The principal goal of the project research was to study the issues of managing large clusters of compute resources, and to develop solutions to the problem of how to use such clusters optimally for large-scale scientific computing. The project was a huge success by any measure. The main output of the project was the Distributed Queuing System (DQS), delivered in versions 2, 3, and 4 over the life of its development. DQS was made freely available and was used by hundreds of cluster sites: dozens of universities and national laboratories; many commercial firms, such as Merrill Lynch, Boeing, Amoco; and even government agencies, such as the British Secret Service.

The DQS project achievements are best documented by the DQS reference documents. Appendix A gives the User Guide to DQS release 3.3.2, which illustrates its use and capabilities. Appendix B shows the DQS Reference Manual for release 3.3.2. Appendix C shows the DQS Installation and Maintenance Manual.

DQS is still freely available and in wide use today, and is distributed in numerous Linux distributions, such as RedHat, Debian, and SuSE. DQS was licensed to Genias Software and became the basis of the commercial product CODINE. CODINE and DQS were eventually licensed by SUN, and became the basis of the SUN Grid Engine. DQS is supported today by Grid Queuing Systems, LLC.¹

¹ See http://gridqueuing.com/index.html
DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.
DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.
Appendix A

DISTRIBUTED QUEUEING SYSTEM - 3.3.2

USER GUIDE

December 12, 2000

Introduction

DQS is actually a simple system, which provides a multitude of options to accommodate the requirements of a wide variety of sites, and users. As the number of options increase, as they do with each succeeding generation of DQS, a user might mistakenly come to view the system as quite complex. This user guide is intended to provide an introduction to DQS for the new user as well as explaining those features most often used by the experienced user. In particular the concept of "resources" is explored with attention focused on the new DQS 3.3.2 feature, "consumable resources".

A DQS job

Any job a user needs to execute on one or more computers can be a "DQS job". For those whose sole contact with computers has been through the means of personal UNIX workstations the concept of running their jobs in a "batch" mode may be somewhat disconcerting. Users accustomed to submitting their jobs to mainframe computers will be more familiar with the attributes of DQS. But, unlike the mainframe system the DQS batch environment customarily includes multiples of autonomous UNIX based computational platforms heterogeneous in hardware architecture and operating system variant.

In its most fundamental form, a DQS job is an extension of a UNIX script used to run an application, as one might even on their own personal workstation. Let us use the "traditional" example of a FORTRAN compilation and execution of a simple application:

f90 test.f -o test
./test

where ".test" simply produces the classical "Hello World" output which is sent to standard error.

If we then wish to run this same application within a UNIX script we would create a file called "test.run" with the following lines:

- #!/bin/sh
- f90 test.f -o test >& test.errors
- ./test > test.out
Note that we redirect the stdout and stderr files to "test.out" and "test.errors" respectively.

This script would then be executed by the user on a machine of their choice, most likely their own workstation.

What then is needed to turn this script into a DQS job? Nothing... as long as one doesn't care what machine it will be executed on. All that is needed is to "submit" this job to the DQS batch queuing system.

Submitting a job

The simple example becomes a "DQS job" by submitting it to the DQS system with the "qsub" utility:

qsub test.run

The "qsub" ancillary utility will contact the qmaster and request that the job be "validated" for execution within the system. This "validation" process of determining whether or not the job requires something that does not exist in the current system. Since our test script makes no obvious requests for resources (the f90 command is not recognized as a request for a compiler resource known by DQS) all that is needed is for any host in this hypothetical cell to be idle, and available to execute the job.

Let us now take advantage of some basic DQS facilities. First we would like to have an email message sent to us upon job termination. We must instruct DQS to perform this task by inserting a "DQS directive" into the test.run script. By default DQS interprets any line of script as "DQS directive" if the first two characters of the line are the string "#$". This can be changed by the user (see "qsub -C" option in the Reference Manual).

Thus we add one line to our script:

- #!/bin/sh
- #$ -me # e-mail notification upon job termination
- f90 test.f -o test >& test.errors
- ./test > test.out

The DQS directive "#$ -me" tells the system that a mail message should be sent to the person submitting the job at the end of the job. We could also have directed that we wish to have a mail message sent at the beginning of the job and also if the job aborts with the directive "#$ -meab". The order of the symbol 'e' 'a' and 'b' in this list is not significant.

Note that the directive could also be communicated with DQS on the "qsub" command line. Instead of inserting the directive in the script, we could perform the submission with:

qsub -me test.run
In cases where only a few directives are needed this approach might be used, but as the user will see many job submissions will benefit from more complex sets of DQS directives which are better "captured" in the job script.

Querying job status

Once a user has relinquished their job to the "welcoming arms" of a queuing system they need a means for monitoring and controlling its destiny. A first step is to query the system to establish the status and "DQS identity" of the job. The "qstat" ancillary utility is used to display the state of queues and jobs. There are three forms of this display:

default (no options) Displays the state of user jobs in summary form

full listing (-f option) Displays summary queue and job status

extended listing (-ext option) Displays the full queue and job descriptions

The simplest command then to get in touch with our job is to execute the command:

qstat

and scan through the output looking for jobs we have submitted. Instead of being deluged with information about every other job in the system one can execute:

qstat -u <my user name>

where <my user name> is the login name of the user who submitted the job.

The output of this variant might look like:

---Pending Jobs---

<my user name> my-job-name dqs-job-number 0:0 QUEUED 03/25 20:40

Which would indicate with some dismay accruing to <my user name> in that the job is not RUNNING on any machine in the system. But it is queued with a priority of zero (the leftmost digit from "0:0"). And our sub-priority is zero (rightmost digit) indicating that there are no prior jobs for this user.

or more optimistically the display might offer:

<table>
<thead>
<tr>
<th>Queue Name</th>
<th>Queue Type</th>
<th>Quan</th>
<th>Load</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue1</td>
<td>batch</td>
<td>1/1</td>
<td>0.14</td>
<td>up</td>
</tr>
<tr>
<td>&lt;my user name&gt;</td>
<td>2183</td>
<td>0:1</td>
<td>r</td>
<td>RUNNING</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>02/12/96 19:25:56</td>
</tr>
</tbody>
</table>
Which would hearten us in our endeavors, because our job is (apparently) executing. The symbols on the output lines may be a bit confusing because the first line shows the status of the queue while the second describes "our job".

Let us examine the queue description first:

**Queue Name**  queue1  each queue is given a unique name by the administrator

**Queue Type**  batch  the default mode of all DQS queues

**Quan**  1/1  one resource ("1/ ") of one available (" /1") is utilized

**Load**  0.14  the load average measured by the queue1 CPU is 0.14

**er**  all of the queue states are displayed in single character symbols. The most important of these are presented between the headings of "Load" and "State". The "e" shows that the queue is ENABLED. The "rr" shows that the queue is RUNNING.

**State**  UP  The normal more of operation will be shown as "UP"

The job description is a bit less cryptic. The entry begins with `<my user name>` and followed by the DQS assigned job number (2183). The values 0:1 give the submission priority of the job, defaulted to zero and the sub-priority :1 which indicates that this is the first job running for this user. The submission priority is assigned by the user with the "qsub" option flag "-p" while the sub-priority is an internal parameter computed during each scheduling pass for all queues.

The command "qstat -ext" produces a comprehensive display of queue and job parameters as well as the status obtained with the "-f" option. Discussion of relevant portions of these extended displays will appear in later sections.

Modifying a job request

Often a user will find one or more of their jobs in the pending queue awaiting assignment to an execution queue. After review of their pending jobs, this user may decide to change the jobs submission parameters to affect the jobs future scheduling. One method for this would be to delete the job and resubmit it. A more convenient technique is to use the "qalter" ancillary utility to modify one or more of the parameters which the user assigned at the time of "qsub", or defaulted by DQS when not explicitly designated by the user.

In the simple example given here, the user provided no parameters to the QSUB command and hence the submission priority has been set to the default value of zero. If the user wishes to increase that priority the "qalter" utility would be invoked with:

qalter -jid <job number>  -p <new priority>
The <job number> is that which DQS assigned to the job in the pending queue, and the <new priority> value must be in the range -1024 to +1023.

Except for the job number, any parameter that can be employed with the "qsub" command can be used with the "qalter" command, including replacing the script file that originally accompanied the "qsub" command. The "qalter" command may not be used for jobs already in the RUNNING state, with exception of the return of "consumable resources" (see below).

Holding, deleting jobs

The user has a number of tools available to work with their jobs once the jobs are in the queuing system. For example they may decide to place a "hold" on one of their jobs in the pending queue so that another job may progress ahead of it or to delay scheduling until some other event or job has occurred. First the user may chose to submit a job to the system with a "hold" placed on the job at the time of the submission. This step involves the use of the "-h" option in the "qsub" command. Once a job is submitted the user can use the "qhold" ancillary utility to place a hold on a job if it is still in the PENDING queue. The "qhold" uses the same "-h" option.

The "-h" option is used for system administration tasks as well as user access. Thus the DQS 3.3.2 Reference Manual describes four alternatives. The user is permitted only the "u" (or user hold) or the "n" (no hold) variants. Thus at job submission the user might place a hold:

```
qsub . . -h u .. test.run
```

Or if the job is in the pending queue:

```
qhold -jid <job number> -h u
```

Once a "hold" has been placed on a job in the pending queue it will not be considered eligible for scheduling until it has either been "released" from the hold or it is deleted from the queue entirely. A job can be released from a user invoked "hold" with the "qrls" ancillary utility:

```
qrls -jid <job number> -h u
```

or the user may modify the "hold" state by using the "qalter" command:

```
qalter -jid <job number> .. .. -h n
```

Which will set the user accessible hold to "none".

A use may delete one or more of their own jobs from the queuing system if the jobs are in either the pending queue or the executing queue:
qdel <job number>

or:

qdel <job number>,<job number>,.....

Note that the job numbers are separated by commas (,) and NOT spaces.

Requesting resources

The simple example we have been using so far (test.run) has made no unusual demands for system resources. It presumes that all queues in the system have a FORTRAN compiler and that the FORTRAN dialect in our test program is consistent with all the compilers. Further, memory, disk-space and data-base locality are also not consequential in this example. These are unrealistic assumptions in most cases. Most sites using DQS contain heterogeneous collections of hardware and software and often subdivide these collections into types of use (long-term jobs, short-term jobs, etc.).

The DQS Administrator is supplied with many tools to organize the system and define the resources available to the user. Typical resources are CPU memory sizes, hardware architecture and operating system versions.

go to toc

**Hard and soft resources**

Most jobs will have one or more imperative requirements. One of the most common is the need for a particular hardware/software system (i.e. AIX-4.3.3). By default requested resources are considered essential (or "hard") unless the user precedes the request in the "qsub" command with the option "-soft".

Requirements for multiples of various resources in parallel jobs, such as 2 or more CPUs can be either "hard" or "soft". Many users choose to request at least 2 CPUs to run their parallel job and then request more processors following the option "-soft" flag in the "qsub" command line or job script. While a non-parallel user might expect to use the "-soft" option for a request of the form "I need at least 32 MB of memory but would be much happier with 64 MB), most site resource allocations will not make effective use of such a request. The most common use of the "-soft" option for non-parallel jobs is to state a preference for a queue without making it a "hard" demand.

**Consumable resources**

Site resources are by and large static over periods of time like days or weeks. CPU memory sizes and CPU computing power are not subject to moment-by-moment changes.
When they are modified the DQS site manager can adjust the resource descriptions to match the new configurations.

There is a class of resources that varies within short periods of time. A very common commercial practice, these days, is to manage software licenses for Compilers, Data Base Managers, etc. dynamically at a given site. Many sites do not purchase licenses for all of their extant platforms. A job submitted to DQS must not be scheduled for execution if that job needs one or more software licenses in order to complete but those licenses are already in use by another job.

Another common form of a time-varying resource would be the amount of shared memory available to a processor in a shared-memory multi-processor system. Shared local disk space might be another resource which is depleted and restored as jobs startup and terminate. Resources of this type are called, by DQS, "consumable resources".

**Forming resource requests**

A user specifies the resources they require in the "qsub" command line or in the DQS script file. A most direct method is to identify a specific queue as the place for the submitted job to execute:

```
qsub ... -q <my queue>
```

That request will require <my queue> for execution. If the user would prefer, but not insist on that queue they might make the command line request:

```
qsub ... -soft -q <my queue>
```

Note that DQS scans the command line and script commands from left to right. During that process any resource requests to the right of a "-hard" or "-soft" option flag will be interpreted as requiring that type of resource. Hence one could mix hard and soft resources thus:

```
qsub ... -soft -q <my queue> ... -hard <some other resource>..
```

The typical job request will not demand a specific queue. Instead the user will request one or more classes of resources which have been established by the DQS administrator. Let us presume a site with three different hardware platform architectures for which there are several CPUs available each. The site administrator has named the resources with their operating system tags, AIX433, IRIX65, SOLARIS27. In addition this example site will own one FORTRAN license each for the different operating systems. The administrator will name these, XLF, SGIFTN and FORTRAN.

To further complicate our example, each brand of CPU has a different amount of memory on each of its three separate CPUs, 32 Megabytes, 64 Megabytes and 128 Megabytes.
The example we have been using (test.run) will now be submitted in a more realistic manner:

```
qsub -me -l AIX433.and.(mem.gt.32).and (XLF.eq.1) test.run
```

The command line now has the resource request appended to it. Requests for resources other than specific queue names begin with the "-l" flag and consist of a string of resource names, interspersed with logical and relational operators. Since the string must have NO embedded blanks, parenthesis may be used to aid readability.

The resource request is interpreted by DQS as follows:

1. The resource request is a "hard" request by default.
2. Any queues that contain the "AIX433" resource AND a "mem" resource greater than 32 and at least one XLF license will be eligible for assignment to the job.
3. When the job is started the XLF available license count will be reduced by 1. When the job terminates the XLF license count will be increased by one.

A command line or DQS script may contain one or more request strings beginning with the "-l" option flag. Each one of these strings will request at least one queue to meet the requirement. Thus:

```
qsub -l AIX433 -l AIX433
```

Would request that two queues/CPUs be allocate to this job. This same request can be restated more simply:

```
qsub -l (qty.eq.2).and.AIX433
```

Depending upon the topology of the DQS site and the requirements of a given job, resource requests can contain a number of elements. Obviously parallel jobs will require more complex resource requests than simple single-processor jobs.

Note: Relational operators can be given in FORTRAN or "C" syntax (.eq. == , .ne. != , .lt. < , .gt. > , .le. <= , .ge. >= ). Logical operators can also be given in either language syntax (.and. && , .or. || , .not. !). For compatibility with DQS 3.3.2 the comma (,) may be used in place of the logical ".and." operator.

The consumable resource "XLF" requested by the job can be returned to the license pool by a RUNNING job by executing the DQS command "qalter" with the "-rc" option:

```
qalter -rc XLF=1
```

This command would return one XLF license to the system.
"Potential" resources

DQS 3.3.2 performs a pre-validation of jobs before accepting them into the queuing system. This pre-validation consists of searching all queue definitions to see if the "hard" resources requested for the job actually exist, even if they may be in use by some other job at the time this job was submitted. If all of the "hard" resources do not exist, the job is rejected, and an error message with the reason for the rejection returned to the "qsub" ancillary utility and displayed for the user.

In some cases a user may be aware that a resource (such as a new) queue will be added or returned to the DQS at some point in the future. They may wish to submit their job and place it into the pending queue to await the appearance of the new resource. This can be accomplished by adding the "FORCE REQUEST" flag ('-F') to the QSUB command line or DQS script:

qsub -F -1 (wild-eyed_scheme).and.mem.gt.1000000

The "-F" flag should be used with care as no pre-validation is performed and a job may have an erroneous resource request which will leave it "orphaned" in the pending queue until either the job's owner or the DQS Administrator deletes it at a later time.

Moving jobs

Once a job has been placed into the RUNNING state and is executing in one or more queues its parameters cannot be modified nor can it be moved to another location in the system. Pending (non-executing) jobs can be moved from one target queue to another by one of the following methods:

1. Implicitly... using the "qalter" command to modify a resource request such that the eligible queues will be changed
2. Explicitly... when the job was submitted with the "-q" option to identify a specific target queue, that target can be changed by using the "qalter" command with a different "-q" target destination-id
3. Intercell ...If a site has more than one DQS cell operating and they are mutually "authenticated" a user may move their jobs from one cell to another if that user has permission to both cells.

Cells and queues

What is a cell? It is the collection of computer hosts and DQS software which make up a single entity managed by a daemon called the "qmaster". In the following diagram are displayed four CPUs. One of these is executing the qmaster daemon. Two processors are executing the dqs_execd daemons. These two processors are related to the queues shown here and would execute any job assigned to those queues. The computer labeled "dqs
host" is not running any of the DQS daemons. It is known to the qmaster because the site administrator has added that name to the qmaster's host list. This action makes this host a "trusted DQS host" as are any hosts running the daemon.

A DQS site may have more than one "cell". The site administrator may choose to keep each cell independent and separate from the others. On the other hand they may organize the system so that one or more cells will have authorized communications with others.

A user logged into a host in one cell can submit jobs to the other cells, or they can perform the QSTAT function for the other cells.

**Qmove**

The user can move one of their jobs in a pending queue in one cell to the pending queue in another cell. The qmove ancillary utility is provided for this inter-cell transfer purpose only. The usual command would be:

```
qmove <job number>@CELL_C2
```

Which would move the numbered job from CELL_C2 to the cell in which the qmove ancillary utility is being executed. Where a user in CELL_C3 wishes to move a job from CELL_C2 to CELL_C1 the command would be:
qmove -cell CELL_C1 <job number>@CELL_C2

The effects of this move process can be somewhat surprising:

1. The job number is changed.
2. The moved job becomes a newly submitted job in the destination queue.
3. The resource requests are interpreted in terms of the new cell's characteristics.
4. A moved job is not pre-validated and thus can become an orphan in the new destination queue.

Queue-Complexes

Queue-complexes are arbitrary resource definitions that once defined, can be associated with queues. These resource definitions can be combinations of available licenses, memory, architecture, available software, etc. Queue-complexes are used by the scheduler at job submission time to determine a best fit between requested and available resources. These resource specifications are completely arbitrary, allowing for highly configurable systems.

A sample queue complex.

```
mem=128
ARC=RIOS
matlab
pvm3
p5
```

Once defined, the complex can be associated with a queue. An arbitrary number of complexes can be assigned to queues. At job submission time, the qmaster matches the users requests with the complex definitions to select the queues which meet the users needs.

Suspending queues and jobs

The user will note that from time to time one or more queue may display the SUSPENDED status. When this occurs, any job executing on that queue is suspended also, but NOT terminated. As the queue is unsuspended the job is continued from the point where it was suspended. During the period of its suspension, all of its files remain open and all memory and paging space allocated to the job remain in that state.

When does a queue get suspended? The DQS administrator and anyone designated as the queue's owner can suspend that queue using the "qmod" command. There is one additional method, which may appear in some site configurations. If a queue is assigned to a host, which is also serving as the personal workstation for some user of the system, they may chose to use the "qidle" command at that workstation. This ancillary utility is a
X-windows facility which monitors the keyboard and mouse on a workstation. If these devices are being used the "qidle" facility will suspend the queues on that workstation.

One additional means by which a queue may be suspended is to designate it as a subordinate queue to another queue. The usual application of this facility is when a host serves both as a parallel and single processor resource. The single processor queue is made subordinate to the parallel queue. When a parallel job is started the subordinate queue and any job being executed there will be suspended.

Parallel jobs

A major feature of DQS is its support for the scheduling and management of parallel jobs to be run on two or more of the hosts in a system. There are three components in submitting parallel jobs:

1. Resource requests must identify the requisite system capabilities and machines for as many hosts as absolutely required ("hard") plus those additional resources that would be desirable ("soft"). This is very important to sound system management because it is possible for a user to request and be assigned a single host, then launch additional processes on other hosts by mistake, or design. Some of the parallel message passing systems do not prevent this abuse(error).

2. The ".par" flag should be used to specify one of several paradigms known to DQS. P4, MPI, PVM and TCGMSG are popular message passing systems, which are supported with DQS system facilities. If none of these schemes match the user's requirements, at a minimum the option "-par GENERIC_ALL" should be selected.

3. Most parallel jobs require the execution of a startup script to initiate the message passing system's daemons, or other mechanism. The MPI system starts with the function "mpirun" for example. A parallel job submission then should identify the script and any parameters as an element in either the ".master", ".q" or ".l" options of the command line or DQS script. For example:

\[ \text{qsub -me -l (qty.eq.4).and.(exec.eq.mpirun).and.AIX433} \]

This will request four AIX433 hosts to run a parallel job. After the job is put into execution, but before the user's job script is executed, the function "mpirun" will be executed in the working directory of that user.

Job execution environment

The simple "test.run" example we have been using so far will have operated with the following characteristics

1. The job will be executed in the user's home director. Using the ".cwd" option can change the job execution to the working directory from which the "qsub" ancillary utility was executed.
2. The standard output and standard error files are written to the directory in which the program is being executed. The "-o" and "-e" and "-j" options can be used to specify what directory should be used for the outputs. The "-j" option can "join" stderr and stdout and put the results where specified.

3. The normal environment, which the user encounters when logging in to run a batch or interactive job, will be used. The "-v" option permits a user to specify new values for existing environment variables or to create new environment variables.

For detailed instructions on changing the jobs' environment please see "qsub" in the DQS reference manual.

DQS scheduling strategies

Once a job has disappeared into the maw of DQS, it is subjected to a variety of manipulations which are intended to utilize the entire system resources in the most optimum way while ensuring that each user is given "fair" access to those resources. The default operation of the scheduler is often adapted by each site to its own requirements. The basic process consists of:

1. First the site administrator may chose one of two options for selecting queues for assignment. The default is to sort queues by the load average they are reporting to the qmaster. The second is to sort the queues according to the internal sequence numbers assigned by the site. This second technique is used where the preference is to allocate host machines on the basis of their computing power instead of their current computing load.

2. Jobs in the queue are sorted by the priority with which they were submitted, and then their internal sub-priority and finally their job identifier. The sub-priority is recalculated each time the qmaster scans the queued jobs for scheduling. It is relative to each user and reflects the total number of jobs in this queue which are ahead of each job. This number includes any of the user's jobs which are in RUNNING state as well as though which are QUEUED.

3. Starting with the first PENDING job in the list, the qmaster attempts to find a queue which is available, not overloaded, and which meets the resource requirements of that job. If a queue is found the job is "handed off" to that queue's dqs_execd daemon, and the next job is examined.

4. If the job's requirements cannot be met, it is passed over and the next one in the list is considered.

5. Each site has a system wide parameter, set by the DQS administrator, which limits the number of jobs for a single user which may be considered during any given scheduling pass over the queue. Those pending jobs which are in this category they are marked "MAXUJOB" by the qmaster and ignored during that pass.

6. There is one additional scheduling filter provided in DQS 3.3.2. Besides the system-wide maximum user job parameter, each queue possesses a maximum job parameter. This is assigned in the queue configuration for each queue and can be
changed by the DQS administrator while the system is in operation. In the default DQS 3.3.2. release the job pre-validation process includes a "user eligibility" criteria in addition the available resource tests.

After a job has been validated as to requesting "real" resources, it is tested against the site's queues to determine which ones it would be eligible for. Of the eligible queues, the values of the "maximum user jobs" for each queue is extracted and the smallest one selected. At the same time the number of jobs in RUNNING state for this user is computed. If the minimum queue-maximum-user-jobs is not greater than the number of that user's jobs RUNNING.. the job is rejected at QSUB time and an error message returned to the user.

This last scheduling pre-validation most certainly may confuse the reader but it is the core of the "fair play" method developed at SCRUCSIT and needs to be used for a while to demonstrate its behavior and value.

Problem Solving

Even when one starts with the simple test case with which we began this User Guide, it is possible to get into one or more dead-ends on one's first, second, or whatever, attempts at using the DQS. We will proceed through a number of typical problems, which a user may encounter along the way:

1. An attempt to execute any DQS ancillary utility fails with error messages such as "Unable to contact qmaster". The host on which the ancillary utility is initiated is not in the authorized list of DQS hosts. The site administrator can verify this with the system error file.

2. An attempt to execute a DQS ancillary utility fails with messages containing "ALARM". "shutdown". The qmaster has died, or is too busy to respond to the utility within a time limit established for the system. The DQS administrator can adjust this value.

3. The submission of a job with "qsub" fails with an error message indicating that one or more resources are not available, or the job exceeds the number permitted for this user at this time. A first step is to resubmit the job with the ",F" (force job submission) flag set. Otherwise consult the DQS administrator for assistance.

4. A submitted job, which has not been forced ("-F" flag) remains in the pending queue without being placed into execution while other later jobs go onto execution. First use the command "qstat -f" and examine the list of resources shown for this job. It is possible that only one or two queues in the system satisfy the resources requested and that the other jobs which appear to have bypassed this jobs are less demanding. If possible the resources requested can me modified with the "qalter" command.

5. A job appears in the "qstat" display as RUNNING on a queue, but never terminates and a process status ("ps") fails to show the job executing in that host. An attempt to perform a "qdel" operation on this job fails. This is a serious
internal error in DQS and the DQS administrator should be contacted immediately.

6. A job appears in the "qstat" display as RUNNING and then disappears as if it has completed execution. However no job results appear and the stdout and stderr files are absent.--- This is a serious internal error in DQS and the site administrator should be contacted.

The DQS error file (err_file) and accounting file (acc_file) contain valuable information which can assist the knowledgeable user the means for analyzing and correcting their problems with the system. Refer to DQS 3.3.2 Error Messages document for further information.
Attention
Note that all default DQS-3.3.2 ancillary commands referenced have the digits "332" appended to the command. That is, if this document cites "qsub", use the default "qsub332" instead.

Table of Contents

Introduction

qalter - Alter a Batch Job
qconf - Queue configuration tasks
qdel - Delete on or more jobs from the system
qhold - Place a hold on one or more jobs to defer scheduling
qidle - X Windows console monitor
qmod - Modify the state of a queue
qmove - Move jobs from one cell to another
qrls - Release hold(s) on one or more jobs
qstat - Display status of selected queues and jobs
qsub - Submit a job to the queuing system

the daemons

Glossary
Introduction

This manual is based on the draft POSIX Specification "Part 2: Shell and Utilities -- Amendment: Batch Environment". DQS 3.3.2 possesses many extensions and refinements to that basic specification. Refer to the DQS 3.3.2 Appendix and the POSIX Specification for a comparison of the differences.

the utilities

qalter- Alter a Batch Job


Description

The attributes of a job are altered by a request to the qmaster. The "qalter" ancillary utility is a user-accessible batch client that requests the alteration of the attributes of one or more jobs.

The "qalter" ancillary utility alters the attributes of those jobs, and only those jobs, for which a job_identifier is presented to the utility.

The "qalter" utility alters the attributes of jobs in the order in which the job_identifiers are presented to the utility.

If the "qalter" utility fails to process a job_identifier successfully, the utility will proceed to process the remaining job_identifiers, if any.

For each job_identifier for which the "qalter" utility succeeds, each attribute of the identified job will be altered as indicated by all of the options presented to the utility.

For each identified job for which the "qalter" utility fails the utility will not alter any attribute of the job.

For each job that the "qalter" utility processes, the utility will not modify any attribute other than those required by the options and option arguments presented to the utility.
The "qalter" utility alters jobs by sending a Modify Job Request to the qmaster. At the time the "qalter" utility exits, it will have modified the job corresponding to each successfully processed job_identifier.

Jobs in the RUNNING state cannot be modified by "qalter".

**Options**

-a date time
Redefines the time at which the job becomes eligible for execution. The "date_time" option argument is described in the Glossary.

The "qalter" utility sets the Execution_Time attribute of the job to the number of seconds since Epoch that is equivalent to the local times expressed by the value of the "date_time" option argument. Specifying a "date_time" option argument that represents a time (number of seconds since Epoch) earlier than the time at which the utility exists will have the same effect on job execution as if the "-a" option had not been presented to the utility.

-A account string
Redefines the account to which the resource consumption of the job should be charged. The "account_string" option argument is described in the Glossary.

-cell cell_name
The cell to which the "qalter" request is to be sent.

-clean
Causes all elements of the job to be reset to the initial default status prior to applying any modifications (if any) appearing in this specific "qalter" command.

-cwd
Redefines the current working directory to be used when the job begins execution.

-C directive_prefix
The directive prefix consists of two ASCII characters which when appearing in the first two bytes of a script line indicate that what follows is a DQS command.

-e path_name
Redefines the path to be used for the standard error stream of the job.

If the "path_name" option argument constitutes an absolute path name, the "qalter" utility will set the Error_Path attribute of the job to the value of the path_name option argument including the hostname, if present.

If the "path_name" option argument constitutes a relative path name and no hostname is specified, the "qalter" utility will set the Error_Path attribute of the job to the value of the absolute path name. The absolute path name is derived from the relative path name by
expanding the "path-name" option argument relative to the current working directory of
the process that executed the "qalter" utility.

If the "path-name" option argument constitutes a relative path name and a hostname is
specified, the "qalter" utility will set the Error_Path attribute of the job to the value of the
option argument without expansion.

If the "path-name" option argument does not include a hostname, the "qalter" utility will
prefix the path name in the Error_Path attribute with "hostname:", where hostname is the
name of the host upon which the "qalter" utility is being executed.

The default file name for standard error output has the following form:

"job_name.e.sequence_number.pid"

If the "-e" option possess no option argument, the utility will clear the Error_Path
attribute of the job so that the default standard error output will be used when the job is
executed.

-h hold list
Redefines the types of holds, if any, on the job. The "qalter -h" will accept a value for the
"hold_list" option argument that is a string of alphanumeric characters in the portable
character set. The "hold_list" option argument is described in the Glossary.

-hard
Signifies that all resource requirements following in the "qalter" command will be hard
requirements and must be satisfied in full before a job can be scheduled.

-help
Prints a listing of all options for qalter.

-j join list
Redefines which streams of the job are to be merged. The "qalter -j" option will accept a
value for the "join_list" option argument that is a string of alphanumeric characters in the portable
character set. The "join_list" option argument is described in the Glossary.

The only characters permitted in this implementation are "y" or "n". Specifies whether
the standard error stream of the job is merged into the standard output stream. The
meaning of the "-j" option is different from the standard.

-jid job_identifier
Defines the job identifier on which the "qalter" command will act.

-l resource list
Redefines the resources that are allowed or required by the job. The "resource_list"
option argument is described in the Glossary.
-m mail options
Redefines the points in the execution of the job at which the server is to send mail about a change in the state of the job.

The "mail_options" option argument is described in the Glossary.

-master parallel destination identifier
Redefines the queue to serve as a master queue for a parallel job. The "parallel_destination_identifier" option argument is described in the Glossary.

-M mail_list (Default - the login name of the user submitting the job)
Redefines the list of users to which the server that executes the job is to send mail, if the server sends mail about the job.

mail_address[,mail_address,....]

The interpretation of mail_address is implementation defined.

-notify
Redefines the Notify attribute for a job. This attribute, when TRUE causes the batch server to send "warning" signals to a running job prior to sending the signals themselves. If a SIGSTOP is pending the job will receive a SIGUSR1 several seconds before the SIGSTOP. If a SIGKILL is pending the job will receive a SIGUSR2 several seconds before the SIGKILL. The amount of time delay is controlled by the "notify" parameter in each queue configuration.

-N name
Redefines the name of a job.

The "qalter" utility will set the Job_Name attribute of the job to the value of the name option, truncated to eight characters, if necessary.

If the "-N" option is not presented to the "qalter" utility, the utility will set the Job_Name attribute of the job to the name of the script -name argument after any directory "path-name" has been removed from the script-name.

If the "-N" option is not presented to the "qalter" utility, and the script is read from standard input, the utility will set the job-name attribute of the job to the value STDIN.

-o path_name
Redefines the path for standard output of the job.

If the "path_name" option argument constitutes an absolute path name, the "qalter" utility will set the Output_Path attribute of the job to the value of the "path_name" option argument including the hostname, if present.
If the "path_name" option argument constitutes a relative path name and no hostname is specified, the "qalter" utility will set the Output_Path attribute of the job to the value of the absolute path name. The absolute path name is derived from the relative path name by expanding the "path_name" option argument relative to the current working directory of the process that executed the "qalter" utility.

If the "path_name" option argument constitutes a relative path name and a hostname is specified, the "qalter" utility will set the Output_Path attribute of the job to the value of the option argument without expansion.

If the "path_name" option argument does not include a hostname, the "qalter" utility will prefix the path name in the Output_Path attribute with "hostname: ", where hostname is the name of the host upon which the "qalter" utility is being executed.

The default file name for standard output has the following form:

job_name.o.sequence_number.pid

If the "-o" option possess no option argument, the utility will clear the Output_Path attribute of the job so that the default standard error output will be used when the job is executed.

-p priority
Redefines the priority of the job.

The "qalter" utility will accept a value for the priority option argument that conforms to the syntax for signed decimal integers, and which is not less than -1024 and not greater than 1023.

-par parallel paradigm
DQS can process, monitor and initiate parallel jobs using one of several message passing system paradigms. This option redefines the paradigm to use for a job. The "parallel_paradigm" option argument is described in the Glossary.

-passwd
Specifies that a password list will be requested for AFS and/or DFS re-authentication. If given, DQS will utilize the password list to authenticate the process prior to executing the job, so that re-authentication will be unnecessary.

-passwd_file filename
Redefines the name of the file containing the AFS passwords. (See AFS support in the User Guide.)

-q destination_identifier
Resets the destination identifier list, i.e. queue name(s), for the job. The "destination_identifier" option argument is described in the Glossary.
-r  y | n
Redefines whether the job is Rerunnable.

If the value of the option argument is 'y' the "qalter" utility will set the Rerunnable
attribute of the job to TRUE.

If the value of the option argument is 'n' the "qalter" utility will set the Rerunnable
attribute of the job to FALSE.

The "qalter" utility will consider it an error if any character other than 'y' or 'n' is specified
in the option argument.

-reauth  seconds
Specifies the time interval for AFS/DFS re-authentication.

-soft
Signifies that all resource requirements following in the "qalter" command will be soft
requirements and will be filled on a "as available" basis before a job can be scheduled.

-S  path_name_list
Redefines the shell that interprets the script at the destination system.

The "qalter" utility will accept a "path_name_list" option argument that conforms to the
following syntax:

path-name[@host][,path-name[@host],...]

The "qalter" utility will accept only one path name that is missing a corresponding host
name. The "qalter" utility will allow only one path name per named host.

The "qalter" utility will add a value to the Shell-Path-List attribute of the job for each
entry in the "path_name_list" option argument.

If the "-S" option has no arguments the utility will set the path-list attribute of the job to
the null string.

-v  variable_list
Redefines the environment variables to be exported to the job.

The "variable_list" option argument is described in the Glossary.

If the -v option is presented to the "qalter" utility, the utility will also add to the
"environment_variable_list" attribute of the job every variable named in the
"variable_list" option argument and, optionally, values of specified variables.
-verify
This does not affect any job attributes. It causes the qalter utility to send the contents of
the "qalter" command to standard out for verification by the user.

-V
Specifies that all environment active within the dqs_execd at the time it initiates the job
be exported to the job.

Operands

Script File

The path to the script of the job. The "Script_File" operand argument is described in the
Glossary.

If the "qalter" utility is presented with a script file it will replace the job script file
originally submitted with the "qsub" command. The "qalter" utility scans the script file
interpreting lines flagged with the DQS_DELIMITER pair of characters. All command
options in the "qalter" command line and the script file are processed and replace or
modify job attributes for the job.

A script can contain directives to the "qalter" utility.

The "qalter" utility will scan the lines of the script for directives until the first line that
begins with a string other than the directive string; if directives occur on subsequent lines,
the utility will ignore those directives.

go to toc

qconf- Queue configuration tasks


Description

The "qconf" utility provides the user interface for managing the queues which make up a
DQS environment.
The "qconf" may be utilized by managers, operators and users with increasing levels of restrictions for each class of user. The DQS manager has unrestricted control over queue management.

**Options**

- **-ac complex_name**
  Adds a complex to the DQS environment. Complex entries contain one or more resources which may be requested by jobs submitted to the system. The entries will conform to the following three forms:

  - string
  - string =string
  - string =integer quantity

  There are no reserved words in complex lists. Thus the "string" may contain any combinations of alphanumeric characters and be up to 1023 bytes in length.

  This option requires root/manager privileges.

- **-acons consumable_name**
  Add a consumable to the DQS environment Consumable list. Each consumable is described by a four-line entry in an internal file with the consumable_name. An entry:

  Consumable consumable_name
  Available = <the amount of resources available>
  Consume_by <quantum by which resource is reduced by a request>
  Current = <currently available resources>

  The Current value starts out with the available value and is reduced or increased as resources are used and returned by executing jobs.

  This option requires root/manager privileges.

- **-ah server_list**
  Adds one or more hosts to the DQS host list. A host must reside in the DQS list to process any other DQS commands. The host on which the qmaster is running is automatically added to the host list. In addition host names are added when they appear in a newly created queue configuration. The host-name_list will conform to the following:

  hostname[,hostname,...]

  This option requires root/manager privileges.

- **-am user list**
  Adds one or more managers to the manager list.
The "user_list" option argument is described in the Glossary.

This option requires root/manager privileges.

-aq [queue_template]
Add A NEW QUEUE
Retrieves a default queue configuration, or the optional "queue_template" and executes the editor specified by the EDITOR environment variable to permit the customization of the queue configuration. As a minimum, the new queue must be assigned a unique queue name and a host.

Upon exit from the editor the queue is registered with the qmaster.

This option requires root/manager privileges.

-au  user_list| name_list
Adds one or more users to the DQS access list (ACL). The user is used for "quusage" authentication.

The "user_list" option argument is described in the Glossary.

This option requires root/manager privileges.

-Aq filename
Add New Queue From File
Adds a new queue description from an ASCII file. The default queue template, "generic_queue" may be found in the "common_dir" and used as a basis for preparation of the queue configuration file.

This option requires root/manager privileges.

-cell  cell_name
Specifies the cell to which the "qconf" request will be Sent.

-cq  destination identifier
Clean Queue(s)
Cleans out residual information from a queue where a job has been completed but for some reason the system has retained the job in the queue. This step is still necessary in DQS 3.3.2 due to some cases which the qmaster cannot verify job completion. It most
often occurs when the DQS manager is starting and terminating the qmaster and dqs_execd processes with a kill -9 while doing system testing. The "destination_identifier" option argument is described in the Glossary.

This option requires root/manager privileges.

**-dc complex_name**
Deletes the named complex.

This option requires root/manager privileges.

**-dcons consumable_name**
Deletes the named Consumable.

This option requires root/manager privileges.

**-dh server_list**
Deletes one or more hosts from the DQS host list. The host-name_list will conform to the following:

hostname[,hostname,...]

This option requires root/manager privileges.

**-dm user list**
Deletes one or more managers from the manager list.

The "user_list" option argument is described in the Glossary.

This option requires root/manager privileges.

**-do user list**
Deletes one or more operators from the operator list.

The "user_list" option argument is described in the Glossary.

This option requires root/manager privileges.

**-dq destination identifier**
Removes the specified queue(s).

The "destination_identifier" option argument is described in the Glossary.

This option requires root/manager privileges.
-du user list
Deletes one or more users from the DQS access list (ACL).

The "user_list" option argument is described in the Glossary.

This option requires root/manager privileges.

-help
Sends a listing of the qconf options to standard output

-mc complex_name
Modify Complex
Invokes the DQS editor and displays the named complex. When the editor is closed the complex is updated by the qmaster.

This option requires root/manager privileges.

-mf consumable_name
Modify Consumable
Invokes the editor which displays the entry for the named consumable. When the editor is closed the consumable values are updated. This action will be used in subsequent passes of the job scheduler however no RUNNING jobs will be affected.

This option requires root/manager privileges.

-mq queue_name
Modify Queue Configuration
Invokes the editor which displays the entry for the named queue. When the editor is closed the queue configuration is updated. This action will be used in subsequent passes of the job scheduler however no RUNNING jobs will be affected.

Note: the "qname" and "qhostname" fields CANNOT be changed with this option. Modification of these fields requires deletion of the queue definition with the "-d q queue_name" option.

This option requires root/manager privileges.

-Mq filename
Modify Queue Configuration using named file
The editor is invoked to apply the named file to the queue configuration. These modifications are then used in subsequent passes of the job scheduler. No running jobs will be affected.

To create the new file for use with this option, copy generic_queue from the DQS directory tree under conf/qmaster/qmaster_hostname/common_dir to a new file and make
the appropriate changes. Then execute qconf -Mq new_file to make the changes take affect.

Note: the "qname" and "qhostname" fields CANNOT be changed with this option. Modification of these fields requires deletion of the queue definition with the "-dq queue_name" option.

This option requires root/manager privileges.

-scl
  Show Complex List Names

-sc complex_list
  Show complexes

-sconf [cell]
  Show master configuration for cell

-scons
  Show the names of the consumables

-scons consumable_name
  Show the named consumable entry

-sh
  Show Hosts

-sm
  Show Managers

-so
  Show Operators

-sq [destination_identifier]
  Show queue configurations

  The "destination_identifier" option argument is described in the Glossary.

-sql
  Show list of "destination identifiers" (queue names)

-su name_list
  Show users

  The "name_list" option argument is described in the Glossary.
qdel- Delete on or more jobs from the system

qdel [-cell cell_name] [-f] [-help] [-verify] job_identifier_list

Description

A job is deleted by sending a request to the batch server that manages the job. A job that has been deleted is no longer subject to management by batch services. The "qdel" utility is user-accessible client of batch services that requests the deletion of one or more jobs.

A "qdel" utility requests a batch server to delete those jobs for which a job_identifier appears in the "job_identifier_list".

The "qdel" utility deletes the specified jobs in the order in which their job_identifiers appear in the "job_identifier_list".

If the "qdel" utility fails to process any job_identifier successfully, the utility will proceed to process the remaining job_identifiers, if any.

The "qdel" utility deletes each job by sending a Delete Job Request to the qmaster.

The "qdel" utility will not exit until the job corresponding to each successfully processed job_identifier has been deleted.

Options

-cell cell_name
The cell to which the "qdel" request is sent.

-f
Attempt to force a deletion even if the job has been handed to a host that cannot be reached. This does not guarantee that running processes are destroyed.

-help
Prints a listing of all options

-verify
Causes DQS to print the request to standard out. no action is taken.
Operands

job identifier list
Specifies the job number(s) of the jobs which are to be deleted. The job numbers were assigned by the batch server when the job was submitted.

The "job_identifier_list" operand argument is described in Glossary.

go to toc

qhold- Place a hold on one or more jobs to defer scheduling

qhold [-h hold_list] [-help] [-verify] [job_identifier_list]

Description

The "qhold" utility provides a means for a user to place one or more types of holds on identified job(s). A job that has one or more holds is not eligible for execution.

"qhold" places holds on jobs in the order in which their identifiers are presented to the utility, by sending a Hold Job Request to the batch server that manages the job.

If "qhold" fails to process any identified job successfully, the utility will proceed to process the remaining job identifiers, if any.

"qhold" will not exit until holds have been placed on the job corresponding to each successfully processed job identifier.

Options

-h hold_list
Defines the types of holds, if any, on the job. The "qhold -h" will accept a value for the "hold_list" option argument that is a string of alphanumeric characters in the portable character set. The "hold_list" option argument is described in the Glossary.

-help
Prints a listing of all options

-verify
Causes DQS to print the request to standard out. No action is taken.

Operands
**job identifier list**
Specifies a list of one or more job sequence numbers that are affected by the "qhold" utility.

If the application presents the server portion of a job identifier to the "qhold" utility, the utility will send the Hold Job Request to the specified server.

The "job_identifier_list" operand argument is described in Glossary.

**qidle- X Windows console monitor**

qidle [-d display] [-s seconds] [-g geometry]

**Description**

"qidle" monitors the local X server for activity, if a user is interactively using that host's console, the local DQS queue(s) is (are) automatically suspended. If the server is idle for a certain period of time, "qidle" re-enables the queue causing the suspended job to resume. The sleep button in the X-windows "qidle" icon allows the console user to modify the length of time for which the queue is suspended.

**Options**

- **-d display**
  Name of the local X display. "qidle" will use "unix:0" as a default display unless the environment variable "DISPLAY" is present. The "-d" option will override these choices for the display name.

- **-s seconds**
  Length of time that the queue is suspended before retrying an enable of the queue

- **-g geometry**
  X windows geometry for the "qidle" window, in the form used to describe location length and width of a window for X-windows applications.

**qmod- Modify the state of a queue**


**Description**
The "qmod" command enables users classified as owners of a workstation to modify the state of a DQS queue for their machine. A manager, operator or the super-user can execute "qmod" for any queue in their system.

**Options**

- **-cell cell_name**
The cell to which the request is sent.

- **-d**
Disables the queue

- **-e**
Enables the queue

- **-f**
Force a modification action for the queue despite the apparent current state of the queue. For example if a queue appears to be suspended but the job execution seems to be continuing the manager can force a suspend operation which will send a SIGSTOP to the job.

- **-help**
Send a list of all options to standard output.

- **-l resource list**
Redefines the resources that are allowed or required by the job.

The "resource_list" option argument is described in the [Glossary](#).

- **-q destination identifier**
Name of the queue(s) to be modified.

The "destination_identifier" option argument is described in the [Glossary](#).

- **-s**
Suspends the queue and any jobs in execution.

- **-soc**
Suspends the queue on completion of currently executing job.

- **-us**
Removes the suspend status from the queue and restarts any jobs which were running when the queue was suspended.

- **-verify**
Send the contents of the "qmod" request to standard output. No actions are taken.
-xsoc
Remove the suspend on completion mandate from the queue.

go to toc

qmove- Move jobs from one cell to another

qmove [-cell cell_name] job_identifier_list

Description
Move a job to <cell_name>. This function is a small subset of the POSIX "qmove" command. If a job must be moved from one queue to another within a cell, the "qalter" utility should be used.

Options
-cell cell_name
The destination cell for the moved jobs. If this option is absent, the cell within which this utility is executing will become the destination.

Operands
-job_identifier_list
Specifies a list of one or more job sequence numbers that are affected by the qmove command.

To use this utility, a user must be authenticated in both cells as well as being the owner of the jobs to be moved.

The "job_identifier_list" operand argument is described in Glossary.

go to toc

qrls- Release hold(s) on one or more jobs

qrls [-h hold_list] job_identifier

Description
A job might have one or more holds which prevent the job from executing. A job from which all the holds have been removed becomes eligible for execution, and is said to have been released.
A job hold is removed by sending a Release Job Request to the qmaster. The "qrls" utility is a user-accessible client of a batch server that requests holds to be removed from one or more jobs.

The "qrls" utility will remove one or more holds from those jobs for which a job_identifier is presented to the utility, in the order in which the job_identifiers are presented to the utility.

If the "qrls" utility fails to process a job_identifier successfully, the utility will proceed to process the remaining job_identifiers, if any.

The "qrls" utility will not exit until the holds have been removed from the job corresponding to each successfully processed job_identifier.

**Options**

-h hold list
Specifies the types of holds to be removed from the job.

The "hold_list" option argument is described in the Glossary.

**Operands**

job_identifier_list
Specifies a list of one or more job sequence numbers that are affected by the "qrls" command.

If the application presents the server portion of a job identifier to the "qrls" utility, the utility will send the Hold Job Request to the specified server.

The "job_identifier_list" operand argument is described in Glossary.

go to toc

**qstat- Display status of selected queues and jobs**


**Description**

The status of a job, queue, or batch server is obtained by a request to the qmaster. The "qstat" utility is a user accessible batch client that requests the status of one or more jobs, queues, or servers, and writes the status information to standard output.
For each successfully processed destination, the "qstat" utility will display information about the corresponding queue.

For each successfully processed server name, the "qstat" utility will display information about the corresponding server.

For each successfully processed resource request, the "qstat" utility will display information about the queues that possess resources that match those specified.

For each successfully processed user id, the "qstat" utility will display information about the jobs owned by that user.

The "qstat" utility acquires all job and queue status information in a single request to the batch server/qmaster. Server information is displayed for each queue with one of the following status symbols

RUNNING - The queue is in the running state or a job is RUNNING.
ENABLED - The queue is enabled for scheduling jobs.
DISABLED - The queue is disabled and will not be included in scheduling.
SUBORDINED - The queue has been subordinated by a "superior" queue.
SUSPENDED - The queue any running jobs in that queue are suspended.
UNDEFINED - The queue's dqs_execd has not reported in to the qmaster.
ALARM - The load average threshold for the queue has been exceeded.

Options

-cell cell_name
The cell to which the "qstat" request is to be forwarded.

-ext
Specifies that an extended description of queues and jobs are to be sent to standard output

-f
Specifies that a "full format" output format is to be used. This format shows the full queue status and list of jobs in the queues, but not the queue and job descriptions provided with the "-ext" option.

-help
Sends a list of all "qstat" options to standard output
-l resource list
This option allows users to define which resources queues must satisfy in order to show up under the "qstat" listing.

The "resource_list" option argument is described in the Glossary.

Examples:

qstat -l ibms
displays all the queues of group "ibms"

qstat -l mem.gt.256
displays all the queues having mem > 256

-q destination identifier
Specify the destination identifier i.e. queue name(s), that "qstat" should pertain to.

The "destination_identifier" option argument is described in the Glossary.

-u user list
Specify the user(s) that "qstat" should pertain to.

The "user_list" option argument is described in the Glossary.

-verify
Causes the contents of the "qstat" command to be sent to standard output, no status is requested or displayed.

Standard Output

If the -f option is not specified, the utility will display the following items on a single line in the stated order, with white space separators, for each successfully processed operand:

-- queue name
-- job identifier
-- job name
-- job owner
-- job state
-- job location
If the -f option is specified, the utility will display the following items for each successfully processed operand:

-- queue name
-- type of queue (execution or routing)
-- maximum number of concurrent jobs permitted
-- total number of jobs in queue
-- enabled or disabled status of the queue
-- started or stopped status of the queue
-- job identifier
-- job name
-- job owner
-- job state
-- job location
-- why a pending job has not been scheduled

go to toc

qsub- Submit a job to the queuing system


Description

To submit a script is to create a job that executes the script. A script is submitted by a request to a batch server. The "qsub" utility is a user accessible batch client that submits a script.
Upon successful completion, the "qsub" utility shall have created a job that will execute the submitted script.

The "qsub" utility shall submit a script by sending a Queue Job Request to a batch server.

The "qsub" utility shall place the value of the following environment variables in the Variable-List attributes of the job:

- HOME
- LOGNAME
- PATH
- MAIL
- SHELL
- TZ

The name of the environment variable shall be the current name prefixed with the string "DQS_O_".

In addition to the variables described above, the "qsub" utility shall add the following variables with the indicated values to the variable list:

- **--DQS_O_WORKDIR** The absolute path of the current working directory of the "qsub" utility process.

- **--DQS_O_HOST** The name of the host on which the "qsub" utility is running.

### Summary of DQS environment variables

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Value at qsub Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>DQS_O_HOME</td>
<td>$HOME</td>
</tr>
<tr>
<td>DQS_O_HOST</td>
<td>client host name</td>
</tr>
<tr>
<td>DQS_O_LOGNAME</td>
<td>$LOGNAME</td>
</tr>
<tr>
<td>DQS_O_PATH</td>
<td>$PATH</td>
</tr>
<tr>
<td>DQS_O_MAIL</td>
<td>$MAIL</td>
</tr>
<tr>
<td>DQS_O_SHELL</td>
<td>$SHELL</td>
</tr>
<tr>
<td>DQS_O_TZ</td>
<td>$TZ</td>
</tr>
<tr>
<td>DQS_O_WORKDIR</td>
<td>current working directory</td>
</tr>
</tbody>
</table>

### Options
-a **date time**
Defines the time at which the job becomes eligible for execution.

The "date_time" option argument is described in the *Glossary*.

The "qsub" utility sets the Execution_Time attribute of the job to the number of seconds since Epoch that is equivalent to the local times expressed by the value of the "date_time" option argument. Specifying a date_time option argument that represents a time (number of seconds since Epoch) earlier than the time at which the utility exists will have the same effect on job execution as if the "-a" option had not been presented to the utility.

-A **account string**
Defines the account to which the resource consumption of the job should be charged.

The "account_string" option argument is described in the *Glossary*.

-cell **cell_name**
The cell to which the "qsub" request is sent.

-cwd
Declares that the job should **begin** execution in the current working directory, rather than in the user's home directory. Obviously this directory must exist on all potential hosts. Inside your job script, you can, of course, set the current directory using the "cd" command, again with the caveat that the named directory must exist on any potential execution host.

Obeying "-cwd" takes place as follows:

1. If execution takes place on the same machine that submitted the job, just "cd" to the submitting directory.
2. If executing on a different machine, then
   a. attempt to automount the submitting directory
   b. use a local directory with the exact same absolute path as the submitting directory
3. If neither option of #2 works, the job silently fails.

-C **directive_prefix**
The directive prefix consists of two ASCII characters which when appearing in the first two bytes of a script line indicate that what follows is a DQS command. Default is "#?"

-e **path_name**
Defines the path to be used for the standard error stream of the job.

If the "path_name" option argument constitutes an absolute path name, the "qsub" utility will set the Error_Path attribute of the job to the value of the "path_name" option argument including the hostname, if present.
If the "path_name" option argument constitutes a relative path name and no hostname is specified, the "qsub" utility will set the Error_Path attribute of the job to the value of the absolute path name. The absolute path name is derived from the relative path name by expanding the "path_name" option argument relative to the current working directory of the process that executed the "qsub" utility.

If the "path_name" option argument constitutes a relative path name and a hostname is specified, the "qsub" utility will set the Error_Path attribute of the job to the value of the option argument without expansion.

If the "path_name" option argument does not include a hostname, the "qsub" utility will prefix the path name in the Error_Path attribute with "hostname:", where hostname is the name of the host upon which the "qsub" utility is being executed.

The default file name for standard error output has the following form:

"job_name.e.sequence_number.pid"

If the "-e" option possess no option argument, the utility will clear the Error_Path attribute of the job so that the default standard error output will be used when the job is executed.

-F
Forces the submission of the job to the queuing system despite the fact that a requested resource (-l option) does not exist in any queue at the time of job submission. This is provided where the user anticipates a resource to be introduced into the system at a later time, and wishes to queue their job to wait for that resource.

-h
Place a "USER" hold on the job. A job that has one or more holds is not eligible for execution. See the "qhold" command.

-hard
Signifies that all resource requirements following in the "qsub" command will be hard requirements and must be satisfied in full before a job can be scheduled.

-help
Prints a listing of all options for "qsub".

-hold_jid jid[,jid,....]
Do not schedule this job until all the cited "jid's" finish.

-j  join list
Redefines which streams of the job are to be merged. The "qsub -j" option will accept a value for the "join_list" option argument that is a string of alphanumeric characters in the portable character set. The "join_list" option argument is described in the Glossary.

The only characters permitted in this implementation are "y" or "n". Specifies whether the standard error stream of the job is merged into the standard output stream. The meaning of the "-j" option is different from the standard.

If the value of the option argument is 'y' the Join_Path attribute of the job is set to TRUE, if the value is 'n' the Join_Path attribute of the job is set to FALSE.

If both the "-j y" and the "-e" options are specified, the batch server will set but ignore the Error_Path attribute.

If the "-j" option is not presented to the "qsub" utility, the utility will set the value of the Join_Path attribute to FALSE.

-l resource_list
Defines the resources that are allowed or required by the job.

The "resource_list" option argument is described in the Glossary.

-m mail_options
Defines the points in the execution of the job at which the server is to send mail about a change in the state of the job.

The "mail_options" option argument is described in the Glossary.

-master parallel_destination_identifier
Defines the queue to serve as a master queue for a parallel job.

The "parallel_destination_identifier" option argument is described in the Glossary.

-M mail_list
Defines the list of users to which the server that executes the job is to send mail, if the server sends mail about the job.

mail_address[,mail_address,...]
(Default- the login name of the user submitting the job)

NOTE: The interpretation of mail_address is implementation defined.

-N name
Defines the name of the job. The name can be any printable set of characters up to 8 chars long. Names in excess of 8 characters will be truncated.
The "qsub" utility will set the Job_Name attribute of the job to the value of the name option, truncated to eight characters, if necessary.

If the "-N" option is not presented to the "qsub" utility, the utility will set the Job_Name attribute of the job to the name of the Script_File argument after any directory "path-name" has been removed. If the script is read from standard input, "qsub" will set the job-name attribute of the job to the value STDIN.

-notify
Defines the Notify attribute for a job. This attribute, when TRUE, causes the batch server to send "warning" signals to a running job prior to sending the signals themselves. If a SIGSTOP is pending the job will receive a SIGUSR1 several seconds before the SIGSTOP. If a SIGKILL is pending the job will receive a SIGUSR2 several seconds before the SIGKILL. The amount of time delay is controlled by the "notify" parameter in each queue configuration.

-o path_name
Redefines the path for standard output of the job.

If the "path_name" option argument constitutes an absolute path name, the "qsub" utility will set the Output_Path attribute of the job to the value of the "path_name" option argument including the hostname, if present.

If the "path_name" option argument constitutes a relative path name and no hostname is specified, the "qsub" utility will set the Output_Path attribute of the job to the value of the absolute path name. The absolute path name is derived from the relative path name by expanding the "path_name" option argument relative to the current working directory of the process that executed the "qsub" utility.

If the "path_name" option argument constitutes a relative path name and a hostname is specified, the "qsub" utility will set the Output_Path attribute of the job to the value of the option argument without expansion.

If the "path_name" option argument does not include a hostname, the "qsub" utility will prefix the path name in the Output_Path attribute with "hostname:", where hostname is the name of the host upon which the "qsub" utility is being executed.

The default file name for standard output has the following form:

job_name.o.sequence_number.pid

If the -o option possess no option argument, the utility will clear the Output_Path attribute of the job so that the default standard error output will be used when the job is executed.
-p priority
Defines the priority of the job relative to other jobs.

"qsub" will accept a value for the priority option argument that conforms to the syntax for signed decimal integers, and which is not less than -1024 and not greater than 1023.

-par parallel paradigm
DQS can process, monitor and initiate parallel jobs using one of several message passing system paradigms. This option defines the paradigm to use for a job. The "parallel_paradigm" option argument is described in the Glossary.

-passwd
Specifies that a password list will be requested for AFS and/or DFS re-authentication. If given, DQS will utilize the password list to authenticate the process prior to executing the job, so that re-authentication will be unnecessary.

-passwd_file filename
Defines the name of the file containing the AFS passwords. (See AFS support in the User Guide.)

-q destination identifier
Assigns the destination identifier, i.e. queue name(s), for the job.

DQS attempts to assign the host based on a first to last traversal of the "destination_identifier" list. If the "destination_identifier" list is specified as a "soft" request and DQS is unable to fill the request based on the respective "destination_identifier" list, DQS will assign a host as available.

The "destination_identifier" option argument is described in the Glossary.

-r y | n
Defines whether the job is Rerunable.

If the value of the option argument is 'y', "qsub" will set the Rerunable attribute of the job to TRUE, if the option argument is 'n', the Rerunable attribute will be set to FALSE.

The "qsub" utility will consider it an error if any character other than 'y' or 'n' is specified in the option argument.

-reauth seconds
Specifies the time interval for AFS/DFS re-authentication.

-soft
Signifies that all resource requirements following in the "qsub" command will be soft requirements and will be filled on a "as available" basis before a job can be scheduled.
-S path_name_list
Defines the shell that interprets the script at the destination system.

"qsub" accepts a "path_name_list" option argument that conforms to the following syntax:

path-name[@host],path-name[@host],...

The "qsub" utility will accept only one path name that is missing a corresponding host name. The "qsub" utility will allow only one path name per named host.

"qsub" adds a value to the Shell_Path_List attribute of the job for each entry in the "path_name_list" option argument.

If the -S option has no arguments the utility will set the path-list attribute of the job to the null string.

-v variable_list
Defines the environment variables to be exported to the job.

The "variable_list" option argument is described in the Glossary.

If the -v option is presented to the "qsub" utility, the utility will also add to the environment_variable_list attribute of the job every variable named in the "variable_list" option argument and, optionally, values of specified variables.

-verify
This does not affect any job attributes. It causes the "qsub" utility to send the contents of the "qsub" command to standard out so it may be verified by the user.

-V
Specifies that all environment variables be exported to the context of the job.

Script File
The path to the script of the job. The "Script_File" operand argument is described in the Glossary.

Defining the Directive Prefix
The "qsub" utility will select the directive prefix from one of the following sources:

-- if the -C option is presented to the utility, the value of the directive_prefix option argument.

-- if the environment variable POSIX2_DPREFIX is defined, the value of that variable.
If the -C option is present in the script file, it will be ignored.

The daemons

The backbone of DQS is the operation of the qmaster and the dqs_execd daemons. Each host which will execute batch jobs will have a dqs_execd running as a background daemon. The qmaster runs as a background daemon on each host functioning as a DQS cell manager. The dqs_execd daemons communicate only with the qmaster except the DSH utility is invoked by a master process during parallel job startup. The qmaster handles all requests from DQS utilities and communicates with the dqs_execd daemons in its own cell and with other qmaster daemons.

qmaster
dqs_execd

Glossary

account string

The account string is a list containing account_names as follows:

Account_Name[@cell][,Account_Name[@cell],[...]]

The Account_Name may be any arbitrary ASCII alphanumeric string of bytes but may contain no blanks or separator characters. The under-bar "_" is considered a non-separator.

date time

The date_time option argument is of the form:

[[CC][Y][Y][M][M][D][D][H][H][M][M][S][S]]

where:

CC  Specifies the first two digits of the year
YY Specifies the last two digits of the year
MM Specifies the month of the year (01 to 12)
DD Specifies the day of the month (01 to 31)
hh Specifies the hour of the day (00 to 23)
mm Specifies the minute of the hour (00 to 59)
ss Specifies the number of seconds in the hour (00 to 59)
(note the decimal point must be present to flag the ss)

destination identifier

The destination_identifier(_list) option argument conforms to the following:

destination_id  "queue" "queue@cell"
destination_id_list  destin_id[,destin_id,...]

parallel destination identifier

For parallel jobs, the destination_identifier can have an additional component:

destination_id_list[-exec.eq. exec_str]

The string "-exec.eq." is reserved for "generic" parallel job execution. At the moment of job initiation the optional execution string will be used to startup a user supplied program.

For example:

qsub ... -master myqueue -exec.eq. "parallel N2 params"

This entry requests "myqueue" as the queue for the master process. DQS will execute the program "parallel" and pass to it the parameters "N2" and "params".

DQS attempts to assign the master host based on a first to last traversal of the destination_identifier list. If the destination_identifier list is specified as a "soft" request and DQS is unable to fill the master request based on the respective destination_identifier list, DQS will assign a master as available.

The exec_str may contain DQS reserved "wildcards" which will be expanded to provide dynamic information to the startup program. The reserved "wildcards" are:
%execer_id Set to DQS

%master_host Set to the master's hostname at random

%my_hostname Set to my hostname at random

%my_nodenum Set to my node number at random

%total_numnodes Set to the total number of nodes at random

%remote_info Set to remote exec_str at random

Using the previous example with a wildcard:

qsub ... -master myqueue -exec.eq. "parallel %total_numnodes params"

---

**hold list**

Utilities accept a value for the "hold_list" option argument that is a string one or more characters 'u' 's' 'o' or the single character 'n'.

For each unique character in the "hold_list", the utility will add a value to the Hold_Type attribute of the job as follows, each representing a different hold type:

- u USER
- s SYSTEM
- o OPERATOR
- n NO_HOLD

It is considered an error if any hold type other than 'n' is combined with hold type 'n'.

---

**job identifier list**

sequence_number[@cell][,sequence_number[@cell],...]

If the application omits the cell portion of a job identifier, the utility will use either:

- the name of the cell specified on the command line, if any
- the name of the cell specified in the environment variable DQS_CELL, if it is set
-- the name of the default cell

**join list**

The utility will accept a "join_list" option argument that consists of one or more characters 'e' and 'o' or the single character 'n'.

All other job output streams specified will be merged into the output stream represented by the character listed first in the "join_list" option argument.

For each unique character in the "join_list" option argument, the utility will add a value to the Join_Path attribute of the job as follows, each representing a different of stream to join:

- The standard error of the batch job JOIN_STD_ERROR
- The standard output of the batch job JOIN_STD_OUTPUT

An existing Join_Path attribute can be cleared by the job type:

- NO_JOIN

If 'n' is specified, then no files are joined. The utility will consider it an error if any type other than 'n' is combined with the join type 'n'.

If the value of the option argument is 'y' the Join_Path attribute of the job is set to TRUE, if the value is 'n' the Join_Path attribute of the job is set to FALSE.

If both the "-j y" and the "-e" options are specified, the batch server will set but ignore the Error_Path attribute.

If the "-j" option is not presented to the utility, the utility will set the value of the Join_Path attribute to FALSE.

**mail options**

The utility will accept a value for the "mail_options" option argument that is a string one or more of the characters:

- MAIL_AT_EXIT
- MAIL_AT_BEGINNING
- MAIL_AT_ABORT
s MAIL_AT_SUSPENSION

If any of these characters are duplicated in the "mail_options" option argument, the
duplicates will be ignored.

n NO_MAIL

If 'n' is specified then mail is not sent. The utility will consider it an error if any mail type
other than 'n' is combined with mail type 'n'.

---

parallel paradigm

The "paradigm" can be one of the following:

p4 - support for ANL p4 language interface

mpi - support for the standard MPI interface

generic_all - a simple parallel interface

pvm - The ORNL PVM interface

tcgmsg - The Argonne / PNL interface

generic_sla - ???

(See parallel job execution in the User Manual.)

---

resource list

The "resource_list" option argument will conform to the following:

resource[relational-op resource-field[logical-op resource[relational-op]]...

The "resource_list" should not contain ANY embedded blanks, as the space character is
used to delimit each of the option fields in commands.

There are two character strings which act as "reserved words" in the "resource_list":

qty The quantity of a requested resource, default=1.

-exec.eq. Identifies special parallel resource definitions. See Parallel Job Submission in
the User Guide.
Each relational-op may be selected from the FORTRAN or "C" list of operators, which consists of
.eq. .ne. .lt. .le. .or. ‘==‘ ‘!=‘ ‘>‘ ‘<‘ ‘>=‘ ‘<=‘

A special relationship is flagged when a single equal-sign is used '='.
This indicates that a comparison of character strings will be used in the relational test rather than a numeric comparison.

Each logical-op may be selected from the FORTRAN or "C" list of operators, which consists of
.and. .or. .not. or ‘&&‘ ‘||‘ ‘!’

The "logical-op" may be preceded or followed by an opening or closing parenthesis to ease readability. For example "(mem.gt.32).and.(disk.gt.100)"

---

**Script File**

If the script is not presented to the utility, or if the specified operand is the single character '-' (minus), the utility will read the script from standard input.

If the script argument represents a partial path, the utility will expand the path relative to the current directory of the process executing the utility.

If the utility is presented with a script file it will replace the job script file originally submitted with the "qsub" command. The utility scans the script file interpreting lines flagged with the DQS_DELIMITER pair of characters. All command options in the command line and the script file are processed and replace or modify job attributes for the job.

A script can contain directives to the utility.

The utility will scan the lines of the script for directives until the first line that begins with a string other than the directive string; if directives occur on subsequent lines, the utility will ignore those directives.

The utility will process a line in the script as a directive if, and only, if, the string of characters from the first non-white space on the line until the first space or tab character on the line match the directive prefix.

The utility will ignore any blank lines and shell comment lines prior to the first executable line.

The utility will process the options and option arguments contained on the directive prefix line using the same syntax as if the options were input on the command line.
The utility will continue to process a directive prefix line until after a `<newline>` or an non-quoted comment character of the shell that will interpret the script is encountered.

If an option is present in both a directive and arguments to the utility, it will ignore the option and the corresponding option argument, if any, in the directive.

If an option that is present in the directive is not present in the arguments to the utility, the utility will process the option and the option argument, if any.

In order of preference, the utility will select the directive prefix from one of the following sources:

-- if the `-C` option is presented to the utility, the value of the directive_prefix option argument.

-- if the environment variable `POSIX2_DPREFIX` is defined, the value of that variable.

If the `-C` option is present in the script file, it will be ignored.

user list

The "user_list" option argument will conform to the following:

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

or

ame,[name,...]

variable list

A variable_list is a set of strings delimited by commas. A string will be of either of these forms:

variable

or

variable = value  (with no intervening spaces)

A variable cannot be repeated in the environment variable_list option argument.
The Distributed Queuing System (DQS) is an experimental batch queueing system which has been under development at the Supercomputer Computations Research Institute (CSIT) at Florida State University for the past 9 years. The first years of this activity were funded by the Department of Energy Contract DE-FC0585ER250000. DQS is freely distributed to all parties with the understanding that it continues to be an evolving development system, and no warranties should be implied by this distribution.
DQS is intended to provide a mechanism for the management of requests for execution of batch jobs on one or more members of a homogeneous or heterogeneous network of computers. Facilities for load-balancing, prioritization and expediting of a wide variety of computational jobs are included to assist each site in tailoring the behavior of the system to their particular environment.

CSIT support

CSIT will make every effort, within its resources to assure that DQS is suitable for operation as a batch queuing system in as many site situations as possible. CSIT staff will respond to requests for assistance as well as investigating bugs, incorporating repairs and updating documentation, from those who are utilizing DQS. However it is not possible, at this time, to make a formal commitment for the long term support and enhancement of this system. Any user or organization which decides to adopt DQS will be assuming all risks from that undertaking.

DQS and future enhancements can be obtained by Internet ftp from "ftp.csit.fsu.edu".

Announcements of new releases and improvements will be emailed to anyone who contacts CSIT to add their name to the announcement list dqs-announce. This is done by filling out the online form at URL:

http://mailer.csit.fsu.edu/mailman/listinfo/dqs-announce

A name can be removed from the announcement list by visiting the same online form and using the "Edit Options" selection with your email address:

Bug reports should be sent to: dqs@csit.fsu.edu

DQS user information exchange is provided by Rensselaer Polytechnic Institute. To add your name and email address to this list:

Send email to dqs-l@vm.its.rpi.edu

Leave the "subj:" line blank

Send a one line message: SUBSCRIBE dqs-l 1stname Lastname

To remove name and email address:
What's New in DQS 3.3.1

The release of DQS 3.0 was a major departure for the DQS evolution. It was based on several years' experience with DQS 2.1 in a variety of computing environments. Although it retained many features of the 2.1 version, DQS 3.0 was a major restructuring and re-coding of the basic system with a major focus on supporting parallel (clustered) computation on two or more UNIX based hardware platforms. The newly emerging message passing scheme (MPI) was considered throughout the DQS 3.0 implementation.

In early 1995 DQS 3.0-3.1 was subjected to extensive testing and the contributions of numerous users were incorporated to produce DQS 3.1.2 which was released in March and augmented over a period of six months to become DQS 3.1.2.4. With the exception of some minor "improvements this system has been fairly stable and in operational use for nine months.

Operational experience at CSIT and other large production sites revealed several features which needed to be added or adapted to make the system easier to use or to manage. Several sites provided the DQS development team with valuable insight, advice and code which has been incorporated into this new release. Although all user interfaces have not been changed (albeit "enhanced") the internals of this system have undergone considerable change, hence the naming of this release as 3.3.1 instead of 3.1.2.5. We took this opportunity to restructure the documentation (one more time!) in response to numerous requests to make it easier to access. In addition to numerous bug-fixes for DQS 3.1.2.4 provided by several very helpful sites (see "acknowledgments") a number of new features have been added to the system.
The "new" features of DQS 3.3.1 tend to be somewhat invisible to the DQS user. The bulk of this effort has been focused on further "bulletproofing" the system to minimize, if not eliminate, the unreported termination of daemons, utilities and jobs. Some features are "semi-visible" such as the revised scheduling system. A few are quite evident to all, as the "job pre-validation" feature returns immediate feedback on the complete absence of a requested resource. With this in mind we list here the major changes which appear in DQS 3.3.1:

**CPU Job Limits**

New hard and soft CPU Job Limits provide for limiting the resource utilization of all processes that make up a job.

**Year 2000 Compliance**

DQS is now Year 2000 ready.

**System Log Interface**

Critical errors are now logged to both the DQS log files and the System Log.

**Decimal Notation For Queue Configuration**

The queue configuration parameters are now entered in decimal notation instead of hexadecimal when you execute the qconf331 program. They are also displayed in decimal with qstat331 -ext.

**Many Memory Leaks Fixed**

There were a number of memory leaks in the qmaster and dqs_execd that were fixed. These leaks were in library code that also is used by the ancillary programs.

**Protocol Modified**

The internal protocol was modified. This is a major departure from 3.2.7 because the 3.2.7 protocol is incompatible with the 3.3.1 protocol. Backward compatibility was built in so the 3.3.1 daemons can start up on 3.2.7 configuration files. The files will be immediately modified for the new protocol and will become incompatible with the 3.2.7 daemons.
Environment Overrides for log_file and err_file

The log_file and err_file names can now be overridden by specifying the new value as an environment variable. The new variables names are LOG_FILE and ERR_FILE.

New make clean target

A new target, clean, was added to the makefiles.

Documentation

The DQS 3.3.1 Documentation was reorganized...again. The POSIX specification has been extricated from the document body and is now an Appendix. The reference manual pertains only to the DQS 3.3.1 implementation and all confusing references to "standard" and "non-standard" options removed.

The documentation consists of three principle chapters and three appendices. The Installation and Maintenance Manual is primarily aimed at the DQS system administrator. The User Guide is obviously targeted at the DQS user community. The Reference Manual will be accessed by both users and administrators. Appendix A contains a catalog of all DQS error messages with information on methods for dealing with the error. Appendix B contains the POSIX specification on which DQS 3.3.1 is based. Appendix C contains several miscellaneous sections, including installation variants and system tuning guidelines.

The documentation is supplied in several forms:

a. Microsoft WORD 97
b. PostScript
c. HTML format (can be viewed with any of the commercial WEB browsers).

DQS Installation

DQS is designed to be installed on almost every existing UNIX platform. This process thus must cope with many differences and idiosyncrasies of the varied hardware configurations and operating systems. DQS 3.3.1 attempts to detect and resolve these differences to minimize the need for operator actions, but with even the simplest installation there will be a need for some input from the DQS administrator.
Obtaining DQS 3.3.1

DQS 3.3.1 can be obtained by ftp download from ftp.csit.fsu.edu/pub/dqs. The README.331 file in that directory will indicate which version should be downloaded. To reduce download bandwidth, improvements and big-fixes will be distributed on a file-by-file replacement basis rather than requiring a complete download of the DQS 3.3.1 system. For this reason we do not envision distributing systems such as DQS 3.3.1.1 in the future. (But you never know.)

Setting up for installation

DQS 3.3.1 is distributed as a compressed TAR file. After this file is uncompressed it is recommended that the DQS system be extracted (with TAR) into a directory which is accessible by all operating systems for which DQS will be built. The DQS installation process will create a separate directory in the subdirectory DQS/ARCS for each different architecture/operating system.

Installation

The installation scripts produce a list of defaults which will be used for the installation. The user is asked to review this list to ensure that it meets their requirements. The default-cell name and default initial queue name are derived from the host-name of the machine on which the installation process is being executed. If the installation is being executed as "root" the system will be setup to use "reserved" ports for communication, otherwise "non-reserved" ports will be utilized. If the installation is being run as non-root then the user doing the "make install" will be automatically added as a manager.

The installation proceeds in stages, as follows:

a. Run the command "make config". The DQS config program then asks the user to provide a base directory to use for the installation of DQS binaries, libraries and documentation as well as the DQS configuration and resolve files and directories. The default paths offered by the dialogue are based on the current working directory (if running as non-root) or /usr/local//DQS (when running as root). This latter path is commonly used at DQS sites as all hosts of a common architecture often share the path "/usr/local". The simple install will only request one starting point for building a DQS331 tree. If the administrator wishes to differentiate the
various components, binaries, libraries, spool directories, etc. They can type "CUSTOM" when asked to enter an alternative base path.

b. Run the command "make". The next step invokes the make operation to create all of the DQS 3.3.1 executables. The binaries are placed in a subdirectory within the ../DQS/ARCS directory named for the specific platform being built. This provides a separate repository for each type of host system in the cluster.

c. Run the command "make installall". This step moves the binaries to the directory from which they will be executed. This process renames the executables by placing a tag "331" at the end of each name. This step also moves the sample conf_file and resolve file to the conf directory. This is done to differentiate these binaries from other DQS versions which might have preceded the DQS331.

d. The next step involves the addition of the three DQS 3.3.1 entries to the /etc/services file on one or more hosts. This step must be done with root permission and by someone familiar with UNIX system administration knowledge. While DQS attempts to identify proper port numbers to be used in the /etc/services file, local conditions may dictate another choice. Upon successful completion of the installation the administrator can proceed to "Testing the Installation". If error messages appear and the installation is aborted the administrator should refer to "Solving Installation Problems".

e. Finally the administrator should proceed to the step "Testing the DQS331 system.

An optional approach is available to the knowledgeable DQS administrator which omits all interaction. This requires the editing of three DQS files used during the make process. Details for this approach may be found in the Appendix C Miscellaneous "Key System Variables and Manual Installation".

The Graphical Interface

The X-window based DQS graphical interface is installed as a separate step. Change directory to DQS/XSRC and follow the instructions provided in the file named INSTALL. The X-Window interface is being restructured and will be integrated fully in future DQS releases.

Testing the installation

The installation process creates a series of directories and subdirectories and two crucial files, the "conf_file" (configuration file) and the "resolve_file". If the system installation was
completed correctly the conf_file will contain information which will be read by every DQS binary file when it is started. This includes the DQS daemons, qmaster and dqs_execd, and the DQS interface "utilities" qsub, qdel, qmod, qconf, qstat, qrls, qhold and qmove. It is best that these two files are accessible through an NFS/AFS/DFS file cross-mounting. If that is not possible then the administrator must ensure that identical copies of these files are present on each host.

Once the binaries have been moved to their execution directory (we will use the path /usr/local/DQS/bin" for all future examples), the qmaster can be started. If during the installation process the administrator chose "FALSE (NO)" when asked the question "Reserved ports?", then the /etc/services file will have been updated (by a root user) with the three entries suggested by the config process (or a rational alternative). The conf_file will contain the names of these entries along with the DEFAULT_CELL name which must match the first entry on the first (non-commented) line in the resolve file. The administrator should make a visual check of these three crucial files, conf_file, resolve_file and /etc/services to make sure that they conform to these requirements.

QMASTER

<The qmaster manages all resources for a single DQS cell.>

Once satisfied that all is well the qmaster can be started by typing
"/usr/local/DQS/bin/qmaster331. On this first occasion, it would be useful to check that the process has actually started by viewing the UNIX process status (ps). If the qmaster name does not appear in the hosts process list, the administrator should check the "err_file" in the qmaster spool directory (chosen during the DQS config stage-default: " /usr/local/DQS/common/conf")).

If the qmaster appears to be operating, it can be tested by executing the command
"/usr/local/DQS/bin/qstat331 -f", on the same host where the qmaster331 is running A normal response to this command would be one or more lines of output describing the status of the current queues. For brand new installations this will be simply a header with no other lines. Error messages may appear if things are not quite "in harmony", refer to "DQS Error Messages" and "Solving Installation Problems: for assistance in this case.

DQS_EXECED

<The dqs_execed is a DQS daemon which resides on each host which has at least one queue and will be executing DQS managed jobs.>

If the "qstat331" command succeeds, it is time to start a dqs_execd, which actually manages a particular queue. For this test, on the same host where the qmaster "dwelleth" type the command "/usr/local/DQS/bin/dqs_execd331". Again the UNIX process status should be examined (ps). If the dqs_execd is not executing, refer to the err_file for
significant error messages. Consult "DQS Error Messages" and "Solving Installation Problems: for assistance.

Executing the command "qconf -aq" (queue configuration, add queue) will produce an edit session with the default editor on that host. If the "qconf" command yields an error message and shuts down, consult "Solving Installation Problems". A queue "template" will be displayed which can be modified using the editor commands. For this test the queue name, and queue host name should be changed to match the name of the host on which the dqs_exed is executing. We will deal with the remaining entries later (see .The Queue Configuration).

<table>
<thead>
<tr>
<th>Q_name</th>
<th>ibm11</th>
</tr>
</thead>
<tbody>
<tr>
<td>hostname</td>
<td>ibm11.csit.fsu.edu</td>
</tr>
<tr>
<td>seq_no</td>
<td>0</td>
</tr>
<tr>
<td>load_mass</td>
<td>1</td>
</tr>
<tr>
<td>load_alarm</td>
<td>175</td>
</tr>
<tr>
<td>priority</td>
<td>0</td>
</tr>
<tr>
<td>type</td>
<td>batch</td>
</tr>
<tr>
<td>rerun</td>
<td>FALSE</td>
</tr>
<tr>
<td>quantity</td>
<td>1</td>
</tr>
<tr>
<td>tmpdir</td>
<td>/tmp</td>
</tr>
<tr>
<td>shell</td>
<td>/bin/csh</td>
</tr>
<tr>
<td>klog</td>
<td>/usr/local/bin/klog</td>
</tr>
<tr>
<td>reauth_time</td>
<td>6000</td>
</tr>
<tr>
<td>last_user_delay</td>
<td>0</td>
</tr>
<tr>
<td>max_user_jobs</td>
<td>4</td>
</tr>
<tr>
<td>notify</td>
<td>60</td>
</tr>
<tr>
<td>owner_list</td>
<td>NONE</td>
</tr>
<tr>
<td>user_acl</td>
<td>NONE</td>
</tr>
<tr>
<td>xuser_acl</td>
<td>NONE</td>
</tr>
<tr>
<td>subordinate_list</td>
<td>NONE</td>
</tr>
<tr>
<td>complex_list</td>
<td>NONE</td>
</tr>
<tr>
<td>consumables</td>
<td>NONE</td>
</tr>
<tr>
<td>s_rt</td>
<td>2147483647</td>
</tr>
<tr>
<td>h_rt</td>
<td>2147483647</td>
</tr>
<tr>
<td>s_cpu_job</td>
<td>2147483647</td>
</tr>
<tr>
<td>h_cpu_job</td>
<td>2147483647</td>
</tr>
<tr>
<td>s_cpu</td>
<td>2147483647</td>
</tr>
<tr>
<td>h_cpu</td>
<td>2147483647</td>
</tr>
<tr>
<td>s_fsize</td>
<td>2147483647</td>
</tr>
<tr>
<td>h_fsize</td>
<td>2147483647</td>
</tr>
<tr>
<td>s_data</td>
<td>2147483647</td>
</tr>
<tr>
<td>h_data</td>
<td>2147483647</td>
</tr>
<tr>
<td>s_stack</td>
<td>2147483647</td>
</tr>
<tr>
<td>h_stack</td>
<td>2147483647</td>
</tr>
<tr>
<td>s_core</td>
<td>2147483647</td>
</tr>
<tr>
<td>h_core</td>
<td>2147483647</td>
</tr>
<tr>
<td>s_rss</td>
<td>2147483647</td>
</tr>
<tr>
<td>h_rss</td>
<td>2147483647</td>
</tr>
</tbody>
</table>
When the queue name and queue host name are modified, exit the editor in the normal manner (ESC-ZZ for vi or CTRL-X CTRL-C for emacs). This will trigger the qconf utility to parse the submitted definition and, if no syntactical errors are discovered will create the requested queue.

<table>
<thead>
<tr>
<th>Queue Name</th>
<th>Queue Type</th>
<th>Quan</th>
<th>Load</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibms30</td>
<td>batch</td>
<td>0/1</td>
<td>0.10</td>
<td>DISABLED</td>
</tr>
</tbody>
</table>

Note that the status entry in the right column of the qstat output will display the word "DISABLED". All new queues are initiated in DISABLED state. To enable the queue we need to invoke another DQS command "/usr/local/DQS/bin/qmod331 -e <queue name>" (modify queue, enable the queue name given here as <queue name>).

Again execute the "/usr/local/DQS/bin/qstat331 -f" command:

<table>
<thead>
<tr>
<th>Queue Name</th>
<th>Queue Type</th>
<th>Quan</th>
<th>Load</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibms30</td>
<td>batch</td>
<td>0/1</td>
<td>0.10</td>
<td>UP</td>
</tr>
</tbody>
</table>

**TEST SCRIPT**

Once the qmaster and at least one daemon are running, a simple test can be performed. In the directory ../DQS/tests directory are a collection of sample scripts. The entire contents of this directory should be copied to a user (non-root) directory owned by the administrator. As a first test change directory to this non-root directory and type "/usr/local/DQS/bin/qsub331 dqs.sh". This will submit the simple script to DQS:

```
#!/bin/ksh
#$ -l qty.eq.1
#$ -N UTESTJOB
#$ -A dummy_account
#$ -cwd
echo 'we are now doing something else'
printenv
sleep 30
echo 'end of script'
```

A message should appear in response to the qsub331 command:

"your job 1 has been submitted".

After 30 seconds the job should complete and in the directory where the job was submitted two output files should appear:

UTESTJOB.e1.25674 and UTESTJOB.o1.25674

The title UTESTJOB was established by the DQS directive "#$ -N UTESTJOB". The next field (either e1 or o1) contains the job
number preceded by the type of file. The stderr file for the job will have an "e" in that position and the stdout file will have an "o". The number at the end of the file name represents the PID the job had when it executed. The UTESTJOB.el.25674 file should be zero length. If not examine its contents for the cause of any error. The stdout file should begin with the line: 'we are now doing something else', followed by a display of the user's environment and ending with the line 'end of script'.

**COMPLETION OF INSTALLATION**

If the test script completes correctly, hosts can be added and additional queues created and more complex job tests can be submitted. If the "Quick Install" method was chosen the time has probably arrived to plan an operational cell organization and setup resource files and queues. In order to layout an effective system it is important to understand how DQS is constructed, the capabilities of its components and how they may be tailored for a specific site.

**System Topology and Operation**

A basic DQS system consists of at least one computer host which is running the qmaster program and at least one instantiation of the dqs_execd daemon which manages the actual execution of jobs on the host which they "inhabit". All of the resources managed and monitored by a qmaster are considered to be a "cell".

![Diagram of DQS system topology and operation](image-url)
Within a cell there are three classes of programs operating. The qmaster daemon, the dqs_execd daemon and the DQS utilities which include qsub, qstat, qmod, qconf, qdel, qhold, qhold, qrls.

1. The qmaster maintains all of the critical files and tables for a cell. There are actually two types of tables managed by the qmaster which are called "queues". The first is the job queue which is a linear, ordered list of all jobs in the system. This list is sorted by job priority, an internal job sub-priority (based on a site parameterized "fair use" policy) and then by the order in which jobs have been submitted. The second table type is a list of "execution queues", where each potential target for running a job is defined by a queue configuration for that target.

2. The qmaster possesses a set of "auxiliary" files which are used to maintain information for system security and to parameterize DQS for specific site characteristics. Access control lists, static and consumable resource definitions, and a table of "trusted hosts" who are permitted to contact the qmaster are "mirrored" in memory and disk at all times so that the qmaster can survive interruptions such as power-outages.

3. The primary mode of operation of the qmaster is "listening and waiting". The qmaster listens for messages from other qmasters (which are managing their own cells), its own dqs_execd daemons and the DQS utilities. Periodically the qmaster examines the job list and attempts to find an execution queue which can satisfy the requirements of one or more jobs in the table.

4. The basic operation of the dqs_execd is "sleep through class.. and wake up in time to answer a teacher's question or hear the end-of-class bell". The "class bell" in this case is a periodic event where the dqs_execd gathers information on the health and state of the host machine on which it resides. This period is defined in the "conf_file" and can be varied be each site. At this point the "load average" is sent to the qmaster to provide the qmaster with information to help it distribute jobs among the available hosts. (If the conf_file parameter "DEFAULT_SORT_SEQ_NO" is set to TRUE, the load average report is subservient to the sequence number of a queue.)

5. The "teacher's question" in this case is a probe from the qmaster for a system integrity test or a system request, usually to begin execution of a job. At this prodding the dqs_execd sets to work as we will see later.

6. In a quiescent system, with no jobs queued, and none executing, the qmaster and dqs_execd daemons continue their "sleepy handshaking" described above. The term "sleepy" was chosen because these programs have been designed to utilize minimal system resources (memory and cpu cycles) on their hosts. Thus both programs are either sleeping or performing the minor handshaking indicated by the a diagram. In DQS331, a qmaster in one cell does not poll or communicate with other qmasters except to request an action such as moving a job from one queue to another.
7. Into the idle system described here, a user submits a job from one of the "trusted hosts" in the system. This could be a host in the cell which also houses the qmaster or a dqs_execd or on a host with neither daemons, but which was made a trusted host by the administrator using the "qconf -ah " command. Two validation steps occur upon invocation of the "qsub" command.

A. The qsub command line and the script file are scanned for DQS directives. DQS directives may occur in either stream, but the scanning stops when a string is encountered which is neither a comment nor a DQS directive. (The default flag for a DQS directive is the character pair `#$`). All DQS directives are "parsed" for syntactical errors and rejected at this point if problems are found.

B. The syntactically verified command line and script file are then sent to the qmaster (shown as [b] in the diagram). The qmaster then performs a "semantic" validation of the job request. By "semantic" here we mean "does the request make sense in the context of this system at this time". The second test compares the user's request for site-defined resources (such as those actually present in the system at the moment. Unless the submitted job possesses the DQS directive "-F" (force the acceptance of the resource request), if one or more of the requested resources do not exist (please note that this test verifies that a resource is present in the system, not whether or not it is in use by another job!).

8. If the job request passes these tests it is placed in the job queue [c]. This queue is 'mirrored' on disk so that it may be recovered after a system restart. When a new job is placed in the list, the qmaster scheduler scans all the jobs in the list and tries to find an execution queue which will satisfy each entry's request for resources. This process does NOT begin with the newly arrived job but begins at the top of the list, so it is possible that the job submission may trigger the scheduling of a previously submitted job and leave this job "awaiting another time".

9. At some point, motivated by the submission of a new job, the termination of a running job or a period of seconds defined by the "SCHEDULE_TIME" parameter in the conf_file, the qmaster will scan the job list and find a job which meets the resource requirements. The job description and script file are "packaged up" and sent to the target dqs_execd [d]. The status information for the target queue is updated to indicate the change of state and the identity of the job's host machine. Where parallel jobs have been specified, the qmaster will assign additional hosts and mark their status as running the selected job. Slave processes, however are initiated by the dqs_execd host for the Master process, and not the qmaster.
10. The dqs_execd first records the job request information in its own "mirror" disk file, so that it may be retrieved in the event of a system restart while the job is executing. Then the job is prepared for execution. This process consists of first creating a separate UNIX process to monitor and manage the executing job. In DQS331 this is called the "shepherd" process. It is the presence of this "shepherd" which permits a single dqs_execd to manage multiple job executions on the same host and deals with the need for AFS re-authentication invisible to the executing jobs.

11. The first step for the "shepherd" is to establish an environment for the job which matches that of the submitting user, modified by the parameters in the job script and on the command line. Next the "shepherd" determines how the system and the user wish to handle the stdout and stderr files for the job. This is directed by DQS directives and the system-wide parameters in the "conf_file".

12. If one of the forms of parallel job execution has been specified (the "-p" option in the DQS directives) the Master dqs_execd will "remote-shell" the DQS task "dsh" (distributed shell) to the target Slave processes. The dqs_execd on each Slave host will start a process to manage the SLAVE task. (In this release of DQS 3.1.3 this task is NOT identical to the process shepherd and does not support AFS re-authentication of the SLAVE process.)

13. After the user's environment has been setup and any SLAVE process managers started on other hosts, the DQS331 "process shepherd" sends the job startup notice (if requested) and then launches the job.

14. The "process shepherd" then enters its own "sleep" loop, occasionally awakening to peek at the running job and copy output files (as directed) to their target directories.

15. Upon job termination the "process shepherd" executes a system defined "add-on script" which usually performs additional job-cleanup operations. The dqs_execd then forms an accounting record including job execution statistics, which is sent to the qmaster, signaling the completion of all activities related to the job[a]. Any SLAVE processes terminate their own portion of a job independently. These SLAVE tasks are usually shut down by their master process, according to the methodology of the specific parallel paradigm, P4, MPI, TCGMSG, or PVM.

16. As with the "qsub" job submission program, all DQS331 utilities interact only with the qmaster. The qmaster rejects any requests if the originating computer is not in the cell's host list. The qmaster then checks to see if the user has permission to perform the actions. For example at most sites any user can request a display of the queue status (qstat command), while only a DQS administrator is permitted to add, delete or disable queues.

17. Thus in this system a valid request by a user to delete one of their running jobs consists of the following sequence qdel <job > → qmaster ; qmaster validates request; qmaster sends a job terminate message to the appropriate dqs_execd; qmaster sends an acknowledgment to the qdel utility; qdel posts a message to the submitting user; dqs_execd sends a
UNIX SIGKILL to the job; job termination triggers the dqs_execd to
gather usage data and send an end-job message to the qmaster; qmaster
logs the accounting information; qmaster deletes all job information;
qmaster marks the queue as available for scheduling.

**Cells, Hosts, Queues**

In the previous section a diagram of the elements constituting a
"cell" were displayed. A DQS331 site may have several
independent cells, or they may be aggregated into a common
operating environment:

This example displays three cells A,B,C, each managed by its
own qmaster QM-A, QM-B or QM-C. The hosts are labeled A1
and A2 for Cell-A, B1, B2 and B3 for Cell-B and C1 and C2 for
Cell-C. For this discussion we will assign the qmasters to a
separate host in each cell. QM-A will thus be on host A0, QM-B
on host B0 and QM-C on host C0.

Communications among the various hosts in a cell and between
cells is structured by the inclusion of a host within a qmaster's host
list. In the above example qmaster QM-A has four hosts in its
table, A0(its own host), A1, A2 and B0(the qmaster host for cell
B). Instead of a completely symmetrical inter-cell arrangement
here we have chosen to not have QM-A linked with QM-C. Thus
neither of these qmasters will have the other cell's qmaster host in
its own hosts table.

An option, which is less secure, is to permit the host from one cell
to contact the qmaster in another cell (as shown by path [c]. In this
case host B3 could execute utilities and perhaps launch jobs in
Cell-C as well as Cell-B. Even without this "sneak path" hosts in
cells A and C can interrogate the status of queues in Cell-B, if the
user permissions allow such an activity.
Note, once again, that a host in a cell may have no queues assigned to it for execution, or it may have one or more queues assigned to it. It is also quite common to have a dqs_execd running on the same host as the qmaster daemon. The DQS331 utilities can be executed on any host in a cell, regardless of whether that host is running a dqs_execd daemon.

The first level of security within DQS is then a "trust" relationship among a cell's hosts and between each cell's qmasters. The next level of security is the level of permissions established by a qmaster's "manager" and "administrator" lists. The third level of security is defined by specific user permissions or exclusions for each queue. Certain activities are permitted to a DQS administrator or manager which a queue owner may not invoke, among them are deleting the queue itself or changing its configuration. A queue owner and the DQS managers may perform activities such as queue suspension, which of course the average user is prohibited from doing.

**System Directories**

To manage system security, queues, jobs and user access, a number of directories are created during the startup process. The DQS administrator will normally not have to deal with these directories nor their contents. However when all DQS files cannot or should not be cross-mounted it is important that the function of these elements are understood so that they can be placed correctly in the system.

**Shared & Local**

As indicated in the installation instructions, the easiest method for managing a DQS is to have all the system files and directories mounted by NFS/AFS or DFS on all hosts. The one exception to this is that the directories containing the binaries for all DQS executables which, of course, should only be shared by hosts with identical architecture and operating system configurations. A knowledgeable administrator may wish to make changes directly to the contents of one of these directories. Where appropriate a hint or two are provided to assist the system manager. A typical installation will posses a directory tree somewhat like: (underlined names are directories, italicized names are files)

Sorry, the display of this tree is hopelessly messed up - h.n.

/user
/local

/DQS

/common /bin

/conf

/qmaster resolve_file conf_file /dqs_execd
act_file log_file

/QM-A host-A1 host_An

/common_dir /exec_dir
complex_file script_file
consumables_file /job_dir
generic_queue job1
host_file job2
man_file ..
op_file ..
seq_num_file ..

acl_file ..
/job_dir /usage_dir
job1 current_usage
job2 /tid_dir
.. tid_##xxxx
    .. tid_##xxxx

/queue_dir pid_file
queue-A1 core
queue-A2
Four system files are classed as "should be shared by all hosts, if at all possible". They are:

**conf_file** --- This file is created during the DQS331 "config" step of the installation or system update. This file contains system-wide configuration which is read by the qmaster, dqs_execd and all DQS utilities when they startup. If it is necessary to make changes to this file, the qmaster and all dqs_execd's should be shutdown and restarted after the changes are complete, so that they will posses the latest configuration. Failure to observe this step may often result in bizarre and unexplained behavior of the system if not an outright collapse. If this file cannot be cross mounted by all hosts, then an IDENTICAL COPY of this file needs to be distributed to all hosts before restarting the qmaster or dqs_execd daemons or any of the command-utilities.

The location from which this file is read is "hard-wired" into the compiled DQS code based in the #define Conf_FILE statement in the dqs.h file which is also created by the DQS "config" step. It is important to understand that the default installation setup places the conf_file in "/usr/local/common/conf" directory, which is also used as the default location for the qmaster and dqs_execd spool directories. While those directories can be relocated by changing the conf_file and restarting the daemons, the location of the resolve_file and conf_file can only be changed by modifying "dqs.h" with an editor or be re-executing the "config" program.

The following are the initial entries in the conf file with a description of each line's effect on the system.

```
QMASTER_SPOOL_DIR
/usr/local/DQS/common/conf
```
This parameter points to the starting directory from which the qmaster's sub-directories are created. While at some sites with several cells the resulting tree can be shared by multiple qmasters, it is only necessary that the qmaster have access to the sub-directories for itself. This tree appears above as "qmaster/QM-A".

**EXECD_SPOOL_DIR**

/usr/local/DQS/common/conf

This parameter points to the starting directory from which all of the dqs_execd's in the cell will find their individual queue management directories. In the default DQS setup all dqs_execd's in a cell use this same directory tree terminating their own specific set of sub-directories. This is illustrated in the preceding diagram by "/dqs_execd/host-A1".

**DEFAULT_CELL** user-network

The system-wide, unique name for a given cell. This can be any arbitrary ASCII string and is defaulted to the qmaster's host domain name during the installation process. If this name is changed the corresponding string in the "resolve_file" must be changed accordingly and vice-versa.

**RESERVED_PORT** TRUE
This parameter indicates that all daemons and utilities in a cell will be using UNIX reserved ports for socket communications. UNIX system port numbers from 0 to 1023 are designated as "reserved". If this parameter is set to TRUE then all of the DQS331 programs MUST execute with root ownership. If this parameter is set to FALSE then the /etc/services port numbers for DQS331 services must be greater than 1024.

**DQS_EXECD_SERVICE dqs33_execd**

Any arbitrary ASCII string can be used to identify the tcp port number to be used when the qmaster or the DQS utility "dsh" is communicating with the dqs_execd. The only requirement is that this name must be unique among all names in the /etc/services file.

**QMASTER_SERVICE dqs33_qmaster**

Any arbitrary ASCII string can be used to identify the tcp port number to be used when the dqs_execd or DQS utilities are communicating with the qmaster. The only requirement is that this name must be unique among all names in the /etc/services file.

**INTERCELL_SERVICE dqs33_intercell**
Any arbitrary ASCII string can be used to identify the tcp port number to be used when the one qmaster is communicating another qmaster. The only requirement is that this name must be unique among all names in the /etc/services file.

**KLOG /usr/local/bin/klog**

The re-authentication process in AFS systems will use the klog program. This entry is only used when AFS support was selected during DQS installation.

**REAUTH_TIME 60**

If AFS has been selected, all daemons and executing jobs will be re-authenticated every period of this number of seconds.

**MAILER /bin/mail**

All jobs can select options to send brief "job startup", "job end" and "job abort" messages to one or more designated users. In addition the DQS331 system will send mail messages to the administrator in the event of extraordinary system events.

**DQS_BIN /usr/local/DQS/bin**

The qmaster, dqs_execd and all user initiated utilities locate their binaries in the BIN_DIR established during
the "config" step of installation. This entry is set by that step, and acts as a "place-holder" for that target directory. This parameter is used, however by the parallel queue management system. If the administrator wishes this parameter can be changed to point to a different directory where PVM,P4,TCGMSG and MPI support programs may reside. Doing so will not affect the continued use of the BIN_DIR for the remaining DQS executables.

ADMINISTRATOR admin@host_machine

On startup of the qmaster this entry is used to identify the primary DQS administrator for this cell. This also forms the email address used to send system error messages.

DEFAULT_ACCOUNT GENERAL

Any arbitrary ASCII string (without separator characters such as blanks, periods, commas) can be used as an account identifier. Each job submission can provide its own account identifier, which overrides this default string. No validation is performed on this or the user submitted account name string. When a job terminates a record is created from hardware and software usage data. The "account string" is appended and the record is appended to the qmaster's "act_file".
LOGMAIL FALSE

By default none of the mail generated by the DQS, either to users or the system's managers is logged. Setting this parameter to TRUE will cause the qmaster to create a mail log file, where all system emails are recorded and time-stamped.

DEFAULT_RERUN FALSE

It is our sincere hope to have the rerun feature if DQS implemented in future versions. In DQS331 this parameter is ignored.

DEFAULT_SORT_SEQ_NO FALSE

During the qmaster's scheduling process two major steps occur. First the jobs themselves are sorted according to their submitted priorities and internal policy criteria. Second all of the available queues are scanned to find one which suits the needs of the first job to be scheduled. The ordering of this queue scanning process can be changed by this parameter. When this parameter is FALSE all of the queue entries are sorted in the decreasing order of their host's usage data (as reported by the dqs_execd). Thus the first queue examined will be the least "busy" queue, in an effort to spread the workload across the system.
If this parameter is set to TRUE the queues are examined in the order of the sequence number assigned by the administrator in each queue configuration. Many sites use this method to ensure that their most powerful hosts are scanned first, by assigning those hosts very low sequence numbers to the corresponding queues.

SYNC_IO FALSE

In multi-host systems utilizing NFS mounted files it is possible for I/O actions to become disordered in their results. The ordering of lines of output sent to stdout or stderr can become totally confused. DQS331 is supposed to have a feature in its "process shepherd" to ensure that all stdout and stderr output is properly time sequenced, even when multiple SLAVE processes are involved. In the initial DQS331 release this feature is not active.

USER_ACCESS ACCESS_FREE

This feature for differentiating levels of access for users or classes of users is not implemented in DQS331.

LOGFACILITY LOG_VIA_COMBO

Many system messages are generated to aid in the maintenance and diagnosis of DQS operation. Three files are used for this
activity, the "err_file", the "log_file" and the "syslog_file". Depending on the level of attention required messages are directed to one of these files. All messages with ERR, CRIT, or WARNING are always sent to err-file. Messages with levels of INFO, WARNING or NOTICE can be sent to the system log or the normal activity log file. The normal mode is to use both the system log and normal log file.

**LOGLEVEL LOG_INFO**

Information is logged depending on the level assigned within the DQS. In increasing order they are LOG_INFO, LOG_NOTICE, LOG_WARNING, LOG_ERR, LOG_CRIT, LOG_ALERT, LOG_EMERG. Setting the LOGFACILITY parameter establishes the minimum level of messages to be recorded. A parameter of LOG_INFO ensures that all messages will appear in the "logfile".

**MIN_UID 10**

**MIN_GID 10**

For security reasons it is desirable to establish a minimum user and group identifier (uid or gid) which will be permitted in execution of any of the DQS utilities. The qmaster and dqs_execd, of course normally operate at root level. The recommended setting is "10" for these parameter values as most UNIX critical processes run
with uid and gid values below "10". It is strongly recommended that these default values be retained.

Attempts to run DQS utilities such as qsub, qalter, qstat, etc. will fail if these default values are used, which is the "correct", albeit confusing (to new system managers) behavior of DQS.

MAXUJOBS 10

There are a number of DQS "system policy" parameters available to the DQS331 administrator. One of these is a system-wide limit on the total number of jobs a user may have considered for scheduling at any one time. This is not a limit on the total number of jobs which a user can have queued up in the system, but it does instruct the qmaster not to consider more than MAXUJOBS for a user during a scheduling pass. The effect of this limit can become quite subtle. For example, if a limit of 10 is established and the user submits 100 jobs, they will be ordered in sequence of their priority and submission time. If the first ten of these jobs require system resources not currently available, they cannot be scheduled. Neither will any following jobs, which may need some resource which is actually available. An additional user
OUTPUT_HANDLING
LEAVE_OUTPUT_FILES

When a job is started by the qmaster it may be able to produce large stdout or stderr files. The writing of these files to a remote, NFS mounted file system can have negative impacts on system performance. In some cases, retaining these files on a host's local file systems could prevent network congestion and minimize I/O delays for the running job. DQS331 provides three options for handling these output files. The default LEAVE_OUTPUT_FILES causes the stdout and stderr files to be left in the working directory established by the user's "qsub" script.

This parameter can be changed to LINK_OUTPUT_FILES. In this case the administrator must create a special file in one or all the dqs_execd spool directories. The name of this file is defaulted to "netpath" during the DQS "config" step. This default name may be changed in the dqs.h file by the administrator, if they are prepared to recompile the entire DQS331 system. The "netpath" file should contain one ASCII line defining the fully qualified network path.
of the target directory into which the stdout and stderr files are to actually be placed.

If the parameter is set to COPY_OUTPUT_FILES the DQS331 process "shepherd" creates temporary standard output and standard error files local to the host executing the job. A special "copy" process is started which wakes up periodically (set by the hard-wired COPY_FILE_DELAY in the dqs.h file), and copies the current contents of those files to their actual destination.

**ADDON_SCRIPT NONE**

At the conclusion of a user's job, and in the working space of that job it is sometimes necessary to conduct system cleanup tasks. This is particularly true of parallel processing tasks which might leave "orphan" daemons running, in the event of unplanned process termination. A system script maintained within the DQS can be created and invoked at the conclusion of EVERY user job. This parameter must then contain the fully qualified path-name to this script file.

**ADDON_INFO NONE**

When OUTPUT_HANDLING is set to anything other than LEAVE_OUTPUT_FILES, the system administrator may
wish to maintain a diagnostic awareness of the "process shepherd" handling of the copying or linking of a user's stdout and stderr files. If this parameter is set to something other than NONE, the parameter string should be a fully qualified path to a file containing an ASCII string to be appended to the "stdout" file along with other job information.

LOAD_LOG_TIME 30

Upon startup the dqs_execd sets this parameter (specified in seconds) as a minimum period for the dqs_execd to deliver system usage statistics to the qmaster.

STAT_LOG_TIME 600

Various system statistics, beyond the host usage provided by the dqs_execd daemons, are gathered periodically, based on the value of this parameter (specified in seconds).

SCHEDULE_TIME 60

The qmaster scans the cell's job queue after every new job is submitted to the system or upon termination of a running job. Absent these occurrences the qmaster will trigger a scheduling pass of the jobs based on this parameter (specified in seconds).

MAX_UNHEARD 90
The qmaster does not poll other daemons for their status. Instead it updates the queue status for each dqs_execd which reports in. If a dqs_execd fails to report in to the qmaster within this threshold (seconds) the qmaster will mark all queues managed by the dqs_execd as "status UNKNOWN". This status is updated every interval, and can be changed from UNKNOWN to UP if the dqs_execd has finally succeeded in updating the qmaster.

ALARMS 3

ALARMM 4

ALARML 5

The admonition to "avoid changing these parameters" in the installation is well founded. These parameters control the amount of time permitted before the UNIX system interrupts an attempt at inter-host communications. The ALARMS value is the time in seconds before a DQS utility such as qsub, qmod is interrupted. A message will appear for the user with message "Alarm Clock Shutdown" indicating that the utility cannot contact the qmaster within "ALARMS" seconds. The ALARMM parameter sets a similar limit on dqs_execd<->qmaster communications attempts. ALARML is the longest
period established for inter-
process interchange attempts,
and is used to control
qmaster<->qmaster
communications.

In systems where the qmaster host is also running other jobs or where the network interconnect can become congested is possible for one or more communications attempts to fail due to an ALARM time-out. If the err_file contains frequent "ALARM CLOCK Shutdown" warnings or utility execution fails often with similar error messages the three ALARM parameters should be increased. These values should be kept as small as practical to prevent a failing DQS element from tying up the host's tcp/ip interface.

**resolve_file** --- This file is also created during the DQS "config" process. It is the equivalent of a combination of the UNIX 
"resolv.conf" and "hosts.equiv" files for managing network security. The default resolve_file is:

```plaintext
# NOTE! blank lines NOT permitted #
# NOTE! fields must be separated by one(1) AND ONLY one space #
# 1st field = cell_name
# 2nd field = primary qmaster
# 3rd field = primary qmaster alias
# 4th field = secondary qmaster
# 5th field = secondary qmaster alias
user-network QM-A0 QM-A0.user.com NONE NONE
```

The comment lines direct the DQS manager as to the format of new entries or entry changes, Some of the aspects of this file need further explanation.
a. The cell name appearing in the first field of the first non-commented line MUST be identical to the name appearing as the DEFAULT_CELL parameter in the conf_file.
b. DQS331 does not yet support alternate qmasters and thus the last two fields of each non-commented line must be "NONE" and "NONE"
c. Additional cells may be defined by adding lines to the resolve_file following the primary cell entry. If a host in one cell is permitted to contact a qmaster in another cell (via a "sneak path") then the cell name and qmaster name for that other cell must appear in the source cell's resolve_file.

err_file --- The master, dqs_exced and all DQS utilities may originate error messages which are directed to a hard-wired filename "err_file". This name is created during the DQS "config" step and implanted in the "dqs.h" include file in the ../DQS/SRC directory. The installation process assumes that all DQS331 programs will have write-access to the path name which appears as QMASTER_SPOOL_DIR in the conf_file. If this path name is inappropriate for ALL DQS programs the administrator may choose to change the definition of ERR_FILE in the include file "dqs.h". This will require recompilation of the entire DQS331 system.

As an alternative, the administrator may choose to let each program write to its own "err_file" and gather and collate all the files when it is necessary to examine error information. In this case, however the path-name accessible by each host must be identical to the QMASTER_SPOOL_DIR name.

log_file --- The master, dqs_exced and all DQS utilities may originate error messages which are directed to a hard-wired filename "log_file". This name is created during the DQS "config" step and implanted in the "dqs.h" include file in the ../DQS/SRC directory. The installation process assumes that all DQS331 programs will have write-access to the path name which appears as QMASTER_SPOOL_DIR in the conf_file. If this path name is inappropriate for ALL DQS programs the administrator may choose to change the definition of ERR_FILE in the include file "dqs.h". This will require recompilation of the entire DQS331 system.

"log_file" and gather and collate all the files when it is necessary to examine error information. In this case, however the path-name
accessible by each host must be identical to the
QMASTER_SPOOL_DIR name.

Qmaster

The qmaster directory contains a major sub-directory for each
qmaster registered in this cell. Each qmaster's directory contains
four sub-directories whose contents change constantly during
DQS331 operation, and hence must permit write operations an all
files. There are also two files created by the qmaster, the pid_file
and stat_file. An additional, unwelcome file may appear here also.
In the event of a qmaster crash, its core file will be placed in this
directory.

common_dir

This directory contains files common to the scheduling and
dispatching of jobs by the qmaster.

complex_file -- This file contains all of the definitions of
complexes created by the add complex command (qconf -ac).

consumables_file -- This file contains all of the definitions
of consumable resources created by the add consumable
resource command (qconf -acons).

generic_queue -- This file is read by the qmaster each time
the create queue command (qconf -aq) is performed and no
name is provided as a parameter following the "-aq" option
flag. The contents of this file form the starting template
presented in the editor for modification by the
administrator.

host_file -- The host_file is read up at startup of the
qmaster and contains a list of all the hosts known to the
qmaster and occasionally called "trusted hosts". Any
program attempting to contact the qmaster must have its
host's name in this list or be rejected. On the initial startup
of the qmaster this file will not be present. The qmaster will
post a warning in the err_file and create the host_file.

man_file -- This file contains the login names of all
individuals identified as cell "managers". A cell "manager"
is given permission to access all DQS331 system files and
to execute every option of every DQS331 utility.
**op_file** -- This file contains the login names of all individuals identified as cell "operator". A cell "operator" is given permission to perform a number of system operations normally reserved to the system manager, and prohibited to the standard system user. The functions qdel, qmod, qmove, and qrls are permitted by operators. Functions such as creating or deleting queues or adding and deleting managers and operators is, of course, limited to cell managers.

**seq_num_file** -- Jobs are assigned an internal sequence number. The next number to be assigned by the qmaster appears as a single binary value in this file. It is thus not possible to manually reset sequence numbers, other than to delete this file, forcing the numbering sequence to begin over with "1".

**acl_file** -- This file contains all of the access control list "acl" names for all queues. This is actually a list of lists. An "acl" is a list of names to be given access to one or more queues. A queue definition can include these individuals by naming the corresponding "acl" in its "user_acl" parameter.

**job_dir**

This directory contains a file for each job currently in the queuing system. Each file contains the submitted script file along with tables and lists created by the qsub operation and used to manage the job while it is in the queue awaiting assignment to a host, as well as during actual job execution.

**queue_dir**

This directory contains a file for each queue. The file name is, in fact the name assigned to that queue. Each file contains the queue configuration, encoded in binary form, along with various tables which the queue manager utilizes to manage the queues.

**tid_dir**

To maintain internal coherency during system operation, in the face of multiple hosts executing multiple processes a unique identifier label is generated by the qmaster and dqs_execd for every inter-host communications. This label is called a "task identifier" or "tid". An empty file for each generated "tid" is created in this An acknowledgment by the receiving host for a
transaction causes the corresponding tid file to be deleted from this
directory.

In the event of aberrant behavior of a hardware or DQS331
software element some "orphan tid's" may be found in this
directory, however the administrator is cautioned to NOT clear out
tid files manually without careful analysis. This scheme was
created to ensure inter-host synchronization despite multiple
restarting of the qmaster or the dqs_execd.

**pid_file** -- This file contains a list of the process id of the running
qmaster. This is a "canonical" location where site procedures may
find this pid for system management actions.

**Stat_file** -- Based on the period defined as "STAT_LOG_TIME"
the qmaster records summary information about all the queues it is
managing. This data is time-stamped so that DQS managers might
determine when queue status changes occur inadvertently.

**dqs_execd**

The dqs_execd directory contains major sub-directories for each
dqs_execd operating in this cell. Each dqs_execd directory contains
four sub-directories plus one file, the "pid_file" which contains the
process id of the dqs_execd. Of course there is also the possibility
of a core file being placed here in the event of a dqs_execd crash.

**exec_dir**

The exec_dir contains the actual job file for the executing job.
When the dqs_execd launches a job, the script file is copied here
and executed.

**job_dir**

The job_dir contains a file for each job which the dqs_execd is
managing (usually only one). In addition to the job's DQS script
this file contains all the tables and information necessary for the
qmaster and the dqs_execd to manage this job.

**rusage_dir**

Upon job termination usage data is collected and formatted into a
"termination record" to be sent to the qmaster. This record is
written to this directory and retained until the qmaster has received
and recorded this information. The procedure is used to prevent
vital data from being lost, particularly from long-running jobs, in
the event of an interruption of dqs_execd or qmaster service.

tid_dir

To maintain internal coherency during system operation, in the
face of multiple hosts executing multiple processes a unique
identifier label is generated by the qmaster and dqs_execd for
every inter-host communication. This label is called "task
identifier" or "tid". An empty file for each generated "tid" is
created. An acknowledgment by the receiving host for a
transaction causes the corresponding tid file to be deleted from this
directory.

Temporary Files

The dqs_execd creates and deletes a number of temporary files in
the "/tmp" directory of its host. These are deleted after use, but if
the dqs_execd has been shut down during job launching and
execution, these files may be left in the "/tmp" directory
inadvertently. Since they are given unique names for the job
execution they will remain until removed by the system manager.

The Queue Configuration

The queue configuration was introduced during the discussion of
setting up an initial DQS33 cell and queue. The queue
configuration is the primary means of tailoring a DQS system to a
particular site's requirements. The queue configuration can be
changed dynamically by the DQS cell manager without requiring a
shutdown and restart of either the qmaster or dqs_execd, unlike the
more static "conf_file". Changing the queue configuration will not
affect any jobs already in execution. The modified configuration
will be considered during the next scheduling pass of the qmaster
after the change has been completed. A description of each
element follows:

Q_name QAI

Any ASCII string of numbers and letters may be used in
the queue name. It must be a unique queue name in a given
cell.

hostname QAI host
The hostname entered here may be any form of the host's name which is used by the network members. DQS will convert the entered name to the fully qualified host name and insert that into the registered queue configuration.

seq_no 0

The seq_no is an arbitrary sequence number assigned by the DQS administrator. It is ignored if the conf_file parameter "DEFAULT_SORT_SEQ_NO" is set to FALSE. If "DEFAULT_SORT_SEQ_NO" is set to TRUE the qmaster will scan the queue list in the order of the sequence numbers starting with zero "0".

The DQS administrator may choose one of several strategies for assigning sequence numbers. At CSIT the lowest sequence number is assigned to the most powerful computing engines, with less powerful machines being assigned higher sequence numbers.

load_masg 1

Each dqs_execd collects information about the state of its host's overall computational and I/O load as reported by the UNIX system through the "rusage" structure. A "total system load" is provided as an integer value representing a fractional percentage of the system usage. A value of 1 represents a load of 0.01, a value of 10 represents a load of 0.10, and a value of 100 represents a load of 1.0.

When DEFAULT_SORT_SEQ_NO is set TRUE the qmaster attempts to assign jobs to the least loaded queues which meet the resources requested by the job. The queues are sorted into increasing order of the load average, weighted by multiplying by the reported load average by the "massage factor" (the load_masg value). The load_masg factor thus permits the administrator to adjust the system wide relationships between different hosts which may be necessitated by variations in usage measurements or background task activity.

load_alarm 175

A threshold value can be set beyond which a queue will not be considered for scheduling by the qmaster. When a host reports a load average greater than this threshold it is
in an "ALARM" state, and this flag is displayed in qstat output. The default load_alarm represents a load average of 1.75.

priority 0

This field may be confusing at this point because jobs also possess a submission priority. The difference is that the job priority determines only how it is ordered among other jobs in competition for system resources. The job submission priority has no influence on the UNIX priority with which that job is executed.

The queue priority field here IS the UNIX priority assigned to any job executed in this queue and thus may range from -19 (low) to +19 (high).

type batch

DQS was designed to support the scheduling and management of batch and interactive jobs. DQS33 supports only batch queues. This parameter is ignored

rerun FALSE

Automatic job rerun is not enabled in DQS331, this field is ignored.

quantity 1

A DQS331 queue can manage more than one job in execution at a time, though this is usually not a practical way to operate a single cpu host.

tmpdir /tmp

During job startup and execution several temporary files are created. This parameter should be the fully qualified path name to the hosts temporary directory.

shell /bin/csh

The default shell for executing jobs in this queue. This default can be overridden by commands in the job script.

klog /usr/local/bin/klog
The path name to the AFS klog executable.

    reauth_time 6000

The time period in milliseconds for performing an AFS re-authentication of the executing job.

    last_user_delay 0

To prevent a single user from dominating the utilization of a queue the administrator can set this time-out value (seconds) during which a user's job will not be considered for scheduling following termination of a previous job for that user.

    max_user_jobs 4

This is the second system parameter available for implementing scheduling policies for DQS331 at a site. The MAXUJOBS parameter in the conf_file limits the total number of jobs a user can have considered for scheduling across the entire system. The queue configuration "max_user_jobs" establishes a limit on the number of jobs a user can have queued which will be considered for scheduling for this queue. See "SCHEDULING" for a more complete discussion of this topic.

    notify 60

A user job may invoke the "-notify" option which instructs the system to send the job a SIGUSR1 or SIGUSR2 signal as a warning in advance of a SIGSTOP or SIGTERM signal. This "notify" parameter in the queue configuration establishes the number of seconds between sending the warning signal and the SIGTERM or SIGSTOP.

    owner_list NONE

In addition to the DQS manager and DQS operator an individual can be designated a queue owner. A queue owner can perform many system management tasks permitted to the managers and operators but limited to this queue. Job deletion, queue suspension, enabling and disabling are among those actions One or more login names can be entered for this parameter.
user_acl NONE

The administrator can create one or more access lists using the "qconf -au" command. This command adds one or more users to a named list. (This named list will be created if it doesn't exist.) These named lists (of names) can be used to include or exclude groups of users in access to a specific queue. This queue configuration parameter "user_acl" can contain a list of one or more acl_list names which will be permitted to use the queue. (That is, the parameter can itself be a list of names of lists of names - confused ?).

xuser_acl NONE

The administrator can create one or more access lists using the "qconf -au" command. That command adds one or more users to a named list. (This named list will be created if it doesn't exist.) These named lists (of names) can be used to include or exclude groups of users in access to a specific queue. This queue configuration parameter "user_acl" can contain a list of one or more acl_list names which will be excluded from access to the queue.

subordinate_list NONE

One or more DQS331 queues can be subordinated to another queue. The queue specifying a list of subordinates with this parameter is called the "superior queue". A "superior queue" can NOT be a subordinate queue to another. A queue can only be subordinated to one other queue. The "subordinate_list" parameter can contain a list of one or more queue names in the same cell as the queue defining this parameter.

Superior queues are analyzed for scheduling in the same manner as all queues. If a job is assigned to a superior queue, the qmaster will suspend the execution of jobs in all of the queues in the superior queue's subordinate list.

complex_list NONE
This parameter can contain one or more names of complexes defined by the "add complex" function of the qconf command (qconf -ac). See "Complexes and Consumables". Any complex name can be preceded by the DQS reserved word "REQUIRED" (must be all caps). This indicates that no job will be scheduled for this queue UNLESS it requests a resource described in that complex.

```
consumables NONE
```

This parameter can contain one or more names of consumable resources defined by the "add consumable " function of the qconf command (qconf -acons). See "Complexes and Consumables". Any consumable name can be preceded by the DQS reserved word "REQUIRED" (must be all caps). This indicates that no job will be scheduled for this queue UNLESS it requests a resource described in that consumable.

```
s_rt       2147483647
h_rt       2147483647
s_cpu      2147483647
h_cpu      2147483647
s_fsize    2147483647
h_fsize    2147483647
s_data     2147483647
h_data     2147483647
s_stack    2147483647
h_stack    2147483647
s_core     2147483647
h_core     2147483647
s_rss      2147483647
h_rss      2147483647
```

These parameters establish the "hard" and "soft" limitations on a host's resource utilization of a job executing under control of this queue. The "hard" limits are transferred to the job's execution environment in the hopes that the host operating system provides support for these limits. Note, however, that if a host does support these limits they apply only on a process-by-process basis!! If a job script contains multiple invocations of processes, as in a FORTRAN compilation and execution, the limits apply to each individual step in the job.

DQS331 does check the "soft" and "hard" real-time limits (s_rt & h_rt) and will terminate jobs based on the values of
those parameters. A job exceeding the "soft" real-time limit is sent a SIGTERM signal which can be intercepted by the job using the "-notify" option in the job script. If the job exceeds the "hard" real-time limits it is sent a SIGKILL signal which cannot be caught by the user job.

Complexes & Consumables

The most valuable aspect of DQS, and easily its most confusing property is the ability to define and utilize a variety of system "resources" which can then be requested in a user's DQS job script. These resource requests are used to differentiate and assign jobs to the variety of system capabilities found in today's heterogeneous computing environments. Let us look at an example of how and why resource definitions are created at a site. The diagram shows five DQS hosts with different capabilities.

Many users will have created an application compiled for one machine architecture, say AIX. In the pictured environment the user could run their application on one of the AIX machines by specifying the queue name, say QN1. The negative aspect of this simple approach is that the job may be kept waiting for QN1 because of a previous job on that machine while either QN3 or QN5 might be available.

![Diagram of five DQS hosts with different capabilities]

The solution for this situation is to create a resource definition for all AIX machines in the cell and name it "AIX1". Then the user can submit a job using the qsub command with the "-l" option. What are the steps needed to accomplish this:

a. A complex is created by typing "qconf -ac AIX1" (create a complex named AIX1)
b. The default text editor is started and an empty page displayed. The administrator enters an arbitrary string such as "our_AIX". Then save the results and close the editor.
c. Now that we have a complex defined (AIX1) we can add that complex to a queue definition.
d. Assuming that the queue has already been defined we will modify it using the qconf command. Typing "qconf -mq QN1" opens up another editor window with the complete queue definition displayed.
e. Replace the parameter entry for "complex_list" from NONE with AIX1. (the name The DQS administrator creates a resource definition called a "complex".
f. given to the complex definition NOT the contents of that definition.
g. In the same manner add the complex name AIX1 to the queues QN4 and QN5.
h. Advertise the resource name "our_AIX" to all users.
i. A user can then direct their jobs to any one of the AIX machines by including the resource request "-l our_AIX" in their DQS job script.

This simple example illustrates two key points.

1. The complex name is used by the administrator to assist in designing and managing collections of resources and queues. The complex name IS NOT USED by the user in resource requests.
2. Resource requests in job submissions use the descriptions within one or more complex definitions.

Let us expand the example slightly and create a new complex which cuts across machine architecture features, but shares a different attribute:

a. Create a complex for systems supporting PVM by typing "qconf -ac PVM1".
b. When the editor window opens enter a single line "our_PVM"
c. Save the file and close the editor and advertise the resource name "our_PVM" to the users.
d. Add the complex name PVM! To the complex_list parameter of queues QM2 and QM4.
e. A user wishing to submit a job to a queue which is running on an AIX machine which provides PVM support would use a resource request "-l our_AIX.and.our_PVM"

So far the sample resource definitions have been a single string such as "our_AIX" or "our_PVM". We could have used an alternative form for describing alternatives as we did with AIX
versus HPUX. This form would replace the string we entered in the complex files: arch=our_AIX and arch=our_HPUX. The string "arch" is one created by the administrator and could be any arbitrary name. A resource request would then have the form "-l arch=our_AIX", or "-l arch=our_HPUX".

Resource definitions can contain numeric values and the corresponding resource requests can perform numeric comparison on these values to satisfy a criteria. A complex called BigMemory could be defined containing the line "mem=128". For our example let QN1 and QN2 both be operating on hosts which have 128 megabytes each. The complex BigMemory would be added to the QN1 and QN2. A request for an AIX machine with at least 64 bytes of memory might be stated as "-l our_AIX.and.mem.ge.64".

Resource definitions can possess more than the single line examples in each named complex. A complex definition named "BIG_HUMMER" might look like:

```
AIX414
mem=1028
Horsepower=10
IO_bandwidth=250
```

A resource request which needs a BIG_HUMMER host would, in this case look like:

```
"-l
AIX414.and.mem.ge.1028.and.Horsepoer.ge.10.and.
IO_bandwidth.ge.250"
```

There is one type of resource we have singled out for special handling in DQS331. These are resources which are not static during the operation of a DQS cell. While machine horsepower, memory size and operating systems and compilers for long periods of times (on the order of days or weeks), shared memory multiprocessor cpus will have varying amounts of shared memory available to them as different jobs are executed on other of its cpus. An increasingly common resource situation is "licensed software" such as compilers and data-base management systems. In many cases there are fewer licenses available within a system than there are hosts to execute the software.

This type of resource is called a "consumable" in DQS331. The definition of a consumable resource is somewhat different than a
DQS "complex", in that the administrator will describe the total number of a resource which is available in a system, and the number of that resources consumed by a satisfied resource request. In the case of a FORTRAN compiler license, a site usually purchases a number of licenses for their system which are managed by a "license server". The consumable resource manager in DQS33 does not supplant a license server nor can it effectively mimic such a server. Instead it provides a mechanism parallel to the license server which attempts to keep track of how many licenses are in use by DQS clients. That is, DQS does not query the license manager for a count of available licenses, it keeps its own count of how many licenses (a consumable resource) are in use by DQS jobs.

The administrator defines a consumable resource by executing the command "qconf -acons FORTRAN" (using the compiler as an example). The default editor will open a window with the following template

```
Consumable     xlf
Available = <the amount of resources available>
Consume_by < quantum by which resource is reduced by a request>
Current = < currently available resources>
```

The field for Available should be filled in with the number of FORTRAN licenses authorized to this system. The Consume_by will be 1 for software such as compilers. The Current field will usually be equal to the Available field, unless there are several licenses in use at the time this Consumable is being defined. The Current field is also used to reset the DQS33 consumable counter when DQS33 gets out of sync with the actual license manager.

Queues which must manage this consumable resource should then have the consumable name added to the consumables parameter list in the queue configuration. The user need not be aware of the distinction between standard complexes and consumables. Their resource requests are stated in the same way: "-1 our_AIX.and.mem.ge.64.and.xlf". The qmaster will determine if an xlf license is available by examining its internal counters (which may NOT match the license server's). If the license and other resources are available the job will be launched. At the time the job is started the consumable count for the FORTRAN resource will be decremented.
Upon job termination this resource count will be incremented. Obviously this is not a satisfactory situation for a user who wishes to submit a job which does a quick FORTRAN compile which produces an executable which is then to run a week long job. The consumable count would remain decremented for the duration of the job while the license manager will have had the license "token" returned at the conclusion of the compilation.

For this situation the cooperation of the user is required, to avoid breaking up jobs into compile-only and compute-only separate jobs. The "qalter" command has been modified to permit any user to execute the "qalter" command but only if it has the "-rc" return consumable, command. The user job would then have a script file which might look like:

```csh
#!/bin/csh
#$ -l xlf.and.our_AIX
xlf my myprogram
Regrettably, "qalter -rc"
not implemented - hen 990806
qalter -rc xlf 1
myprogram mydata
```

The qalter command here specifies the name of the resource being returned followed by the quantity being returned. When resources such as high performance disk or shared memory are being defined as a consumable resource often a "quanta" of the resource is granted and recovered. An example might be that a UNIX page is the minimum quanta or an integral number of pages could be the "quanta". Where licenses are normally doled out one at a time, memory might be allocated 1 MB at a time. Hence the Consume_by field in the consumable definition.

**REQUIRED Complexes and Consumables**

A job submission may contain one or more resource requests (the "-l" option). A job with no specific resource requests is thus a candidate for assignment to any available queue. In many installations some queues are best utilized by very specific job configurations. An example might be a site which possesses a heterogeneous collection of cpus with very wide differences in computing capacity. The more robust computers should not be assigned to "tiny" but persistent jobs in some cases. DQS 3.3.1 provides a special keyword "REQUIRED" which can precede any complex or consumable which a user MUST request in order for that job to be considered for scheduling on that queue.
Job Scheduling

The crux of any resource allocation and management system is its ability to provide resources in an "efficient" and "fair" manner. "Efficiency" is usually measured in terms of maximizing job throughput and effective utilization of the available resources. "Efficiency" can be quantified in ways usually referred to the hardware hosts in a system. "Fairness" is less easily described, is often measured by perceptions and is most often referred to the human users of a system. Further, priorities for efficiency and fairness and their relative values can vary widely from site to site. The burden of meeting these objectives falls upon the system job scheduling mechanism.

Forty years of experience with attempts at creating comprehensive job scheduling algorithms have demonstrated several points:

a. It is virtually impossible to produce a "one size fits all" algorithm which will satisfy the demands for efficiency plus fairness at every site.

b. Scheduling systems which attempt to provide a 'flexible' software solution do so by offering to the administrator numerous parameters for adjusting the methods used for allocating resources. The plethora of variables presented is ultimately confusing if not confounding.

c. Most sites with complex requirements and knowledgeable support personnel end up writing their own scheduling code or modifying the code provided with the system.

DQS331 therefore attempts to provide only a minimal amount of job scheduling technology. Hopefully small sites will be able to achieve a good level of balance in host usage and perceived "fairness" with the system as it is delivered. As a site develops experience with batch job management the staff will experiment with the few parameters provided in DQS. At some point the administrator will want to probe the module dqs_schedule.c, adding or subtracting from its capabilities. To that end we will describe the basic features of DQS scheduling and try to illuminate the routines most likely to be modified.

A user job passes through two screening processes before being considered by the qmaster for scheduling:

1. At the time of job submission a user job is checked to see if it meets two system criteria:
a. Are resources present in the system which meet the
requirements specified for the job (usually through
the "-1" parameter in a qsub script)?

b. Is this user under the maximum threshold
established for using system resources?

If a job fails these tests it is rejected at the time of
submission and an error message returned to the user
submitting the job. (In the event that a job is submitted in
anticipation of resources being added to the system, such a
new host architecture, the user can choose to override the
first test by using the "force" option ("-F") in the qsub
command.

2. Once a user job has been accepted into the system it will be
placed into the qmaster's job list where it will remain until
it has been executed or deleted. If a job's submission
exceeds the MAXUJOBS limit placed in the conf_file, it
will remain in the queue BUT it will not be considered
during scheduling passes by the qmaster.

The qmaster conducts an examination (or "pass") over the job list:

a. Every time a job is added to the list
b. Every time a job terminates

c. If neither of these steps occur, the qmaster will scan the list
on a periodic basis based on the number of seconds in the
"SCHEDULE_TIME" parameter in the conf_file.

The scanning process consists of sorting the jobs according to their
submitted priority (the "-p" option), then by an internally generated
"subpriority" and finally by the job sequence number (establishing
its submission order). After the jobs are sorted they are examined
in order, testing each available queue (each ordered by load
average or sequence number) looking for the first one which
matches the resources requested by that job. If a match is found the
job is dispatched and the next job is examined.

Manipulation of a job's subpriority before the sorting step is the
easiest way to affect the basic scheduling algorithm. In DQS331
this simply consists of increasing the subpriority field of a job
based on the number of previously submitted jobs (at the same
priority level) for that user. Thus two or more users with several
jobs queued at the same priority and for the same system resource
will have their jobs interleaved, so that no one user can dominate a
resource by submitting a large quantity of jobs.
The system administrator will probably experiment with this subpriority computation as a first step in customizing DQS. Flirting with the resource matching is considered to be a more risky affair as the side effects of such changes are harder to predict or detect.

**AFS Operation**

DQS331 provides a minimal AFS support capability. The introduction of the "process shepherd" has made the job re-authentication in DQS conform to AFS security requirements. The output file handling feature addresses the 'cross platform' security problems of dealing with stdout and stderr.

**Multi-Cell Operation**

A limited multi-cell operation capability is provided in DQS331. Jobs may be moved from cell to cell if they are not yet in execution, and users authenticated in one cell can view the status of the queues in another cell.

**Accounting**

Site accounting methods vary as widely as any aspect of a batch processing system. DQS331 records as much information as possible about a job's scheduling and execution in a single ASCII line of text. These entries are preceded by an ASCII string of the standard UNIX GMT time of the entry.

Extraction of the accounting information simply requires using a structure definition for the act_file entries in one's "c" extraction program. An example of this technique may be found in the program acte.c which can be found in the ../DQS/tools directory. Included in the tools directory is a script "dostats" which employs acte to create a series of system summary files for the administrator.

**System Management**

The process of DQS system management first consists of laying out the physical and logical structure of a cell. The physical organization is described by adding hosts and assigning them to queues. The logical organization consists of defining resource "complexes" and consumable resources and assigning these to their appropriate queue hosts. Finally setting system parameters in the
conf_file and each queue configuration establishes the operating environment for DQS operation.

The ongoing management steps should include:

a. review of the queue status information to spot queues in UNKNOWN or ALARM state; (DQS331 will send email to the administrator whenever possible, but a sudden crash of a daemon may only be detected from the qstat command display)

b. regular review of the err_file, log_file, stat_file and act_file looking for operational anomalies; Some will be obvious, such as dqs_execd's which have vanished or been restarted. One key thing to look for is a sequence of jobs aborting on the same host (a potential problem with DQS or the host) or a sequence of jobs aborting for the same user (may point to a problem with the user's jobs or the user's permissions). (Job aborts may be detected by examining the exit_status of jobs in the act_file.

c. Changing queue parameters, adding and deleting jobs and performing queue suspend/unsuspend, or disable/enable operations as required.

The majority of the DQS331 utilities set and their options are provided for the system management function. While users may employ the qalter command, for example, to change the characteristics of a submitted job, more often the administrator will avail themselves of this function. A not-uncommon occurrence is for the administrator to increase the submission priority of a job to move it ahead of other jobs in the scheduling.

One utility should be highlighted here, the "qidle" function. Many DQS hosts may actually reside on someone's desk and serve as their personal; workstation. At the same time these machines are utilized for their computational capabilities in a cell. To serve both functions, it must be possible for the workstation user to have priority access to their machine and not suffer keyboard and mouse response deficiencies because the host is being shared with DQS. A first step is to make the "owner" of the workstation also an "owner" of all queues assigned to that host. Then when the workstation owner wishes to have exclusive use of the machine they will have DQS permission to suspend any queues on that machine.
Enter the "qidle" utility. This is an X-Window based program, since we presume that workstation users will be operating with X-Window. It can be started at any workstation and performs the following functions on behalf of the workstation "owner" who the administrator has also designated a queue "owner" in the queue configuration.

a. If the workstation mouse and keyboard are used in some way (mouse movement, button clicks, keyboard typing), all queues on that host are suspended.

b. If the keyboard and mouse have not been used for a period of time specified in the qidle command, then all queue suspensions are removed.

What happens in the case where more than one user may have access to a workstation. The "system console" is an example where many users may be permitted to operate the keyboard and mouse. Making all users "owners" of that station's queues could result in an unmanageable list and is a potential security problem, since a queue owner has privileges beyond queue suspension actions.

The qidle in DQS331 has thus been modified from its DQS 3.1.2.4 form. It is now a member of the DQS331 utilities group and communicates directly with the qmaster rather than indirectly through the qmod utility. It can be started on any workstation by any user who has permission to login to that workstation. Once started it performs the same functions described above.

**Problem Solving**

**Solving Installation Problems**

Most installation difficulties can be divided into three categories (in the order of probability)

a. One or more bugs remain in the DQS 3.3.1 installation procedure. This release has not been tested on all available UNIX platforms (hardware or software versions 0).

b. The interactive interface has produced messages or questions which may confuse the reader. Some of these are natural warnings from the make process or compiler. A few will be labeled "error" when they do not effect the installation process. These often occur when an installation
is being performed over an old one and the target directories already exist.

c. The administrator is running as non-root and attempting operations not permitted in that mode.

d. Host machines to be used for qmaster and/or dqs_exec do not have uniform access (through NFS or AFS or DFS) to the DQS binary files, or the spool directories defined during the installation procedure.

e. Attempts to use qstat331, qsub331, etc receive a message ".. unable to contact qmaster". This is usually due to a user trying to invoke one of the DQS utilities on a host not known to the qmaster. The qmaster maintains a list of all "trusted hosts" in the cell which it manages. Hosts are added automatically when a queue is configured for them: "qmaster331 -aq" or by an explicit host addition "qconf331 -ah <host name > ".

Identify the symptoms of the installation failure and refer to one of the following sections:

**INSTALL fails during the make process of the "config" program.**

The GNU configure program uses the "Makefile.in" template in the DQS/CONFIG directory to produce the Makefile for the DQS config utility. It is possible that a new configuration of compilers or linkers can cause the GNU facility to create an erroneous Makefile. Visually check the Makefile for correctness.

Although DQS331 installation has been tested on many platforms, variants of the compiler or operating systems can create WARNING messages during the compilation which we have not made provision for. Even different versions of GNU "C" yield different warning messages. If the error is fatal to the compilation please contact the DQS331 support team for assistance.

**INSTALL fails during the execution of the DQS config program.**

During the config process the system attempts to create a number of directories and sub-directories. The default starting point for this process is the current working directory of the user if running as non-root or /usr/local/DQS if running as root. If any of the directories exist, an error message is displayed on stdout, but the config program continues. If the user discovers that they have erroneously specified directory names, config can be interrupted by typing CTRL-C. This will unwind many aspects of the configuration process, however NO DIRECTORIES will be
removed. The administrator will have to cleanup any relevant directories manually. After reviewing the directory already exists messages the administrator can choose to ignore those which are expected because the directories were previously created.

**INSTALL fails during the "make" process**

During the DQS config step, all of the target directories are created except for the ones associated with the compiled output object ('.o' files) and the interim executables (qmster, dqs_execd). If a previous installation occurred under a "root" user and the current "make" is being done as a "non-root" the attempt to create the ARCS sub-directories will fail for lack of permissions. The solution is to perform the "make" as root or change the owner of the ARCS sub-directories to the user doing the installation of DQS331.

The GNU CC compiler is chosen as the default compiler or the "make" process if it is available. Some sites may experience a large number of "gcc" warning messages if there have been local modifications to the gnu include files. If this occurs or if the site prefers to use the native "C" compilers then the following steps should be taken:

a. Stop the "make" operation. The GNU configure program and the DQS config utility will have been executed and all Makefile templates will contain the GCC default. Change directory to DQS/SRC and edit the Makefile.proto file.

b. Search the Makefile.proto for any lines which match "CC=gcc" and replace the string "gcc" with the native compiler name, (usually "cc").

c. Change directory back to the base directory, DQS, and type "make" to restart the process.

If only "warning" messages appear in the stdout results you can feel reasonably secure with the installation. However we will try to eliminate these in future releases and would appreciate receiving information on these occurrences. If an error fatal to the compilation occurs please contact the DQS support staff.

**INSTALL fails during the "make installbin" phase**

Once the make process has created the temporary executables in the ARCS directory they should be moved to their "final resting
place" as chosen during the DQS config step. For operational installations this step should be performed as root. If the INSTALL script was started as non-root and the target directory requires root permissions the INSTALL process will fail at this point.

If this occurs the administrator should switch to "root", change directory to /DQS and type "make installbin".

Since the DQS config process attempts to create the BIN target directory, this phase may generate several warning messages that "directory already exists". Ignore these warnings. If, however the message is "error, permission denied", the process should be repeated in "root" mode.

To prevent confusion between DQS331 binaries and previously installed versions we have appended the string "331" during the installbin process. The usual next step is to provide soft-links in /usr/local/bin to these binaries something of the form:

```
ln -s /usr/local/DQS/bin/qaster331 /usr/local/bin/qmaster331
```

**INSTALL fails during the "make instalconf" phase**

After the binaries have been installed in their directory the "resolve_file" and "conf_file" will be moved to their target directory, ( a possible default might be "/usr/local/DQS/common/conf" ). In our "quick install example" this process should proceed automatically. If the INSTALL script was initiated by a non-root user and the destination directory is restricted to a root-user this step will fail with a "permission denied" error message. However when a series of different platform types are being aggregated into a single cell, only one conf_file and resolve_file need be moved to the common/conf directory. If this has already been done then this step can be skipped.

**Startup of the qmaster fails.**

The principle reason for the qmaster not executing during initial testing is the absence of the /etc/services entries directed by the installation process. The err_file should be examined. Warning messages about absent hosts, acl and complex files should be ignored. Look for an entry "Bad Service" which points to the /etc/services file.
An obvious error, but one which occurs often is trying to start the qmaster in user-mode while the RESERVED_PORTS TRUE appears in the conf_file.

If attempts at starting the qmaster fail, after checking root-mode and the /etc/services file, the administrator should set the environment variable DEBUG to 1 and then restart the qmaster as follows: "qmaster331 >&debug.out &" (assuming a C shell environment). After the qmaster crashes send the file "debug.out" to the DQS support staff.

**Startup of the dqs_execd fails**

The principle reason for the dqs_execd not executing during initial testing is the absence of the /etc/services entries on its host as directed by the installation process. The err_file should be examined. Warning messages should be ignored. Look for an entry "Bad Service" which points to the /etc/services file.

An obvious error, but one which occurs often is trying to start the dqs_execd in user-mode while the RESERVED_PORTS TRUE appears in the conf_file.

If the dqs_execd is not able to check-in with the qmaster during dqs_execd startup the daemon will shut down (once executing the dqs_execd will not shut down if the qmaster is absent). Make sure the qmaster is running before attempting to start the dqs_execd.

If attempts at starting the dqs_execd fail, after checking root-mode and the /etc/services file, the administrator should set the environment variable DEBUG to 1 and then restart the qmaster as follows: "dqs_execd331 >&debug.out &" (assuming a C shell environment). After the qmaster crashes send the file "debug.out" to the DQS support staff.

**Startup of qconf fails**

If the first attempt at using qconf produces error messages and the qconf terminates there are several possible causes:

a. The user is attempting to execute qconf in root-mode while the MIN_UID and MIN_GID are non zero. For security reasons root users are not normally permitted to execute DQS utilities unless the MIN_UID is set to zero.

b. qconf is being started in user-mode but the utility itself is NOT owned by root and does not have the permissions for
the owner set correctly. This can occur when a manager uses a path to the ARCS directory for the utility rather than the BIN_DIR target where installbin is supposed to put all DQS binaries.

c. qconf is being started on a host not yet known to the qmaster. Here we have a cart-and-horse situation. We need to use the qconf function to add hosts, but cannot execute qconf because its host is not "legal". The only solution is to initiate qconf on the same host where the qmaster resides.

**qstat display shows queue status as UNKNOWN**

During the initial test phase, the manager will have created one queue using qconf. After it has been created, execution of qstat should show the presence of a queue and a status of DISABLED. An UNKNOWN status indicates a failure of the dqs_execd to contact the qmaster in the time prescribed as MAX_UNHEARD in the conf_file. Check the err_file for messages relating to the dqs_execd being unable to contact the qmaster. Since the dqs_execd would not even start if it could not check in with the qmaster, some new problem must have developed. Check to see if the dqs_execd is still running.

**qsub fails to submit test job**

The test script should be accepted by the DQS system at this point with no problem, since utility<->qmaster interaction has been operating successfully in the previous steps. The most likely reason for a failure of this qsub test is represented by a message of the form "ALARM CLOCK shutdown". This is due to the qmaster or the network interfaces being overburdened. Often the host on which the qmaster is running may be executing some non-DQS managed computational hog. If the ALARM message occurs try increasing the ALARM values in the conf file and re-executing the qsub command. (note that for this experiment the dqs_execd and qmaster need not be restarted after changing the conf_file, as the qsub is the only one complaining. However if the new values of ALARM's parameters prove satisfactory the daemons should be restarted as soon as practicable.)

**test job end with no output**

If the permissions for the user submitting the test script are not sufficient for the target host the job launching process will be terminated and a message sent to the err_file. An accounting
record will also be sent to the DQS act_file. Check these files for information.

**Test script produces a non-zero length stderr file**

The test script should create two output files, one containing stdout information and the other the stderr output. If the stderr output is not zero length than some "very unlikely" event occurred during the job execution. Examine this stderr file and the err_file to determine what the cause was.

**Operational errors**

Once the system has succeeded in running the test script, the administrator will configure hosts, queues and resources for its operational settings. A myriad of situations can then occur which may appear to be, or in fact are, DQS system errors. For this reason DQS produces a large number of informational, warning and error messages which are posted to the system err_file.

In the event that an operational aberration is detected the err_file should be examined closely. If no explanations are obvious, The DQS support staff should be contacted and sent a relevant extraction from the err_file and act_file.