hop-client Documentation

SCiMMA

Jul 29, 2020

CONTENTS

`hop-client` is a pub-sub client library for Multimessenger Astrophysics.

CHAPTER

ONE

USER'S GUIDE

1.1 Installation

You can install hop-client either via pip, conda, or from source.

To install with pip:

```
pip install -U hop-client
```

To install with conda, you must use the channel from the SCiMMA Anaconda organization:

```
conda install --channel scimma hop-client
```

To install from source:

```
tar -xzf hop-client-x.y.z.tar.gz
cd hop-client-x.y.z
python setup.py install
```

1.2 Quickstart

1.2.1 Using the CLI

By default, authentication is enabled, reading in configuration settings from `config.toml`. The path to this configuration can be found by running `hop auth locate`. One can initialize this configuration with default settings by running `hop auth setup`. To disable authentication in the CLI client, one can run `--no-auth`.

Publish messages

```
hop publish kafka://hostname:port/gcn -f CIRCULAR example.gcn3
```

Example messages are provided in `tests/data` including:

- A GCN circular (`example.gcn3`)
- A VOEvent (`example_voevent.xml`)

Consume messages

```
hop subscribe kafka://hostname:port/gcn -s EARLIEST
```

This will read messages from the gcn topic from the earliest offset and read messages until an end of stream (EOS) is received.

1.2.2 Using the Python API

Publish messages

Using the python API, we can publish various types of messages, including structured messages such as GCN Circulars and VOEvents:

```
from hop import stream
from hop.models import GCNCircular

# read in a GCN circular
with open("path/to/circular.gcn3", "r") as f:
    circular = GCNCircular.load(f)

with stream.open("kafka://hostname:port/topic", "w") as s:
    s.write(circular)
```

In addition, we can also publish unstructured messages as long as they are JSON serializable:

```
from hop import stream

with stream.open("kafka://hostname:port/topic", "w") as s:
    s.write({"my": "message"})
```

By default, authentication is enabled for the Hop broker, reading in configuration settings from `config.toml`. In order to modify various authentication options, one can configure a `Stream` instance and pass in an `Auth` instance with credentials:

```
from hop import Stream
from hop.auth import Auth

auth = Auth("my-username", "my-password")
stream = Stream(auth=auth)

with stream.open("kafka://hostname:port/topic", "w") as s:
    s.write({"my": "message"})
```

To explicitly disable authentication, one can set `auth` to `False`.

Consume messages

One can consume messages through the python API as follows:

```
from hop import stream

with stream.open("kafka://hostname:port/topic", "r") as s:
    for message in s:
        print(message)
```

This will listen to the Hop broker, listening to new messages and printing them to stdout as they arrive until there are no more messages in the stream. By default, this will only process new messages since the connection was opened. The `start_at` option lets you control where in the stream you can start listening from. For example, if you'd like to listen to all messages stored in a topic, you can do:

```
from hop import stream
from hop.io import StartPosition

stream = Stream(start_at=StartPosition.EARLIEST)

with stream.open("kafka://hostname:port/topic", "r") as s:
    for message in s:
        print(message)
```

1.3 Streaming

1.3.1 The Stream Object

The `Stream` object allows a user to connect to a Kafka broker and read in a variety of alerts, such as GCN circulars. It also allows one to specify default settings used across all streams opened from the Stream instance.

Let's open up a stream and show the `Stream` object in action:

```
from hop import Stream

stream = Stream(persist=True)
with stream.open("kafka://hostname:port/topic", "r") as s:
```

(continues on next page)

(continued from previous page)

```
    for message in s:
        print(message)
```

The `persist` option allows one to listen to messages forever and keeps the connection open after an end of stream (EOS) is received. This is to allow long-lived connections where one may set up a service to process incoming GCNs, for example.

A common use case is to not specify any defaults ahead of time, so a shorthand is provided for using one:

```
from hop import stream

with stream.open("kafka://hostname:port/topic", "r") as s:
    for message in s:
        print(message)
```

A complete list of configurable options in `Stream` are:

- `auth`: A *bool* or `auth.Auth` instance to provide authentication
- `start_at`: The message offset to start at, by passing in an `io.StartPosition`
- `persist`: Whether to keep a long-live connection to the client beyond EOS

In addition, `stream.open` provides an option to retrieve Kafka message metadata as well as the message itself, such as the Kafka topic, key, timestamp and offset. This may be useful in the case of listening to multiple topics at once:

```
from hop import stream

with stream.open("kafka://hostname:port/topic1,topic2", "r", metadata=True) as s:
    for message, metadata in s:
        print(message, metadata.topic)
```

1.4 Authentication

1.4.1 Configuration

Since connections to the Hopskotch server require authentication, there are several utilities exposed to generate and provide credentials for both the CLI and python API. `hop auth` provides command line options to generate a configuration file with proper credentials needed to authenticate.

In order to generate a configuration file, one can run `hop auth setup`, which prompts the user for a username and password to connect to Hopskotch to publish or subscribe to messages.

The default location for the configuration file can be found with `hop auth locate`, which points by default to `${HOME}/.config/hop/config.toml`, but can be configured by setting the `XDG_CONFIG_PATH` variable.

1.4.2 Using Credentials

Authentication is enabled by default and will read credentials from the path resolved by `hop auth locate`.

For the python API, one can modify various authentication options by passing in an `Auth` instance with credentials to a `Stream` instance. This provides a similar interface to authenticating as with the requests library.

```
from hop import Stream
from hop.auth import Auth

auth = Auth("my-username", "my-password")
stream = Stream(auth=auth)

with stream.open("kafka://hostname:port/topic", "w") as s:
    s.write({"my": "message"})
```

In order to disable authentication in the command line interface, you can pass `--no-auth` for various CLI commands. For the python API, you can set `auth` to `False`.

1.5 Message Formats

The hop client provides a few in-memory representations for common message types for easy access to various message properties, as well as loading messages from their serialized forms or from disk. These message formats, or models, can be sent directly to an open `Stream` to provide seamless serialization of messages through Hopskotch.

1.5.1 Structured Messages

Currently, the structured messages available are `VOEvent` and `GCNCircular`. To give an example of its usage:

```
from hop import Stream
from hop.auth import load_auth
from hop.models import VOEvent

xml_path = "/path/to/voevent.xml"
voevent = VOEvent.load_file(xml_path)

stream = Stream(auth=load_auth())
with stream.open("kafka://hostname:port/topic", "w") as s:
    s.write(voevent)
```

1.5.2 Unstructured Messages

Unstructured messages can be sent directly to an open `Stream` instance and will be serialized appropriately. Any python object that can be JSON serialized can be sent. Examples include a dictionary, a byte64 encoded string, and a list.

1.6 Commands

hop-client provides a command line interface for various tasks:

- `hop auth`: Authentication utilities
- `hop publish`: Publish messages such as GCN circulars and notices
- `hop subscribe`: Listen to messages such as GCN circulars and notices
- `hop version`: Show version dependencies of `hop-client`

1.6.1 `hop auth`

This command allows a user to handle auth-based configuration.

```
usage: hop auth [-h] <command> ...

Authentication utilities.

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

Commands:
  <command>
    locate    display authentication config path
    setup     set up authentication config with defaults
```

1.6.2 `hop publish`

This command allows a user to publish various structured and unstructured messages, including:

- RFC 822 formatted GCN circular
- An XML formatted GCN/VOEvent notice
- Unstructured messages such as byte-encoded or JSON-serializable data.

Structured messages such as GCN circulars and VOEvents are published as JSON-formatted text.

```
usage: hop publish [-h] [--no-auth] [-f {CIRCULAR,VOEVENT,BLOB}]
                   URL MESSAGE [MESSAGE ...]

Parse and publish messages.

positional arguments:
  URL                   Sets the URL (kafka://host[:port]/topic) to publish
                        messages to.
  MESSAGE               One or more messages to publish.

optional arguments:
  -h, --help            show this help message and exit
  --no-auth             If set, disable authentication.
  -f {CIRCULAR,VOEVENT,BLOB}, --format {CIRCULAR,VOEVENT,BLOB}
                        Specify the message format. Defaults to BLOB for an
                        unstructured message.
```

1.6.3 hop subscribe

This command allows a user to subscribe to messages and print them to stdout.

```
usage: hop subscribe [-h] [--no-auth] [-s {EARLIEST,LATEST,PRODUCER}] [-p]
                     [-j]
                     URL

Receive and parse messages.

positional arguments:
  URL                   Sets the URL (kafka://host[:port]/topic) to publish
                        messages to.

optional arguments:
  -h, --help            show this help message and exit
  --no-auth             If set, disable authentication.
  -s {EARLIEST,LATEST,PRODUCER}, --start-at {EARLIEST,LATEST,PRODUCER}
                        Set the message offset offset to start at. Default:
                        LATEST.
  -p, --persist         If set, persist or listen to messages indefinitely.
                        Otherwise, will stop listening when EOS is received.
  -j, --json            Request message output as raw json
```

1.6.4 hop version

This command prints all the versions of the dependencies

```
usage: hop version [-h]

List all the dependencies' versions.

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

CHAPTER

TWO

API REFERENCE

2.1 hop-client API

2.1.1 hop.auth

class hop.auth.**Auth**(*user*, *password*, *ssl=True*, *method=<SASLMethod.SCRAM_SHA_512: 3>*, ***kwargs*)

Attach SASL-based authentication to a client.

Returns client-based auth options when called.

user [*str*] Username to authenticate with.

password [*str*] Password to authenticate with.

ssl [*bool*, optional] Whether to enable SSL (enabled by default).

method [*SASLMethod*, optional] The SASL method to authenticate, default = SASLMethod.SCRAM_SHA_512. See valid SASL methods in SASLMethod.

ssl_ca_location [*str*, optional] If using SSL via a self-signed cert, a path/location to the certificate.

hop.auth.**get_auth_path**()

Determines the default location for auth configuration.

Returns: The path to the authentication configuration file.

hop.auth.**load_auth**(*authfile='/home/docs/.config/hop/config.toml'*)

Configures an Auth instance given a configuration file.

Args:

authfile: Path to a configuration file, loading from the default location if not given.

Returns: A configured Auth instance.

Raises: KeyError: An error occurred parsing the configuration file.

2.1.2 hop.cli

hop.cli.**add_client_opts**(*parser*)
Add general client options to an argument parser.

Args: parser: An ArgumentParser instance to add client options to.

2.1.3 hop.io

class hop.io.**Deserializer**(*value*)
An enumeration.

classmethod deserialize(*message*)
Deserialize a stream message and instantiate a model.

Args: message: A serialized message.

Returns: A data container corresponding to the format in the serialized message.

Raises:

ValueError: If the message is incorrectly formatted or if the message format is not recognized.

class hop.io.**Stream**(*auth=True*, *start_at=<ConsumerStartPosition.LATEST: 2>*, *persist=False*)
Defines an event stream.

Sets up defaults used within the client so that when a stream connection is opened, it will use defaults specified here.

Args:

auth: A *bool* or *Auth* instance. Defaults to loading from *auth.load_auth()* if set to True. To disable authentication, set to False.

start_at: The message offset to start at in read mode. Defaults to LATEST. persist: Whether to listen to new messages forever or stop

when EOS is received in read mode. Defaults to False.

open(*url*, *mode='r'*, *metadata=False*)
Opens a connection to an event stream.

Args: url: Sets the broker URL to connect to.

Kwargs: mode: Read ('r') or write ('w') from the stream. metadata: Whether to receive message metadata along

with payload (read only).

Returns: An open connection to the client, either an adc Producer instance in write mode or an adc Consumer instance in read mode.

Raises:

ValueError: If the mode is not set to read/write or if more than one topic is specified in write mode.

2.1.4 hop.publish

2.1.5 hop.subscribe

hop.subscribe.**print_message**(*message_model*, *json_dump=False*)
Print the content of a message.

Args: message_model: dataclass model object for a message json_dump: boolean indicating whether to print as raw json

Returns: None

2.1.6 hop.models

class hop.models.**GCNCircular**(*header: dict*, *body: str*)
Defines a GCN Circular structure.

The parsed GCN circular is formatted as a dictionary with the following schema:

{'headers': {'title': ..., 'number': ..., ...}, 'body': ...}

asdict()
Represents the GCN Circular as a dictionary.

Returns: The dictionary representation of the Circular.

classmethod load(*email_input*)
Create a new GCNCircular from an RFC 822 formatted circular.

Args: email_input: A file object or string.

Returns: The GCNCircular.

classmethod load_file(*filename*)
Create a new GCNCircular from an RFC 822 formatted circular file.

Args: filename: The GCN filename.

Returns: The GCNCircular.

serialize()
Wrap the message with its format and content.

Returns: A dictionary with "format" and "content" key-value pairs.

class hop.models.**MessageBlob**(*content: str*, *missing_schema: bool = False*)
Defines an unformatted message structure.

This is included to mirror the implementation of structured formats.

asdict()
Represents the message as a dictionary.

Returns: The dictionary representation of the message.

classmethod load(*blob_input*)
Create a blob message from input text.

Args: blob_input: The unstructured message text or file object.

Returns: The Blob.

classmethod load_file(*filename*)
Create a blob message from an input file.

Args: filename: A filename.

Returns: The Blob.

serialize()
Wrap the message with its format and content.

Returns: A dictionary with "format" and "content" key-value pairs

class hop.models.**VOEvent**(*ivorn: str*, *role: str = 'observation'*, *version: str = '2.0'*, *Who: dict = <factory>*, *What: dict = <factory>*, *WhereWhen: dict = <factory>*, *How: dict = <factory>*, *Why: dict = <factory>*, *Citations: dict = <factory>*, *Description: dict = <factory>*, *Reference: dict = <factory>*)
Defines a VOEvent 2.0 structure.

Implements the schema defined by: http://www.ivoa.net/Documents/VOEvent/20110711/

asdict()
Represents the VOEvent as a dictionary.

Returns: A dictionary representation of the VOEvent.

classmethod load(*xml_input*)
Create a new VOEvent from an XML-formatted VOEvent.

Args: xml_input: A file object, string, or generator.

Returns: The VOEvent.

classmethod load_file(*filename*)
Create a new VOEvent from an XML-formatted VOEvent file.

Args: filename: Name of the VOEvent file.

Returns: The VOEvent.

serialize()
Wrap the message with its format and content.

Returns: A dictionary with "format" and "content" key-value pairs.

2.1.7 hop.version

hop.version.**get_packages**()
Returns the package dependencies used within hop-client.

hop.version.**print_packages_versions**()
Print versions for the passed packages.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

h

INDEX