00A0 $\exists$ $\forall$ $\subseteq$ 2713x 27FA$\Longleftrightarrow$ $\surd$ $\sqrt[3]{}$ $\oplus$ $\otimes$

roslibpy-docs-zh

0.6.0

2020 07 08

Gramazio Kohler Research

Wu Hsin

:

Python ROS Bridge library **roslibpy** Python IronPython ROS WebSockets rosbridge 2.0 publishing subscribing service calls actionlib TF ROS

rospy **ROS** Linux ROS

roslibpy API roslibjs

2019 6 8 **roslibpy** 0.6.0

CHAPTER 1

Python ROS Bridge library **roslibpy** Python IronPython ROS WebSockets rosbridge 2.0 publishing subscribing service calls actionlib TF ROS

rospy **ROS** Linux ROS

roslibpy API roslibjs

1.1

- (Topic)
- (Service)
- ROS (get/set/delete)
- ROS API
- Actionlib
- tf2_web_republisher TF Client

Roslibpy Python 2.7 Python 3.x IronPython 2.7

1.2

`pip` **roslibpy**:

```
pip install roslibpy
```

IronPython `pip` :

```
ipy -X:Frames -m pip install --user roslibpy
```

1.3

1.4

- roslibpy
-
- :

```
pip install -r requirements-dev.txt
```

pyinvoke

- `invoke clean`:
- `invoke check`:
- `invoke docs`:
- `invoke test`:
- `invoke`:

1.5

roslibpy

- semver
  - `patch`: BUG
  - `minor`: Feature
  - `major`:
- `CHANGELOG.rst`
- :

```
invoke release [patch|minor|major]
```

- Profit!

1.6 Credits

roslibjs Python roslibjs

CHAPTER 2

roslibpy ROS

: ROS **rosbridge server** **TF2 web republisher** *ROS*

ROS `host` `'localhost'` ROS master *IP*

: `port` `9090` `rosbridge` `9090`

2.1

`roslibpy`:

```
>>> import roslibpy
```

:

```
>>> ros = roslibpy.Ros(host='localhost', port=9090)
>>> ros.run()
```

:

```
>>> ros.is_connected
True
```

(• ω •)y

2.2

Python

ros-hello-world.py

```
import roslibpy

client = roslibpy.Ros(host='localhost', port=9090)
client.run()
print('Is ROS connected?', client.is_connected)
client.terminate()
```

:

```
$ python ros-hello-world.py
```

ROS

2.3

run() ROS *roslibpy.Ros* run_forever()

roslibpy.Ros.on_ready()

run_forever() on_ready()

```
from __future__ import print_function
import roslibpy

client = roslibpy.Ros(host='localhost', port=9090)
client.on_ready(lambda: print('Is ROS connected?', client.is_connected))
client.run_forever()
```

: run() run_forever()

2.4 Hello World: Topics

ROS Hello World / talker listener ROS

roslibpy

2.4.1 talker

ROS Ctrl+C

```
import time

import roslibpy
```

( )

( )

```
client = roslibpy.Ros(host='localhost', port=9090)
client.run()

talker = roslibpy.Topic(client, '/chatter', 'std_msgs/String')

while client.is_connected:
    talker.publish(roslibpy.Message({'data': 'Hello World!'}))
    print('Sending message...')
    time.sleep(1)

talker.unadvertise()

client.terminate()
```

- ros-hello-world-talker.py

: ROS std_msgs/String Python

2.4.2 listener

Listener

```
from __future__ import print_function
import roslibpy

client = roslibpy.Ros(host='localhost', port=9090)
client.run()

listener = roslibpy.Topic(client, '/chatter', 'std_msgs/String')
listener.subscribe(lambda message: print('Heard talking: ' + message['data']))

try:
    while True:
        pass
except KeyboardInterrupt:
    client.terminate()
```

- ros-hello-world-listener.py

2.4.3

talker :

```
$ python ros-hello-world-talker.py
```

listener :

```
$ python ros-hello-world-listener.py
```

:	Ros master

2.5 Services

ROS

get_loggers

```
import roslibpy

client = roslibpy.Ros(host='localhost', port=9090)
client.run()

service = roslibpy.Service(client, '/rosout/get_loggers', 'roscpp/GetLoggers')
request = roslibpy.ServiceRequest()

print('Calling service...')
result = service.call(request)
print('Service response: {}'.format(result['loggers']))

client.terminate()
```

- ros-service-call-logger.py

2.6 Services

ROS

ROS std_srvs/SetBool :

```
import roslibpy

def handler(request, response):
    print('Setting speed to {}'.format(request['data']))
    response['success'] = True
    return True

client = roslibpy.Ros(host='localhost', port=9090)

service = roslibpy.Service(client, '/set_ludicrous_speed', 'std_srvs/SetBool')
service.advertise(handler)
print('Service advertised.')

client.run_forever()
client.terminate()
```

- ros-service.py

:

```
$ python ros-service.py
```

Ctrl+C

:

- ros-service-call-set-bool.py

:

```
$ python ros-service-call-set-bool.py
```

: **roslibpy** *API*

2.7 Actions

ROS **Actions**

roslibpy action *roslibpy.actionlib.SimpleActionServer* action

action action actionlib_tutorials

2.7.1 Action

```
import roslibpy
import roslibpy.actionlib

client = roslibpy.Ros(host='localhost', port=9090)
server = roslibpy.actionlib.SimpleActionServer(client, '/fibonacci', 'actionlib_
↪tutorials/FibonacciAction')

def execute(goal):
    print('Received new fibonacci goal: {}'.format(goal['order']))

    seq = [0, 1]

    for i in range(1, goal['order']):
        if server.is_preempt_requested():
            server.set_preempted()
            return

        seq.append(seq[i] + seq[i - 1])
        server.send_feedback({'sequence': seq})

    server.set_succeeded({'sequence': seq})


server.start(execute)
client.run_forever()
```

- ros-action-server.py

  :

```
$ python ros-action-server.py
```

action Ctrl+C

2.7.2 Action

action

action

```
from __future__ import print_function
import roslibpy
import roslibpy.actionlib

client = roslibpy.Ros(host='localhost', port=9090)
client.run()

action_client = roslibpy.actionlib.ActionClient(client,
                                                '/fibonacci',
                                                'actionlib_tutorials/FibonacciAction')

goal = roslibpy.actionlib.Goal(action_client,
                               roslibpy.Message({'order': 8}))

goal.on('feedback', lambda f: print(f['sequence']))
goal.send()
result = goal.wait(10)
action_client.dispose()

print('Result: {}'.format(result['sequence']))
```

- ros-action-client.py

  :

```
$ python ros-action-client.py
```

action

roslibpy.actionlib.Goal.wait() result

CHAPTER 3

API

Robot Web Tools ROS bridge suite WebSockets ROS
ROS bridge protocol JSON publishing, subscribing, service calls, actionlib, TF ROS

3.1 ROS

ROS `rosbridge`

ROS master **rosbridge suite**:

```
sudo apt-get install -y ros-kinetic-rosbridge-server
sudo apt-get install -y ros-kinetic-tf2-web-republisher
```

:

```
roslaunch rosbridge_server rosbridge_websocket.launch
rosrun tf2_web_republisher tf2_web_republisher
```

: `roslaunch` `roscore` `roscore`

3.2 ROS

ROS *Ros*

ROS

class roslibpy.**Ros**(*host*, *port=None*, *is_secure=False*)
Connection manager to ROS server.

`call_in_thread`(*callback*)
Call the given function in a thread.

The threading implementation is deferred to the factory.

Args: callback (`callable`): Callable function to be invoked.

`call_later`(*delay*, *callback*)
Call the given function after a certain period of time has passed.

Args: delay (`int`): Number of seconds to wait before invoking the callback. callback (`callable`): Callable function to be invoked when ROS connection is ready.

`close`()
Disconnect from ROS master.

`connect`()
Connect to ROS master.

`emit`(*event_name*, **args*)
Trigger a named event.

`get_action_servers`(*callback*, *errback=None*)
Retrieve list of action servers in ROS.

`get_message_details`(*message_type*, *callback*, *errback=None*)
Retrieve details of a message type in ROS.

`get_node_details`(*node*, *callback*, *errback=None*)
Retrieve list subscribed topics, publishing topics and services of a specific node name.

`get_nodes`(*callback*, *errback=None*)
Retrieve list of active node names in ROS.

`get_params`(*callback*, *errback=None*)
Retrieve list of param names from the ROS Parameter Server.

`get_service_request_details`(*type*, *callback*, *errback=None*)
Retrieve details of a ROS Service Request.

`get_service_response_details`(*type*, *callback*, *errback=None*)
Retrieve details of a ROS Service Response.

`get_service_type`(*service_name*, *callback*, *errback=None*)
Retrieve the type of a service in ROS.

`get_services`(*callback*, *errback=None*)
Retrieve list of active service names in ROS.

`get_services_for_type`(*service_type*, *callback*, *errback=None*)
Retrieve list of services in ROS matching the specified type.

`get_topic_type`(*topic*, *callback*, *errback=None*)
Retrieve the type of a topic in ROS.

`get_topics`(*callback*, *errback=None*)
Retrieve list of topics in ROS.

`get_topics_for_type`(*topic_type*, *callback*, *errback=None*)
Retrieve list of topics in ROS matching the specified type.

`id_counter`
Generate an auto-incremental ID starting from 1.

Returns: int: An auto-incremented ID.

`is_connected`
Indicate if the ROS connection is open or not.

Returns: bool: True if connected to ROS, False otherwise.

`off`(*event_name*, *callback=None*)
Remove a callback from an arbitrary named event.

Args: event_name (`str`): Name of the event from which to unsubscribe. callback: Callable function. If `None`, all callbacks of the event will be removed.

`on`(*event_name*, *callback*)
Add a callback to an arbitrary named event.

Args: event_name (`str`): Name of the event to which to subscribe. callback: Callable function to be executed when the event is triggered.

`on_ready`(*callback*, *run_in_thread=True*)
Add a callback to be executed when the connection is established.

If a connection to ROS is already available, the callback is executed immediately.

Args: callback: Callable function to be invoked when ROS connection is ready. run_in_thread (`bool`): True to run the callback in a separate thread, False otherwise.

`run`(*timeout=None*)
Kick-starts a non-blocking event loop.

Args: timeout: Timeout to wait until connection is ready.

`run_forever`()
Kick-starts a blocking loop to wait for events.

Depending on the implementations, and the client applications, running this might be required or not.

`send_on_ready`(*message*)
Send message to the ROS Master once the connection is established.

If a connection to ROS is already available, the message is sent immediately.

Args: message (*Message*): ROS Bridge Message to send.

`send_service_request`(*message*, *callback*, *errback*)
Send a service request to the ROS Master once the connection is established.

If a connection to ROS is already available, the request is sent immediately.

Args: message (*Message*): ROS Bridge Message containing the request. callback: Callback invoked on successful execution. errback: Callback invoked on error.

`terminate`()
Signals the termination of the main event loop.

3.3 ROS

3.3.1

ROS ROS , messages **ROS messages** *Message* *Topics* /

class roslibpy.Message(*values=None*)
Message objects used for publishing and subscribing to/from topics.

A message is fundamentally a dictionary and behaves as one.

class roslibpy.Topic(*ros*, *name*, *message_type*, *compression=None*, *latch=False*, *throttle_rate=0*, *queue_size=100*, *queue_length=0*)
Publish and/or subscribe to a topic in ROS.

Args: ros (*Ros*): Instance of the ROS connection. name (`str`): Topic name, e.g. `/cmd_vel`. message_type (`str`): Message type, e.g. `std_msgs/String`. compression (`str`): Type of compression to use, e.g. *png*. Defaults to *None*. throttle_rate (`int`): Rate (in ms between messages) at which to throttle the topics. queue_size (`int`): Queue size created at bridge side for re-publishing webtopics. latch (`bool`): True to latch the topic when publishing, False otherwise. queue_length (`int`): Queue length at bridge side used when subscribing.

advertise()
Register as a publisher for the topic.

is_advertised
Indicate if the topic is currently advertised or not.

Returns: bool: True if advertised as publisher of this topic, False otherwise.

is_subscribed
Indicate if the topic is currently subscribed or not.

Returns: bool: True if subscribed to this topic, False otherwise.

publish(*message*)
Publish a message to the topic.

Args: message (*Message*): ROS Bridge Message to publish.

subscribe(*callback*)
Register a subscription to the topic.

Every time a message is published for the given topic, the callback will be called with the message object.

Args: callback: Function to be called when messages of this topic are published.

unadvertise()
Unregister as a publisher for the topic.

unsubscribe()
Unregister from a subscribed the topic.

3.3.2

/ ROS *Services* /

class roslibpy.Service(*ros*, *name*, *service_type*)
Client/server of ROS services.

This class can be used both to consume other ROS services as a client, or to provide ROS services as a server.

Args: ros (*Ros*): Instance of the ROS connection. name (`str`): Service name, e.g. `/add_two_ints`. service_type (`str`): Service type, e.g. `rospy_tutorials/AddTwoInts`.

`advertise`(*callback*)
Start advertising the service.

This turns the instance from a client into a server. The callback will be invoked with every request that is made to the service.

If the service is already advertised, this call does nothing.

Args:

callback: Callback invoked on every service call. It should accept two parameters: ***service_req***
service_response. It should return *True* if executed correctly, otherwise *False*.

`call`(*request*, *callback=None*, *errback=None*, *timeout=None*)
Start a service call.

The service can be used either as blocking or non-blocking. If the `callback` parameter is `None`, then the call will block until receiving a response. Otherwise, the service response will be returned in the callback.

Args: request (*ServiceRequest*): Service request. callback: Callback invoked on successful execution. errback: Callback invoked on error. timeout: Timeout for the operation, in seconds. Only used if blocking.

Returns: object: Service response if used as a blocking call, otherwise `None`.

`is_advertised`
Service servers are registered as advertised on ROS.

This class can be used to be a service client or a server.

Returns: bool: True if this is a server, False otherwise.

`unadvertise()`
Unregister as a service server.

`class roslibpy.ServiceRequest`(*values=None*)
Request for a service call.

`class roslibpy.ServiceResponse`(*values=None*)
Response returned from a service call.

3.3.3

ROS *Param*

`class roslibpy.Param`(*ros*, *name*)
A ROS parameter.

Args: ros (*Ros*): Instance of the ROS connection. name (`str`): Parameter name, e.g. `max_vel_x`.

`delete`(*callback=None*, *errback=None*)
Delete the parameter.

Args: callback: Callable function to be invoked when the operation is completed. errback: Callback invoked on error.

`get`(*callback=None*, *errback=None*)
Fetch the current value of the parameter.

Args: callback: Callable function to be invoked when the operation is completed. errback: Callback invoked on error.

set(*value*, *callback=None*, *errback=None*)
Set a new value to the parameter.

Args: value: Value to set the parameter to. callback: Callable function to be invoked when the operation is completed. errback: Callback invoked on error.

3.4 Actionlib

ROS **actionlib** ROS Actions

Actions *ActionClient* *Goals* goal Action **status result feedback timeout**

class roslibpy.actionlib.Goal(*action_client*, *goal_message*)
Goal for an action server.

After an event has been added to an action client, it will emit different events to indicate its progress:

- `status`: fires to notify clients on the current state of the goal.
- `feedback`: fires to send clients periodic auxiliary information of the goal.
- `result`: fires to send clients the result upon completion of the goal.
- `timeout`: fires when the goal did not complete in the specified timeout window.

Args: action_client (*ActionClient*): Instance of the action client associated with the goal. goal_message (*Message*): Goal for the action server.

cancel()
Cancel the current goal.

is_finished
Indicate if the goal is finished or not.

Returns: bool: True if finished, False otherwise.

send(*result_callback=None*, *timeout=None*)
Send goal to the action server.

Args: timeout (`int`): Timeout for the goal's result expressed in seconds. callback (`callable`): Function to be called when a result is received. It is a shorthand for hooking on the `result` event.

wait(*timeout=None*)
Block until the result is available.

If `timeout` is `None`, it will wait indefinitely.

Args: timeout (`int`): Timeout to wait for the result expressed in seconds.

Returns: Result of the goal.

class roslibpy.actionlib.ActionClient(*ros*, *server_name*, *action_name*, *timeout=None*, *omit_feedback=False*, *omit_status=False*, *omit_result=False*)
Client to use ROS actions.

Args: ros (*Ros*): Instance of the ROS connection. server_name (`str`): Action server name, e.g. `/fibonacci`. action_name (`str`): Action message name, e.g. `actionlib_tutorials/FibonacciAction`. timeout (`int`): **Deprecated.** Connection timeout, expressed in seconds.

`add_goal`(*goal*)
Add a goal to this action client.

Args: goal (*Goal*): Goal to add.

`cancel`()
Cancel all goals associated with this action client.

`dispose`()
Unsubscribe and unadvertise all topics associated with this action client.

`class roslibpy.actionlib.SimpleActionServer`(*ros*, *server_name*, *action_name*)
Implementation of the simple action server.

The server emits the following events:

- `goal`: fires when a new goal has been received by the server.
- `cancel`: fires when the client has requested the cancellation of the action.

Args: ros (*Ros*): Instance of the ROS connection. server_name (`str`): Action server name, e.g. `/fibonacci`. action_name (`str`): Action message name, e.g. `actionlib_tutorials/FibonacciAction`.

`is_preempt_requested`()
Indicate whether the client has requested preemption of the current goal.

`send_feedback`(*feedback*)
Send feedback.

Args: feedback (`dict`): Dictionary of key/values of the feedback message.

`set_preempted`()
Set the current action to preempted (cancelled).

`set_succeeded`(*result*)
Set the current action state to succeeded.

Args: result (`dict`): Dictionary of key/values to set as the result of the action.

`start`(*action_callback*)
Start the action server.

Args: action_callback: Callable function to be invoked when a new goal is received. It takes one paramter containing the goal message.

`class roslibpy.actionlib.GoalStatus`
Valid goal statuses.

3.5 TF

ROS TF2 **roslibpy** *TFClient* tf2_web_republisher

`class roslibpy.tf.TFClient`(*ros*, *fixed_frame='/base_link'*, *angular_threshold=2.0*, *translation_threshold=0.01*, *rate=10.0*, *update_delay=50*, *topic_timeout=2000.0*, *server_name='/tf2_web_republisher'*, *repub_service_name='/republish_tfs'*)
A TF Client that listens to TFs from tf2_web_republisher.

Args: ros (*Ros*): Instance of the ROS connection. fixed_frame (`str`): Fixed frame, e.g. `/base_link`. angular_threshold (`float`): Angular threshold for the TF republisher. translation_threshold (`float`): Translation threshold for the TF republisher. rate (`float`): Rate for the TF republisher. update_delay (`int`): Time expressed in milliseconds to wait after a new subscription to update the TF republisher's list of TFs. topic_timeout (`int`): Timeout parameter for the TF republisher expressed in milliseconds. repub_service_name (`str`): Name of the republish tfs service, e.g. `/republish_tfs`.

`dispose()`
Unsubscribe and unadvertise all topics associated with this instance.

`subscribe`(*frame_id*, *callback*)
Subscribe to the given TF frame.

Args: frame_id (`str`): TF frame identifier to subscribe to. callback (`callable`): A callable functions receiving one parameter with *transform* data.

`unsubscribe`(*frame_id*, *callback*)
Unsubscribe from the given TF frame.

Args: frame_id (`str`): TF frame identifier to unsubscribe from. callback (`callable`): The callback function to remove.

`update_goal()`
Send a new service request to the tf2_web_republisher based on the current list of TFs.

CHAPTER 4

4.1

pull request

1. Fork clone
2. `virtualenv conda`
3. :

```
pip install -r requirements-dev.txt
```

4. :

```
invoke test
```

5.
6. :

```
invoke test
```

7. `AUTHORS.rst`
8. Commit+push GitHub
9. GitHub pull request

pyinvoke

- `invoke clean`:
- `invoke check`:

- `invoke docs`:
- `invoke test`:
- `invoke`:

4.2

/ / docstrings API
reStructuredText Sphinx HTML
:

```
invoke docs
```

4.3 BUG

BUG

-
- ROS
-
- BUG

4.4 Feature

GitHub issue feature

-
-

CHAPTER 5

5.1

- Gramazio Kohler Research @gramaziokohler
- Gonzalo Casas <casas@arch.ethz.ch> @gonzalocasas
- Mathias Ldtke @ipa-mdl

5.2

- Wu Hsin @XinArkh

CHAPTER 6

changelog Keep a Changelog (Semantic Versioning)

0.6.0

Changed

- For consistency, `timeout` parameter of `Goal.send()` is now expressed in **seconds**, instead of milliseconds.

Deprecated

- The `timeout` parameter of `ActionClient()` is ignored in favor of blocking until the connection is established.

Fixed

- Raise exceptions when timeouts expire on ROS connection or service calls.

Added

- Support for calling a function in a thread from the Ros client.
- Added implementation of a Simple Action Server.

0.5.0

Changed

- The non-blocking event loop runner now waits for the connection to be established in order to minimize the need for `on_ready` handlers.

Added

- Support blocking and non-blocking service calls.

Fixed

- Fixed an internal unsubscribing issue.

0.4.1

Fixed

- Resolve reconnection issues.

0.4.0

Added

- Add a non-blocking event loop runner

0.3.0

Changed

- Unsubscribing from a listener no longer requires the original callback to be passed.

0.2.1

Fixed

- Fix JSON serialization error on TF Client (on Python 3.x)

0.2.0

Added

- Add support for IronPython 2.7

Changed

- Handler `on_ready` now defaults to run the callback in thread

Deprecated

- Rename `run_event_loop` to the more fitting `run_forever`

0.1.1

Fixed

- Minimal documentation fixes

0.1.0

Added

- Initial version

CHAPTER 7

roslibpy

Sphinx + reStructuredText + readthedocs

GitHub star ~

>> **roslibpy** roslibpy <<

>> roslibpy-docs-zh <<

Wu Hsin

2019.4.26.

2019.6.8. edited.

Until the end of the world.

CHAPTER 8

- genindex
- modindex
- search

r

T

U

W