pyMedoc Documentation

Release 0.1.0

Cosan Lab

Feb 21, 2018

Contents

Python package for wireless network triggering of Medoc Pathway Thermal Stimulation System

CHAPTER 1

Installation

```
pip install git+https://github.com/cosanlab/pymedoc
```

1.1 Usage

1.1.1 Basic usage

To use this package, simply initialize the Pathway device and call its methods to interface with the Medoc system. pyMedoc automatically tries to establish a connection on initialization. Below are some example. See the API for a full list of commands.

```
from pymed.devices import Pathway
# Obviously this is a fake IP
my_pathway = Pathway(ip='10.10.10.10',port_number=9991)

# Check status
response = my_pathway.check_status()

# Start protocol 18
response = my_pathway.test_program(18)

# Stop a running program
response = my_path.stop()
```

1.1.2 Complete example of timed stimulation

This is a fully working script that ensures the execution of a program designed to deliver 10s of stimulation. The script assumes that program is numbered '100' on the Medoc system internally and configured for external control.

```python
from pymedoc.devices import Pathway
import time

# Get settings from Medoc system
ip = 'X.X.X.X.X'
port = 20121

# We're going to trigger a protocol that stimulates for 10s
# Stimulation occurs at 46.5 deg but the system baseline is 32 deg
# The fastest possible ramp up is 10 deg/s so we need to account for
# ramp up and ramp down time before the trial ends for real
stimulation_time = 10
ramp_time = (46.5 - 32) / 10.
total_time = stimulation_time + ramp_time

# Establish initial connection test
medoc = Pathway(ip,port,verbose=False)

# Pause for a second
time.sleep(1)

# Issue the command to load the program
print("Starting program")
resp = medoc.program(100)

# The system has a variable length "pre-test" phase before a trigger can be sent.
# Any triggers sent during this time will be completely missed by the system.
# More annoying is that the length of the "pre-test" phase is uknowable ahead of time
# Here we use a convenient method built specifically for this situation
# It repeatedly checks the system 'test_state' until its value is'RUNNING'
# where it can reliably get a trigger
# Print each poll attempt with the verbose flag
ready = medoc.poll_for_change('test_state','RUNNING',verbose=True)

# Trigger the start of the stimulation
print("Triggering")
resp = medoc.trigger()

# Wait the duration of stimulation
time.sleep(total_time)

# Send the stop signal
print("Stopping")
resp = medoc.stop()

# Again check until the system has ACTUALLY stopped to do anything else
ready = medoc.poll_for_change('test_state','IDLE',verbose=True)

print("END")
```

1.2 API Reference

This reference provides detailed documentation for all the features in pyMedoc

1.2.1 `pymedoc.devices`: pyMedoc Devices

`class Pathway`(*ip*, *port_number*, *timeout=5.0*, *verbose=True*, *buffer_size=1024*)

Pathway is a class to communicate with the Medoc Pathway thermal stimulation machine.

Parameters

- **`ip`** (*`str`*) – device ip address
- **`port_number`** (*`int`*) – port the device is listening on
- **`timeout`** (*`float`*) – seconds until connection timeouts; default 5s
- **`verbose`** (*`bool`*) – flag whether to print responses; default True
- **`buffer_size`** (*`int`*) – size of connection buffer; default 1024

`call`(*command*, *protocol=None*, *reuse_socket=False*, *verbose=False*)

Send command to device.

Parameters

- **`command`** (*`str/int`*) – command name or command_id number to send to device
- **`protocol`** (*`str/int`*) – protocol number on device to issue command to (only needed for command TEST_PROGRAM)
- **`reuse_socket`** (*`bool`*) – try to reuse the last created socket connection; *NOT CURRENTLY FUNCTIONAL*
- **`verbose`** (*`bool`*) – whether to print out the device callback

Returns response from Medoc system

Return type response (dict)

`poll_for_change`(*to_watch*, *desired_value*, *poll_interval=0.5*, *poll_max=-1*, *verbose=False*, *server_lag=1.0*, *reuse_socket=False*)

Poll system for a value change. Useful for waiting until the Medoc system has transitioned to a specific state in order to issue another command, but the transition length is unknowable.

Parameters

- **`to_watch`** (*`str`*) – the response field we should be monitoring; most often 'test_state' or 'pathway_state'
- **`desired_value`** (*`str`*) – the desired value of the field to wait on, i.e. keep checking until response_field has this value
- **`poll_interval`** (*`float`*) – how often to poll; default .5s
- **`poll_max`** (*`int`*) – upper limit on polling attempts; default -1 (unlimited)
- **`verbose`** (*`bool`*) – print poll attempt number and current state
- **`server_lag`** (*`float`*) – sometimes if the socket connection is pinged too quickly after a value change the subsequent command after this method is called can get missed. This adds an additional layer of timing delay before returning from this method to prevent this; default 1s
- **`reuse_socket`** (*`bool`*) – try to reuse the last created socket connection; *NOT CURRENTLY FUNCTIONAL*

Returns whether desired_value was achieved

Return type *status* (bool)

status()
Convenience method.

program(*protocol*)
Convenience method.

start()
Convenience method.

pause()
Convenience method.

trigger()
Convenience method.

stop()
Convenience method.

abort()
Convenience method.

yes()
Convenience method.

no()
Convenience method.

CHAPTER 2

API

CHAPTER 3

Usage

Index