
Developer manual of RasPy

Release 0.0.1

bibi21000

January 11, 2015

1	Installation	1
1.1	Raspbian	1
1.2	Update packages	1
1.3	Download it	1
1.4	Configure your system	1
1.5	Installation	2
1.6	Run the tests	2
1.7	Start it	2
1.8	Read the doc	2
2	Before starting	3
2.1	Phylosophy	3
2.2	Documentation	3
2.3	Tests	3
2.4	GitHub	4
3	Develop	5
3.1	A new device	5
3.2	A new server	5
4	raspy package	7
4.1	Subpackages	7
4.2	Module contents	21
	Python Module Index	23
	Index	25

Installation

The following lines “clone” a GitHub repository. If you want to submit “pull requests”, you need to “fork” RasPy using this [guide](#).

1.1 Raspbian

Install official raspbian from here : <http://www.raspbian.org/>.

Newbies can install it from : <http://www.raspberrypi.org/downloads/>.

Developpers or others can also install it on standard distributions (Ubuntu, Debian, RedHat, ...).

1.2 Update packages

You can now update packages

```
sudo apt-get -y update
sudo apt-get -y dist-upgrade
```

We need to install some packages to download and build RasPy:

```
sudo apt-get -y install build-essential python-dev python-minimal python python2.7-dev python2.7-r
```

Some packages need to be removed as new versions are available from eggs :

```
sudo apt-get remove python-zmq libzmq libzmq-dev python-nose pylint
```

1.3 Download it

You should now download RasPy using git. You should not download and install RasPy with root user. Ideally, you should create a special user for running RasPy or the pi user. Keep in mind root is baaaddddd.

```
git clone https://github.com/bibi21000/RasPy.git
```

1.4 Configure your system

- access rights
- sudo nopasswd
- ...

1.5 Installation

If you want to develop for RasPy, you need to install it in develop mode :

```
sudo make develop
```

Otherwise install it normally ... but not now ;) :

```
sudo make install
```

And be patient ... installation need to compile zmq ... It takes a while ...

If something goes wrong during install or if you want to remove RasPy from you computer, you type :

```
sudo make uninstall
```

If you want to remove dependencies, look at setup.py to get the list and use the following command for every package:

```
sudo pip uninstall package
```

1.6 Run the tests

Check that the SLEEP constant in tests/common.py ist set to 1.0 or 1.5

```
vim tests/common.py
```

You can now check that everything is fine running the tests :

```
make tests
```

If it fails ... run it again :) At last, copy / paste the full screen output and send it to the core team.

1.7 Start it

In the next monthes, you should be abble to start it :

```
make start
```

1.8 Read the doc

- docs/pdf
- docs/html

Before starting

If you want to develop you surely need vim :

```
sudo apt-get -y install vim-nox vim-addon-manager
```

2.1 Philosophy

Tests, tests, ... and tests :

- A bug -> a test -> a patch
- A new feature -> many test

And documentation

- A new feature -> documentation

2.2 Documentation

If you want to generate the documentation, you need to install some packages :

```
sudo apt-get -y install python-sphinx graphviz
```

And some eggs :

```
sudo pip install seqdiag sphinxcontrib-seqdiag
sudo pip install blockdiag sphinxcontrib-blockdiag
sudo pip install nwdiag sphinxcontrib-nwdiag
sudo pip install actdiag sphinxcontrib-actdiag
```

You can now generate the full documentation using :

```
make docs
```

You can also generate a part of it, for example :

```
cd docs
make html
```

2.3 Tests

Nosetests and pylint are used to test quality of code. There reports are here :

- Nosetests report

- Coverage report
- Pylint report

Coverage is not the goal but it's one : a module must have a coverage of 90% to be accepted by core team. Otherwise it will block the packaging process. Of course, a FAILED test will also.

Keep in mind that all tests must succeed before submitting pull request. But :

- if a test is a work in progress, you can skip it using `self.wipTest()`
- if a test can only be run on Raspberry (ie onewire), it must call `self.skipTest(message)` at its start.

There is 2 ways to launch the tests. The first one to use on a Raspberry :

```
make tests
```

You can also run the developers tests (without skipped one) on a standard computer running :

```
make devtests
```

If you're on a raspberry, you can run the full tests like this :

```
make tests
```

Running only one test module :

```
/usr/local/bin/nosetests --verbosity=2 --cover-package=raspy --with-coverage --cover-inclusive --cover-branches --cover-conditions
```

You can follow automatic tests on [travis-ci](#).

2.4 GitHub

You can test the code, build the doc and commit it using the following command :

```
make git
```

You may use `ssh_keys` to do it automatically without typing password.

Develop

3.1 A new device

3.2 A new server

raspy package

4.1 Subpackages

4.1.1 raspy.common package

Subpackages

raspy.common.devices package

Submodules

raspy.common.devices.device module Devices.

class `raspy.common.devices.device.BaseDevice` (*json=None*)

Bases: `object`

The base device object

What is a device :

- a temperature sensor
- a wind sensor
- a camera
- the clock RTC
- a dimmer
- a TV
- ...

What can we do with a device :

- get value of a sensor
- dim a dimmer
- take a photo with camera
- ...

We should do auto-mapping :

- python object <-> json
- python object <-> html

We should manage complex devices, ie a TV : it groups a channel selector (+, -, and direct access to a channel), a volume selector, ... In an ideal world we should not be obliged to create each sub-devices.

Naming convention of devices on the network : (MDP.routing_key(hostname, service)).{device_name}[,subdevice]

check (*json=None*)

Check that the JSON is a valid device

fullname (*prefix*)

The fullname of the device

json

Check that the JSON is a valid device

name

The name of the device Must be unique for the instance server.

new (*json=None*)

Create a new device and return it

oid = 'base'

The Object Identifier It should be given by the core team as it can break other devices. Need to define a naming convention : sensor, sensor.temperature, media.camera, ...

template

The template of the device

templates = {}

The templates dictionary Every device must add an entry for its config Will be used to check the device

class `raspy.common.devices.device.DeviceRegister`

Bases: object

The device register

All devices must register to this register (in main module)

check (*json=None*)

Check that the JSON is a valid device

new (***kwargs*)

Create a new device and return it

register (*device_type*)

Register a device_type under key

raspy.common.devices.media module Media devices

class `raspy.common.devices.media.MediaCamera` (***kwargs*)

Bases: `raspy.common.devices.media.MediaDevice`

The camera device object

new (***kwargs*)

Create a new device and return it

oid = 'media.camera'

The Object Identifier It should be given by the core team as it can break other devices. Need to define a naming convention : sensor, sensor.temperature, media.camera, ...

class `raspy.common.devices.media.MediaDevice` (***kwargs*)

Bases: `raspy.common.devices.device.BaseDevice`

The sensor device object

check (*json=None*)

Check that the JSON is a valid device

oid = 'media'

The Object Identifier It should be given by the core team as it can break other devices. Need to define a naming convention : sensor, sensor.temperature, media.camera, ...

class `raspy.common.devices.media.MediaTV` (***kwargs*)

Bases: `raspy.common.devices.media.MediaDevice`

The temperature sensor device object

new (***kwargs*)

Create a new device and return it

oid = 'media.tv'

The Object Identifier It should be given by the core team as it can break other devices. Need to define a naming convention : sensor, sensor.temperature, media.camera, ...

raspy.common.devices.sensor module Sensors devices

class `raspy.common.devices.sensor.SensorDevice` (***kwargs*)

Bases: `raspy.common.devices.device.BaseDevice`

The sensor device object

check (*json=None*)

Check that the JSON is a valid device

oid = 'sensor'

The Object Identifier It should be given by the core team as it can break other devices. Need to define a naming convention : sensor, sensor.temperature, media.camera, ...

class `raspy.common.devices.sensor.SensorTemperature` (***kwargs*)

Bases: `raspy.common.devices.sensor.SensorDevice`

The temperature sensor device object

new (***kwargs*)

Create a new device and return it

oid = 'sensor.temperature'

The Object Identifier It should be given by the core team as it can break other devices. Need to define a naming convention : sensor, sensor.temperature, media.camera, ...

class `raspy.common.devices.sensor.SensorWind` (***kwargs*)

Bases: `raspy.common.devices.sensor.SensorDevice`

The wind sensor device object

new (***kwargs*)

Create a new device and return it

oid = 'sensor.wind'

The Object Identifier It should be given by the core team as it can break other devices. Need to define a naming convention : sensor, sensor.temperature, media.camera, ...

Module contents

Submodules

raspy.common.MDP module

exception `raspy.common.MDP.ClientError` (*value*)

Bases: `raspy.common.MDP.GenericError`

Client side exception

exception `raspy.common.MDP.GenericError` (*value*)

Bases: `exceptions.Exception`

Generic exception

exception `raspy.common.MDP.ServerError` (*value*)

Bases: `raspy.common.MDP.GenericError`

Server side exception

`raspy.common.MDP.logger` = `<logging.Logger object at 0x2aded235ed90>`

Majordomo Protocol definitions

`raspy.common.MDP.routing_key` (*hostname, service*)

raspy.common.client module

class `raspy.common.client.Client` (*hostname='localhost', service='worker', broker_ip='127.0.0.1', broker_port=5514, poll=1500, ttl=900*)

Bases: `raspy.common.executive.Executive`

The generic worker

From <http://zguide.zeromq.org/py:all#header-48>

request (*service=None, data=['mmi.echo'], callback=None, args=(), kwargs={}*)

Request a job to a worker

run ()

Run the client

status (*uuid*)

Request a job to a worker

raspy.common.dynamic module

raspy.common.executive module

class `raspy.common.executive.Executive` (*hostname='localhost', service='executive', broker_ip='127.0.0.1', broker_port=5514*)

Bases: `object`

The Executive mother class for all workers

todo :

- bug : can't stop when jobs in queues

destroy ()

Wait for threads and destroy contexts.

get_instance_id ()

Return the instance of the executive

... todo : must be multihost and multithread.

run ()

Run the executive

shutdown ()

Shutdown executive.

class `raspy.common.executive.ExecutiveProcess` (*executive, executive_name*)

Bases: `multiprocessing.process.Process`

Process executing tasks from a given tasks queue

run ()

shutdown ()

Method to deactivate the client connection completely.

Will delete the stream and the underlying socket.

Warning: The instance MUST not be used after `shutdown ()` has been called.

Return type None

raspy.common.kvcliapi module

kvsimple - simple key-value message class for example applications

Author: Min RK <benjaminrk@gmail.com>

From : <http://zguide.zeromq.org/py:kvsimple>

class `raspy.common.kvcliapi.KvPublisherClient` (*hostname='localhost', broker_ip='127.0.0.1', broker_port=5514*)

Bases: object

KeyValue Protocol Client API, Python version.

Implements the client defined at <http://zguide.zeromq.org/page:all#Working-with-Subtrees>

From <https://raw.githubusercontent.com/imatix/zguide/master/examples/Python/clonecli4.py>

destroy ()

Destroy object

send (*subtree='subtree', key='key', body='body'*)

Send the update

class `raspy.common.kvcliapi.KvSubscriberClient` (*hostname='localhost', subtree='subtree', broker_ip='127.0.0.1', broker_port=5514, speed=1.0*)

Bases: object

KeyValue Protocol Client API, Python version.

Implements the client defined at <http://zguide.zeromq.org/page:all#Working-with-Subtrees> From <https://raw.githubusercontent.com/imatix/zguide/master/examples/Python/clonecli4.py>

destroy ()

Destroy object

run ()

Run the poller

shutdown ()

Shutdown the broker.

raspy.common.kvsimple module

kvsimple - simple key-value message class for example applications

Author: Min RK <benjaminrk@gmail.com>

From : <http://zguide.zeromq.org/py:kvsimple>

class `raspy.common.kvsimple.KVMsg` (*sequence, key=None, body=None*)

Bases: object

Message is formatted on wire as 3 frames:

- frame 0: key (OMQ string)

- frame 1: sequence (8 bytes, network order)
- frame 2: body (blob)

body = None

dump()

Dump me to a string

key = None

classmethod recv(*socket*)

Reads key-value message from socket, returns new kvmsg instance.

send(*socket*)

Send key-value message to socket; any empty frames are sent as such.

sequence = 0

store(*dikt*)

Store me in a dict if I have anything to store

raspy.common.mdcliapi module

Majordomo Protocol Client API, Python version.

Implements the MDP/Worker spec at <http://rfc.zeromq.org/spec:7>.

Author: Min RK <benjaminrk@gmail.com> Based on Java example by Arkadiusz Orzechowski

class `raspy.common.mdcliapi.MajorDomoClient` (*broker*)

Bases: `object`

Majordomo Protocol Client API, Python version.

Implements the MDP/Worker spec at <http://rfc.zeromq.org/spec:7>.

broker = None

client = None

ctx = None

destroy()

Destroy object

poller = None

reconnect_to_broker()

Connect or reconnect to broker

retries = 3

send(*service, request*)

Send request to broker and get reply by hook or crook.

Takes ownership of request message and destroys it when sent. Returns the reply message or None if there was no reply.

timeout = 2500

verbose = False

raspy.common.mdwrkapi module

Majordomo Protocol Worker API, Python version

Implements the MDP/Worker spec at <http://rfc.zeromq.org/spec:7>.

Author: Min RK <benjaminrk@gmail.com> Based on Java example by Arkadiusz Orzechowski


```
class raspy.common.mdwrkapi.MajorDomoWorker (broker, service)
```

Bases: object

Majordomo Protocol Worker API, Python version

Implements the MDP/Worker spec at <http://rfc.zeromq.org/spec:7>.

HEARTBEAT_LIVENESS = 3

broker = None

ctx = None

destroy ()

Destroy object

expect_reply = False

heartbeat = 2500

heartbeat_at = 0

liveness = 0

reconnect = 2500

reconnect_to_broker ()

Connect or reconnect to broker

recv (reply=None)

Send reply, if any, to broker and wait for next request.

reply_to = None

send_to_broker (command, option=None, msg=None)

Send message to broker.

. If no msg is provided, creates one internally

service = None

shutdown ()

Shutdown executive.

status = True

The status of the worker. Should be update by callback in the future

timeout = 2500

verbose = False

worker = None

raspy.common.runner module

raspy.common.server module

```
class raspy.common.server.Server (hostname='localhost', service='worker', bro-
                                   ker_ip='127.0.0.1', broker_port=5514)
```

Bases: `raspy.common.executive.Executive`, `raspy.common.statistics.Statistics`

The generic worker

From <http://zguide.zeromq.org/py:all#header-48>

worker_mmi ()

Retrieve mmi informations of the worker

raspy.common.statistics module

class `raspy.common.statistics.SNMP` (*oid='module.snmp.key', doc='A statistic integer value', initial=0*)

Bases: `object`

Abstract statistic item

set (*value*)

Set a value to the snmp object

class `raspy.common.statistics.SNMPCounter` (*oid='module.snmp.key', doc='A statistic counter value with overflow', initial=0, overflow=4294967296*)

Bases: `raspy.common.statistics.SNMP`

Long (32bits) with overflow

set (*value=1*)

Add value (default=1) to current value. Also manage overflow.

class `raspy.common.statistics.SNMPFloat` (*oid='module.snmp.key', doc='A statistic float value', initial=0.0*)

Bases: `raspy.common.statistics.SNMP`

Float counter

class `raspy.common.statistics.SNMPString` (*oid='module.snmp.key', doc='A statistic string value', initial=''*)

Bases: `raspy.common.statistics.SNMP`

Float counter

class `raspy.common.statistics.Statistics`

Bases: `object`

The statistics manager

add_statistic ()

Add a new statistic to the manager

remove_statistic (*oid*)

Remove a statistic from the manager

update_statistic (*oid=''*)

Add a new statistic to the manager

worker_statistics ()

Send statistics via mmi

raspy.common.supervisor module

class `raspy.common.supervisor.Supervisor` (*runner=None*)

The worker supervisor

Start executives in separate process see futures Each executive start multiples threads of workers

todo :

- bug : can't stop when jobs in queues

get_instance_id ()

Return the instance of the worker : must be multihost and multithread.

reload ()

Request the workers configuration against the configurator.

Will unregister all workers, stop all timers and ignore all further messages.

Warning: The instance MUST not be used after `shutdown()` has been called.

Return type None

run()

Start the IOLoop instance

shutdown()

Shutdown supervisor.

Will unregister all workers, stop all timers and ignore all further messages.

Warning: The instance MUST not be used after `shutdown()` has been called.

Return type None

stop_executives()

Shutdown executives.

raspy.common.zhelpers module

Helper module for example applications. Mimics ZeroMQ Guide's zhelpers.h.

`raspy.common.zhelpers.dump(msg_or_socket)`

Receives all message parts from socket, printing each frame neatly

`raspy.common.zhelpers.set_id(zsocket)`

Set simple random printable identity on socket

`raspy.common.zhelpers.zpipe(ctx)`

build inproc pipe for talking to threads

mimic pipe used in czmq zthread_fork.

Returns a pair of PAIRs connected via inproc

Module contents

4.1.2 raspy.http package

Submodules

raspy.http.http module

raspy.http.views module

Module contents

4.1.3 raspy.servers package

Submodules

raspy.servers.broker module

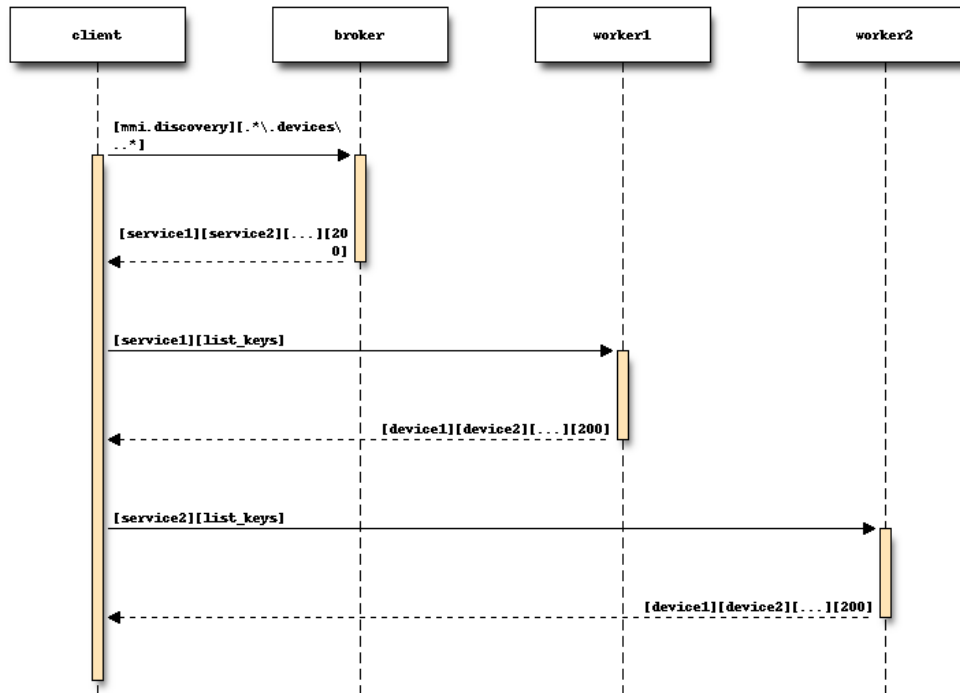
class `raspy.servers.broker.Broker` (*hostname='localhost', service='broker', broker_ip='127.0.0.1', broker_port=15514*)

Bases: `raspy.common.executive.Executive`

Majordomo Protocol broker and key/value proxy.

Discovery process

Here is the way for a client to discover all devices on network



You can do the same for crons and scenarios

HEARTBEAT_EXPIRY = 7500

HEARTBEAT_INTERVAL = 2500

HEARTBEAT_LIVENESS = 3

INTERNAL_SERVICE_PREFIX = 'mmi.'

ctx = None

delete_worker (*worker, disconnect*)

Deletes worker from all data structures, and deletes worker.

destroy ()

Disconnect all workers, destroy context.

dispatch (*service, msg*)

Dispatch requests to waiting workers as possible

heartbeat_at = None

poller = None

process_client (*sender, msg*)

Process a request coming from a client.

process_worker (*sender, msg*)

Process message sent to us by a worker.

purge_workers ()

Look for & kill expired workers.

Workers are oldest to most recent, so we stop at the first alive worker.

require_service (*name*)

Locates the service (creates if necessary).

require_worker (*address*)
Finds the worker (creates if necessary).

run ()
Main broker work happens here

send_heartbeats ()
Send heartbeats to idle workers if it's time

send_to_worker (*worker, command, option, msg=None*)
Send message to worker.
. If message is provided, sends that message.

service_internal (*service, msg*)
Handle internal service according to 8/MMI specification

services = None

shutdown ()
Shutdown the broker.

socket = None

waiting = None

worker_waiting (*worker*)
This worker is now waiting for work.

workers = None

class raspy.servers.broker.**Proxy** (*hostname='localhost', service='broker', broker_ip='*', broker_port=5514, speed=1.0*)
Bases: threading.Thread
The publisher
Tree :

- /event/
- /scenario/
- /device/

from : <http://zguide.zeromq.org/page:all#Working-with-Subtrees>

run ()
Run the proxy

send_single (*key, kvmsg, route*)
Send one state snapshot key-value pair to a socket

shutdown ()
Shutdown the proxy.

class raspy.servers.broker.**Route** (*socket, identity, subtree*)

class raspy.servers.broker.**Service** (*name*)
Bases: object
a single Service
name = None
requests = None
waiting = None

class raspy.servers.broker.**Worker** (*identity, address, lifetime*)
Bases: object
a Worker, idle or active

```
address = None
expiry = None
identity = None
service = None
```

raspy.servers.core module

```
class raspy.servers.core.Core(hostname='localhost', service='core', broker_ip='127.0.0.1',
                               broker_port=5514)
Bases: raspy.common.server.Server
```

The Core server

•Cron

- Generate events in the publisher

•Scenario

- a scenario can run in background at startup (ie thermostat) or fired by an event (ie cron, sun is down, temperature is under 0°C)
- a scenario can be a loop so it must be launch in a separate thread : start filling, loop until water level is ok : need to call self._stopevent.isSet() in it so that the tread can shutdown.
- look for updates in entries list (cron, sensors, variables in the publisher) : a list mapped in friendly user's names (using store)
- it can publish some values with publisher
- do some work using inline code python :
- send commands to devices, cron jobs, start other scenario, update some variables in publisher
- we can export/import scenarios : share with friends

•NTP / Sytem Time / RTC Sync

- sync from ntp to rtc
- sync from trc to system : using sudo with no password

```
worker_cron()
```

Create a worker to handle cron requests

```
worker_scenario()
```

Create a worker to handle scenario's requests

```
worker_scenarios()
```

Create a worker to handle scenarios requests (list_keys, ...)

```
class raspy.servers.core.Cron
```

Bases: object

A cron job

```
aps_job = None
```

```
class raspy.servers.core.CronManager
```

Bases: object

The manager of cron job

```
jobs = {}
```

```
class raspy.servers.core.Scenario(name='scenar1', publisher=None)
```

Bases: threading.Thread

A scenario

code = None
The code we must exec

conf = {}
The configuration of the scenario

entries = {}
The entries we must look at for an event driven scenario

fire ()
Check if the scenario must be fired using entries and that is not already running. If so, call `self.run()`

Returns True if the thread must be launch (`self.run()`), False otherwise

Return type boolean

load (store)
Load the scenario from titanic store

Parameters `store` – the store to get info from

Type `titani_store`

Returns True if the scenario was loaded from store

Return type boolean

run ()
Run the scenario

running = False
Is the scenario running

shutdown ()
Shutdown the scenario

store (store)
Store the scenario to titanic store

class `raspy.servers.core.ScenarioManager (publisher=None)`
Bases: `object`

The manager of scenarios

<http://etutorials.org/Programming/Python+tutorial/Part+III+Python+Library+and+Extension+Modules/Chapter+13.+Control>
<http://lucumr.pocoo.org/2011/2/1/exec-in-python/> <http://late.am/post/2012/04/30/the-exec-statement-and-a-python-mystery>

add (name='scenar1', entries={}, code=None, conf={})
Add a scenario

delete (name='scenar1')
Delete a scenario

list ()
Return all scenarios with conf, entries, ... as json dict

list_keys ()
Return all scenarios key (=name) ... as json list

load (store)
Load the scenarios from titanic store

store keys :

- `scenario.main.conf` : a json dict for configuration of scenario
- `scenario.main.keys` : a json list of the scenario's names
- `scenario.key1.conf` : a json dict for configuration of scenario key1
- `scenario.key1.entries` : a json dict of entries of scenario key1

- `scenario.key1.code` : a json string of code of scenario key1

scenarios = {}

The scenarios

shutdown ()

Shutdown the scenario manager

store (store)

Store the scenarios to titanic store

update (name='scenar1', entries={}, code=None, conf={})

Update a scenario

raspy.servers.fake module

```
class raspy.servers.fake.Fake (hostname='localhost', service='fake', broker_ip='127.0.0.1',  
                                broker_port=5514)
```

Bases: `raspy.common.server.Server`

A fake server to test RasPy

- we must developp “real” fake device : ie a temperature sensors must not send random values
- a cyclic sensor : parameters : cycle length, min, max and unit. Will do cycle from min temp to max temp (at cycle/2) and fall back tp min temp at end of if
- a linear sensor
-

worker_devices ()

Create a worker to handle devices requests

raspy.servers.onewire module

```
class raspy.servers.onewire.OneWire (hostname='localhost', service='onewire', broker_ip='127.0.0.1',  
                                       broker_port=5514, devices_dir='/sys/bus/w1/devices')
```

Bases: `raspy.common.server.Server`

The OneWire server

You need to load kernel module :

```
sudo vim /etc/modules
```

```
# /etc/modules: kernel modules to load at boot time.  
#  
# This file contains the names of kernel modules that should be loaded  
# at boot time, one per line. Lines beginning with "#" are ignored.  
# Parameters can be specified after the module name.  
w1-therm  
w1-gpio pullup=1  
i2c-dev  
i2c-bcm2708  
spi-bcm2708  
snd-bcm2835
```

And check that blacklist is correct :

```
sudo vim /etc/modprobe.d/raspi-blacklist.conf
```



```
# blacklist spi and i2c by default (many users don't need them)
blacklist spi-bcm2708
blacklist i2c-bcm2708
blacklist snd-soc-pcm512x
blacklist snd-soc-wm8804
```

At last, we must load the module in init script so that we don't need to update this.

From <https://www.modmypi.com/blog/ds18b20-one-wire-digital-temperature-sensor-and-the-raspberry-pi>

worker_devices()

Create a worker to handle devices requests

raspy.servers.titanic module

```
class raspy.servers.titanic.Titanic (hostname='localhost',          service='titanic',
                                     broker_ip='127.0.0.1',         broker_port=5514,
                                     data_dir='/tmp/raspy')
```

Bases: `raspy.common.executive.Executive`

The Titanic helper

Also integrates a store for keys/values

From <http://zguide.zeromq.org/py:all#Disconnected-Reliability-Titanic-Pattern>

<http://zguide.zeromq.org/py:all#Service-Oriented-Reliable-Queuing-Majordomo-Pattern>

<https://github.com/imatix/zguide/tree/master/examples/Python>

reply_filename(uuid)

Returns freshly allocated reply filename for given UUID

request_filename(uuid)

Returns freshly allocated request filename for given UUID

run()

Run the hub

service_success(client, uuid)

Attempt to process a single request, return True if successful

store_filename(service)

Returns store filename for given service

titanic_close()

Create a worker to handle titanic.close

titanic.close: confirm that a reply has been stored and processed.

titanic_reply()

Create a worker to handle titanic.service

titanic.reply: fetch a reply, if available, for a given request UUID.

titanic_request(pipe)

Create a worker to handle titanic.request

titanic.request: store a request message, and return a UUID for the request.

titanic_store()

Create a worker to handle store services

Module contents

4.2 Module contents

r

- [raspy](#), 21
- [raspy.common](#), 15
 - [raspy.common.client](#), 10
 - [raspy.common.devices](#), 9
 - [raspy.common.devices.device](#), 7
 - [raspy.common.devices.media](#), 8
 - [raspy.common.devices.sensor](#), 9
 - [raspy.common.executive](#), 10
 - [raspy.common.kvcliapi](#), 11
 - [raspy.common.kvsimple](#), 11
 - [raspy.common.mdcliapi](#), 12
 - [raspy.common.MDP](#), 9
 - [raspy.common.mdwrkapi](#), 12
 - [raspy.common.server](#), 13
 - [raspy.common.statistics](#), 14
 - [raspy.common.supervisor](#), 14
 - [raspy.common.zhelpers](#), 15
- [raspy.servers](#), 21
 - [raspy.servers.broker](#), 15
 - [raspy.servers.core](#), 18
 - [raspy.servers.fake](#), 20
 - [raspy.servers.onewire](#), 20
 - [raspy.servers.titanic](#), 21

A

`add()` (raspy.servers.core.ScenarioManager method), 19
`add_statistic()` (raspy.common.statistics.Statistics method), 14
`address` (raspy.servers.broker.Worker attribute), 17
`aps_job` (raspy.servers.core.Cron attribute), 18

B

`BaseDevice` (class in raspy.common.devices.device), 7
`body` (raspy.common.kvsimple.KVMsg attribute), 12
`Broker` (class in raspy.servers.broker), 15
`broker` (raspy.common.mdcliapi.MajorDomoClient attribute), 12
`broker` (raspy.common.mdwrkapi.MajorDomoWorker attribute), 13

C

`check()` (raspy.common.devices.device.BaseDevice method), 8
`check()` (raspy.common.devices.device.DeviceRegister method), 8
`check()` (raspy.common.devices.media.MediaDevice method), 8
`check()` (raspy.common.devices.sensor.SensorDevice method), 9
`Client` (class in raspy.common.client), 10
`client` (raspy.common.mdcliapi.MajorDomoClient attribute), 12
`ClientError`, 9
`code` (raspy.servers.core.Scenario attribute), 18
`conf` (raspy.servers.core.Scenario attribute), 19
`Core` (class in raspy.servers.core), 18
`Cron` (class in raspy.servers.core), 18
`CronManager` (class in raspy.servers.core), 18
`ctx` (raspy.common.mdcliapi.MajorDomoClient attribute), 12
`ctx` (raspy.common.mdwrkapi.MajorDomoWorker attribute), 13
`ctx` (raspy.servers.broker.Broker attribute), 16

D

`delete()` (raspy.servers.core.ScenarioManager method), 19

`delete_worker()` (raspy.servers.broker.Broker method), 16
`destroy()` (raspy.common.executive.Executive method), 10
`destroy()` (raspy.common.kvcliapi.KvPublisherClient method), 11
`destroy()` (raspy.common.kvcliapi.KvSubscriberClient method), 11
`destroy()` (raspy.common.mdcliapi.MajorDomoClient method), 12
`destroy()` (raspy.common.mdwrkapi.MajorDomoWorker method), 13
`destroy()` (raspy.servers.broker.Broker method), 16
`DeviceRegister` (class in raspy.common.devices.device), 8
`dispatch()` (raspy.servers.broker.Broker method), 16
`dump()` (in module raspy.common.zhelpers), 15
`dump()` (raspy.common.kvsimple.KVMsg method), 12

E

`entries` (raspy.servers.core.Scenario attribute), 19
`Executive` (class in raspy.common.executive), 10
`ExecutiveProcess` (class in raspy.common.executive), 10
`expect_reply` (raspy.common.mdwrkapi.MajorDomoWorker attribute), 13
`expiry` (raspy.servers.broker.Worker attribute), 18

F

`Fake` (class in raspy.servers.fake), 20
`fire()` (raspy.servers.core.Scenario method), 19
`fullname()` (raspy.common.devices.device.BaseDevice method), 8

G

`GenericError`, 10
`get_instance_id()` (raspy.common.executive.Executive method), 10
`get_instance_id()` (raspy.common.supervisor.Supervisor method), 14

H

`heartbeat` (raspy.common.mdwrkapi.MajorDomoWorker attribute), 13

- heartbeat_at (raspy.common.mdwrkapi.MajorDomoWorker attribute), 13
- heartbeat_at (raspy.servers.broker.Broker attribute), 16
- HEARTBEAT_EXPIRY (raspy.servers.broker.Broker attribute), 16
- HEARTBEAT_INTERVAL (raspy.servers.broker.Broker attribute), 16
- HEARTBEAT_LIVENESS (raspy.common.mdwrkapi.MajorDomoWorker attribute), 13
- HEARTBEAT_LIVENESS (raspy.servers.broker.Broker attribute), 16
- I**
- identity (raspy.servers.broker.Worker attribute), 18
- INTERNAL_SERVICE_PREFIX (raspy.servers.broker.Broker attribute), 16
- J**
- jobs (raspy.servers.core.CronManager attribute), 18
- json (raspy.common.devices.device.BaseDevice attribute), 8
- K**
- key (raspy.common.kvsimple.KVMsg attribute), 12
- KVMsg (class in raspy.common.kvsimple), 11
- KvPublisherClient (class in raspy.common.kvcliapi), 11
- KvSubscriberClient (class in raspy.common.kvcliapi), 11
- L**
- list() (raspy.servers.core.ScenarioManager method), 19
- list_keys() (raspy.servers.core.ScenarioManager method), 19
- liveness (raspy.common.mdwrkapi.MajorDomoWorker attribute), 13
- load() (raspy.servers.core.Scenario method), 19
- load() (raspy.servers.core.ScenarioManager method), 19
- logger (in module raspy.common.MDP), 10
- M**
- MajorDomoClient (class in raspy.common.mdcliapi), 12
- MajorDomoWorker (class in raspy.common.mdwrkapi), 12
- MediaCamera (class in raspy.common.devices.media), 8
- MediaDevice (class in raspy.common.devices.media), 8
- MediaTV (class in raspy.common.devices.media), 9
- N**
- name (raspy.common.devices.device.BaseDevice attribute), 8
- name (raspy.servers.broker.Service attribute), 17
- new() (raspy.common.devices.device.BaseDevice method), 8
- new() (raspy.common.devices.device.DeviceRegister method), 8
- new() (raspy.common.devices.media.MediaCamera method), 8
- new() (raspy.common.devices.media.MediaTV method), 9
- new() (raspy.common.devices.sensor.SensorTemperature method), 9
- new() (raspy.common.devices.sensor.SensorWind method), 9
- O**
- oid (raspy.common.devices.device.BaseDevice attribute), 8
- oid (raspy.common.devices.media.MediaCamera attribute), 8
- oid (raspy.common.devices.media.MediaDevice attribute), 9
- oid (raspy.common.devices.media.MediaTV attribute), 9
- oid (raspy.common.devices.sensor.SensorDevice attribute), 9
- oid (raspy.common.devices.sensor.SensorTemperature attribute), 9
- oid (raspy.common.devices.sensor.SensorWind attribute), 9
- OneWire (class in raspy.servers.onewire), 20
- P**
- poller (raspy.common.mdcliapi.MajorDomoClient attribute), 12
- poller (raspy.servers.broker.Broker attribute), 16
- process_client() (raspy.servers.broker.Broker method), 16
- process_worker() (raspy.servers.broker.Broker method), 16
- Proxy (class in raspy.servers.broker), 17
- purge_workers() (raspy.servers.broker.Broker method), 16
- R**
- raspy (module), 21
- raspy.common (module), 15
- raspy.common.client (module), 10
- raspy.common.devices (module), 9
- raspy.common.devices.device (module), 7
- raspy.common.devices.media (module), 8
- raspy.common.devices.sensor (module), 9
- raspy.common.executive (module), 10
- raspy.common.kvcliapi (module), 11
- raspy.common.kvsimple (module), 11
- raspy.common.mdcliapi (module), 12
- raspy.common.MDP (module), 9
- raspy.common.mdwrkapi (module), 12
- raspy.common.server (module), 13

raspy.common.statistics (module), 14
 raspy.common.supervisor (module), 14
 raspy.common.zhelpers (module), 15
 raspy.servers (module), 21
 raspy.servers.broker (module), 15
 raspy.servers.core (module), 18
 raspy.servers.fake (module), 20
 raspy.servers.onewire (module), 20
 raspy.servers.titanic (module), 21
 reconnect (raspy.common.mdwrkapi.MajorDomoWorker attribute), 13
 reconnect_to_broker() (raspy.common.mdcliapi.MajorDomoClient method), 13
 reconnect_to_broker() (raspy.common.mdwrkapi.MajorDomoWorker method), 17
 recv() (raspy.common.kvsimple.KVMsg class method), 12
 recv() (raspy.common.mdwrkapi.MajorDomoWorker method), 13
 register() (raspy.common.devices.device.DeviceRegister method), 8
 reload() (raspy.common.supervisor.Supervisor method), 14
 remove_statistic() (raspy.common.statistics.Statistics method), 14
 reply_filename() (raspy.servers.titanic.Titanic method), 21
 reply_to (raspy.common.mdwrkapi.MajorDomoWorker attribute), 13
 request() (raspy.common.client.Client method), 10
 request_filename() (raspy.servers.titanic.Titanic method), 21
 requests (raspy.servers.broker.Service attribute), 17
 require_service() (raspy.servers.broker.Broker method), 16
 require_worker() (raspy.servers.broker.Broker method), 16
 retries (raspy.common.mdcliapi.MajorDomoClient attribute), 12
 Route (class in raspy.servers.broker), 17
 routing_key() (in module raspy.common.MDP), 10
 run() (raspy.common.client.Client method), 10
 run() (raspy.common.executive.Executive method), 10
 run() (raspy.common.executive.ExecutiveProcess method), 10
 run() (raspy.common.kvcliapi.KvSubscriberClient method), 11
 run() (raspy.common.supervisor.Supervisor method), 15
 run() (raspy.servers.broker.Broker method), 17
 run() (raspy.servers.broker.Proxy method), 17
 run() (raspy.servers.core.Scenario method), 19
 run() (raspy.servers.titanic.Titanic method), 21
 running (raspy.servers.core.Scenario attribute), 19
 scenarios (raspy.servers.core.ScenarioManager attribute), 20
 send() (raspy.common.kvcliapi.KvPublisherClient method), 11
 send() (raspy.common.kvsimple.KVMsg method), 12
 send() (raspy.common.mdcliapi.MajorDomoClient method), 12
 send_heartbeats() (raspy.servers.broker.Broker method), 17
 send_single() (raspy.servers.broker.Proxy method), 17
 send_to_broker() (raspy.common.mdwrkapi.MajorDomoWorker method), 13
 send_to_worker() (raspy.servers.broker.Broker method), 17
 SensorDevice (class in raspy.common.devices.sensor), 9
 SensorTemperature (class in raspy.common.devices.sensor), 9
 SensorWind (class in raspy.common.devices.sensor), 9
 sequence (raspy.common.kvsimple.KVMsg attribute), 12
 Server (class in raspy.common.server), 13
 ServerError, 10
 Service (class in raspy.servers.broker), 17
 service (raspy.common.mdwrkapi.MajorDomoWorker attribute), 13
 service (raspy.servers.broker.Worker attribute), 18
 service_internal() (raspy.servers.broker.Broker method), 17
 service_success() (raspy.servers.titanic.Titanic method), 21
 services (raspy.servers.broker.Broker attribute), 17
 set() (raspy.common.statistics.SNMP method), 14
 set() (raspy.common.statistics.SNMPCounter method), 14
 set_id() (in module raspy.common.zhelpers), 15
 shutdown() (raspy.common.executive.Executive method), 10
 shutdown() (raspy.common.executive.ExecutiveProcess method), 11
 shutdown() (raspy.common.kvcliapi.KvSubscriberClient method), 11
 shutdown() (raspy.common.mdwrkapi.MajorDomoWorker method), 13
 shutdown() (raspy.common.supervisor.Supervisor method), 15
 shutdown() (raspy.servers.broker.Broker method), 17
 shutdown() (raspy.servers.broker.Proxy method), 17
 shutdown() (raspy.servers.core.Scenario method), 19
 shutdown() (raspy.servers.core.ScenarioManager method), 20
 SNMP (class in raspy.common.statistics), 14
 SNMPCounter (class in raspy.common.statistics), 14
 SNMPPFloat (class in raspy.common.statistics), 14
 SNMPString (class in raspy.common.statistics), 14
 socket (raspy.servers.broker.Broker attribute), 17
 Statistics (class in raspy.common.statistics), 14
 status (raspy.common.mdwrkapi.MajorDomoWorker

S

Scenario (class in raspy.servers.core), 18
 ScenarioManager (class in raspy.servers.core), 19

attribute), 13
status() (raspy.common.client.Client method), 10
stop_executives() (raspy.common.supervisor.Supervisor method), 15
store() (raspy.common.kvsimple.KVMsg method), 12
store() (raspy.servers.core.Scenario method), 19
store() (raspy.servers.core.ScenarioManager method), 20
store_filename() (raspy.servers.titanic.Titanic method), 21
Supervisor (class in raspy.common.supervisor), 14

T

template (raspy.common.devices.device.BaseDevice attribute), 8
templates (raspy.common.devices.device.BaseDevice attribute), 8
timeout (raspy.common.mdcliapi.MajorDomoClient attribute), 12
timeout (raspy.common.mdwrkapi.MajorDomoWorker attribute), 13
Titanic (class in raspy.servers.titanic), 21
titanic_close() (raspy.servers.titanic.Titanic method), 21
titanic_reply() (raspy.servers.titanic.Titanic method), 21
titanic_request() (raspy.servers.titanic.Titanic method), 21
titanic_store() (raspy.servers.titanic.Titanic method), 21

U

update() (raspy.servers.core.ScenarioManager method), 20
update_statistic() (raspy.common.statistics.Statistics method), 14

V

verbose (raspy.common.mdcliapi.MajorDomoClient attribute), 12
verbose (raspy.common.mdwrkapi.MajorDomoWorker attribute), 13

W

waiting (raspy.servers.broker.Broker attribute), 17
waiting (raspy.servers.broker.Service attribute), 17
Worker (class in raspy.servers.broker), 17
worker (raspy.common.mdwrkapi.MajorDomoWorker attribute), 13
worker_cron() (raspy.servers.core.Core method), 18
worker_devices() (raspy.servers.fake.Fake method), 20
worker_devices() (raspy.servers.onewire.OneWire method), 21
worker_mmi() (raspy.common.server.Server method), 13
worker_scenario() (raspy.servers.core.Core method), 18
worker_scenarios() (raspy.servers.core.Core method), 18

worker_statistics() (raspy.common.statistics.Statistics method), 14
worker_waiting() (raspy.servers.broker.Broker method), 17
workers (raspy.servers.broker.Broker attribute), 17

Z

zpipe() (in module raspy.common.zhelpers), 15