Advanced Dependencies

The next step is to build a more robust dependency system. Building on what we have learnt in the previous tutorials this tutorial will:

- Give example of JobLabel for linking jobs
- Give example of Dependency linking to JobLabel
- Shows correct dependency graph in the Qube! GUI

Feel free to download and run the script below. It sets up a job that will :

- Create a Parent "Sleep job" with a range of 60
- Create a Blocked Child "Sleep job" that links to the Parent job and waits for a complete status before starting

Advanced_Dependency.py

```
#!/usr/bin/python


# Below are required imports for the script to run
import os, sys


# The below few lines of code are to determine the OS of the machine that your running

# this script from and then define the location of the Qube! API
if 'QBDIR' in os.environ:
 sys.path.append('%s/api/python' % os.environ['QBDIR']);
elif os.uname()[0] == 'Darwin':
 sys.path.append('/Applications/pfx/qube/api/python');
elif os.uname()[0] == 'Linux':
 sys.path.append('/usr/local/pfx/qube/api/python');
else:
 sys.path.append('c:/program files/pfx/qube/api/python');


# The below line of code is to import the above defined Qube! API
import qb


# Below is the main function to run in this script
def main():

    # ----------------Start creation of Parent
Job----------------------------------------

    # Below defines an empty list for combining all tasks in the dependancy chain
    task = []

    # Below creates an empty dictionary to be filled by the following lines of code
    job = {}

    # Below defines a label for the dependancy to be used internally within this
script
    job['label']= 'ParentLabel'

    # Below defines the name of the Qube! job
    job['name'] = 'python parent job'

    # Below defines how many Instances/subjobs the job is to spawn
    job['cpus'] = 1
    # Below defines the internal Qube! jobtype to be used to execute the job
    job['prototype'] = 'cmdrange'

    # The below parameters are explained further in the "Job submission with job
package explained" page
    package = {}
    job['package'] = package
    job['package']['cmdline'] = 'sleep QB_FRAME_NUMBER'

 # Below defines the Agenda/Range of the job this will fill the Frames/Work section of
the Qube! GUI
 # "0-60x10" is range 0-60 in chunks of 10 frames
    agendaRange = '0-60x10'

    # Below defines the internal command required to generate the agenda
    agenda = qb.genframes(agendaRange)

    # Below defines the job details for the agenda
    job['agenda'] = agenda

    # Below appends the details of this task to the job dictionary for later
submission
    task.append(job)

 # ----------------Start creation of Child Job-----------------------------------------


 # Below creates an empty dictionary to be filled by the following lines of code
    job = {}

    # Below defines a label for the dependancy to be used internally within this
script
    job['label']= 'ChildLabel'

    # Below defines the dependancy of this job see below for possible dependancy
strings
    job['dependency'] = 'link-complete-job-ParentLabel'

    # Below defines the name of the Qube! job
    job['name'] = 'python child job'

    # Below defines how many Instances/subjobs the job is to spawn
    job['cpus'] = 1

    # Below defines how many Instances/subjobs the job is to spawn
    job['prototype'] = 'cmdrange'

    # The below parameters are explained further in the "Job submission with job
package explained" page
    package = {}
    job['package'] = package
    job['package']['cmdline'] = 'sleep 20'

    # Below appends the details of this task to the job dictionary for later
submission
    task.append(job)

 # Below submits the task list to Qube!
    listOfSubmittedJobs = qb.submit(task)
    # Below prints out a list of jobs that have been submitted by name
    for job in listOfSubmittedJobs:
     print '%(name)15s: %(id)s' % job

# Below runs the "main" function
if __name__ == "__main__":
    main()
    sys.exit(0)
```

This script differs from the rest quite a lot:

`task = []`

To create a list of jobs that are submitted:

`task.append(job)`

Combines the list of jobs for final submission with:

`listOfSubmittedJobs = qb.submit(task)`

With this method the jobs are not submitted per "job = {}" instead combined and submitted once all tasks have completed:

`job['label']= 'ParentLabel'`

This creates a internal label for the job which is assessed at submission time:

`job['dependency'] = 'link-complete-job-ParentLabel'`

The Child job then uses the internal label to link to the Parent job.

Here are some examples of how you can link the jobs:

`job['dependency'] = 'link-complete-job-ParentLabel'`

This will run once the Parent job is complete.

`job['dependency'] = 'link-failed-job-ParentLabel'`

This will run once the Parent job is failed.

`job['dependency'] = 'link-killed-job-ParentLabel'`

This will run if the Parent job has been killed.

`job['dependency'] = 'link-done-job-ParentLabel'`

This will run once the Parent job returns a status of done. Done means if the job completes,fails or has been killed.

You can also link jobs by different types.

`job['dependency'] = 'link-done-work-ParentLabel'`

This will run depending on the status of the jobs work.

`job['dependency'] = 'link-done-subjob-ParentLabel'`

This will run depending on the status of the jobs subjobs.

See Also

Job Dependency Attribute Syntax