

### Rappture Integration with Submit

Parsing error resulted in empty content. Displaying raw markup below.

== Overview ==

It is possible to use the submit command to execute simulation jobs generated by Rappture interfaces remotely. A common approach is to create a shell script which can be exec'd or forked from an application wrapper script. This approach has been applied to TCL, Python, Perl wrapper scripts. To avoid consumption of large quantities of remote resources it is imperative that the submit command be terminated when directed to do so by the application user (Abort button).

```
{{{<br><br>}}}
```

=== Python Wrapper Script ===

Submit can be called from a python Rappture wrapper script for remote batch job submission. An example of what code to insert in the wrapper script is detailed here.

An initial code segment is required to catch the Abort button interrupt.

```
{{{
import os
import sys
import stat
import Rappture
import signal
import re

def sig_handler(signal, frame):
    if Rappture.tools.commandPid > 0:
        os.kill(Rappture.tools.commandPid,signal.SIGTERM)

signal.signal(signal.SIGINT, sig_handler)
signal.signal(signal.SIGHUP, sig_handler)
signal.signal(signal.SIGQUIT, sig_handler)
signal.signal(signal.SIGABRT, sig_handler)
signal.signal(signal.SIGTERM, sig_handler)
}}}
```

A second code segment is used to build an executable script that can be

## RAPPTURE INTEGRATION WITH SUBMIT

---

executed using `Rappture.tools.getCommandOutput`. The trap statement will catch the interrupt thrown when the wrapper script execution is Aborted. Putting the submit command in the background allows for the possibility of issuing multiple submit commands from the script. The wait statement forces the shell script to wait for all submit commands to terminate before exiting.

```
{{{
submitScriptName = 'submit_app.sh'
submitScript      = ""#!/bin/sh

trap cleanup HUP INT QUIT ABRT TERM

cleanup()
{
    echo "Abnormal termination by signal"
    kill -s TERM `jobs -p`
    exit 1
}

""

submitScript += "submit -v u2-grid python foo.py -i bar.in"
submitScript += "\nwait"

submitScriptPath = os.path.join(os.getcwd(),submitScriptName)

fp = open(submitScriptPath,'w')
if fp:
    fp.write(submitScript)
    fp.close()

os.chmod(submitScriptPath,
          stat.S_IRWXU|stat.S_IRGRP|stat.S_IXGRP|stat.S_IROTH|stat.S_IX
OTH)
}}}
```

In the previous piece of code you must edit the following line to accompany what file you want to be remotely executed (e.g. `foo.py`), any input files you may need (e.g. `bar.in`), and the grid you want to run it on (e.g. `u2-grid`):

```
{{{
submitScript += "submit -v u2-grid python foo.py -i bar.in"
}}}
```

## RAPPTURE INTEGRATION WITH SUBMIT

---

Also when running this script on vhub you must make sure to include the path of files if they are not in your {{{<}}}tool{{{>}}}/bin directory. This can be done by using the 'TOOLDIR' environment variable that holds the tool directory in /apps

The standard method for wrapper script execution of commands can now be used. This will stream the output from all submit commands contained in submit\_script.sh to the GUI display. The same output will be retained in the variable stdout.

```
{{{
exitStatus,stdout,stderr = Rappture.tools.getCommandOutput(submit
ScriptPath)
}}}
```

Each submit command creates files to hold COMMAND standard output and standard error. The file names are of the form JOBID.stdout and JOBID.stderr, where JOBID is an 8 digit number. These results can be gathered as follows.

```
{{{
re_stdout = re.compile(".*.stdout$")
re_stderr = re.compile(".*.stderr$")

out2 = ""
errFiles = filter(re_stderr.search,os.listdir(os.getcwd()))
if errFiles != []:
    for errFile in errFiles:
        errFilePath = os.path.join(os.getcwd(),errFile)
        if os.path.getsize(errFilePath) > 0:
            f = open(errFilePath,'r')
            outFileLines = f.readlines()
            f.close()
            stderr = ''.join(outFileLines)
            out2 += '\n' + stderr
            os.remove(errFilePath)

outFiles = filter(re_stdout.search,os.listdir(os.getcwd()))
if outFiles != []:
    for outFile in outFiles:
        outFilePath = os.path.join(os.getcwd(),outFile)
        if os.path.getsize(outFilePath) > 0:
            f = open(outFilePath,'r')
            outFileLines = f.readlines()
            f.close()
            stdout = ''.join(outFileLines)
}}}
```

## RAPPTURE INTEGRATION WITH SUBMIT

---

```
        out2 += 'n' + stdoutput
    os.remove(outFilePath)
}}}
```

The script file should be removed.

```
{{{
os.remove(submitScriptPath)
}}}
```

The output is presented as the job output log.

```
{{{
lib.put("output.log", out2, append=1)
}}}
```

All other result processing can proceed as normal.

A complete file of the following code maybe downloaded here: [\[\[File\(submit.py\)\]\]](#)

```
{{{<br><br>}}}
```

=== Notes ===

If the file that gets called remotely writes to a file (e.g. foo.out) and you want to open that file once the remote file is done executing you must first get the path to the output file:

Instead of this:

```
{{{
output = open('foo.out', 'r')
}}}
```

You must instead open it by preceding the file name with the path:

```
{{{
outputName = 'foo.out'
outputPath = os.path.join(os.getcwd(),outputName)

output = open(outputPath, 'r')
}}}
```

```
{{{<br>}}}
```

## RAPPTURE INTEGRATION WITH SUBMIT

---

You can get help with the submit command by using the {{{--help}}} option

```
{{{
#> submit --help
Usage: submit [options]
```

### Options:

```
-v, --venue           Remote job destination
-i, --inputfile       Input file
-n NCPUS, --nCpus=NCPUS
                        Number of processors for MPI execution
-N PPN, --ppn=PPN     Number of processors/node for MPI execution
-w WALLTIME, --wallTime=WALLTIME
                        Estimated walltime hh:mm:ss or minutes
-e, --env             Variable=value
-m, --manager         Multiprocessor job manager
-M, --metrics         Report resource usage on exit
-W, --wait            Wait for reduced job load before submission
-h, --help            Report command usage
```

Currently available DESTINATIONS are:

```
u2-grid
```

Currently available MANAGERS are:

```
ccni-bgl-CO
ccni-bgl-VN
ccni-opteron_lammps
mpi
parallel
sbbnl-bgl-CO
sbbnl-bgl-VN
sbbnl-bgp-DUAL
sbbnl-bgp-SMP
sbbnl-bgp-VN
```

```
}}}
```

```
{{{<br>}}}
```

For more information please visit: [[https://hubzero.org/documentation/0.9.0/tooldevs/grid.rappture\\_submit](https://hubzero.org/documentation/0.9.0/tooldevs/grid.rappture_submit) HUBzero Submit Documentation]