enuActor Documentation

Release

Author

June 05, 2014

Contents:

CHAPTER

ONE

ENUACTOR PACKAGE

1.1 Subpackages

1.1.1 enuActor.Commands package

Submodules

enuActor.Commands.BiaCmd module

enuActor.Commands.EnuCmd module

enuActor.Commands.ShutterCmd module

Module contents

1.1.2 enuActor.Devices package

Submodules

enuActor.Devices.Device module

class enuActor.Devices.Device.**Device**(*actor=None*, *cfg_path=None*)

Bases: enuActor.QThread.QThread

All device (Shutter, BIA,...) should inherit this class

Attributes:

- link : TTL, SERIAL or ETHERNET
- ser : serial object from serial module @todo: change into link object
- mode : operation or simulated
- cfg_path : path of the communication and parameter config files It should contains *devices_communication.cfg* and *devices_parameters.cfg* file.

available_link = ['TTL', 'SERIAL', 'ETHERNET']

cfg_files = {'communication': 'devices_communication.cfg', 'parameters': 'devices_parameters.cfg'}

fail(**args*)

getStatus()

return status of shutter (FSM)

Returns 'LOADED','IDLE','BUSY',...

handleTimeout()
Override method QThread.handleTimeout(). Process while device is idling.

Returns @todo

Raises CommErr

initialise()

Overriden by subclasses:

- (Re)Load parameters from config files
- Check communication

Todo

Add load cfg file routine

---

load_cfg(*device*)
Load configuration file of the device.

Parameters **device** (*str.*) – name of the device ('SHUTTER','BIA',...)

Returns dict config

Raises CfgFileErr`

printstateonchange(*e*)
What to display when state change

Parameters **e** – event

send(*input_buff=None*)
To be overriden virtual method

startFSM()
Instantiate the MyFSM class (create the State Machine).

start_communication(**args*, ***kwargs*)
To be overriden virtual method

start_ethernet()
To be overriden virtual method

start_serial(*input_buff=None*)
To be overriden virtual method

start_ttl()
To be overriden virtual method

class enuActor.Devices.Device.**DualModeDevice**(*actor=None*)
Bases: enuActor.Devices.Device.OperationDevice, enuActor.Devices.Device.SimulationDevice

Switch between class following the device mode

check_status()

send(**args*, ***kwargs*)

start_communication(**args*, ***kwargs*)

start_ethernet(**args*, ***kwargs*)

start_serial(**args*, ***kwargs*)

start_ttl(**args*, ***kwargs*)

class enuActor.Devices.Device.**OperationDevice**(*actor=None*, *cfg_path='/home/tpegot/mhsls/devel/enuActor/python/en*
Bases: enuActor.Devices.Device.Device

Device in operation mode:

•Communication is implemented

•Starting, sending and receiving message is implemented

op_send(*input_buff=None*)
Send string to interface

Parameters **input_buff** (*str.*) – string to send to check com.

Returns returns from com.

Raises CommErr

op_start_communication(**args*, ***kwargs*)
Docstring for start_communication.

Note: Need first to specify config file and device by calling `load_cfg()` or in the header of `start_communication()`

Parameters

- **device** (*str.*) – device name
- **startCmd** (*str.*) – starting command to check the communication
- ****kwargs** – remaining keywords are not treated

Returns Communication object (example: `serial.Serial` object)

Raises CfgFileErr

op_start_ethernet()
@todo: Docstring for start_ethernet.

Returns @todo

Raises @todo

op_start_serial(*input_buff=None*)
Start a serial communication

Parameters **input_buff** (*str.*) – Send at start to check communication

Returns `serial.Serial`

op_start_ttl()
@todo: Docstring for start_ttl.

Returns @todo

Raises @todo

class enuActor.Devices.Device.**SimulationDevice**(*actor=None*,
cfg_path='/home/tpegot/mhsls/devel/enuActor/python/enuActor/Dev
Bases: enuActor.Devices.Device.Device

Device in simulation mode:

Almost nothing

sim_check_status()

sim_send(*input_buff=None*)

sim_start_communication(**args*, ***kwargs*)

sim_start_ethernet()

sim_start_serial(*input_buff=None*)

sim_start_ttl()

enuActor.Devices.Device.**interlock**(*self_position*, *target_position*, *target*)
Interlock between self device and target device

Note: Choice of iterable is exclusive either `self_position` or `target_position`

Parameters

- **self_position** (*str, int, float, iterable (list, tuple,...)*) – position(s) from current device class ('*' for all position)
- **target_position** (*str, int, float, iterable.*) – position from target device class. ('*' for all position)
- **target** – target device class

enuActor.Devices.Error module

exception enuActor.Devices.Error.**CfgFileErr**(*reason*, *lvl=1*)
Bases: enuActor.Devices.Error.RuleError

Docstring for CommErr. .. todo:: Specify file error

exception enuActor.Devices.Error.**CommErr**(*reason*, *lvl=1*)
Bases: enuActor.Devices.Error.RuleError

CommErr are all the error related to the communication between PC and Device.

exception enuActor.Devices.Error.**DeviceErr**(*device*, *reason*, *lvl=1*)
Bases: enuActor.Devices.Error.RuleError

DeviceErr are all the error related to the device and controller. When a DeviceErr occures the current state of the FSM go to fail.

exception enuActor.Devices.Error.**RuleError**(*reason*, *lvl=1*)
Bases: exceptions.Exception

Define rule and how it is displaied

PRIORITY_DEFAULT = 1

errno

strerror

enuActor.Devices.bia module

class enuActor.Devices.bia.**Bia**(*actor=None*)
Bases: enuActor.Devices.Device.DualModeDevice

SW device: BIA

Attributes:

- currPos : current position of the BIA

bia(**args*)

initialise()

Initialise Bia :

- Load *cfg/device_parameters.cfg* file
- ...todo

Returns @todo

Raises @todo

op_check_status()
Can not check status yet

setConfig(*freq=None*, *dur=None*, *intensity=None*)
It specifies parameters for light and strobe mode.

Note: Default parameters are located in *cfg/devices_parameters.cfg* file. This function only change default parameters of session.

Todo

Check values and types

Parameters

- **freq** – frequency of strobe mode in *Hz*
- **dur** – duration of strobe mode in :math:‘mu‘s
- **intensity** – intensity of light

Returns @todo

Raises @todo

enuActor.Devices.shutter module

class enuActor.Devices.shutter.**Shutter**(*actor=None*)
Bases: enuActor.Devices.Device.DualModeDevice

SW device: Shutter

Attributes:

- currPos : current position of the shutter

MASK_ERROR_SB_1 = [0, 0, 1, 1]

MASK_ERROR_SB_3 = [1, 1, 1, 1, 1, 1, 1]

MASK_ERROR_SB_4 = [0, 0, 1, 1]

MASK_ERROR_SB_5 = [1, 1, 1, 1, 1, 1, 1]

MASK_ERROR_SB_6 = [0, 0, 1, 1]

STATUS_BYTE_1 = ['S_blade_A_offline', 'S_blade_B_offline', 'S_CAN_comm_error', 'S_error_interlock']

STATUS_BYTE_3 = ['S_motor_to_origin_timeout', 'S_threshold_error', '', 'S_limit_switch', 'S_unknown_command', 'S

STATUS_BYTE_4 = ['S_blade_open', 'S_blade_closed', 'S_error_LED', 'S_error_interlock']

STATUS_BYTE_5 = ['S_motor_to_origin_timeout', 'S_threshold_error', '', 'S_limit_switch', 'S_unknown_command', 'S

STATUS_BYTE_6 = ['S_blade_open', 'S_blade_closed', 'S_error_LED', 'S_error_interlock']

initialise(**args*)

op_check_status()
Check status byte 1, 3, 4, 5 and 6 from Shutter controller and return current list of status byte.

Returns [sb1, sb3, sb5, sb6] with sbi list of byte from status byte

Raises CommErr

parseStatusByte(*sb*)
Send status byte command and parse reply of device

Parameters **sb** – byte 1, 3, 4, 5 or 6

Returns array_like defining status flag

Raises CommErr

positions = ['undef.', 'open', 'closed(A)', 'closed(B)']

shutter(**args*)

shutter_id = ['red', 'blue', 'all']

terminal()
launch terminal connection to shutter device

Returns @todo

Module contents

:RivCreateContent * Contents:

Convention naming

The aim of this interface is to follow this naming convention at large:

`enu <device> <command> [arguments [= value]]`

Also others convention are defined like those for motorized devices:

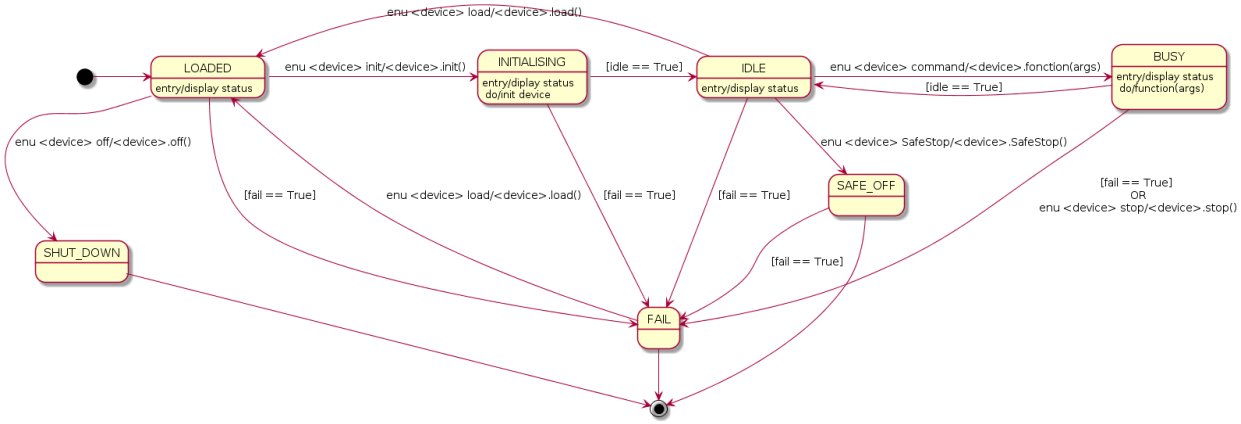
- `enu <motorized-device> SetHome = [value|CURRENT]`: Set Home position to value or current position
- `enu <motorized-device> GetGome`: Get Home position
- `enu <motorized-device> GoHome` : Go to Home

Here are devices classified :

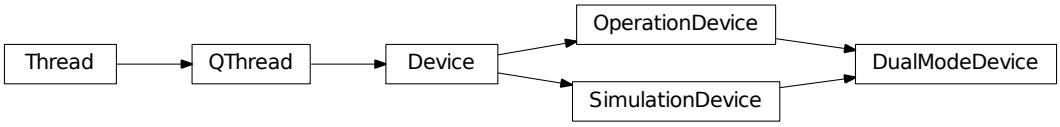
NON MOTORIZED		MOTORIZED			
BIA	IISOURCE	Environment	Shutter	REXM	FPS
todo	todo	todo	todo	todo	todo

Note: Shutter is a motorized device but the SW device won't provide motorized features.

The State Machine



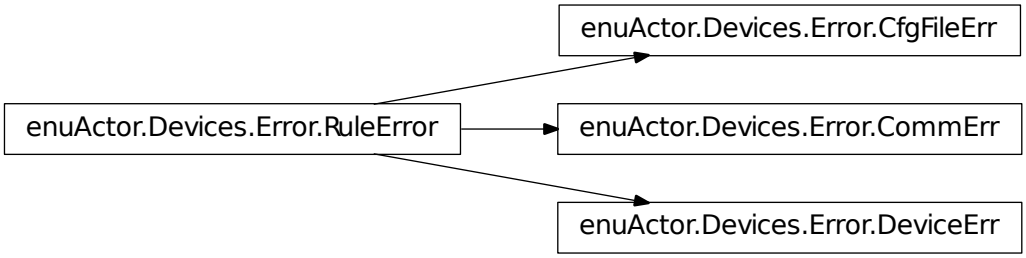
The Devices



Device behaves like an interface for each device such as:

Composed of different common parts for each device:

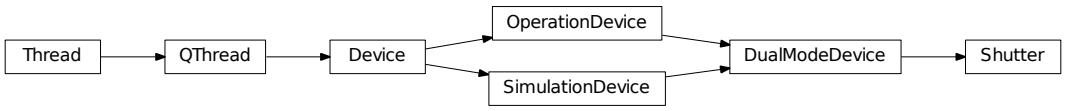
- **FSM :**
  - Here the common state table is defined
  - Start the FSM
  - Diplay rule when state change
  - Each device should implement an `initialise` method which correspod to the INITIALISATION state of the FSM.
- **Communication handling :**
  - Load communication & parameter config file
  - Send message following protocol



Error

Description:

- `CommErr`: error related to communication between PC and Device
- `DeviceErr`: error returned by controller (device) implying a **FAIL** state in the FSM.
- `CfgFileErr`: error from parsing configuration file



Shutter Shutter is open or close ...

Todo

add more details

BIA

Todo

add more details

REXM

Todo

add more details

IISOURCE

Todo

add more details

ENU

Todo

add more details

FPSA

Todo

add more details

1.2 Submodules

1.3 enuActor.MyFSM module

class enuActor.MyFSM.**Fysom**(*cfg*)

Bases: object

Wraps the complete finite state machine operations.

can(*event*)

Returns if the given event be fired in the current machine state.

cannot(*event*)

Returns if the given event cannot be fired in the current state.

is_finished()

Returns if the state machine is in its final state.

isstate(*state*)

Returns if the given state is the current state.

trigger(*event*)

Triggers the given event. The event can be triggered by calling the event handler directly, for ex: fsm.eat() but this method will come in handy if the event is determined dynamically and you have the event name to trigger as a string.

exception enuActor.MyFSM.**FysomError**
Bases: exceptions.Exception

Raised whenever an unexpected event gets triggered.

enuActor.MyFSM.**transition**(*during_state*, *after_state=None*)
Decorator enabling the function to trigger state of the FSM.

Parameters

- **during_state** – event at beginning of the function
- **after_state** – event after the function is performed if specified

Returns function return

Raises DeviceErr

1.4 enuActor.QThread module

class enuActor.QThread.**QMsg**(*method*, **argl*, ***argd*)
Bases: object

DEFAULT_PRIORITY = 5

class enuActor.QThread.**QThread**(*actor*, *name*, *timeout=2*, *isDaemon=True*, *queueClass=<class Queue.PriorityQueue at 0x1fce258>*)
Bases: threading.Thread

exitMsg(*cmd=None*)
handler for the "exit" message. Spits out a message and arranges for the .run() method to exit.

handleTimeout()
Called when the .get() times out. Intended to be overridden.

pingMsg(*cmd=None*)
handler for the 'ping' message.

putMsg(*method*, **argl*, ***argd*)
send ourself a new message.

Parameters

- **method** – a function or bound method to call
- ***argl** – the arguments to the method.
- ***argd** – the arguments dict to the method

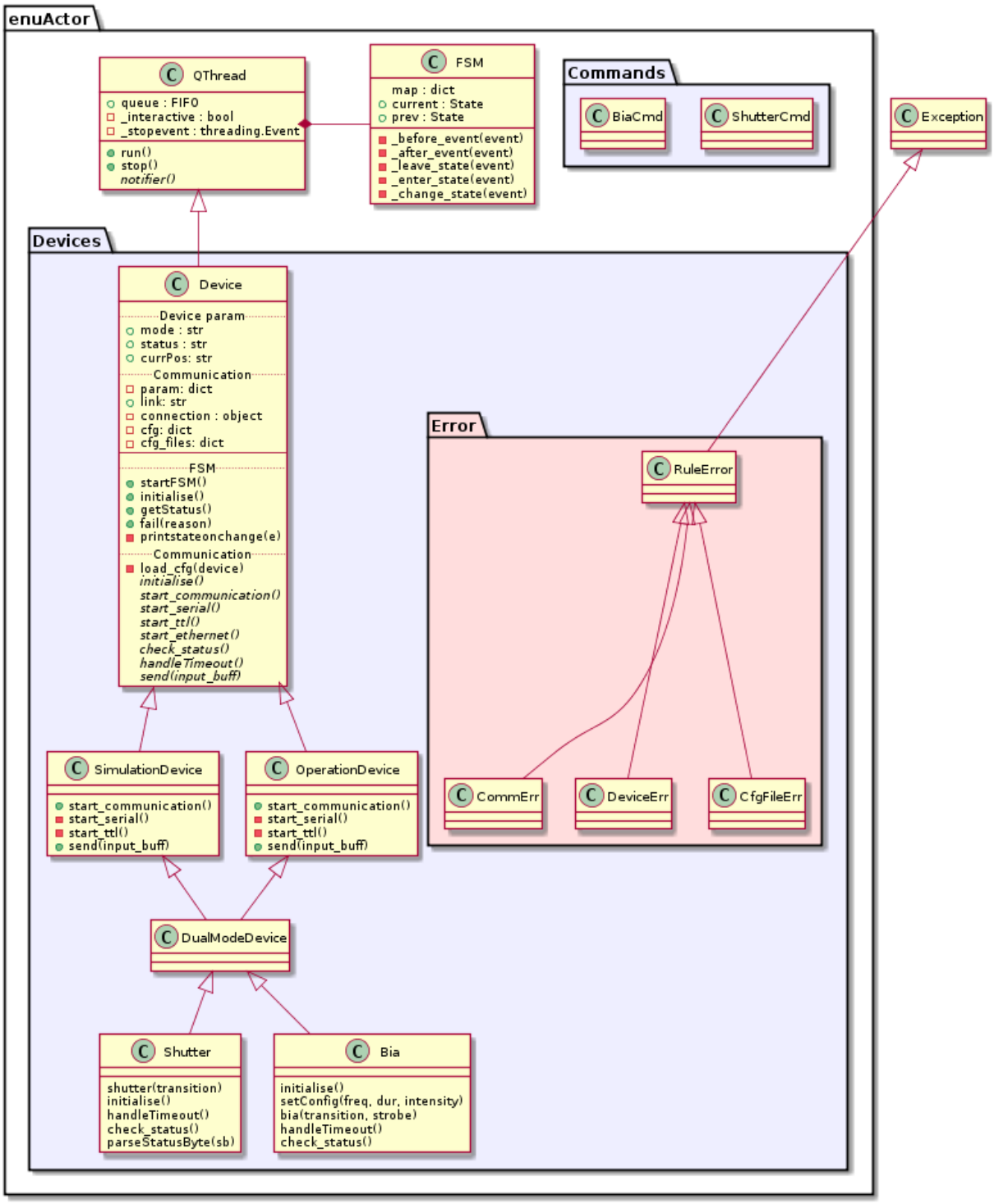
run()
Main run loop for this thread.

sendLater(*msg*, *deltaTime*, *priority=1*)
Send ourself a QMsg after deltaTime seconds.

1.5 enuActor.main module

1.6 Module contents

1.6.1 Class diagram



CHAPTER

TWO

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

e

P

Q

R

S

T