Hermes Documentation

Release 1.0

Nils Diefenbach

Aug 08, 2018

Contents

Contents:

CHAPTER 1

Hermes - Reference

Module loader.

Node Class to pull together receivers and publishers.

Functions as the smallest available unit with which can be communicated in a cluster.

It offers slots for a Publisher and Receiver object. Each of these must implement at least a start() and stop() function, as well as a recv() (Receiver) and publish() (Publisher) method. The passed objects are therefore not limited to `hermes.Publisher` and `hermes.Receiver` objects.

When left unmodified, the Node will simply pass data from the receiver to the publisher.

`hermes.Node` supports the *with* statement and will start up all facilities it has stored in its instance's `hermes.Node.facilities` property. These will also be stopped after leaving the with block, respectively.

class `hermes.node.`**Node**(*name*, *receiver=None*, *publisher=None*)

Basic Node Class.

Provides a basic interface for starting and stopping a node.

Extend this as necessary.

facilities

Return the names of facilities registered with this `hermes.Node` instance.

publish(*channel*, *data*)

Publish the given data to channel, if it is available.

The object must implement `hermes.Publisher.publish()` method, otherwise a :exception:`NotImplementedError` is raised.

The topic is generated from channel and `hermes.Node.name`.

Parameters

- **channel** – topic tree
- **data** – Data object to send via the publisher.

Returns `None`

recv(*block=False*, *timeout=None*)
Receive data from the receiver instance, if available.

The object at :attr:`hermes.node.Node.receiver` must implement a `recv(block, timeout)` method, otherwise an `NotImplementedError` is raised.

run()
Execute the main loop, which can be extended as necessary.

If not extended, the following loop will be executed while `hermes.Node._running` is True:

1. call `hermes.Node.recv()` and check if there's a message
2. **if a message was received:** call `hermes.Node.publish()` and send message.
3. Repeat.

start()
Start the `hermes.Node` instance and its facilities.

stop()
Stop the `hermes.Node` instance and its facilities.

Receiver Component for usage in Node class.

class hermes.receiver.**Receiver**(*sub_addr*, *name*, *topics=None*, *exchanges=None*)
Class providing a connection to one or many ZMQ Publisher(s).

join(*timeout=None*)
Join the `hermes.Receiver` instance.

Clears the `hermes.Receiver._is_running` flag, causing a graceful shutdown of the run loop.

Parameters **timeout** – timeout in seconds passed to `threading.Thread.join()`

Returns `None`

recv(*block=False*, *timeout=None*)
Wrap around `Queue.get()`.

Returns the popped value or `None` if the `queue.Queue` is empty.

Returns data or `None`

run()
Execute the custom run loop for the `hermes.Receiver` class.

It connectos to a ZMQ publisher on the local machine using the ports found in `hermes.Receiver.ports`. If this is empty, it simply loops doing nothing.

Returns `None`

stop(*timeout=None*)
Stop the `hermes.Receiver` instance.

Parameters **timeout** – time in seconds until `TimeOutError` is raised

Returns `None`

Publisher component for use in a Node class.

class hermes.publisher.**Publisher**(*target_addr*, *name*, *ctx=None*, *socket_type=None*)
Allows publishing data to subscribers.

The publishing is realized with ZMQ's Publisher sockets, and supports publishing to multiple subscribers.

The `hermes.Publisher.run()` method continuously checks for data on the internal q, which is fed by the `hermes.Publisher.publish()` method.

join(*timeout=None*)
Join the `hermes.Publisher` instance and shut it down.

Clears the `hermes.Publisher._running` flag to gracefully terminate the run loop.

Parameters **timeout** – timeout in seconds to wait for `hermes.Publisher.join()` to finish

Returns `None`

publish(*envelope*)
Publish the given data to all current subscribers.

Parameters **envelope** – `hermes.Envelope` instance

Returns None

run()
Customized run loop to publish data.

Sets up a ZMQ publisher socket and sends data as soon as it is available on the internal Queue at `hermes.Publisher.q`.

Returns **:cls:'None'**

stop(*timeout=None*)
Stop the `hermes.Publisher` instance.

Parameters **timeout** – time in seconds until `TimeOutError` is raised

Returns `None`

Basic XPub/XSub Proxy Interface for a cluster.

class `hermes.proxy.`**PostOffice**(*proxy_in*, *proxy_out*, *debug_addr=None*)
Class to forward subscriptions from publishers to subscribers.

Uses `zmq.XSUB` & `zmq.XPUB` ZMQ sockets to act as intermediary. Subscribe to these using the respective PUB or SUB socket by binding to the same address as XPUB or XSUB device.

debug_addr
Return debug socket's address.

run()
Serve XPub-XSub Sockets.

Relays Publisher Socket data to Subscribers, and allows subscribers to sub to that data. Offers the benefit of having a single static address to connect to a cluster.

Returns `None`

running
Check if the thread is still alive and running.

stop(*timeout=None*)
Stop the thread.

Parameters **timeout** – timeout in seconds to wait for join

Data structs for use within the hermes ecosystem.

class `hermes.structs.`**Envelope**(*topic_tree*, *origin*, *data*, *ts=None*)
Transport Object for data being sent between hermes components via ZMQ.

It is encouraged to use `hermes.Message` as data for more complex data objects, but all JSON-serializable built-in data types are supported.

They track topic and origin of the data they transport, as well as the timestamp it was last updated at. Updates occur automatically whenever `hermes.Envelope.serialize()` is called. This timestamp can be used to detect Slow-Subscriber-Syndrome by `hermes.Receiver` and to initiate the suicidal snail pattern.

convert_to_frames(*encoding=None*)
Encode the `hermes.Envelope` attributes as a list of json-serialized strings.

Parameters **encoding** – the encoding to us for `str.encode()`, default UTF-8

Returns list of `bytes`

expected_message_type
alias of *Message*

classmethod load_from_frames(*frames*, *encoding=None*)
Load json to a new `hermes.Envelope` instance.

Automatically converts to string if the passed object is a `bytes.encode()` object.

Parameters

- **frames** – Frames, as received by `zmq.socket.recv_multipart()`
- **encoding** – The encoding to use for `bytes.encode()`; default UTF-8

Returns `hermes.Envelope` instance

update_ts()
Update the `hermes.Envelope` timestamp.

class hermes.structs.**Message**(*ts=None*)
Basic Struct class for data sent via an `hermes.Envelope`.

Provides basic and dynamic load and dump functions to easily load data to and from it.

If you have complex data types, consider extending this class, as it requires less overhead than, for example, dictionaries, by using __slots__.

The class's timestamp attribute (ts) denotes the time of which the data was received.

classmethod load(*data*)
Load data into a new data struct.

Parameters **data** – iterable, as transported by `hermes.Envelope`

Returns `hermes.Message`

serialize(*encoding='UTF-8'*)
Serialize this data struct to `bytes`.

Parameters **encoding** – Encoding to use in str.encode()

Returns data of this struct as `bytes`

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

h

Index