vmdpipe Documentation

Release 1

Salvatore M Cosseddu

Sep 02, 2016

CONTENTS

CHAPTER ONE

DESCRIPTION:

VMDpipe provides a set of api to use vmd from python executing tcl code and scripts.

CHAPTER

TWO

NOTES:

In the present version, the module does not provide any class, python interpreter in VMD is not supported and only a single VMD instance is allowed.

VMD executable can be set using vmdpipe.Vsetpath(path) and retrieved using vmdpipe.Vgetpath(path)

vmdpipe.printout (boolean) = True is useful for interactive: vmd stdout is printed to screen instead of being returned as strings

defaultTimeout specifies the wait time (in seconds) before an error is raised if VMD does not respond. In this case VMD instance is not closed. You can wait further using vmdpipe.ping() or kill the instance with vmdpipe.Vkill()

ioLag defines a default time interval for reading vmd stdout after sending a command using vmdpipe.send_string()

Module is implemented using subprocess module and vmd stderr is accessible via vmdpipe._vmdin.stderr (See subprocess manual)

Contents:

2.1 vmdpipe package

2.1.1 vmdpipe module

VMDpipe provides a set of api to use vmd from python executing tcl code and scripts.

In the present version, the module does not provide any class, python interpreter in VMD is not supported and only a single VMD instance is allowed.

VMD executable can be set using vmdpipe.Vsetpath(path) and retrieved using vmdpipe.Vgetpath(path)

vmdpipe.printout (boolean) = True is useful for interactive: vmd stdout is printed to screen instead of being returned as strings

defaultTimeout specifies the wait time (in seconds) before an error is raised if VMD does not respond. In this case VMD instance is not closed. You can wait further using vmdpipe.ping() or kill the instance with vmdpipe.Vkill()

ioLag defines a default time interval before reading vmd stdout after sending a command using vmdpipe.send_string()

Module is implemented using subprocess module and vmd stderr is accessible via vmdpipe._vmdin.stderr (See subprocess manual)

`vmdpipe.`**`Vclose`**`(`*timeout=10*`)`
    close the vmd instance opened by Vopen() and return the returncode

`vmdpipe.`**`Vgetpath`**`()`
    get path of vmd used by the module

`vmdpipe.`**`Vkill`**`()`
kill the vmd instance opened by Vopen()

`vmdpipe.`**`Vopen`**`(`*gui=True*, *timeout=15*, *returnInitStdout=False*`)`
open a vmd instance, use only for interactive/test purposes set text to False for interactive use with gui

An error is raised if VMD does not respond within timeout sectonds. In this case VMD instance is not closed. You can wait further, observe using ping() or kill the instance with Vkill()

if returnInitStdout is True, the function return the init stdout of VMD as string

if vmd.printout is True, init stdout is printed to screen (useful for interactive use)

`vmdpipe.`**`Vsetpath`**`(`*p*`)`
set path of vmd, default vmd from bash env

`vmdpipe.`**`aspylist`**`(`*x*`)`
convert tcl list in python list

`vmdpipe.`**`astcllist`**`(`*x*`)`
convert python list in tcl list

`vmdpipe.`**`callback`**`(`*signal*, *capture_stdout*`)`
listen vmd for a signal and capture stdout

`vmdpipe.`**`isVMDopen`**`()`

`vmdpipe.`**`ping`**`(`*timeout=15*, *signal='vmdpipesignal'*`)`
Send signal to vmd and wait timeout seconds for the response. Finally return the stdout

`vmdpipe.`**`runAndReturn`**`(`*script*, *addexit=True*`)`
Execute a vmd script in a independent vmd instance, close and return the stdout. Both file paths and strings as accepted as script. If script is a string, "exit 0" statement is added at the end. This should be generally fine but if, for any reason, you want to change this default behavior, use addexit=False.

`vmdpipe.`**`send_string`**`(`*commandString*, *timeout=15*, *returnAll=False*, *latency=0.01*`)`
send tcl code to vmd instance created with Vopen() - timeout : an error is raised if VMD does not respond within timeout seconds. VMD process is not killed.

> You further observe the process using ping() or kill it using Vkill(). Increase timeout for commands that take long time.

- if returnAll=False (default), function tries to return only the final return value from the tcl interpreter; if returnAll=True, function returns all the tcl stdout from the command as string
- Increase latency if

if vmdpipe.printout is true vmd stdout is not retured but printed on screen, useful for interactive use.

`vmdpipe.`**`source`**`(`*filename*, ***kwargs*`)`
source a file in the vmd instance created with Vopen()

2.2 Tutorial

Vmdpipe provides useful functions to use VMD either iteractively or in a python script. Few example of it usage are here listed.

2.2.1 Interactive mode:

Open an interactive session with GUI:

```python
import vmdpipe as vmd
vmd.Vopen()
```

that correspond to:

```python
import vmdpipe as vmd
vmd.printout=True        # default, VMD output will be printed on screen
vmd.Vopen(text=False)    # default, open vmd
```

Now you can send some command: vmdpipe will try to capture the return value:

```python
molID=vmd.send_string('mol pdbload 1k4c')    # load a molecule and store molID
print("mol {} loaded".format(molID))
```

By default vmdpipe wait 15s before raising an error:

```python
molID=send_string('sleep 20')    # an error is raised
```

Vmd is not killed, but output to that point is lost. This is made to prevent issues to underlying vmd process to block your script or workflow. You can check if VMD is still alive, send a signal to check if it is responsive, or kill it:

```python
if isVMDopen():
    print("I'm still alive!")

try:
    vmd.ping(10)
except:
    vmd.Vkill()
```

If you know your command will take longer than 15s, increase the timeout (in seconds):

```python
t=vmd.send_string('set t test; sleep 20', timeout=100)   # now it is ok!
print(t)
```

As you can see, no return value was captured. Because, by default, send_string will capture return value of the very last command. If you prefere otherwise, you can save all the stdout printed as a result of your command:

```python
t=vmd.send_string("""
set t test
set g {2 3}
set h [list $t $g]
""", returnAll=True)   # now everything is stored
print(t)
```

As seen, send_string() accepts very complex list of commands. Simpler way to do so is using:: “”” ...”“”

Alternatively you can store your commands in a file, and source them:

```python
t=vmd.source("test.tcl")
```

source() accepts same options of send_string().

Vmdpipe provides a function to convert tcl lists in python lists:

```python
# tcl --> python
t=vmd.aspylist(vmd.send_string("set h [list [list 2 3] [list 4 5] [list 6 7]]"))
```

and back:

```
# python --> tcl
t=vmd.aspylist(vmd.send_string("set h [list [list 2 3] [list 4 5] [list 6 7]]"))
```

To close the vmd instance use:

```
vmd.Vclose()                                        # close vmd
```

2.2.2 Text mode:

Text mode is useful for scripting purposes. In a script, it is safest to run the script opening and closing each time a vmd instance. This is done using:

```
runAndReturn(script)
```

However, many times you want to maintain the vmd instance opened and communicate with it. In these cases you can open the session with:

```
import vmdpipe as vmd
vmd.printout=False                  # nothing will be printed to screen
vmd.Vopen(text=True)               # open vmd in text mode
```

and use all functions above described. It is important to have proper communications with the underlying vmd instance. Option "latency" in vmdpipe.send_string() set time interval before reading vmd stdout after sending a command. It can be changed globally by setting:

```
vmd.ioLag=0.001
```

Default (0.01) should be fine in most cases, however you can play a bit reducing it to improve performance or increasing it if you notice vmd hanging.

PYTHON MODULE INDEX

INDEX