
jps Documentation

Release

Author

June 11, 2016

1	Introduction	3
1.1	How to install	3
1.2	How to write pub/sub	3
1.3	How to run and use tools	4
2	Simple Subscriber	5
3	ActionServer and ActionClient	7
4	Serializer	9
5	jps_topic	11
5.1	jps_topic pub	11
5.2	jps_topic echo	12
5.3	jps_topic list	12
5.4	jps_topic record	12
5.5	jps_topic play	13
6	jps package	15
6.1	Module contents	15
6.2	Submodules	18
6.3	jps.utils module	18
6.4	jps.launcher module	18
6.5	jps.args module	18
6.6	jps.tools module	19
7	Environmental variables	21
8	Indices and tables	23
	Python Module Index	25

jps is simple pub/sub system for python. It is very easy to install and it is easy to understand if you have experience of ROS(Robot Operating System).

Contents:

Introduction

jps(json pub/sub) is small wrapper of [zeromq](#). It provides simple Pub/Sub system and command line tools, which is strongly inspired by [ROS\(Robot Operating System\)](#). jps is easier to install than ROS, and it does not have serialization. Let's use json format.

1.1 How to install

You can use pip to install jps

```
$ sudo pip install jps
```

It installs jps python module, jps_master and *jps_topic* command.

1.2 How to write pub/sub

publisher.py

```
import jps
import time

pub = jps.Publisher('/hoge1')
i = 0
while True:
    pub.publish('hello! jps{0}'.format(i))
    i += 1
    time.sleep(0.5)
```

subscriber.py

```
import jps

def callback(msg):
    print msg

sub = jps.Subscriber('/hoge1', callback)
sub.spin()
```

1.3 How to run and use tools

You need three consoles to test the program.

```
$ jps_master  
$ python publisher.py  
$ python subscriber.py
```

To get the list of the topics, you can use `jps_topic list`

```
$ jps_topic list
```

If you want to see the data in `/hoge1` topic,

```
$ jps_topic echo /hoge1
```

Simple Subscriber

If your subscriber is simple you can write it easier than ROS style. Simple means below here.

- subscribe only one topic
- just while loop main function

simple_subscriber

```
import jps

for msg in jps.Subscriber('/hoge1'):
    print msg
```

This prints /hoge1 messages.

You can mix the two styles if you want.

```
import jps

def callback(msg):
    print 'hoge2 = {}'.format(msg)

sub2 = jps.Subscriber('/hoge2', callback)
for msg in jps.Subscriber('/hoge1'):
    print 'hoge1 is here!{}'.format(msg)
    sub2.spin_once()
```

ActionServer and ActionClient

Do you know about actionlib of ROS? jps provide simple ActionServer and ActionClient. If you want to provide some action which takes long time, how about using these classes. It is made by pub/sub only, but it is possible to handle the response correctly, because it manage what is the response of the request.

Below is a sample of ActionServer

```
import jps
import time

def callback(req):
    time.sleep(1)
    print req + ' received'
    return True

s = jps.ActionServer('move_to', callback)
s.spin()
```

Below is a sample of ActionClient

```
import jps
import json
import time

c = jps.ActionClient('move_to')
time.sleep(0.1) # need this sleep
future = c(json.dumps({'x': 10.0, 'y': 0.1}))
print 'do something during waiting response'
time.sleep(1)
result = future.wait()
print result
```

It does not contain feedback topic, it is the difference between ROS.

Serializer

jps itself does not have serializer. If you want to serialize your payload, You have to handle yourself.

Only serialization by json is supported. If you set environ like below (for bash),

```
export JPS_SERIALIZE=json
```

payload will be automatically serialized as json. Actually, it does just `json.dumps(payload)` before published, `json.loads(payload)` after subscription.

You can use any serializer if you pass it to publisher/subscriber.

```
def my_serialize(payload):  
    return payload + 'hoge'  
  
def my_deserialize(payload):  
    return payload + 'hoge'  
  
pub = jps.Publisher('topic1', serializer=my_serialize)  
sub = jps.Subscriber('topic1', serializer=my_deserialize)
```

jps_topic

`jps_topic` is similar tool to `rostopic`. It is automatically installed by `pip install jps`.

If you want to know how to use, try `-h` option.

```
$ jps_topic -h
usage: jps_topic [-h] {pub,echo,list,record,play} ...

jps pub/sub tool

positional arguments:
  {pub,echo,list,record,play}
                        command
  pub                   publish topic from command line
  echo                 show topic data
  list                 show topic list
  record               record topic data
  play                 play recorded topic data

optional arguments:
  -h, --help            show this help message and exit
```

For all commands, you can set the host of `jps_master` by `-host HOST`, if you want to use `jps` from remote computer.

5.1 jps_topic pub

pub command publishes json text data.

```
$ jps_topic pub topic_name "{\"data\": 1.0}"
```

It publishes only once if the `--repeat` option is not specified. see `-h` option for more detail.

```
$ jps_topic pub -h
usage: jps_topic pub [-h] [--host HOST] [--publisher_port PUBLISHER_PORT] [--repeat REPEAT] topic_name data

positional arguments:
  topic_name            name of topic
  data                 json string data to be published

optional arguments:
  -h, --help            show this help message and exit
  --host HOST           master host
  --publisher_port PUBLISHER_PORT, -p PUBLISHER_PORT
```

```

        publisher port
--repeat REPEAT, -r REPEAT
        repeat in hz

```

5.2 jps_topic echo

echo command prints json text data.

```

$ jps_topic echo -h
usage: jps_topic echo [-h] [--host HOST] [--subscriber_port SUBSCRIBER_PORT] [--num NUM] topic_name

positional arguments:
  topic_name            name of topic

optional arguments:
  -h, --help            show this help message and exit
  --host HOST            master host
  --subscriber_port SUBSCRIBER_PORT, -s SUBSCRIBER_PORT
                        subscriber port
  --num NUM, -n NUM     print N times and exit

```

5.3 jps_topic list

list command collects all topics and create list of topic names, which is published now. It subscribes 1[sec] to create the list. If the topic is published less than 1[Hz], the list command may not catch the name. You can use `--timeout` option to catch the slow topics.

```

$ jps_topic list --help
usage: jps_topic list [-h] [--host HOST] [--subscriber_port SUBSCRIBER_PORT] [--timeout TIMEOUT]

optional arguments:
  -h, --help            show this help message and exit
  --host HOST            master host
  --subscriber_port SUBSCRIBER_PORT, -s SUBSCRIBER_PORT
                        subscriber port
  --timeout TIMEOUT, -t TIMEOUT
                        timeout in sec

```

5.4 jps_topic record

record command is like `roscat` command. It records the topic data to the file. You can replay the data by `play` command. You can use `--file` option to specify the output file name. Default is `record.json`. You can set the topic name to be recorded. If the `topic_names` is empty, all topics will be recorded. Because the file format is normal json, you can read/parse it by any json reader if you want.

```

$ jps_topic record -h
usage: jps_topic record [-h] [--host HOST] [--subscriber_port SUBSCRIBER_PORT] [--file FILE] [topic_names]

positional arguments:
  topic_names            topic names to be recorded

```



```
optional arguments:
  -h, --help            show this help message and exit
  --host HOST            master host
  --subscriber_port SUBSCRIBER_PORT, -s SUBSCRIBER_PORT
                        subscriber port
  --file FILE, -f FILE  output file name (default: record.json)
```

5.5 jps_topic play

play command replays the saved data by `jps_topic record`.

```
$ jps_topic play -h
usage: jps_topic play [-h] [--host HOST] [--publisher_port PUBLISHER_PORT] file

positional arguments:
  file                input file name

optional arguments:
  -h, --help            show this help message and exit
  --host HOST            master host
  --publisher_port PUBLISHER_PORT, -p PUBLISHER_PORT
                        publisher port
```

jps package

6.1 Module contents

class `jps.Publisher` (*topic_name*, *host=None*, *pub_port=None*, *serializer='DEFAULT'*)

Bases: `object`

Publishes data for a topic.

Example:

```
>>> pub = jps.Publisher('special_topic')
>>> pub.publish({'name': 'hoge'})
```

Parameters

- **topic_name** – Topic name
- **host** – host of subscriber/forwarder
- **pub_port** – port of subscriber/forwarder
- **serializer** – this function is applied before publish (default: None)

publish (*payload*)

Publish payload to the topic

Note: If you publishes just after creating Publisher instance, it will causes lost of message. You have to add sleep if you just want to publish once.

```
>>> pub = jps.Publisher('topic')
>>> time.sleep(0.1)
>>> pub.publish('{data}')
```

Parameters payload – data to be published. This is ok if the data is not json.

class `jps.Subscriber` (*topic_name*, *callback=None*, *host=None*, *sub_port=None*, *deserializer='DEFAULT'*)

Bases: `object`

Subscribe the topic and call the callback function

Example:

```
>>> def callback(msg):
...     print msg
...
>>> sub = jps.Subscriber('topic_name', callback)
>>> sub.spin()
```

or you can use python generator style

```
>>> import jps
>>> for msg in jps.Subscriber('/hogel'):
...     print msg
```

Parameters

- **topic_name** – topic name
- **host** – host name of publisher/forwarder
- **sub_port** – port of publisher/forwarder
- **deserializer** – this function is applied after received (default: None)

deserialize (*msg*)

next ()

receive next data (block until next data)

spin (*use_thread=False*)

call callback for all data forever (until C-c)

Parameters **use_thread** – use thread for spin (do not block)

spin_once (*polling_sec=0.01*)

Read the queued data and call the callback for them. You have to handle KeyboardInterrupt (C-c) manually.

Example:

```
>>> def callback(msg):
...     print msg
>>> sub = jps.Subscriber('topic_name', callback)
>>> try:
...     while True:
...         sub.spin_once():
...         time.sleep(0.1)
...     except KeyboardInterrupt:
...         pass
```

class jps.ArgumentParser (*subscriber=True, publisher=True, service=False, *args, **kwargs*)

Bases: `argparse.ArgumentParser`

Create ArgumentParser with args (host/subscriber_port/publisher_port)

Example:

```
>>> parser = jps.ArgumentParser(description='my program')
>>> args = parser.parse_args()
>>> args.host
'localhost'
>>> args.subscriber_port
54321
>>> args.publisher_port
54320
```

Parameters

- **add subscriber_port** (default (*subscriber*) – True)
- **add publisher_port** (default (*publisher*) – True)

class `jps.ServiceServer` (*callback*, *host=None*, *res_port=54323*)

Bases: `object`

Example:

```
>>> def callback(req):
...     return 'req = {req}'.format(req=req)
...
>>> service = jps.ServiceServer(callback)
>>> service.spin()
```

spin (*use_thread=False*)

call callback for all data forever (until C-c)

Parameters *use_thread* – use thread for spin (do not block)

spin_once ()

class `jps.ServiceClient` (*host=None*, *req_port=54322*)

Bases: `object`

call (*request*)

class `jps.ActionServer` (*base_topic_name*, *callback*, *host=None*, *pub_port=None*, *sub_port=None*, *serializer='DEFAULT'*, *deserializer='DEFAULT'*)

Bases: `object`

serve the service which takes some long time

Example:

```
>>> import jps
>>> import time
>>> def callback(req):
...     time.sleep(1)
...     return req + ' received'
>>> s = jps.ActionServer('move_to', callback)
# subscribe 'move_to/request', publish 'move_to/response'
>>> s.spin()
```

spin (*use_thread=False*)

spin_once ()

class `jps.ActionClient` (*base_topic_name*, *host=None*, *pub_port=None*, *sub_port=None*, *serializer='DEFAULT'*, *deserializer='DEFAULT'*)

Bases: `object`

Call an action

Example:

```
>>> import jps
>>> import json
>>> c = jps.ActionClient('move_to')
>>> future = c(json.dumps({'x': 10.0, 'y': 0.1}))
# do something if you are busy to do something during waiting.
>>> result = future.wait()
```

```
class jps.Bridge(upload_topic_names, download_topic_names, remote_host=None, re-
                 mote_pub_port=None, remote_sub_port=None)
    Bases: object

    spin()

exception jps.Error
    Bases: exceptions.Exception
```

6.2 Submodules

6.3 jps.utils module

```
class jps.utils.JsonMultiplePublisher
    Bases: object

    publish multiple topics by one json message

    Example:
```

```
>>> p = JsonMultiplePublisher()
>>> p.publish('{"topic1": 1.0, "topic2": {"x": 0.1}}')
```

```
    publish(json_msg)
        json_msg = '{"topic1": 1.0, "topic2": {"x": 0.1}}'
```

```
class jps.utils.MultiplePublisher(base_topic_name)
    Bases: object

    publish(msg, topic_suffix='')
```

6.4 jps.launcher module

```
jps.launcher.launch_modules(module_names)
    launch module.main functions in another process
```

6.5 jps.args module

```
class jps.args.ArgumentParser(subscriber=True, publisher=True, service=False, *args, **kwargs)
    Bases: argparse.ArgumentParser

    Create ArgumentParser with args (host/subscriber_port/publisher_port)
```

Example:

```
>>> parser = jps.ArgumentParser(description='my program')
>>> args = parser.parse_args()
>>> args.host
'localhost'
>>> args.subscriber_port
54321
>>> args.publisher_port
54320
```

Parameters

- **add subscriber_port** (default (*subscriber*) – True)
- **add publisher_port** (default (*publisher*) – True)

6.6 jps.tools module

`jps.tools.echo` (*topic_name*, *num_print*=None, *out*=<open file '<stdout>', mode 'w'>, *host*='localhost', *sub_port*=54321)
print the data for the given topic forever

`jps.tools.play` (*file_path*, *host*='localhost', *pub_port*=54320)
replay the recorded data by record()

`jps.tools.pub` (*topic_name*, *json_msg*, *repeat_rate*=None, *host*='localhost', *pub_port*=54320)
publishes the data to the topic

Parameters

- **topic_name** – name of the topic
- **json_msg** – data to be published
- **repeat_rate** – if None, publishes once. if not None, it is used as [Hz].

`jps.tools.record` (*file_path*, *topic_names*=[], *host*='localhost', *sub_port*=54321)
record the topic data to the file

`jps.tools.show_list` (*timeout_in_sec*, *out*=<open file '<stdout>', mode 'w'>, *host*='localhost', *sub_port*=54321)
get the name list of the topics, and print it

`jps.tools.topic_command` ()
command line tool for jps

Environmental variables

jps uses below environmental variables. These are optional. You don't need to set these variables (default will be used.)

- JPS_MASTER_HOST: set default master host. This will be overwritten by args. (default: "localhost")
- JPS_SUFFIX: Add this to all topic names. This is for multi robot system. (default: "")
- JPS_MASTER_PUB_PORT: port number for publishers (default: 54320)
- JPS_MASTER_SUB_PORT: port number for subscribers (default: 54321)
- JPS_SERIALIZE: default serialzier. Only 'json' is supported. (default: None)
- JPS_REMAP: remap topic names. If you set 'export JPS_REMAP="hoge=foo"', topic 'hoge' will be changed to 'foo'

Indices and tables

- `genindex`
- `modindex`
- `search`

j

- [jps](#), 15
- [jps.args](#), 18
- [jps.launcher](#), 18
- [jps.tools](#), 19
- [jps.utils](#), 18

A

ActionClient (class in jps), 17
ActionServer (class in jps), 17
ArgumentParser (class in jps), 16
ArgumentParser (class in jps.args), 18

B

Bridge (class in jps), 17

C

call() (jps.ServiceClient method), 17

D

deserialize() (jps.Subscriber method), 16

E

echo() (in module jps.tools), 19
Error, 18

J

jps (module), 15
jps.args (module), 18
jps.launcher (module), 18
jps.tools (module), 19
jps.utils (module), 18
JsonMultiplePublisher (class in jps.utils), 18

L

launch_modules() (in module jps.launcher), 18

M

MultiplePublisher (class in jps.utils), 18

N

next() (jps.Subscriber method), 16

P

play() (in module jps.tools), 19
pub() (in module jps.tools), 19

publish() (jps.Publisher method), 15
publish() (jps.utils.JsonMultiplePublisher method), 18
publish() (jps.utils.MultiplePublisher method), 18
Publisher (class in jps), 15

R

record() (in module jps.tools), 19

S

ServiceClient (class in jps), 17
ServiceServer (class in jps), 17
show_list() (in module jps.tools), 19
spin() (jps.ActionServer method), 17
spin() (jps.Bridge method), 18
spin() (jps.ServiceServer method), 17
spin() (jps.Subscriber method), 16
spin_once() (jps.ActionServer method), 17
spin_once() (jps.ServiceServer method), 17
spin_once() (jps.Subscriber method), 16
Subscriber (class in jps), 15

T

topic_command() (in module jps.tools), 19