. The second command uses the qmod -f option in addition to force registration of the status change in sge_qmaster in case the corresponding sge_execd is not reachable, e.g. due to network problems.

Note – Suspending / unsuspending as well as disabling / enabling queue requires queue owner, Sun Grid Engine manager or operator permission (see the section, “Managers, Operators and Owners” on page 67).

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

Customizing QMON

The look and feel of QMON is largely defined by a specifically designed resource file. Reasonable defaults are compiled in and a sample resource file is available under <sge_*root*>/qmon/Qmon.

The cluster administration may install site specific defaults in standard locations such as /usr/lib/X11/app-defaults/Qmon, by including QMON specific resource definitions into the standard .Xdefaults or .Xresources files or by putting a site specific Qmon file to a location referenced by standard search paths such as XAPPLRESDIR. Ask your administrator if any of the above is relevant in your case,

In addition, the user can configure personal preferences by either copying and modifying the Qmon file into the home directory (or to another location pointed to by the private XAPPLRESDIR search path) or by including the necessary resource definitions into the user‘s private .Xdefaults or .Xresources files. A private Qmon resource file may also by installed via the xrdb command during operation or at start-up of the X11 environment, e.g. in a .xinitrc resource file.

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

Another means of customizing QMON has been explained for the Job Control and Queue Control Customization dialogue boxes shown in FIGURE 5-2 and in FIGURE 5-13. In both dialogue boxes, you can use the Save button to store the filtering and display definitions configured with the customization dialogue boxes to the file, `.qmon_preferences`, in the user's home directory. Upon being restarted, QMON reads this file and reactivates the previously defined behavior.