TERMITE OS

THE TELESCOPE OPERATING SYSTEM

termiteOS Documentation

Release alpha

Nacho Mas

July 28, 2018

CONTENTS

A telescope Operating System base on zmq and Protocol Buffers. Documentation https://nachoplus.github.io/termiteOS/index.html

CHAPTER ONE

INTRODUCTION

Termite OS is a modular, adaptable, easily extendable telescope operating system developed primarily, but not exclusively, in python.

It wants to answer to several limitations that most of the commercial mount controllers have. It wants to be a platform where you can incorporate all kinds of functionalities that the professional or amateur astronomer may need in a simple way.

Posible use case:

- To motorize a DIY mount or retrofit a old one.
- Make your telescope be able to track the ISS or any other satellite.
- Implement new protocol commands.
- New native protocols, i.e. indilib
- Control over the objects catalogs built into your telescope.
- Integrate pointing model in your mount.
- Connect a GPS to your mount and use his data for location and time.
- Define the horizon of your observatory.
- WiFi or Bluetooth access to your mount
- Develope new motion estrategies.

With termiteOS you will address all this things and more.

Termite OS is a **work in progress** but much of the functionality is already available:

- Stepper controller using Raspberry PI and the integrated DRV8825 widely used in the 3D printer world
- LX200 command set
- Slew and celestial track
- Satellite tracking

Ongoing funtionality:

- Arduino base hardware
- BLDC motors
- Servo motors
- Web interface
- Constellation pointing

CHAPTER TWO

ARCHITECTURE

Each termiteOS functionality is implemented as a separate program called a'node'. The nodes communicate each other using the zmq protocol. The organization between the nodes is hierarchical thus a node can have several children but has only one parent or none in the case of the'root node'.

ZMQ (http://zeromq.org/) is used for transport and on https://developers.google.com/protocol-buffers/ for message definitions and serialization.

Each node has its own ZMQ port and a set of commands and responds to through that port. Each node opens connections with its parent node and with all its children so that messages can be exchanged.

These nodes can run on the same or different CPUs taking advantage of all the features of the ZMQ protocol.

CHAPTER

THREE

LICENSE

CHAPTER

FOUR

TECHNICAL INFORMATION

Launcher

- **Launcher** program launch a set of node following the instruction in yaml file

This program allow to launch at once all nodes needed for a specific hardware/funtionality.

Example of a yaml configuration file:

```
simple:
 type: telescope
 host: localhost
 port: 5000
 nodes:
    - LX200:
         type: tcpproxy
         host: localhost
         port: 5001
         params: {'tcpport':6001,'End':'#'}
    - tracker:
         type: TLEtracker
         host: localhost
         port: 5002
         nodes:
            - trackertcp:
                 type: tcpproxy
                 host: localhost
                 port: 5003
                 params: {'tcpport':6003}
```

This example is equivalent to run on the command shell all following commands:

```
miteTelescope   --port 5000 --name simple
mitetcpproxy    --port 5001 --name LX200 --parent_host localhost --parent_port 5000 --params {'tc
miteTLEtracker  --port 5002 --name tracker --parent_host localhost --parent_port 5000
mitetcpproxy    --port 5003 --name trackertcp --parent_host localhost --parent_port 5002 --params
```

You can find other examples in ‘termoteOS/machines/’

Command line:

miteLaunch

Launch all nodes defined in YAMLFILE

```
miteLaunch [OPTIONS] YAMLFILE
```

Arguments

YAMLFILE
Required argument

API

Launch tools to run a rig. Run several daemons at once

`termiteOS.launch.`**`launchmachine`**(*yamlfile*)
Launch an arragement of daemons defined in a yaml file

`termiteOS.launch.`**`launchnode`**(*nodedict*, *parent_host=''*, *parent_port=False*)
Launch a node defined in dictionary. Called recursively

`termiteOS.launch.`**`run_in_separate_process`**(*func*, **args*, ***kwds*)
Run function in a separate process. To background executables

Nodes

Nodes are separate programs. The nodes communicate each other using the zmq protocol. The organization between the nodes is hierarchical thus a node can have several children but has only one parent or none in the case of the'root node'.

Nodes are based on http://zeromq.org/ for transport and on https://developers.google.com/protocol-buffers/ for message definitions and serialization.

Each node has its own ZMQ port and a set of commands and responds to through that port. Each node opens connections with its parent node and with all its children so that messages can be exchanged.

These nodes can run on the same or different CPUs taking advantage of all the features of the ZMQ protocol.

All nodes are derive from nodeSkull base class which implements all the comunication logic and basic commands.

TLEtracker

- **TLEtracker** node calculate satellite TLE speed and RA/DEC and send to the mount

With this node the mount is able to follow any object with TLE.

Command line:

miteTLEtracker

Launch a TLEtracker node

```
miteTLEtracker [OPTIONS]
```

Options

`--name` `<name>`
module name

`--port` `<port>`
Port listen

`--parent_host` `<parent_host>`
Parent host to connect to. If False the node become a ROOTHUB

--parent_port <parent_port>
Parent port to connect to

API

TLEtracker

class `termiteOS.nodes.TLEtracker.`**`TLEtracker`**(*name*, *port*, *parent_host*, *parent_port*)
Command to the parent node to track satellites base on his TLE

`HasChildren`()

Return **True** if the node has childrens. **False** otherwise

`addCMDs`(*CMDs*)
add commands explicitely

`circle`(*re*, *dec*, *r*, *v*)
Used as test

`cmd`(*cmd*)
Execute the cmd command

Parameters **`cmd`** – string contain the command with his parameters

Returns A message containing the answer

`cmd_follow`(*arg*)
Follow satellite

`cmd_help`(*arg*)
Print help text. Do nothing. Normaly overloaded by a child class

`cmd_ls`(*arg*)
list the commands

`cmd_nodes`(*arg*)
list the children nodes:

Returns a python list with all the children names

`cmd_ping`(*arg*)

`cmd_tree`(*arg*)
Print all node and children availabled commands

`cmddummy`(*arg*)
Default cmd to execute when not knowed cmd match. Do nothing

`deregister`(*arg*)
Close the zmq socket and deleted node from the children list

`end`(*arg=''*)
End all

`exenodeCmd`(*arg*)
Execute the command in the children

`gearInit`()
Get the gear info

`heartbeat`(*arg*)
heartbeat Parent part

`nodeheartbeat`(**args*, ***kwargs*)

`observerInit`()
Recover Observer data (lat, lon,...) from parent node

register(*arg=''*)
Call to parent registrar

registrar(*arg*)
Registrar parent part

run()
Main loop. Obtain RA/DEC actual values and do the trackSatellite() call

satPosition(*sat*)
Calculate satellite RA/DEC from TLE

satSpeed(*sat*)
Calculate satellite speed from TLE

scanCMDs()
scanCMD add all methods starting with '**cmd_**' as a commands

sendSlew(*RA, DEC*)
send slew to parent node primitive

sendTrackSpeed(*vRA, vDEC*)
send speed to parent node primitive

signal_handler(*signal, frame*)
Capture Ctril+C key

trackSatellite(*sat*)
send track speed to parent node. If to far from target send also a slew

zmqQueue(**args, **kwargs*)

termiteOS.nodes.TLEtracker.**runTLEtracker**(*name, port, parent_host='', parent_port=False*)
ENTRYPOINT calling this fuction start the node

hub

- **hub** node has not own commands. Only used to connect other nodes

Command line:

miteHub

Launch a hub node

```
miteHub [OPTIONS]
```

Options

--name <name>
module name

--port <port>
Port listen

--parent_host <parent_host>
Parent host to connect to. If False the node become a ROOTHUB

--parent_port <parent_port>
Parent port to connect to

API

Conection HUB

class `termiteOS.nodes.hub.`**`hub`**(*name*, *port*, *parent_host*, *parent_port*)

`HasChildren`()

Return **True** if the node has childrens. **False** otherwise

`addCMDs`(*CMDs*)
add commands explicitely

`cmd`(*cmd*)
Execute the cmd command

Parameters **`cmd`** – string contain the command with his parameters

Returns A message containing the answer

`cmd_help`(*arg*)
Print help text. Do nothing. Normaly overloaded by a child class

`cmd_ls`(*arg*)
list the commands

`cmd_nodes`(*arg*)
list the children nodes:

Returns a python list with all the children names

`cmd_ping`(*arg*)

`cmd_tree`(*arg*)
Print all node and children availabled commands

`cmddummy`(*arg*)
Default cmd to execute when not knowed cmd match. Do nothing

`deregister`(*arg*)
Close the zmq socket and deleted node from the children list

`end`(*arg=''*)
End all

`exenodeCmd`(*arg*)
Execute the command in the children

`heartbeat`(*arg*)
heartbeat Parent part

`nodeheartbeat`(**args*, ***kwargs*)

`register`(*arg=''*)
Call to parent registrar

`registrar`(*arg*)
Registrar parent part

`run`()
Dummy. Normaly overloaded by a child class

`scanCMDs`()
scanCMD add all methods starting with '**cmd_**' as a commands

`signal_handler`(*signal*, *frame*)
Capture Ctril+C key

`zmqQueue`(**args*, ***kwargs*)

`termiteOS.nodes.hub.`**`runhub`**`(`*name*, *port*, *parent_host=''*, *parent_port=False*`)`

joystick

- **joystick** node to manual move the mount with a joystick

Command line:

miteJoy

Launch a joystick node

```
miteJoy [OPTIONS]
```

Options

`--name` `<name>`
module name

`--port` `<port>`
Port listen

`--parent_host` `<parent_host>`
Parent host to connect to. If False the node become a ROOTHUB

`--parent_port` `<parent_port>`
Parent port to connect to

API

`termiteOS.nodes.joystick.`**`runjoystick`**`(`*name*, *port*, *parent_host*, *parent_port*`)`

class `termiteOS.nodes.joystick.`**`stick`**`(`*name*, *port*, *parent_host*, *parent_port*`)`

`HasChildren``()`

Return **True** if the node has childrens. **False** otherwise

`addCMDs``(`*CMDs*`)`
add commands explicitely

`cmd``(`*cmd*`)`
Execute the cmd command

Parameters **`cmd`** – string contain the command with his parameters

Returns A message containing the answer

`cmd_help``(`*arg*`)`
Print help text. Do nothing. Normaly overloaded by a child class

`cmd_ls``(`*arg*`)`
list the commands

`cmd_nodes``(`*arg*`)`
list the children nodes:

Returns a python list with all the children names

`cmd_ping``(`*arg*`)`

cmd_tree (*arg*)
Print all node and children availabled commands

cmddummy (*arg*)
Default cmd to execute when not knowed cmd match. Do nothing

deregister (*arg*)
Close the zmq socket and deleted node from the children list

end (*arg=''*)
End all

exenodeCmd (*arg*)
Execute the command in the children

heartbeat (*arg*)
heartbeat Parent part

nodeheartbeat (**args, **kwargs*)

register (*arg=''*)
Call to parent registrar

registrar (*arg*)
Registrar parent part

run ()

scanCMDs ()
scanCMD add all methods starting with '**cmd_**' as a commands

sendTrackSpeed (*vRA, vDEC*)

signal_handler (*signal, frame*)
Capture Ctril+C key

zmqQueue (**args, **kwargs*)

tcpproxy

- **tcproxy** node acts as a proxy connector between other node and an especific TCP port

All commands recived throught the TCP port are relay to the myCmdPort port of the node conected to.

Using this node allow us to connect to a specific node (the parent node of tcpproxy node) with a regular *telnet host port*

Command line:

mitetcpproxy

Launch a tcpproxy node

```
mitetcpproxy [OPTIONS]
```

Options

--name <name>
module name

--port <port>
Port listen

--parent_host <parent_host>
Parent host to connect to. If False the node become a ROOTHUB

--parent_port <parent_port>
Parent port to connect to

--params <params>
Dictionary with extra parameters default={"tcpport":6001,"End":"#"}

API

To mimic a tty serial port: *socat TCP:localhost:6000,reuseaddr pty,link=/tmp/lx200*

`termiteOS.nodes.tcpproxy.`**`runtcpproxy`**(*name*, *port*, *parent_host*, *parent_port*, *params*)
ENTRYPOINT calling this fuction start the node

class `termiteOS.nodes.tcpproxy.`**`tcpproxy`**(*name*, *port*, *parent_host*, *parent_port*, *params*)

`HasChildren`()

Return **True** if the node has childrens. **False** otherwise

`addCMDs`(*CMDs*)
add commands explicitely

`clientthread`(*conn*, *parent_host*, *parent_port*)
Function for handling connections. This will be used to create threads

`cmd`(*cmd*)
Execute the cmd command

Parameters **cmd** – string contain the command with his parameters

Returns A message containing the answer

`cmd_help`(*arg*)
Print help text. Do nothing. Normaly overloaded by a child class

`cmd_ls`(*arg*)
list the commands

`cmd_nodes`(*arg*)
list the children nodes:

Returns a python list with all the children names

`cmd_ping`(*arg*)

`cmd_tree`(*arg*)
Print all node and children availabled commands

`cmddummy`(*arg*)
Default cmd to execute when not knowed cmd match. Do nothing

`deregister`(*arg*)
Close the zmq socket and deleted node from the children list

`end`(*arg=''*)
Close all and exit

`exenodeCmd`(*arg*)
Execute the command in the children

`heartbeat`(*arg*)
heartbeat Parent part

`nodeheartbeat`(**args*, ***kwargs*)

recv_end(*conn*)
: Parse cmd lines

register(*arg=''*)
: Call to parent registrar

registrar(*arg*)
: Registrar parent part

run()
: Main loop. Dispach incoming messages

scanCMDs()
: scanCMD add all methods starting with '**cmd_**' as a commands

signal_handler(*signal, frame*)
: Capture Ctril+C key

startserver(*port*)
: Function to open the TCP incoming port

zmqQueue(**args, **kwargs*)

Telescope

- **telescope** node implement a telescope mount basic commands (goto,slew,track..)

Two axis equatorial mount telescope.

Command line:

miteTelescope

Launch a telescope node

```
miteTelescope [OPTIONS]
```

Options

--name <name>
: module name

--port <port>
: Port listen

--parent_host <parent_host>
: Parent host to connect to. If False the node become a ROOTHUB

--parent_port <parent_port>
: Parent port to connect to

API

ENGINE

termiteOS.nodes.telescope.**runtelescope**(*name, port, parent_host='', parent_port=False*)
: **ENTRYPOINT** calling this fuction start the node

class termiteOS.nodes.telescope.**telescope**(*name, port, parent_host, parent_port*)

HasChildren()

> **Return** **True** if the node has childrens. **False** otherwise

addCMDs(*CMDs*)
: add commands explicitely

altAz_of(*ra*, *dec*)

cmd(*cmd*)
: Execute the cmd command

> **Parameters** **cmd** – string contain the command with his parameters
>
> **Returns** A message containing the answer

cmd_ack(*arg*)

cmd_align2target(*arg*)

cmd_firware_date(*arg*)

cmd_firware_ver(*arg*)

cmd_getLocalDate(*arg*)

cmd_getLocalTime(*arg*)

cmd_getSideralTime(*arg*)

cmd_getTargetDEC(*arg*)

cmd_getTargetRA(*arg*)

cmd_getTelescopeDEC(*arg*)

cmd_getTelescopeRA(*arg*)

cmd_help(*arg*)
: Print help text. Do nothing. Normaly overloaded by a child class

cmd_info(*arg*)

cmd_ls(*arg*)
: list the commands

cmd_nodes(*arg*)
: list the children nodes:

> **Returns** a python list with all the children names

cmd_ping(*arg*)

cmd_pulseE(*arg*)

cmd_pulseN(*arg*)

cmd_pulseS(*arg*)

cmd_pulseW(*arg*)

cmd_setMaxSlewRate(*arg*)

cmd_setTargetDEC(*arg*)

cmd_setTargetRA(*arg*)

cmd_slew(*arg*)

cmd_slewRate(*arg*)

cmd_stopSlew(*arg*)

cmd_tree(*arg*)
: Print all node and children availabled commands

`cmddummy`(*arg*)
  Default cmd to execute when not knowed cmd match. Do nothing

`deregister`(*arg*)
  Close the zmq socket and deleted node from the children list

`end`(*arg=''*)

`exenodeCmd`(*arg*)
  Execute the command in the children

`getDEC`(*arg*)

`getGear`(*arg*)

`getObserver`(*arg*)

`getRA`(*arg*)

`heartbeat`(*arg*)
  heartbeat Parent part

`hourAngle`(*ra*)

`nodeheartbeat`(**args*, ***kwargs*)

`observerInit`()

`register`(*arg=''*)
  Call to parent registrar

`registrar`(*arg*)
  Registrar parent part

`run`()

`scanCMDs`()
  scanCMD add all methods starting with '**cmd_**' as a commands

`setTrackSpeed`(*arg*)

`signal_handler`(*signal*, *frame*)
  Capture Ctril+C key

`track`()

`values`(*arg*)

`zmqQueue`(**args*, ***kwargs*)

Drivers

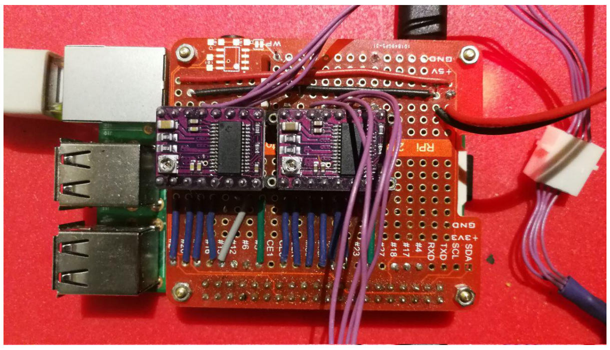
Drivers interact with the real hardware and normaly are used in the **node** code.

Motors

Raspberry Drivers

rpiDRV8825Hat

This is a DIY Raspberry Pi Hat base on the popular stepper control chip DRV8825



API DIY DRV8825 driver Hat interface.

This board has two DRV8825 able to driver 2 motors See hardware on termiteOS/driver/rpi/hardware

INTERFACE TO OTHER MODULES:

- motorBeta
- pinout
- microsteps
- clutch()
- reset()
- sleep()
- set_microsteps(microsteps)
- sync(motorBeta)

class `termiteOS.drivers.rpi.rpiDRV8825Hat.`**`rpiDRV8825Hat`**(*raspberry*, *driverID*, ***kwds*)

This class define the PIN mapping and basic methods for the rpiDVR8825 Hat

Note: Posible values of driverID can be 0 or 1.

`clutch`(*ON_OFF*)
Engage or Disengage(free spinnig) the motors

`fault`(*gpio*, *level*, *tick*)
Callback function to check internal driver faults

`reset`(*ON_OFF*)
Reset the driver circuit

`set_dir`(*dir*)
Set the direction of motion

`set_microsteps`(*microsteps*)
Set microstepping mode of the driver

`sleep`(*ON_OFF*)
Sleep or wake up the driver circuit

`stepcounter`(*gpio*, *level*, *tick*)
Callback function to update internal position counter

`sync`(*position*)
Set the actual internal position == position

`test`()
Test the Hat sending steps

rpiSpeedPWM

A PWM driver

API Raspberry PWM motor driver.

INTERFACE:

- **inherits several methods from rpiDRV8825Hut base class**
  - betaMotor
  - pinout
  - microsteps
  - clutch()
  - reset()
  - sleep()
  - set_microsteps(microsteps)
  - sync(position)
- **Own methods:**
  - setSpeed (radians/seconds)
  - setRPM(RPM)
  - SetPoint(setpoint)
  - goto() -> absolute SetPoint
  - move() -> relative SetPoint
  - stop()
  - isStopped
  - gotoEnd
  - pos

class `termiteOS.drivers.rpi.rpiSpeedPWM.`**`rpiSpeedPWM`**(*driverID*, *microsteps*, *FullTurnSteps*, *gear=1*, *name='Axis'*, *raspberry='localhost'*)

This class do the PWM control calling the underlying pigpiod daemon.

Note: Up to dates only PID control is implemented.

`SetPoint`(*setpoint*)
Establish the _SetPoint value

`clutch`(*ON_OFF*)
Engage or Disengage(free spinnig) the motors

`fault`(*gpio*, *level*, *tick*)
Callback function to check internal driver faults

`goto`(*setpoint*, *blocking=False*)
Absolute movement

`gotoEnd`
True if the axis finally arrive to destination(_SetPoint), False otherwise

isStopped
True if is stopped, False otherwise

move (*relsetpoint*)
Relative movement

pos
Actual position (Corrected motorBeta)

rampUp (*v*, *deltaT*, *out_min=-750*, *out_max=750*)
Limit motor speed changes to avoid axis stalling

reset (*ON_OFF*)
Reset the driver circuit

run (**args*, ***kwargs*)

setRPM (*rpm*)
Set and start PWM to obtain rpm

setSpeed (*v*)
Set and start PWM to obtain radians/seconds

set_dir (*dir*)
Set the direction of motion

set_microsteps (*microsteps*)
Set microstepping mode of the driver

sleep (*ON_OFF*)
Sleep or wake up the driver circuit

stepcounter (*gpio*, *level*, *tick*)
Callback function to update internal position counter

stop ()
Not implemented

stopPWM ()
Stop PWM generation without any check

sync (*newposition*)
Establish newposition as current possition (motorBeta)

test ()
Test the Hat sending steps

trackSpeed (*trackSpeed*)
Set axis track speed. Track speed*timestep is add to the _SetPoint value

`termiteOS.drivers.rpi.rpiSpeedPWM.`**threaded** (*fn*)
Multithread wrapper. Used as a function decorator

Miscellaneus

TBD

Indices and tables

- genindex
- modindex
- search