zmq-plugin Documentation

Release 0.1.post43.dev225771696

Christian Fobel

Jul 20, 2017

Contents

Contents:

CHAPTER 1

Project Modules

zmq_plugin Package

`hub` Module

class `zmq_plugin.hub.`**`Hub`**(*query_uri*, *name='hub'*)

Bases: `object`

Central hub to connect a network of plugin instances.

Note: Thread-safety

All socket configuration, registration, etc. is performed *only* when the *reset* method is called explicitly. Thus, all sockets are created in the thread that calls the *reset* method.

By creating sockets in the thread the calls *reset*, it is straightforward to, for example, run a *Plugin* in a separate process or thread.

Parameters

- **`query_uri`** (`str`) – The URI address of the **hub** query socket.

  Plugins connect to the query socket to register and query information about other sockets.

- **`name`** (`str`) – Unique name across all plugins.

`command_socket`

zmq.Socket – Plugins send command requests to the command socket.

`command_uri`

str – The URI address of the command socket.

Command URI is determined at time of binding (bound to random port).

host
str – Host name or IP address.

name
str – Hub name (**MUST** be unique across all plugins).

publish_socket
zmq.Socket – Hub broadcasts messages to plugins over the publish socket.

publish_uri
str – The URI address of the publish socket.

Publish URI is determined at time of binding (bound to random port).

query_socket
zmq.Socket – Plugins connect to the query socket to register and query information about other sockets.

query_uri
str – The URI address of the query socket.

registry
OrderedDict – Registry of connected plugins.

transport
str – Transport (e.g., "tcp", "inproc").

Attributes

`logger`	*logging.Logger* – Class-specific logger.

Methods

`on_command_recv`(msg_frames)	Process multi-part message from *command* socket.
`on_execute__ping`(request)	Send "pong" response to requesting plugin.
`on_execute__register`(request)	Add name of client to registry and respond with registry contents.
`on_query_recv`(msg_frames)	Process multi-part message from query socket.
`query_send`(message)	Send message to query socket.
`reset`()	Reset the plugin state.
`reset_command_socket`()	Create and configure *command* socket (existing socket is destroyed if it exists).
`reset_publish_socket`()	Create and configure *publish* socket (existing socket is destroyed if it exists).
`reset_query_socket`()	Create and configure *query* socket (existing socket is destroyed if it exists).

logger
logging.Logger – Class-specific logger.

Logger configured with a name in the following form:

```
<module_name>.<class_name>.<method_name>->"<self.name>"
```

on_command_recv(*msg_frames*)
Process multi-part message from *command* socket.

Only `execute_request` and `execute_reply` messages are expected.

Messages are expected under the following scenarios:

1. A plugin submitting an execution request or reply to another plugin.
2. A plugin submitting an execution request or reply to the **hub**.

In case 1, the `source` and `target` in the message header **MUST** both be present in the local registry (i.e., *`registry`*).

In case 2, the `source` in the message header **MUST** be present in the local registry (i.e., *`registry`*) and the `target` **MUST** be equal to *`name`*.

This method may, for example, be called asynchronously as a callback in run loop through a `zmq.eventloop.ZMQStream(...)` configuration.

See here for more details.

Parameters **`msg_frames`** (*`list`*) – Multi-part ZeroMQ message.

`on_execute__ping`(*request*)

Send "pong" response to requesting plugin.

Useful to, for example, test connection from plugins.

Returns "pong"

Return type str

`on_execute__register`(*request*)

Add name of client to registry and respond with registry contents.

Returns Registry contents.

Return type OrderedDict

`on_query_recv`(*msg_frames*)

Process multi-part message from query socket.

This method may, for example, be called asynchronously as a callback in run loop through a `zmq.eventloop.ZMQStream` configuration.

See here for more details.

Parameters **`msg_frames`** (*`list`*) – Multi-part ZeroMQ message.

`query_send`(*message*)

Send message to query socket.

Parameters **`message`** (*`str`*) – Encoded json reply.

`reset`()

Reset the plugin state.

This includes:

- Resetting the execute reply identifier counter.
- Resetting the `publish`, `query`, and `command` sockets.

`reset_command_socket`()

Create and configure *command* socket (existing socket is destroyed if it exists).

`reset_publish_socket`()

Create and configure *publish* socket (existing socket is destroyed if it exists).

reset_query_socket()
Create and configure *query* socket (existing socket is destroyed if it exists).

`plugin` Module

class zmq_plugin.plugin.**Plugin**(*name*, *query_uri*, *subscribe_options=None*)
Bases: *zmq_plugin.plugin.PluginBase*

Attributes

logger	*Return logger configured with a name in the following form* – <module_name>.<class_name>.<method_name>->"<self.name>"

Methods

close()	Close all sockets.
execute(target_name, command[, timeout_s, ...])	Send request to execute the specified command to the identified target and return decoded result object.
execute_async(target_name, command[, ...])	Send request to execute the specified command to the identified target.
on_command_recv(frames)	Process multi-part message from command socket.
on_execute__ping(request)	
on_subscribe_recv(msg_frames)	Process multi-part message from subscribe socket.
query(request, **kwargs)	Send request message to **hub**, receive response, and return decoded reply message.
register()	Register as a plugin with the central **hub**.
reset()	Reset the plugin state.
reset_command_socket()	Create and configure `command_socket` socket (existing socket is destroyed if it exists).
reset_query_socket()	Create and configure `query_socket` socket (existing socket is destroyed if it exists).
reset_subscribe_socket()	Create and configure `subscribe_socket` socket (existing socket is destroyed if it exists).
send_command(request)	Send command message request through **hub**.

on_execute__ping(*request*)

class zmq_plugin.plugin.**PluginBase**(*name*, *query_uri*, *subscribe_options=None*)
Bases: `object`

Plugin which can be connected to a network of other plugin instances through a central **hub** (i.e., *zmq_plugin.hub.Hub*).

Note: Thread-safety

All socket configuration, registration, etc. is performed *only* when the *reset* method is called explicitly. Thus, all sockets are created in the thread that calls the *reset* method.

By creating sockets in the thread the calls *reset*, it is straightforward to, for example, run a *Plugin* in a separate process or thread.

Parameters

- **`name`** (*`str`*) – Unique name across all plugins.
- **`query_uri`** (*`str`*) – The URI address of the **hub** query socket.
- **`subscribe_options`** (*`dict, optional`*) – See *`subscribe_options`*.

`callbacks`
OrderedDict – Registry of functions to call upon receiving `execute_reply` messages, keyed by the `session` field of the `execute_request`/`execute_reply` header.

`command_socket`
zmq.Socket – Used to send command requests to the **hub** command socket.

`execute_reply_id`
itertools.count – Reply message count iterator.

Increments by one each time a reply message is sent.

`host`
str – Host name or IP address.

`hub_name`
str – Name of hub.

`query_socket`
zmq.Socket – Connects to the **hub** query socket to register and query information about other sockets on the **hub**.

`query_uri`
str – The URI address of the query socket.

`subscribe_options`
dict – Each `(key, value)` item in dictionary is applied to *`subscribe_socket`* using the `setsockopt()` method.

This is useful, for instance, to set the subscription filter.

`subscribe_socket`
zmq.Socket – Hub broadcasts messages to all plugins over the publish socket.

`transport`
str – Transport (e.g., “tcp”, “inproc”).

Attributes

`logger`	*Return logger configured with a name in the following form* – <module_name>.<class_name>.<method_name>->”<self.name>”

Methods

close()	Close all sockets.
execute(target_name, command[, timeout_s, ...])	Send request to execute the specified command to the identified target and return decoded result object.
execute_async(target_name, command[, ...])	Send request to execute the specified command to the identified target.
on_command_recv(frames)	Process multi-part message from command socket.
on_subscribe_recv(msg_frames)	Process multi-part message from subscribe socket.
query(request, **kwargs)	Send request message to **hub**, receive response, and return decoded reply message.
register()	Register as a plugin with the central **hub**.
reset()	Reset the plugin state.
reset_command_socket()	Create and configure *command_socket* socket (existing socket is destroyed if it exists).
reset_query_socket()	Create and configure *query_socket* socket (existing socket is destroyed if it exists).
reset_subscribe_socket()	Create and configure *subscribe_socket* socket (existing socket is destroyed if it exists).
send_command(request)	Send command message request through **hub**.

close()

Close all sockets.

execute(*target_name*, *command*, *timeout_s=None*, *wait_func=None*, *silent=False*, *extra_kwargs=None*, ***kwargs*)

Send request to execute the specified command to the identified target and return decoded result object.

N.B., this method blocking, i.e., it waits for a response. See *execute_async* method for non-blocking variant with *callback* argument.

Parameters

- **target_name** (*str*) – Name (i.e., ZeroMQ identity) of the target.
- **command** (*str*) – Name of command to execute.
- **timeout_s** (*float, optional*) – If `timeout_s` is set, `IOError` is raised if response is not received within `timeout_s` seconds.
- **wait_func** (*function, optional*) – If `wait_func` is set, the `wait_func` function is called repeatedly until response is received.

  This is useful to prevent *execute()* from completely blocking thread execution.
- **silent** (*bool, optional*) – A boolean flag which, if `True`, signals the plugin to execute this code as quietly as possible.

  If `silent` is set to `True`, reply will *not* broadcast output on the IOPUB channel.
- **extra_kwargs** (*dict*) – Extra keyword arguments to be passed to command.

  Useful to, for example, include keyword arguments whose name conflict with arguments of *execute_async()*/*execute()*.
- ****kwargs** (*dict*) – Keyword arguments for command.

Returns Result from remotely executed command.

Return type object

See also:

`execute_async()`

`execute_async`(*target_name*, *command*, *callback=None*, *silent=False*, *extra_kwargs=None*, ***kwargs*)

Send request to execute the specified command to the identified target.

N.B., this method is non-blocking, i.e., it does not wait for a response. For a blocking wrapper around this method, see *execute* method below.

Parameters

- **`target_name`** (`str`) – Name (i.e., ZeroMQ identity) of the target.
- **`command`** (`str`) – Name of command to execute.
- **`callback`** (`function, optional`) – Function to call on received response.

  Callback signature is `callback_func(reply)`, where `reply` is an `execute_reply` message.

  Callback is added to `callbacks`, keyed by session identifier of request.
- **`silent`** (`bool, optional`) – A boolean flag which, if `True`, signals the plugin to execute this code as quietly as possible.

  If `silent` is set to `True`, reply will *not* broadcast output on the IOPUB channel.
- **`extra_kwargs`** (`dict`) – Extra keyword arguments to be passed to command.

  Useful to, for example, include keyword arguments whose name conflict with arguments of `execute_async()`/`execute()`.
- **`**kwargs`** (`dict`) – Keyword arguments for command.

Returns Session identifier for request.

Return type str

See also:

`execute()`

`logger`

Return logger configured with a name in the following form – <module_name>.<class_name>.<method_name>->"<self.name>"

`on_command_recv`(*frames*)

Process multi-part message from command socket.

This method may, for example, be called asynchronously as a callback in run loop through a `zmq.eventloop.ZMQStream` configuration.

See here for more details.

Parameters **`msg_frames`** (`list`) – Multi-part ZeroMQ message.

`on_subscribe_recv`(*msg_frames*)

Process multi-part message from subscribe socket.

This method may, for example, be called asynchronously as a callback in run loop through a `zmq.eventloop.ZMQStream` configuration.

See here for more details.

Parameters **`msg_frames`** (`list`) – Multi-part ZeroMQ message.

query (*request*, ***kwargs*)
Send request message to **hub**, receive response, and return decoded reply message.

Parameters **dict** (*request*) – <...>_request message.

register ()
Register as a plugin with the central **hub**.

Registration also updates the local plugin registry, which contains the name of all plugins registered with the **hub** at the time of registration.

Note that this method is safe to execute multiple times. This provides a mechanism to refresh the local plugin registry.

reset ()
Reset the plugin state.

This includes:

- Resetting the execute reply identifier counter.
- Resetting the `command_socket`, `query_socket`, and `subscribe_socket` sockets.
- Registering with the central **hub**.

reset_command_socket ()
Create and configure `command_socket` socket (existing socket is destroyed if it exists).

reset_query_socket ()
Create and configure `query_socket` socket (existing socket is destroyed if it exists).

reset_subscribe_socket ()
Create and configure `subscribe_socket` socket (existing socket is destroyed if it exists).

send_command (*request*)
Send command message request through **hub**.

Parameters **request** (*dict*) – Command request message.

schema Module

zmq_plugin.schema.**MESSAGE_SCHEMA**
dict – ZeroMQ Plugin message format as json-schema (inspired by IPython messaging format).

See here for information on content transfer encodings.

class zmq_plugin.schema.**PandasJsonEncoder** (*skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *encoding='utf-8'*, *default=None*)
Bases: `json.encoder.JSONEncoder`

Example

```
>>> data = pd.Series(range(10))
>>> df_data = pd.DataFrame([data.copy() for i in xrange(5)])
>>> combined_dump = json.dumps([df_data, data], cls=PandasJsonEncoder)
>>> loaded = json.loads(combined_dump, object_hook=pandas_object_hook)
>>> assert(loaded[0].equals(df_data))
>>> assert(loaded[1].equals(data))
```

See also:

`pandas_object_hook()`

Methods

`default`(o)	
`encode`(o)	Return a JSON string representation of a Python data structure.
`iterencode`(o[, _one_shot])	Encode the given object and yield each string representation as available.

`default`(*o*)

`zmq_plugin.schema.`**`decode_content_data`**(*message*)
Validate message and decode data from content according to mime-type.

Parameters **`message`** (`dict`) – One of the message types defined in `MESSAGE_SCHEMA`.

Returns Return deserialized object from `content['data']` field of message.

Return type object

Raises `RuntimeError` – If `content['error']` field is set.

`zmq_plugin.schema.`**`encode_content_data`**(*data*, *mime_type='application/python-pickle'*, *transfer_encoding='BASE64'*)

`zmq_plugin.schema.`**`get_connect_reply`**(*request*, *content*)
Construct a `connect_reply` message.

Parameters

- **`request`** (`dict`) – The `connect_request` message corresponding to the reply.
- **`content`** (`dict`) – The content of the reply.

Returns A `connect_reply` message.

Return type dict

`zmq_plugin.schema.`**`get_connect_request`**(*source*, *target*)
Construct a `connect_request` message.

Parameters

- **`source`** (`str`) – Source name/ZMQ identifier.
- **`target`** (`str`) – Target name/ZMQ identifier.

Returns A `connect_request` message.

Return type dict

`zmq_plugin.schema.`**`get_execute_reply`**(*request*, *execution_count*, *status='ok'*, *error=None*, *data=None*, *mime_type='application/python-pickle'*, *transfer_encoding='BASE64'*, *silent=None*)
Construct an *execute_reply* message.

Parameters

- **`request`** (`dict`) – The *execute_request* message corresponding to the reply.

- **execution_count** (*int*) – The number execution requests processed by plugin, including the request corresponding to the reply.
- **status** (*str, optional*) – One of *'ok'*, *'error'*, *'abort'*.
- **error** (*exception, optional*) – Exception encountered during processing of request (if applicable).
- **data** (*dict, optional*) – Result data.
- **mime_type** (*dict, optional*) – Mime-type of requested data serialization format.

  By default, data is serialized using **:module:'pickle'**.
- **transfer_encoding** (*str, optional*) – If `BASE64`, encode binary payload as base 64 string.
- **silent** (*bool, optional*) – A boolean flag which, if `True`, signals the plugin to execute this code as quietly as possible. If `silent=True`, reply will *not* broadcast output on the IOPUB channel. If `None`, silent setting from request will be used.

Returns An `execute_reply` message.

Return type dict

`zmq_plugin.schema.`**`get_execute_request`**(*source*, *target*, *command*, *data=None*, *mime_type='application/python-pickle'*, *transfer_encoding='BASE64'*, *silent=False*, *stop_on_error=False*)

Construct an `execute_request` message.

Parameters

- **source** (*str*) – Source name/ZMQ identifier.
- **target** (*str*) – Target name/ZMQ identifier.
- **command** (*str*) – Name of command to execute.
- **data** (*dict, optional*) – Keyword arguments to command.
- **mime_type** (*dict, optional*) – Mime-type of requested data serialization format.

  By default, data is serialized using **:module:'pickle'**.
- **silent** (*bool, optional*) – A boolean flag which, if `True`, signals the plugin to execute this code as quietly as possible. If `silent=True`, reply will *not* broadcast output on the IOPUB channel.
- **stop_on_error** (*bool, optional*) – A boolean flag, which, if `False`, does not abort the execution queue, if an exception is encountered. This allows the queued execution of multiple `execute_request` messages, even if they generate exceptions.

Returns An `execute_request` message.

Return type dict

`zmq_plugin.schema.`**`get_header`**(*source*, *target*, *message_type*, *session=None*)

Construct message header.

Parameters

- **source** (*str*) – Source name/ZMQ identifier.
- **target** (*str*) – Target name/ZMQ identifier.

- **message_type** (*str*) – Type of message, one of 'connect_request', 'connect_reply', 'execute_request', 'execute_reply'.
- **session** (*str, optional*) – Unique session identifier (automatically created if not provided).

Returns Message header including unique message identifier and timestamp.

Return type dict

zmq_plugin.schema.**get_schema**(*definition*)

zmq_plugin.schema.**mime_type**(*mime_type_override=None*)

Decorator to specify mime type of return type.

The mime_type attribute of the function is set accordingly.

zmq_plugin.schema.**pandas_object_hook**(*obj*)

Example

```
>>> data = pd.Series(range(10))
>>> df_data = pd.DataFrame([data.copy() for i in xrange(5)])
>>> combined_dump = json.dumps([df_data, data], cls=PandasJsonEncoder)
>>> loaded = json.loads(combined_dump, object_hook=pandas_object_hook)
>>> assert(loaded[0].equals(df_data))
>>> assert(loaded[1].equals(data))
```

See also:

PandasJsonEncoder

zmq_plugin.schema.**validate**(*message*)

Validate message against message types defined in *MESSAGE_SCHEMA*.

Parameters message (*dict*) – One of the message types defined in *MESSAGE_SCHEMA*.

Returns Message. A jsonschema.ValidationError is raised if validation fails.

Return type dict

Subpackages

bin Package

bin Package

zmq_plugin.bin.**verify_tornado**()

Verify that *tornado* package is installed.

The *tornado* module is not included in the *install_requires* list because it is not required for basic usage of the package without a tornado run-loop.

hub Module

monitor Module

`plugin` Module

examples Package

`demo` Module

`hub` Module

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index