Job Requirements

Synopsis

Jobs can have requirements, which are resources or other conditions which must be available before the job can run. Qube!'s requirement specification is expression based. The syntax for specifying the expression is similar to Perl or C. The proper use of these expressions will allow a user to specify the host and/or the conditions required.

An expression consists of operators and operands. Operators are either text or symbolic. These are equivalent:

```
eq, ==, =
```

String and numeric comparisons are automatically resolved based upon the values they resolve to.

Quoting

Since a job requirement can include a number of operator characters, any reference to a property or resource that includes an operator should be quoted so the interpreter can differentiate between the literal character and the operator.

Operators

Operator	Definition	Expression	Result
min	minimum	10 min 12	10
max	maximum	10 max 12	12
sub, -	subtract	10 sub 8	2
add, +	addition	1 + 2	3
mul, *	multiplication	3 * 4	12
div, /	division	14 / 7	2
xor, ^	XOR	12 xor 8	4
mod, %	modulus	10 % 4	2
in	value in list (string with commas)	"v" in "x,y,v"	true
has	list (string with commas) has value	"x,y,v" has "v"	true
not, !	NOT	not 1	false
eq, =, ==	equal	10 == 10	true
ne, <>, !=	NOT equal	10 != 10	false
and, &&	AND	1 and 0	false
or, ||	OR	1 or 0	true
&	bitwise AND	12 & 8	8
|	bitwise OR	8 | 4	12
lt, <	less than	5 < 10	true
gt, >	greater than	5 > 10	false
le, <=	less than or equal	4 >= 6	false
ge, >=	greater than or equal	4 <= 6	true
rs, >>	bitwise right shift (used to divide by 2^n)	4 >> 1	2
ls, <<	bitwise left shift (used to multiply by 2^n)	4 << 1	8

The reason for multiple definitions for most operators is to allow a programmer more flexibility in the case of Unix command line applications where reserved characters such as ">", unless otherwise escaped, will be interpreted by the shell.

Operands

Operands in Qube! also have a syntax. They all follow a base `class.type` format.

Host.*type* operands

Operand	Values
host.os	"linux", "irix", "winnt", "osx"
host.processor_speed	CPU speed in MHz
host.processor_make	"GenuineIntel", "AuthenticAMD"
host.processor_model	"pentium"
host.kernel_version	Version reported by the operating system.
host.architecture	"intel", "mips"
host.name	Host name
host.groups	Comma delimited list of group names
host.cluster	Cluster specification string
host.state	Host state
host.restrictions	List of restricted cluster specification strings
host.flags	Numeric representation of the Worker's flags
host.qube_version	Worker version of Qube!
host.jobtypes	Comma delimited list of job types
host.flag.*name*	true if the flag exists
host.duty.*property*	Comma delimited list of job properties for jobs on the worker.

Resource operands

are slightly different and include those defined by your administrator host.

Operand (resource)	Values
host.processors.[used | avail | total]	Number of processors available on the worker
host.memory.[used | avail | total]	Memory in Mb available on the worker
host.swap.[used | avail | total]	Swap space available in Mb on the worker

Job operands

The possible operands for a job.*type* are:

Operand	Description
job.name	job name
job.id	job id
job.pid	job's parent id
job.pgrp	job process group
job.priority	job priority

job.label	job's label
job.user	job's owner
job.status	job status
job.prototype,job.type	job type
job.cluster	job's cluster value
job.restrictions	restrictions list
job.kind	user defined job "kind"
job.reservations	job's reservations
job.requirements	job's requirements
job.flags	job's flags numeric value
job.flag.[name]	true if the flag exists
job.kind	job kind

Examples

Syntax	Explanation
`% qbsub --requirements "host.processors.total > 10" set`	Command line example that uses a host resource expression
`host.os eq linux`	Run my job only on Linux hosts
`"host.os == 'winnt' and host.processor_speed >= 3000"`	Run on a Windows machine that has a processor speed of at least 3GHz
`host.name ne "qb001"`	Run my job on any host except qb001
`"maya" in host.jobtypes`	Run the job on a host with the Maya job type
`host.processors.total == 2`	Run my job only on dual processor hosts
`not (job.id in host.duty.id)`	Run my job only if there isn't already one of this job's instances running on it
`job.kind = 'test'` (or any other value, your choice...) `not(job.kind in host.duty.kind)` (Also see How to restrict a host to only one instance of a given kind of job, but still allow other jobs)	Run only one "kind" of job on a worker at the same time (this will allow other kinds of jobs still to run, different from reserving all job slots)