qbwrk.conf File Format

The `qbwrk.conf` file format is very similar to the `qb.conf` format, with just the inclusion of a macro and template inheritance system. Note that quoted strings in the `qbwrk.conf` file *must* be done with *double-quotes*. Using single-quotes may result in unexpected behavior, including values being completely ignored. Click on each section heading below to expand the description.

Comments

Comments are preceded by the "#" character; everything after the character is ignored by the parser.

Example

key = value #and then the comment

#comment key is equal to value

Hostname Expansion

When defining configuration for a range of hosts which are numbered, it is simpler to group them in numeric range definitions. This is accomplished in the header of the configuration:

```
[pfx[000-100]]
```

Example

```
[pfx[000-100]]
```

worker_cluster = /project

- Here we define the configurations for all named Workers pfx000 through pfx100. If you don't want to pad out the numeric portion of the host name, simply leave out the initial "0" characters:

```
[fast[1-100]]
```

```
worker_cluster = /other
```

- Ranged host name entries will accept templates, just like individual host entries.

```
[group]
```

worker_groups = "mygroup"

```
[sqb[10-20]] : group
```

Macro Variables

To simplify definition of templates as well as Worker configurations, the qbwrk.conf also supports the use of macro variables. These are in the format:

- $*template.name*
- "${*template.name*}"

The template names self and this are reserved words and can be used to reference local settings. The variable $_ is also reserved for the local template's name as well as to represent child templates which inherit the template.

Format

```
[template]key = value[host]otherkey = ${template.key}
```
evaluates to:
```
[template]key = value[host]otherkey = value
```

Example

```
[ref]
worker_cluster = /project[logs]
worker_logfile = "/logdir/${_}.workerlog"
```

```
[qb001] : template
```

- inherit template

`worker_cluster = "${ref.worker_cluster}/vfx"`

- For Worker qb001, worker_cluster will evaluate to "/project/vfx".

`[qb002] : template`

- inherit template

worker_cluster = $ref.worker_cluster

- In the case of Worker qb002, worker_cluster is defined to be the worker_cluster entry from template ref: "/project".

Key / Value Pairs

The qbwrk.conf file is broken up into sections containing key/value pair settings. Each section is described by a header between brackets "[" and "]". Section names may only consist of the characters a-z, A-Z, 0-9 and "_". As for the key/value pair format, the key's name is always represented first, followed by an "=" sign and finally the value. While whitespace such as spaces or tabs don't matter, it is important that the key/value pair remain on a single line unless curly braces are used to describe the value.

Format

[section]
key = valuekey = "value1 value2"
key = { value1 value2 }

Example

[qb001]

- hostname

worker_cluster = /projects/movie

- config entry

Predefined Platform Sections

There are a few "special" section names that can be used for the different platforms. These will be associated with the Workers using the respective platform.

> ⚠ The 'default' template is applied to **all** workers, regardless of operating system, and is inherited by every other template. So
>
> - [winnt]
> - [winnt] : default
>
> are equivalent.

- `[default]`
- `[winnt]`
- `[osx]`
- `[linux]`

Example

```
[linux]
worker_cluster = /mylinuxboxes
```

Template Inheritance

The advantage of the macro file format is the use of template inheritance and value replacement. A section is allowed to "inherit" another section's key/value pairs. This can be accomplished by adding a colon and a space-delimited list of templates.

> ⚠ The `qbwrk.conf` file is read from top to bottom. A template can only inherit from an already-defined template that appears before it in the file.

Example

The section:

```
[qube]
key9 = value00

[section]
key = value
key1 = value1

[section2] : section
key = value3

[section3] : section2 qube
```

evaluates to:

```
[qube]
key9 = value00

[section]
key = value
key1 = value1

[section2]
key = value3
key1 = value1

[section3]
key = value3
key1 = value1
key9 = value00
```

The [global_config] Section (New in 6.9-1)

A `[global_config]` section may be defined in the qbwrk.conf file to set up global qbwrk.conf configurations.

Currently, it supports the following two parameters, used primarily to optimize the loading time of qbwrk.conf:

`templates`: explicitly list all template names in the qbwrk.conf file

`non_existent`: explicitly list all non_existent hostnames that are listed in qbwrk.conf

During the loading of the qbwrk.conf file, if a section name like `[name]` is encountered, the "name" is assumed to be a valid hostname of a worker that is online and accessible, and the supervisor tries to look up the IP address from the name.

In certain network setups, if the "name" is actually a qbwrk.conf template name or a hostname of a non-existent (offline or inaccessible) worker, this can slow things down quite a bit since each such lookup needs to time out. In those situations, the above two parameters, "templates" and "non_existent" in the `[global_config]` section can be properly set up to optimize the loading time for qbwrk.conf, which in turn speeds up the execution of `qbadmin w -reconfig` and also the supervisor boot process.

Example:

```
[global_config]

templates = centos,ubuntu

non_existent = render[071-100],render123,render155


[centos]

worker_cluster = /projects/foobar


[ubuntu]

worker_cluster = /projects/secret


[render[001-200]]

worker_groups = "dedicated"
```

In the example above, the section names "centos" and "ubuntu" are listed in the global_config's "templates" parameter, and also the machines "render[071-100],render123,render155" are listed as "non_existent". The supervisor, when loading the qbwrk.conf file, skips the IP address lookup for these listed names, thereby speeding up the process.

Note that in the example, it is assumed that the "non_existent" machines are not online, perhaps taken down for maintenance, and therefore the site administrator has listed them there.

Also note that numerical range expansions are allowed in the value of the "non_existent" parameter (as in "render[071-100]"), to conveniently specify a contiguous chunk of hostnames.

Working Example

As an example, let's say you need all machines to define proxy_account and proxy_password, you would then:

```
[default]
proxy_account = render
proxy_password = cbda878cd5ad5dcdab8967dbc86bc786a857ada57bc # < -
generated by qbhash
```

Then let's say you want to define a worker path map, but it will be different based on the OS, you would then:

```
[osx]
worker_path_map = {
 "\\" : "/",
 "X:/project" : "/Volumes/xsan/project",
 "H:/" : "/home",
}

[winnt]
worker_path_map = {
 "/Volumes/xsan/project" : "X:/project",
 "/home" : "H:/",
 "/" : "\\",
}
```

Now let's say you want hosts01 - host05 to be in a group called "groupA" and a cluster called "/foo"; and host06 - host10 to be in a group called "groupB" and a cluster called "/bar", and then host11-15 need to be in group "groupB" and cluster "/bar" and define a worker_restriction of "/bar/+":

```
[groupa_rule]
worker_groups = groupA
worker_clsuter = /foo

[groupb_rule]
worker_groups = groupB
worker_cluster = /bar

[host[01-05]] : groupa_rule

[host[06-10]] : groupb_rule

[host[11-15]] : groupb_rule
worker_restriction = /bar/+
```

Now let's say that host07 is special and needs to be a member of both groupB and nvidia, but not lose its worker_cluster. Just redefine it afterwards (the file is read top to bottom):

```
[host07]
worker_groups = groupB,nvidia
```

Here's the whole thing together:

```
[default]
proxy_account = render
proxy_password = cbda878cd5ad5dcdab8967dbc86bc786a857ada57bc

[osx]
worker_path_map = {
 "\\" : "/",
 "X:/project" : "/Volumes/xsan/project",
 "H:/" : "/home",
}

[winnt]
worker_path_map = {
 "/Volumes/xsan/project" : "X:/project",
 "/home" : "H:/",
 "/" : "\\",
}

[groupa_rule]
worker_groups = groupA
worker_clsuter = /foo

[groupb_rule]
worker_groups = groupB
worker_cluster = /bar

[host[01-05]] : groupa_rule

[host[06-10]] : groupb_rule

[host[11-15]] : groupb_rule
worker_restriction = /bar/+

[host07]
worker_groups = groupB,nvidia
```

To pull it all together, if host12 were a windows machine, it would get the following config:

```
proxy_account = render # from default
proxy_password = cbda878cd5ad5dcdab8967dbc86bc786a857ada57bc # from
default
worker_path_map = {
 "/Volumes/xsan/project" : "X:/project",
 "/home" : "H:/",
 "/" : "\\",
} # from [winnt]
worker_groups = groupB # from [host12] ( [host[11-15]] ) which inherited
from [groupb_rule]
worker_cluster = /bar # from [host12] ( [host[11-15]] ) which inherited
from [groupb_rule]
worker_restriction = /bar/+ # from [host12] ( [host[11-15]] )
```

- We never had to inherit from [osx] or [winnt], as those were already applied to all OSX and Windows machines before we even got into [host*] sections.
- There are no machines called "group*_rule" - those are just placeholders, if you will.
- You can do all of this through WranglerView on the supervisor (you must be running it on the supervisor for this to work). Simply multi-select workers on the worker layout, right-click > "Configure on supervisor", then change all the parameters you choose and click OK. This is somewhat less flexible, but many people prefer GUIs.