enuActor Documentation

Release

Author

CONTENTS

1	enu <i>A</i>	Actor package
	1.1	Subpackages
	1.2	Submodules
	1.3	enuActor.MyFSM module
	1.4	enuActor.QThread module
	1.5	enuActor.main module
	1.6	Module contents
2	Indi	ces and tables
Рy	thon]	Module Index
In	dex	

Contents:

CONTENTS 1

2 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 (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:
```

1.1. Subpackages 5

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

class enuActor.Devices.bia.**Bia** (actor=None)

SW device: BIA

Bases: enuActor.Devices.Device.DualModeDevice

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
```

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

 $\texttt{MASK_ERROR_SB_1} = [0,0,1,1]$

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

 $MASK_ERROR_SB_4 = [0, 0, 1, 1]$

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

 $MASK_ERROR_SB_6 = [0, 0, 1, 1]$

1.1. Subpackages 7

```
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:

- 1 Convention naming
- 2 The State Machine
- 3 The Devices
 - 3.1 Shutter
 - 3.2 BIA
 - 3.3 REXM
 - 3.4 IISOURCE
 - 3.5 ENU
 - 3.6 FPSA

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:

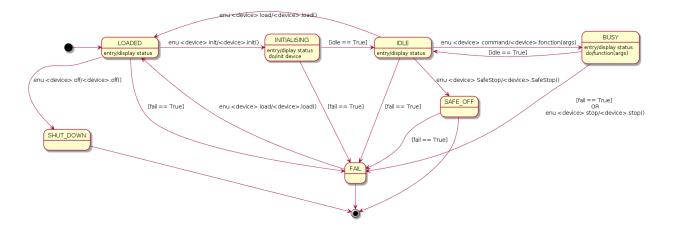
- enu <motorized-device> SetHome = [value|CURRENT]: Set Home position to value or current position
- enu <motorized-device> GetGome: Get Home position
- enu <motorized-device> GoHome: Go to Home

Here are devices classified:

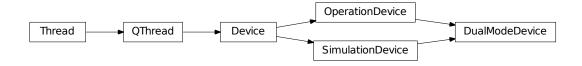
NON	MOTORIZED	MOTORIZED			
BIA	IISOURCE	Environment	Shutter	REXM	FPS
todo	todo	todo	todo	todo	todo

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:

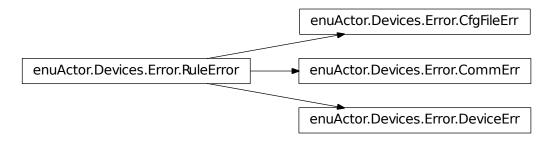
1.1. Subpackages 9

• 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 INITIALISA-TION 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.

1.2. Submodules

```
exception \verb| 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

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

pingMsq(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.

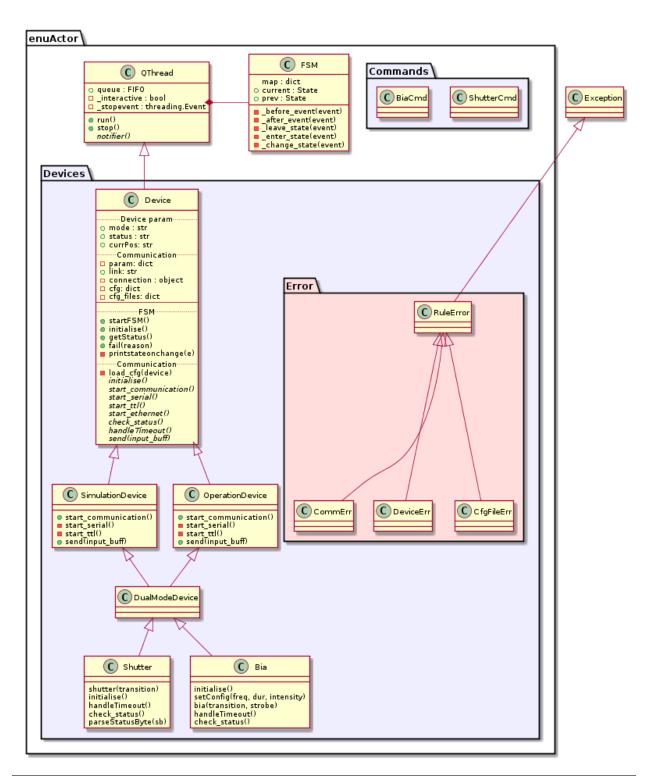
sendLater (msg, deltaTime, priority=1)

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

PYTHON MODULE INDEX

е

```
enuActor,14
enuActor.Commands,3
enuActor.Devices,8
enuActor.Devices.bia,6
enuActor.Devices.Device,3
enuActor.Devices.Error,6
enuActor.Devices.shutter,7
enuActor.MyFSM,11
enuActor.QThread,12
```

enuActor D	ocumentation	. Release
------------	--------------	-----------

18 Python Module Index

A	G
available_link (enuActor.Devices.Device at-	${\tt getStatus()} \ (enuActor.Devices.Device.Device\ method), \ 3$
tribute), 3	Н
B	handleTimeout() (enuActor.Devices.Device.Device
Bia (class in enuActor.Devices.bia), 6 bia() (enuActor.Devices.bia.Bia method), 7	method), 4 handleTimeout() (enuActor.QThread.QThread method),
С	12
can() (enuActor.MyFSM.Fysom method), 11 cannot() (enuActor.MyFSM.Fysom method), 11 cfg_files (enuActor.Devices.Device.Device attribute), 3 CfgFileErr, 6 check_status() (enuActor.Devices.Device.DualModeDevice method), 4 CommErr, 6	initialise() (enuActor.Devices.bia.Bia method), 7 initialise() (enuActor.Devices.Device.Device method), 4 initialise() (enuActor.Devices.shutter.Shutter method), 8 interlock() (in module enuActor.Devices.Device), 6 is_finished() (enuActor.MyFSM.Fysom method), 11 isstate() (enuActor.MyFSM.Fysom method), 11
D	L
DEFAULT_PRIORITY (enuActor.QThread.QMsg attribute), 12	load_cfg() (enuActor.Devices.Device.Device method), 4
Device (class in enuActor.Devices.Device), 3 DeviceErr, 6 DualModeDevice (class in enuActor.Devices.Device), 4 E enuActor (module), 14 enuActor.Commands (module), 3 enuActor.Devices (module), 8 enuActor.Devices.bia (module), 6 enuActor.Devices.Device (module), 3 enuActor.Devices.Error (module), 6 enuActor.Devices.Shutter (module), 7 enuActor.MyFSM (module), 11 enuActor.QThread (module), 12 errno (enuActor.Devices.Error.RuleError attribute), 6 exitMsg() (enuActor.QThread.QThread method), 12	MASK_ERROR_SB_1 (enu- Actor.Devices.shutter.Shutter 7 MASK_ERROR_SB_3 (enu- Actor.Devices.shutter.Shutter 7 MASK_ERROR_SB_4 (enu- Actor.Devices.shutter.Shutter 7 MASK_ERROR_SB_5 (enu- Actor.Devices.shutter.Shutter 7 MASK_ERROR_SB_6 (enu- Actor.Devices.shutter.Shutter 7 MASK_ERROR_SB_6 (enu- Actor.Devices.shutter.Shutter 7
F	0
fail() (enuActor.Devices.Device.Device method), 3 Fysom (class in enuActor.MyFSM), 11 FysomError, 11	op_check_status() (enuActor.Devices.bia.Bia method), 7 op_check_status() (enuActor.Devices.shutter.Shutter method), 8

op_send() (enuActor.Devices.Device.OperationDevice	sim_start_serial() (enu-
method), 5	Actor.Devices.Device.SimulationDevice
op_start_communication() (enu-	method), 6
Actor.Devices.Device.OperationDevice	sim_start_ttl() (enuActor.Devices.Device.SimulationDevice
method), 5	method), 6
op_start_ethernet() (enu-	SimulationDevice (class in enuActor.Devices.Device), 5
Actor.Devices.Device.OperationDevice	start_communication() (enuActor.Devices.Device.Device
method), 5	method), 4
op_start_serial() (enuActor.Devices.Device.OperationDevices.Devices.OperationDevices.Devices.OperationDevices.Devices.OperationDevices.Operati	cstart_communication() (enu-
method), 5	Actor.Devices.Device.DualModeDevice
op_start_ttl() (enuActor.Devices.Device.OperationDevice	method), 4
method), 5	start_ethernet() (enuActor.Devices.Device.Device
OperationDevice (class in enuActor.Devices.Device), 5	method), 4
	start_ethernet() (enuActor.Devices.Device.DualModeDevice
P	method), 4
parseStatusByte() (enuActor.Devices.shutter.Shutter	start_serial() (enuActor.Devices.Device.Device method),
method), 8	4
pingMsg() (enuActor.QThread.QThread method), 12	start_serial() (enuActor.Devices.Device.DualModeDevice
positions (enuActor.Devices.shutter.Shutter attribute), 8	method), 4
	start_ttl() (enuActor.Devices.Device.Device method), 4
printstateonchange() (enuActor.Devices.Device.Device	start_ttl() (enuActor.Devices.Device.DualModeDevice
method), 4	method), 4
PRIORITY_DEFAULT (enu-	startFSM() (enuActor.Devices.Device.Device method), 4
Actor.Devices.Error.RuleError attribute),	7.
6	STATUS_BYTE_1 (enuActor.Devices.shutter.Shutter at-
putMsg() (enuActor.QThread.QThread method), 12	tribute), 7
\circ	STATUS_BYTE_3 (enuActor.Devices.shutter.Shutter at-
Q	tribute), 8
QMsg (class in enuActor.QThread), 12	STATUS_BYTE_4 (enuActor.Devices.shutter.Shutter at-
QThread (class in enuActor.QThread), 12	tribute), 8
D	STATUS_BYTE_5 (enuActor.Devices.shutter.Shutter at-
R	tribute), 8
RuleError, 6	STATUS_BYTE_6 (enuActor.Devices.shutter.Shutter at-
run() (enuActor.QThread.QThread method), 12	tribute), 8
	strerror (enuActor.Devices.Error.RuleError attribute), 6
S	-
send() (enuActor.Devices.Device.Device method), 4	Т
send() (enuActor.Devices.Device.DualModeDevice	terminal() (enuActor.Devices.shutter.Shutter method), 8
method), 4	transition() (in module enuActor.MyFSM), 12
sendLater() (enuActor.QThread.QThread method), 12	trigger() (enuActor.MyFSM.Fysom method), 11
setConfig() (enuActor.Devices.bia.Bia method), 7	
Shutter (class in enuActor.Devices.shutter), 7	
shutter() (enuActor.Devices.shutter,), /	
* *	
shutter_id (enuActor.Devices.shutter.Shutter attribute), 8	
sim_check_status() (enu-	
Actor.Devices.Device.SimulationDevice	
method), 5	
sim_send() (enuActor.Devices.Device.SimulationDevice	
method), 6	
sim_start_communication() (enu-	
Actor.Devices.Device.SimulationDevice	
method), 6	
sim_start_ethernet() (enu-	
Actor.Devices.Device.SimulationDevice	
method), 6	

20 Index