pyhaystack Documentation

Release 3.0.0

Christian Tremblay, P.Eng.

Sep 03, 2020

Contents

CHAPTER 1

pyhaystack

1.1 What is this ?

Pyhaystack is a module that allow python programs to connect to a haystack server using semantic data model for buildings (project-haystack).

Browse a campus, building, floor… find VAV boxes, AHU units, etc. Then extract history data from them and get the results ready for analysis using pandas or your own database implementation.

1.2 Which clients are implemented ?

Actually, connection can be established with :

- Niagara4 by Tridium
- NiagaraAX by Tridium
- Widesky by VRT
- Skyspark by SkyFoundry (version 2 and 3+)

Connection to Niagara AX or Niagara 4 requires the nHaystack module by J2 Innovations to be installed and properly configured on your Jace. Refer to documentation of nHaystack for details.

1.3 How do I install pyhaystack ?

```
pip install pyhaystack
```

Or you can also git clone the develop branch and use

```
python setup.py install
```

Note: Some users reported problems when installing pyhaystack using the Python version provided by their OS (Mac OS users). We recommend to try the virtual environment approach when you are unsure about the python version our modules dependencies.

1.4 Using virtual env

You can find more information on how to use **virtualenv_** but here is a short way of making it work.

```
sudo pip install virtualenv
mkdir your project folder
cd project
virtualenv venv
source venv/bin/activate
```

Note: Once you are in your virtual env DO NOT use sudo to pip install. (in fact, this is the part that made me think of permission issue as I read somewhere that we should never sudo pip install anything)

So now you are in your virtual env (it's in parenthesis in the console) and you

```
pip install requests
pip install hszinc
pip install pyhaystack
```

(note that this time you won't see any weird message when trying to install pandas and you need xcode to perform the install. . . .) You are now able to

```
import hszinc
hszinc.MODE_ZINC
from pyhaystack.client.skyspark import SkysparkHaystackSession
```

1.5 What is project-haystack ?

As stated in the web site

> "Project Haystack is an open source initiative to streamline working with data from the Internet of Things. We standardize semantic data models and web services with the goal of making it easier to unlock value from the vast quantity of data being generated by the smart devices that permeate our homes, buildings, factories, and cities. Applications include automation, control, energy, HVAC, lighting, and other environmental systems."
>
> —Project-Haystack

1.6 Actual implementation

Pyhaystack is robust and will be ready for asynchronous development.

We have chosen a state machine approach with observer pattern. See the docs for more informations.

This implementation has been mostly supported by VRT and Servisys. We are hoping that more people will join us in our effort to build a well working open-source software that will open the door of building data analysis to Python users.

1.7 Dependency

Pyhaystack highly depends on hszinc which is a special parser for zinc encoded data. Zinc was created for project-haystack as a CSV replacement.

For analysis, we also suggest using Pint to deal with units. It will bring a lot of possibilities to pyhaystack (ex. unit conversion)

CHAPTER 2

How to use it

2.1 Connecting to a haystack server

Pyhaystack has support modules for various Project Haystack implementations, as well as an extensible framework for adding support for more implementations.

The core object in Pyhaystack is the *pyhaystack.client.session.HaystackSession* object, which manages the user credentials, implements caching and provides the interface through which you or your scripts interact. This is an abstract class, for which several sub-classes exist.

There are two ways to create a session instance:

- Directly: by importing the relevant class and calling its constructor.
- Via the `pyhaystack.connect` (also known as *pyhaystack.client.get_instance()*) factory.

The former works well for ad-hoc usage in terminal sessions such as Jupyter Notebook, ipython and the plain Python shell. The latter is recommended for scripts that instantiate a session from a configuration file.

2.1.1 Usage hints

Logging messages in Jupyter Notebook

For interactive users using the Jupyter Notebook, the Jupyter notebook configures its logs to only show messages with severity "warning" or higher. To display logging messages at a lower level, run the following in your session:

```
import logging
logging.getLogger().setLevel(logging.INFO)       # or DEBUG
```

Then log messages will appear in your session.

On-Demand connection

Pyhaystack uses a feature called "lazy evaluation" to handle the actual log-in session, and so will remain dormant after the session instance is connected until a request is made.

If, when making a request, pyhaystack detects that it has been disconnected, it will attempt to re-connect automatically.

Loading from a file

The `connect` approach lends itself well to storing the connection details in a plaintext file using either JSON or YAML format. e.g. given the file *my-haystack-server.json*:

```
{
    "implementation": "skyspark",
    "uri": "http://ip:port",
    "username": "user",
    "password": "password",
    "project": "my_project",
    "pint": true
}
```

This can be instantiated like this:

```
import json
import pyhaystack
session = pyhaystack.connect(
    **json.load(
        open("my-haystack-server.json","r")
    )
)
```

(Similarly for YAML, `import yaml` and use `yaml.safe_load`.)

2.1.2 Base session options

pyhaystack.client.session.HaystackSession has the following base arguments. All subclasses should pass these values though as keyword arguments, so all should be usable.

- `uri`: This is the base URI used to access the Project Haystack server.
- `grid_format`: This selects the grid serialisation format for the Project Haystack server. Some (e.g. nHaystack/Niagara AX) only support ZINC serialisation, where as others (such as WideSky and Niagara4) work better with JSON. Most of the time, the default here will be selected appropriate for the underlying server implementation. Valid values are `zinc` and `json`.
- `http_client`: This selects which HTTP client implementation to use for the session instance. pyhaystack at the moment just has two implementations:
  - *pyhaystack.client.http.sync.SyncHttpClient*: a synchronous HTTP client based on the Python Requests library. (default)
  - `pyhaystack.client.http.dummy.DummyHttpClient`: an asynchronous dummy HTTP client used for writing unit tests.

  There are plans to implement asynchronous HTTP clients for various Python asynchronous frameworks (e.g. asyncio, TornadoWeb, Twisted, etc) in the future.

- `http_args`: This is a `dict` of keyword arguments that are passed to the constructor of the `http_client` class used to create a HTTP client instance. If `None` is given, then it is assumed that no arguments are required.
- `tagging_model`: This is used by the high-level entity interface to allow Python objects to be created using various mix-ins based on the tags attached to the entity. The default model, *`pyhaystack.client.entity.models.haystack.HaystackTaggingModel`* assumes a standard Project Haystack tagging model and should suit most users.
- `pint`: This boolean passed to the `hszinc` module to enable use of the `pint` quantity classes (providing on-the-fly unit conversion). By default, this is `False`.
- `log`: An instance of `logging.Logger` used for session logging messages. If not given, then a logger named `pyhaystack.client.${CLASS_NAME}` is created.
- `cache_expiry`: An integer or floating-point value representing the period of time before `about`/`formats`/`ops` response cache expires. The default is one hour.

HTTP client options (`http_client` and `http_args`)

The HTTP client interface is written with asynchronous HTTP clients in mind using a callback-style interface. Internally, the `HaystackSession` class does this:

```
if http_args is None:
    http_args = {}

# ... etc ...

# Create the HTTP client object
if bool(http_args.pop('debug',None)) and ('log' not in http_args):
    http_args['log'] = log.getChild('http_client')
self._client = http_client(uri=uri, **http_args)
```

With the exception of the `debug` and `log` parameters, everything else is passed through verbatim to the underlying HTTP client class. The following are the most useful arguments for `http_args`:

- `log`: If given, this is an instance of a `logging.Logger` class that will be used for log messages from the HTTP client itself.
- `debug`: A boolean flag that enables HTTP client debugging. If given, the `HaystackSession` will create a new `logging.Logger` class for the HTTP client (actually, a child logger of its own logger) for HTTP client debug messages.
- `timeout`: an integer or floating-point value that specifies the HTTP request time-out delay in seconds.
- `proxies`: A `dict` that maps the host name and protocol of a target server to the URI for a local HTTP proxy server to use for that server.
- `tls_verify`: TLS verification of server certificates. If set to `True`, then the server's HTTP certificate will be verified against CA certificates known to the Python process. If you use a custom CA, then this should be set to the full filesystem path to where that CA certificate is stored. Verification can be skipped by setting this to `False` (*not* recommended).

  When using a custom CA, the full certificate chain is required. This is usually done by converting all relevant intermediate certificates to PEM format (aka `.crt` files) and concatenated in order, that is:

  1. the certificate for the CA that signed your server's certificate
  2. the certificate for the CA that signed *that* CA's certificate
  3. ... etc

4. the certificate for the root CA.

If the root CA signed the server's certificate, then the chain is literally the root CA's certificate itself. Note that your server's certificate is *NOT* part of the bundle.

See next section on CA certificates for more details on ways to add certificates to the python process (requests).

- `tls_cert`: TLS client certificate. This is used to authenticate the Pyhaystack client to a Project Haystack server using TLS client authentication. It should either be the full path to a combined certificate/key in PEM format, or a `tuple` of the form `(tls_client_cert, tls_client_key)` where both `tls_client_cert` and `tls_client_key` are full paths to the relevant files.

The base class also supports some additional parameters that may be helpful in very specialised environments.

- `params`: a `dict` of URI query parameters to add to *all* requests.
- `headers`: a `dict` of HTTP headers to add to *all* requests.
- `cookies`: a `dict` of HTTP cookies to add to *all* requests.
- `auth`: Authentication credentials, in the form of a *`pyhaystack.client.http.auth.AuthenticationCredentials`* sub-class.

2.1.3 CA Certificates

Certifi

When connecting to a server using SSL certificates, you may face specific warning from requests. Using *http_args={tls_verify: False}* should not be a permanent solution.

Following requests recommandations, it is possibile to use certifi. Certifi is a collection of curated Root Certificates. This adds a file in your site-packages folder that will hold trusted certificates used by Requests. You can find this file using

```
import certifi
certifi.where()

# Returns something like
'/usr/local/lib/python2.7/site-packages/certifi/cacert.pem'
```

You can then add trusted certificates to this file (just append the trusted .pem text file to the end of cacert.pem).

Requests will accept all certificate from this file and consider your server secure. This will also prevent SSL warnings filling your session debug.

Manual removing of warnings

Another way of dealing with warning log messages is to use specific code to disable warnings

```
import requests
from requests.packages.urllib3.exceptions import InsecureRequestWarning
requests.packages.urllib3.disable_warnings(InsecureRequestWarning)
```

Disabling warning is done at your own risks. Use those features carefully.

SubjectAltWarning

By default, pyhaystack disable the SubjectAltWarning

```
# extract from __init__.py
import requests.packages.urllib3
from requests.packages.urllib3.exceptions import SubjectAltNameWarning
requests.packages.urllib3.disable_warnings(SubjectAltNameWarning)
```

This is motivated by this issue : urllib3_issue523

2.1.4 Connecting to specific Haystack server implementations

Niagara4 (nHaystack)

Specific arguments

In addition to those supported by the base class, the following constructor arguments are supported:

- `username`: The username to use when authenticating with nHaystack
- `password`: The password to use when authenticating with nHaystack

Direct approach

```
from pyhaystack.client.niagara import Niagara4HaystackSession
session = Niagara4HaystackSession(uri='http://ip:port',
                                  username='user',
                                  password='myPassword',
                                  pint=True)
```

`connect()` approach

```
import pyhaystack
session = pyhaystack.connect(implementation='n4',
                             uri='http://ip:port',
                             username='user',
                             password='myPassword',
                             pint=True)
```

Niagara AX (nHaystack)

Specific arguments

In addition to those supported by the base class, the following constructor arguments are supported:

- `username`: The username to use when authenticating with nHaystack
- `password`: The password to use when authenticating with nHaystack

Direct approach

```
from pyhaystack.client.niagara import NiagaraHaystackSession
session = NiagaraHaystackSession(uri='http://ip:port',
                                 username='user',
                                 password='myPassword',
                                 pint=True)
```

connect() approach

```
import pyhaystack
session = pyhaystack.connect(implementation='ax',
                         uri='http://ip:port',
                         username='user',
                         password='myPassword',
                         pint=True)
```

VRT Widesky

Specific arguments

In addition to those supported by the base class, the following constructor arguments are supported:

- `username`: The email address to use when authenticating with WideSky
- `password`: The password to use when authenticating with WideSky
- `client_id`: The OAuth2 client identity to use when authenticating with WideSky.
- `client_secret`: The OAuth2 client secret to use when authenticating with WideSky.
- `impersonate`: This is an optional parameter. If this is set and the caller

  has the required permission to act as the target user. Then all subsequent requests coming from this HTTP session will be based on the target user's READ/WRITE permissions.

Direct approach

```
from pyhaystack.client.widesky import WideskyHaystackSession
session = WideskyHaystackSession(
                uri='https://yourtenant.on.widesky.cloud/reference',
                username='user', password='my_password',
                client_id='my_id', client_secret='my_secret'
                pint=True, impersonate='user_id')
```

connect() approach

```
import pyhaystack
session = pyhaystack.connect(implementation='widesky',
        uri='https://yourtenant.on.widesky.cloud/reference',
        username='user', password='my_password',
```

(continues on next page)

(continued from previous page)

```
        client_id='my_id', client_secret='my_secret',
        pint=True, impersonate='user_id')
```

Skyspark

Specific arguments

In addition to those supported by the base class, the following constructor arguments are supported:

- `username`: The username to use when authenticating with SkySpark
- `password`: The password to use when authenticating with SkySpark
- `project`: The name of the SkySpark project instance.

Direct approach

```
from pyhaystack.client.skyspark import SkysparkHaystackSession
session = SkysparkHaystackSession(uri='http://ip:port',
                                  username='user',
                                  password='my_password',
                                  project='my_project'
                                  pint=True)
```

`connect()` approach

```
import pyhaystack
session = pyhaystack.connect(implementation='skyspark',
                                  uri='http://ip:port',
                                  username='user',
                                  password='my_password',
                                  project='my_project'
                                  pint=True)
```

Next steps

Having created a session instance, you're ready to start issuing requests, which is covered in the next section.

2.2 Operation basics

Pyhaystack's API is written around the concept of finite state machines. At the base level, a simple state machine is used to retrieve the response from one of the Project Haystack server operations.

When one of the low-level functions is called, it takes the arguments, does a little processing then returns a state machine object. Depending on the implementation of the HTTP client being used, it either executes *synchronously*, and will be returned to you in a "finalised" state, or it may execute *asynchronously* in the background.

2.2.1 Operation state machine interface

The base class for all operations is the *pyhaystack.util.state.HaystackOperation*. The following methods and properties are significant for client use:

- *pyhaystack.util.state.HaystackOperation.result*: The result of the state machine. If the result was an exception, the exception will be re-raised.
- *pyhaystack.util.state.HaystackOperation.is_done*: Returns *True* if the operation is complete, *False* otherwise.
- *pyhaystack.util.state.HaystackOperation.is_failed*: Returns *True* if the operation failed, *False* otherwise.
- *pyhaystack.util.state.HaystackOperation.wait()*: This blocks the current thread until the operation completes (or if *timeout* is specified, until that number of seconds expires).
- `pyhaystack.util.state.HaystackOperation.done_sig`: This is a `signalslot.Signal` class that is "emitted" when the operation completes.

Synchronous usage

If you are using an asynchronous HTTP client running in a separate thread, you can optionally block your local thread either temporarily or indefinitely using the *wait* method.

When using the synchronous HTTP client, the *wait* is a no-op, since the state machine is returned to the caller in a resolved state. Thus, in synchronous code, it is recommended to do the following:

```
op = session.someoperation(arg1, arg2, arg3)
op.wait()

res = op.result
# do something with res
```

This ensures that the operation is complete prior to retrieving its result.

Operation states

The individual states of an operation depends on the type of state machine being inspected, however all have a final state that can be checked by inspecting the *is_done* property. An operation is "done" if:

- the operation succeeded, in which case see the *result* property to retrieve the return value.
- the operation failed, in which case reading *result* will re-raise the exception.

Signals

Pyhaystack uses the **:py:module:'signalslot'** module to provide a signal-based interface using the observer pattern. If you've ever worked with Qt, you'll be familiar with how this works.

```
def _on_op_done(operation, **kwargs):
    assert op.is_done # <- should not fire
    # Operation is done, do something with result.

op = session.someoperation(arg1, arg2, arg3)
op.done_sig.connect(_on_op_done)
```

The signal object has a single method, `signalslot.Signal.connect()`, which takes a method or function as an argument. The passed-in method or function needs to accept keyword arguments, and will receive a single argument, *operation*, which will point to the instance of the *HaystackOperation* that emitted it.

Asynchronous Exceptions

When using signals, the behaviour is undefined if your "slot" throws an exception, thus you should catch exceptions in your slots and handle those elsewhere. One helper class you can use for doing this is *pyhaystack.util.asyncexc.AsynchronousException*:

```
from pyhaystack.asyncexc import AsynchronousException

def async_func(callback):
    try:
        res = do_something()
    except:
        # Whoopsie!
        res = AsynchronousException()

    callback(res)
```

In the callback function, you can do something like this:

```
def callback_from_async_func(result):
    try:
        if isinstance(result, AsynchronousException):
            result.reraise()
    except:
        # Handle your exception
```

If *result* is an exception, it'll be re-raised, allowing you to handle it in your code.

2.3 Your first request

You defined a session, now you want to connect to the server. The first request you could make is called "about".

About

The about op queries basic information about the server.

Request: empty grid

Response: single row grid with following columns:

- haystackVersion: Str version of REST implementation, must be "2.0"
- tz: Str of server's default timezone
- serverName: Str name of the server or project database
- serverTime: current DateTime of server's clock
- serverBootTime: DateTime when server was booted up
- productName: Str name of the server software product
- productUri: Uri of the product's web site
- productVersion: Str version of the server software product

- moduleName: module which implements Haystack server protocol if its a plug-in to the product
- moduleVersion: Str version of moduleName

—http://project-haystack.org/doc/Ops

Using a synchronous request, you would use

```
op = session.about()
op.wait()
```

The output of *op.result* would print

```
<Grid>
            Columns:
                    productName
                    moduleName
                    productVersion
                    serverTime
                    tz
                    moduleUri
                    serverName
                    productUri
                    serverBootTime
                    haystackVersion
                    moduleVersion
            Row    0: productName='Niagara AX', moduleName='nhaystack',␣
↪productVersion='3.8.41.2', serverTime=datetime.datetime(2016, 4, 28, 21, 31, 33,␣
↪882000, tzinfo=<DstTzInfo 'America/Montreal' EDT-1 day, 20:00:00 DST>), tz='Montreal
↪', moduleUri=Uri('https://bitbucket.org/jasondbriggs/nhaystack'), serverName=
↪'Servisys', productUri=Uri('http://www.tridium.com/'), serverBootTime=datetime.
↪datetime(2016, 4, 5, 15, 9, 8, 119000, tzinfo=<DstTzInfo 'America/Montreal' EDT-1␣
↪day, 20:00:00 DST>), haystackVersion='2.0', moduleVersion='1.2.5.18.1'
</Grid>
```

The return response is a `hszinc.Grid` instance.

2.3.1 Session.nav()

Session.nav() let you navigate the structure of the Project Haystack server in a manner native to that implementation of Project Haystack. The following is an example of the responses typically seen out of nHaystack.

```
op = session.nav()
op.wait()
nav = op.result
print(nav)

Out[9]:
    <Grid>
            Columns:
                    dis
                    navId
            Row    0: dis='ComponentSpace', navId='slot:/'
            Row    1: dis='HistorySpace', navId='his:/'
            Row    2: dis='Site', navId='sep:/'
    </Grid>
```

(continues on next page)

(continued from previous page)

```
op = session.nav(nav_id='his:/')
op.wait()
nav = op.result
print(nav)

Out[10]:
    <Grid>
            Columns:
                    dis
                    stationName
                    navId
            Row    0: dis='mySite', stationName='mySite', navId='his:/mySite'
    </Grid>
</Grid>
```

2.4 Higher Level Interface

The session instance also provides a higher-level interface that exposes the entities within Project Haystack as Python objects. The two functions that retrieve these entities are:

- *pyhaystack.client.session.HaystackSession.get_entity()* and
- *pyhaystack.client.session.HaystackSession.find_entity()*

Both are wrappers around the *read* operation that retrieve *pyhaystack.client.entity.entity.Entity* instances for the entities returned.

get_entity expects a list of one or more fully qualified identifiers, and will perform a *read* query listing those identifiers as given.

find_entity expects a filter expression, and performs a *read* specifying the given string as the *filter* argument. (Note: *find_entity* takes an argument named *filter_expr* to avoid a clash with the built-in function `filter()`.)

In both cases, a `dict` is returned, where the keys are the identifiers of matching entities and the values are the *Entity* instances themselves. Depending on the tags present, and the *tagging_model* passed to the session, these *Entity* instances may include other mix-in classes as well.

2.4.1 Building a filter string

As a convenience, it is possible to build up a filter string using Python objects, then take a string representation of that composite object to generate a filter string.

The classes are in **:py:module:'pyhaystack.util.filterbuilder'**. An example:

:: from pyhaystack.util import filterbuilder as fb # for brevity

op = session.find_entity(fb.Field('site') &

((fb.Field('tz') == fb.Scalar('Brisbane'))

(fb.Field('tz') == fb.Scalar('Montreal'))))

op.wait() sites_in_brisbane_and_montreal = op.result

would return all sites that are in the Brisbane or Montreal timezones.

This is helpful in scenarios where you have to construct a filter programmatically and wish to avoid the possibility of unsanitised data corrupting your filter string.

2.4.2 Querying Sites

The site is

> "A site entity models a single facility using the site tag. A good rule of thumb is to model any building with its own street address as its own site. For example a campus is better modeled with each building as a site, versus treating the entire campus as one site."
>
> —project-haystack

To browse a site you will use

```
op = session.find_entity(filter_expr='site')
op.wait()
site = op.result
```

and get a dict containing all the information provided

```
{'S.site': <@S.site: {area=BasicQuantity(0.0, 'ft²'), axSlotPath='slot:/site', axType=
↪'nhaystack:HSite', dis='site', geoAddr='2017', geoCity='thisTown', geoCountry=
↪'myCountry', geoLat=0.0, geoLon=0.0, geoPostalCode='', geoState='myState',␣
↪geoStreet='myStreet', navName='site', site, tz='New_York'}>}
```

Using the default tagging model, because the entity has a *site* tag and a *tz* tag, the resulting *Entity* class returned here will be subclasses of the following:

- *pyhaystack.client.entity.entity.Entity* (base class)
- *pyhaystack.client.entity.mixins.site.SiteMixin* (mixin class)
- pyhaystack.client.entity.mixins.site.TzMixin (mixin class)

A session have typically one site attached to it, but there could be more. As a shortcut, pyhaystack provides properties on session to get the site:

```
# Target the first site (returns a SiteTzEntity)
session.site

# Get a dict with all sites
session.sites
```

2.5 Tags

A list of tags can be found here : http://project-haystack.org/tag

For a detailed explanation of tag model, please read this: http://project-haystack.org/doc/TagModel

Pyhaystack let you find what you look for via the "find_entity" functions.

Let see how. . .

2.5.1 Finding sensors

Let say I want to find every sensors on a site which are temperature sensors used in zone.

```
op = session.find_entity(filter_expr='sensor and zone and temp')
op.wait()
```

This will find what you are looking for in the form of a "FindEntityOperation" object. To use the values of this object, you will need to retrive the results using

```
znt = op.result
```

2.5.2 Exploring points and tags

This will return a `dict` object that contains all of the Project Haystack entities that matched the given filter string. The entities are keyed by identifier. When exploring interactively, you can get a list of all the matching entities' identifiers by calling:

```
list(znt.keys())
```

To retrieve a specific entity, you give its identifier as the key:

```
my_office = znt['S.SERVISYS.Salle-Conf~e9rence.ZN~2dT']
```

Having done this, it is possible to interrogate the tags attached to this entity. These are accessed by the `tags` property, which also returns a *pyhaystack.client.entity.tags.MutableEntityTags* if your server supports making changes via the Project Haystack API (currently only WideSky), or *pyhaystack.client.entity.tags.ReadOnlyEntityTags* otherwise.

Both classes function like a `dict`.

```
my_office.tags

{air, axAnnotated,
  axSlotPath='slot:/Drivers/BacnetNetwork/MSTP1/PCV$2d2$2d008/points/ZN$2dT',
  axStatus='ok', axType='control:NumericPoint', cur, curStatus='ok',
  curVal=BasicQuantity(23.4428, '°C'),
  dis='SERVISYS Salle Conférence Salle Conférence ZN-T',
  equipRef=Ref('S.SERVISYS.Salle-Conf~e9rence', None, False),
  his, kind='Number', navName='ZN~2dT', point,
  precision=1.0, sensor, siteRef=Ref('S.SERVISYS', None, False),
  temp, tz='Montreal', unit='°C', zone}
```

You can access specific tags, again, by giving the tag's name as the key

```
val = my_office.tags['curVal']
# That will return BasicQuantity(23.4428, '°C')
# from which you can retrieve
val.value
val.unit
```

What is the ~nn codes I keep seeing?

This is a feature specific to nHaystack. Project Haystack entity identifiers have a restricted character set, and only support a small subset of possible ASCII characters. nHaystack derives the entity's identifier from the name supplied by the user in the back-end configuration.

To encode other forms of characters (from the ISO8859-1 character set), the character is replaced by the sequence, `~nn` where `nn` is the hexadecimal character code for that character. In this case, you'll see `~2d` in place of -, and `~e9` in place of é.

2.5.3 Adding, Changing and Deleting tags

From this interface, it is also possible to update the values of these tags. This requires a back-end server that supports "CRUD" operations (Create, Read, Update & Delete). If your server supports these operations (and pyhaystack supports using them), the `tags` property will be of type *pyhaystack.client.entity.tags.MutableEntityTags*.

Again, this object functions like a `dict`:

```
# Change the display text
znt.tags['dis'] = 'A new display text string'

# Delete the 'his' tag
del znt.tags['his']

# Add a new tag
znt.tags['space'] = hszinc.Quantity(4, 'm²')
```

The changes are held in memory until such time as you either commit them, or revert them. When changes are stored locally, the `is_dirty` property will return `True`.

To forget these changes and roll it back to what's live on the server, call `revert`. This can take an optional list (or other iterable sequence) of tag names that you specifically wish to revert.

Alternatively, to push these changes, call `commit`, which takes an optional callback function. The return value of `commit` is a state machine that returns an instance of the updated entity on success (or raises an exception with the error):

```
assert znt.is_dirty      # assert will pass because of changes
op = znt.commit()
op.wait()

assert op.result is znt # ← this assert will pass
assert not znt.is_dirty # assert will pass because we've "committed"
                        # our changes back to the server.
```

2.6 Histories

Histories are a really important part of building data. Pyhaystack allows retrieving histories as zinc grid, pandas series or pandas Dataframe depending on your needs.

2.6.1 Range

The range parameter is named `rng` to avoid a naming clash with the Python `range` keyword. Its value can be any of the following: [ref : his_read]

- `"today"`
- `"yesterday"`
- `"first"` (WideSky only: returns the very first record)
- `"last"` (WideSky only: returns the very last record)
- `"{date}"`
- `"{date},{date}"`

- `"{dateTime},{dateTime}"`
- `"{dateTime}"` (anything after given timestamp)

As a convenience, pyhaystack also understands, and will translate a `datetime.date`, `datetime.datetime` or `slice` object, so the following will work:

- `datetime.date(2017, 5, 7)` will be sent as `"2017-05-07"`
- `pytz.timezone('Australia/Brisbane').localize(datetime.datetime(2017,5,7, 17,41))` will be sent as `"2017-05-07T17:41+10:00 Brisbane"`
- `slice(datetime.date(2017, 5, 8), datetime.date(2017, 5, 10))` will be sent as `"2017-05-08,2017-05-10"`.

How do I generate a timestamp in a point's local time zone?

Any Project Haystack entity that carries the `tz` tag when retrieved using `find_entity` will automatically be a sub-class of the *`pyhaystack.client.entity.mixins.tz.TzMixin`* class.

This mix-in class defines three properties:

- `hs_tz`: The Project Haystack timezone name (equivalent to `entity.tags['tz']`), e.g. `"Brisbane"`
- `iana_tz`: The IANA name for the timezone, e.g. `"Australia/Brisbane"`
- `tz`: A `datetime.tzinfo` object that represents the timezone.

The timezone handling is built on the `pytz` module, the full documentation for which is on the pytz website.

`site` entities usually have a corresponding `tz` tag attached to them, knowing this:

```
import datetime

# Retrieve a single entity, by ID
op = session.get_entity('S.SERVISYS', single=True)
op.wait()
site = op.result        # is the entity S.SERVISYS

# The current time at S.SERVISYS (to the microsecond)
now = datetime.datetime.now(tz=site.tz)

# A specific time, at S.SERVISYS
sometime = site.tz.localize(datetime.datetime(2017, 5, 7, 18, 11))
```

To retrieve all historical samples between `sometime` and `now`, one could use:

```
op = session.his_read('S.SERVISYS.Bureau-Christian.ZN~2dT',
     rng=slice(sometime, now))
op.wait()
history = op.result
```

A gotcha regarding the precision of timestamps

Some project haystack servers, notably anything that is based on nodehaystack will reject timestamps that have greater than millisecond-precision. This may require that you round the timestamp down to the nearest millisecond first before passing it to pyhaystack.

One way this can be done is using the following lambda function:

```
truncate_msec = lambda dt : dt - datetime.timedelta( \
        microseconds=dt.microsecond % 1000)
```

which is then used like this:

```
now_to_the_msec = truncate_msec(now)
```

2.6.2 Zinc Date and time format

[ref : http://project-haystack.org/doc/Zinc]

- <date> := YYYY-MM-DD
- <time> := hh:mm:ss.FFFFFFFFF
- <dateTime> := YYYY-MM-DD'T'hh:mm:ss.FFFFFFFFFz zzzz

```
op = session.his_read('S.SERVISYS.Bureau-Christian.ZN~2dT', rng='today')
op.wait()
op.result
```

For more details about Pandas : pandas_datastructure

```
op = session.his_read_series('S.SERVISYS.Bureau-Christian.ZN~2dT', rng='today')
op.wait()
op.result
```

In the following example, we will retrive all the historical value from 'today' for all zone temperature sensors.

```
# Retrieve some points
op = session.find_entity(filter_expr='sensor and zone and temp')
op.wait()
znt = op.result

# Read today's history for all found points
op = session.his_read_frame(znt, rng= 'today')
op.wait()
b = op.result
b
```

We use `find_entity` first, then we call `his_read_frame` over the result.

Describe is a Pandas function that gives you some information about a Dataframe or a serie.

Here is an example from the room_temp_serie

```
room_temp_serie.describe()
```

```
count    55.000000
mean     23.454680
std       0.388645
min      22.551900
25%      23.169800
50%      23.689800
75%      23.748750
max      23.806300
dtype: float64
```

2.7 Quantity

Quantity is a way to attach a unit to a float value. Created by hszinc it comes in two flavours : BasicQuantity and PintQuantity

BasicQuantity is a simple parse of the unit read in the result of the haystack request. Each variable has a value property and a unit property. Which can be used in your analysis.

PintQuantity is an interpretation of value and units as physical quantities with relation between them.

> **"Pint is a Python package to define, operate and manipulate physical** quantities: the product of a numerical value and a unit of measurement. It allows arithmetic operations between them and conversions from and to different units."
>
> – Pint

It will allow for example, simple unit conversion on the spot.

2.7.1 How to configure

You choose between Quantities when defining the session.

```
session = NiagaraHaystackSession(uri='http://server', username='user', password=
↪'myComplicatedPassword', pint=True)
```

By default, Pint is not activated. It's possible to modify the choice dynamically using

```
session.config_pint(False) # or True
```

2.7.2 Pros and Cons

For analysis tasks, using PintQuantity is a good thing. You can easily convert between units and keep coherence in your analysis.

```
from pyhaystack import Q_
temp = Q_(13,'degC')
temp.to('degF')
```

But when it's time to write to a haystack server, things get complicated. Hard work has been done to convert from haystack units to Pint. The reverse process is really difficult because of the non-standard nature of units in project-haystack.

2.7.3 Unit database

Pyhaystack uses a custom unit dictionnary built at run time. For more details about that, please refer to hszinc documentation.

2.7.4 Pandas

When reading series and DataFrame, value stored inside are not Quantity. We extact the value property only. But for each serie, we add Metadata to store the unit so you know what's behind.

```
room_temp_serie.meta['units']
```

```
<UnitsContainer({'degC': 1.0})>
```

2.8 Using pyhaystack in a synchronous way

Exploring a site is the first thing we do to start analysing it. Here are some tips on the way you can explore your site.

We assume here that the session is created and you defined

```
site = session.site
```

2.8.1 Browsing a site

A site is typically filled with equipments. Pyhaystack assumes that if you use bracket request over a site, you probably want to explore :

- tags (area, dis, geoAddr, tz, etc...)
- equipments (VAV, AHU, etc...)
- anything else

Lookup will be made in this order. If the key you passed can't be found in tags, pyhaystack will start building a list of equipments under the site.

Read tags

All tags can be retrieved using site['tagName']:

```
site ['area']

# Returns
BasicQuantity(0.0, 'ft²')
```

Find equipments

Equipments can be found using the same syntax. So if you write

```
my_equip = site['myEquip']
Reading equipments for this site...
```

If the equipment exist, it will be returned

Once the first read is done, you can access the list using

```
site.equipments
# Returns a list of EquipSiteRefEntity
```

Note: The key provided will be compared to the ID, the dis and the navName. And will return the first hit.

Find points under equipments

The same logic we saw for site can be applied to equipment. Equipments are typically filled with points that we need to access to. Using the bracket syntax should allow us to do so. So if you write

```
zone_temp = my_equip['ZN~2dT']
Reading points for this equipment...
```

Note: This time again, a list is populated under the object. All points for the equipment will be accessible using the simple syntax *equip.points*. This list is also used to iterate rapidly over point when making search. This way, pyhaystack doesn't need to poll the server.

Finding something else using a filter

If the square bracket search doesn't find tag or equipment or point, it will also try to use find_entity using the provided key. This way, you can also use this simple syntax to look for more complicated results

```
air_sensors = my_equip['sensor and air']
# Returns all the points corresponding to this search.
```

Note: The square bracket search is context sensitive. The find_entity function will be called "where the search is done". This means that when using this search under an equipment, it will look under the equipment. You can also use this search under a site, the search will be done under this particular site.

2.8.2 Histories

Histories are a big parts of pyhaystack if you're using it for numerical analysis.

Pyhaystack provides functions to retrieve histories from your site allowing you to get your result in the form you want it (simple grid, Pandas Series or Pandas Dataframe).

As we want to do numerical analysis, I'll focus on Pandas Series and Dataframe.

As we saw earlier, we can retrieve entities using pyhaystack. Those entities can be used to retrieve histories.

Let's say we would want to retrieve every room temperature sensors on site.

```
room_temp_sensors = session.find_entity(filter_expr='sensor and zone and temp').result
room_temp_sensors_df = session.his_read_frame(room_temp_sensors, rng= 'today').result
room_temp_sensors_df.tail()
```

It's also possible to get a serie out of a sensor :

```
room_temp = session.find_entity(filter_expr='sensor and zone and temp').result
room_temp_serie = session.his_read_series(room_temp['S.SERVISYS.Corridor.ZN~2dT'],␣
↪rng= 'today').result
room_temp_serie
```

```
2016-06-29 00:00:01.937000-04:00    23.8063
2016-06-29 00:15:01.510000-04:00    23.8011
2016-06-29 00:30:01.599000-04:00    23.8020
```

(continues on next page)

(continued from previous page)

```
2016-06-29 00:45:01.931000-04:00    23.7959
2016-06-29 01:00:03.847000-04:00    23.7961
2016-06-29 01:15:01.486000-04:00    23.7956
2016-06-29 01:30:01.884000-04:00    23.7946
2016-06-29 01:45:01.663000-04:00    23.7944
2016-06-29 02:00:01.820000-04:00    23.7932
2016-06-29 02:15:01.766000-04:00    23.7929
2016-06-29 02:30:01.587000-04:00    23.7854
2016-06-29 02:45:01.413000-04:00    23.7606
2016-06-29 03:00:02.369000-04:00    23.7487
2016-06-29 03:15:01.584000-04:00    23.7490
2016-06-29 03:30:02.019000-04:00    23.7488
2016-06-29 03:45:01.478000-04:00    23.7474
2016-06-29 04:00:01.638000-04:00    23.7467
2016-06-29 04:15:01.756000-04:00    23.7450
2016-06-29 04:30:01.865000-04:00    23.7450
2016-06-29 04:45:01.782000-04:00    23.7254
2016-06-29 05:00:01.586000-04:00    23.7142
2016-06-29 05:15:01.370000-04:00    23.6986
2016-06-29 05:30:01.931000-04:00    23.6977
2016-06-29 05:45:01.758000-04:00    23.6969
2016-06-29 06:00:01.920000-04:00    23.6954
2016-06-29 06:15:01.498000-04:00    23.6922
2016-06-29 06:30:01.810000-04:00    23.6946
2016-06-29 06:45:00.236000-04:00    23.6898
2016-06-29 07:00:01.763000-04:00    23.6569
2016-06-29 07:15:01.751000-04:00    23.6571
2016-06-29 07:30:01.604000-04:00    23.6137
2016-06-29 07:45:01.762000-04:00    23.6046
2016-06-29 08:00:02.015000-04:00    22.9552
2016-06-29 08:15:01.482000-04:00    22.6888
2016-06-29 08:30:01.687000-04:00    22.9885
2016-06-29 08:45:00.155000-04:00    23.2589
2016-06-29 09:00:02.063000-04:00    23.4131
2016-06-29 09:15:01.586000-04:00    22.8142
2016-06-29 09:30:01.694000-04:00    22.5519
2016-06-29 09:45:01.475000-04:00    22.9732
2016-06-29 10:00:01.994000-04:00    23.2174
2016-06-29 10:15:01.652000-04:00    23.4262
2016-06-29 10:30:01.596000-04:00    23.4417
2016-06-29 10:45:01.891000-04:00    22.8423
2016-06-29 11:00:01.873000-04:00    22.7915
2016-06-29 11:15:01.775000-04:00    23.1458
2016-06-29 11:30:01.641000-04:00    23.4154
2016-06-29 11:45:01.652000-04:00    23.6271
2016-06-29 12:00:02.147000-04:00    22.9879
2016-06-29 12:15:01.527000-04:00    22.6588
2016-06-29 12:30:01.819000-04:00    22.8726
2016-06-29 12:45:01.590000-04:00    23.1938
2016-06-29 13:00:01.880000-04:00    23.4289
2016-06-29 13:15:01.609000-04:00    22.8838
2016-06-29 13:30:00.607000-04:00    22.8446
dtype: float64
```

As seen when we covered Quantities, you can extract metadata from Series and get the unit.

```
room_temp_serie.meta['units']
```

```
<UnitsContainer({'degC': 1.0})>
```

When using the square bracket search to retrieve a point, you can also chain the his function to the result

```
pcv6 = site['PCV~2d11~2d012_BVV~2d06']
zone_temp = pcv6['ZN~2dT']
zone_temp.his()

# Return a Pandas Series with today's history by default.
```

Refer to the chapter on histories for more details.

2.9 Plugin for Niagara

2.9.1 BQL

[ref: Getting Started with Niagara | docUser.pdf | Niagara documentation] BQL is one Scheme used to Query in the Niagara Framework. An Ord is made up of one or more Queries. A Query includes a Scheme and a body. The bql Scheme has a body with one of the following formats:

- BQL expression
- Select projection FROM extent Where predicate

You can create the Ord Qualifier, Select, FROM and Where portions of a Query.

Usage

Technically, BQL can be used to retrieve any information from a Niagara station. One really useful usage would be to request data from the alarm database or from the audit history.

```
bql_request = "station:|alarm:/|bql:select timestamp, alarmData.sourceName, \
normalTime,ackTime,alarmData.timeInAlarm, msgText, user, alarmData.lowLimit, \
alarmData.highLimit,alarmData.alarmValue, alarmData.notes, sourceState, ackState"

yesterday_alarms = bql_request + ' where timestamp in bqltime.yesterday'

alarms = session.get_bql(yesterday_alarms).result
```

The result will be a pandas dataframe.

2.10 Modules documentation

2.11 pyhaystack package

2.11.1 Subpackages

pyhaystack.client package

Subpackages

pyhaystack.client.entity package

Subpackages

pyhaystack.client.entity.mixins package

Submodules

pyhaystack.client.entity.mixins.equip module

'equip' related mix-ins for high-level interface.

class pyhaystack.client.entity.mixins.equip.**EquipMixin**
Bases: object

A mix-in used for entities that carry the 'equip' marker tag.

find_entity(*filter_expr=None*, *limit=None*, *single=False*, *callback=None*)
Retrieve the entities that are linked to this equipment. This is a convenience around the session find_entity method.

points
First call will force reading of points and create local list

refresh()
Re-create local list of equipments

class pyhaystack.client.entity.mixins.equip.**EquipRefMixin**
Bases: object

A mix-in used for entities that carry an 'equipRef' reference tag.

get_equip(*callback=None*)
Retrieve an instance of the equip this entity is linked to.

pyhaystack.client.entity.mixins.point module

'point' related mix-ins for high-level interface.

class pyhaystack.client.entity.mixins.point.**HisMixin**
Bases: object

A mix-in used for 'point' entities that carry the 'his' marker tag.

his(*rng='today'*, *tz=None*, *series_format=None*, *callback=None*)
Shortcut to read_series

his_read_series(*rng*, *tz=None*, *series_format=None*, *callback=None*)
Read the historical data of the this point and return it as a series.

Parameters

- **rng** – Historical read range for the 'point'
- **tz** – Optional timezone to translate timestamps to
- **series_format** – Optional desired format for the series

his_write_series(*series*, *tz=None*, *callback=None*)
Write the historical data of this point.

Parameters

- **series** – Historical series to write
- **tz** – Optional timezone to translate timestamps to

class pyhaystack.client.entity.mixins.point.**PointMixin**
Bases: object

value

pyhaystack.client.entity.mixins.site module

'site' related mix-ins for high-level interface.

class pyhaystack.client.entity.mixins.site.**SiteMixin**
Bases: object

A mix-in used for entities that carry the 'site' marker tag.

equipments
site.equipments returns the list of equipments under a site

First read will force a request and create local list

find_entity(*filter_expr=None*, *single=False*, *limit=None*, *callback=None*)
Retrieve the entities that are linked to this site. This is a convenience around the session find_entity method.

refresh()
Re-create local list of equipments

class pyhaystack.client.entity.mixins.site.**SiteRefMixin**
Bases: object

A mix-in used for entities that carry a 'siteRef' reference tag.

get_site(*callback=None*)
Retrieve an instance of the site this entity is linked to.

pyhaystack.client.entity.mixins.tz module

Mix-ins for entities exposing a 'tz' tag

class pyhaystack.client.entity.mixins.tz.**TzMixin**
Bases: object

A mix-in used for entities that carry the 'tz' tag.

hs_tz
Return the Project Haystack timezone type.

iana_tz
Return the IANA (Olson) database timezone name.

tz
Return the timezone information (datetime.tzinfo) for this entity.

Module contents

pyhaystack.client.entity.models package

Submodules

pyhaystack.client.entity.models.haystack module

Tagging Model Interface for Project Haystack

class pyhaystack.client.entity.models.haystack.**HaystackTaggingModel**(*session*)
Bases: *pyhaystack.client.entity.model.TaggingModel*

An implementation of the Project Haystack tagging model.

Initialise a new tagging model.

Module contents

pyhaystack.client.entity.ops package

Submodules

pyhaystack.client.entity.ops.crud module

Entity CRUD state machines. These are state machines that perform CRUD operations on entities at a high-level.

class pyhaystack.client.entity.ops.crud.**EntityTagUpdateOperation**(*entity*, *updates*)
Bases: *pyhaystack.util.state.HaystackOperation*

Tag update state machine. This returns the entity instance that was updated on success.

Initialise a request for the named IDs.

Parameters **session** – Haystack HTTP session object.

go()
Start the request, check cache for existing entities.

Module contents

Submodules

pyhaystack.client.entity.entity module

High-level Entity interface.

class pyhaystack.client.entity.entity.**DeletableEntity**(*session*, *entity_id*)
Bases: *pyhaystack.client.entity.entity.Entity*

Class to represent entities that can be deleted from the Haystack server (the server implements the 'delete' operation).

Initialise a new high-level entity object.

Parameters

- **session** – The connection session object.
- **entity_id** – The entity's fully qualified ID.

delete(*callback=None*)
Delete the entity.

class pyhaystack.client.entity.entity.**Entity**(*session*, *entity_id*)
Bases: object

A base class for Project Haystack entities. This is a base class that is then combined with mix-ins depending on the tags present for the entity and the tagging model in use (by default, we use the "Project Haystack" tagging model, but others such as ISA-95 may exist).

This base class just exposes the tags, and if supported by the server, may expose the ability to update those tags.

Initialise a new high-level entity object.

Parameters

- **session** – The connection session object.
- **entity_id** – The entity's fully qualified ID.

dis
Return the description field of the entity.

id
Return the fully qualified ID of this entity.

tags
Return the tags of this entity.

exception pyhaystack.client.entity.entity.**StaleEntityInstanceError**
Bases: exceptions.Exception

Exception thrown when an entity instance is "stale", that is, the entity class type no longer matches the tag set present in the entity.

pyhaystack.client.entity.model module

Tagging Model Interface.

class pyhaystack.client.entity.model.**TaggingModel**(*session*)
Bases: object

A base class for representing tagging models. The tagging model is responsible for considering the tags present on a new entity then instantiating an appropriate data type based on those tags seen.

Initialise a new tagging model.

create_entity(*entity_id*, *tags*)
Create an Entity instance based on the tags present.

pyhaystack.client.entity.tags module

Entity tag interface. This file implements the interfaces that will be used to access and store tags of an entity.

class pyhaystack.client.entity.tags.**BaseEntityTags**(*entity*)
Bases: object

A base class for storing entity tags.

Initialise a new high-level entity tag storage object.

Parameters

- **session** – The connection session object.
- **entity_id** – The entity's fully qualified ID.

class pyhaystack.client.entity.tags.**BaseMutableEntityTags**(*entity*)
Bases: *pyhaystack.client.entity.tags.BaseEntityTags*

A base class for entity tags that supports modifications to the tag set.

commit(*callback=None*)
Commit any to-be-sent updates for this entity.

is_dirty
Returns true if there are modifications pending submission.

revert(*tags=None*)
Revert the named attribute changes, or all changes.

class pyhaystack.client.entity.tags.**MutableEntityTags**(*entity*)
Bases: *pyhaystack.client.entity.tags.BaseMutableEntityTags*, _abcoll.MutableMapping

class pyhaystack.client.entity.tags.**ReadOnlyEntityTags**(*entity*)
Bases: *pyhaystack.client.entity.tags.BaseEntityTags*, _abcoll.Mapping

Initialise a new high-level entity tag storage object.

Parameters

- **session** – The connection session object.
- **entity_id** – The entity's fully qualified ID.

Module contents

pyhaystack.client.http package

Submodules

pyhaystack.client.http.auth module

Base HTTP client authentication classes. These classes simply act as containers for authentication methods defined in the HTTP spec.

class pyhaystack.client.http.auth.**AuthenticationCredentials**
Bases: object

A base class to represent authentication credentials.

class pyhaystack.client.http.auth.**BasicAuthenticationCredentials**(*username*, *password*)
Bases: *pyhaystack.client.http.auth.UserPasswordAuthenticationCredentials*

A class that represents Basic authentication.

class pyhaystack.client.http.auth.**DigestAuthenticationCredentials**(*username*, *password*)

Bases: *pyhaystack.client.http.auth.UserPasswordAuthenticationCredentials*

A class that represents Digest authentication.

class pyhaystack.client.http.auth.**UserPasswordAuthenticationCredentials**(*username*, *password*)

Bases: *pyhaystack.client.http.auth.AuthenticationCredentials*

A base class that represents username/password type authentication.

pyhaystack.client.http.base module

Base HTTP client handler class. This wraps a HTTP client library in a consistent interface to make processing and handling of requests more convenient and to aid portability of pyhaystack.

class pyhaystack.client.http.base.**CaseInsensitiveDict**(**args*, ***kwargs*)

Bases: dict

A dict object that maps keys in a case-insensitive manner.

class pyhaystack.client.http.base.**HTTPClient**(*uri=None*, *params=None*, *headers=None*, *cookies=None*, *auth=None*, *timeout=None*, *proxies=None*, *tls_verify=None*, *tls_cert=None*, *accept_status=None*, *log=None*, *insecure_requests_warning=True*, *requests_session=True*)

Bases: object

The base HTTP client interface. This class defines methods for making HTTP requests in a unified manner. The interface presented is an asynchronous one, even for synchronous implementations.

Instantiate a HTTP client instance with some default parameters. These parameters are made accessible as properties to be modified at will by the caller as needed.

Parameters

- **uri** – Base URI for all requests. If given, this string will be pre-pended to all requests passed through this client.
- **params** – A dictionary of key-value pairs to be passed as URI query parameters.
- **headers** – A dictionary of key-value pairs to be passed as HTTP headers.
- **cookies** – A dictionary of key-value pairs to be passed as cookies.
- **auth** – An instance of a HttpAuth object.
- **timeout** – An integer or float giving the default maximum time duration for requests before timing out.
- **proxies** – A dictionary mapping the hostname and protocol to a proxy server URI.
- **tls_verify** – For TLS servers, this determines whether the server is validated or not. It should be the path to the CA certificate file for this server, or alternatively can be set to 'True' to verify against CAs known to the client. (e.g. OS certificate store)

- **tls_cert** – For TLS servers, this specifies the certificate used by the client to authenticate itself with the server.
- **log** – If not None, then it's a logging object that will be used for debugging HTTP operations.
- **requests_session** – Request sessions handles a lot of things inclusding cookies and it is problematic with some implementations like Skyspark. This flag allows to disable this Session and eliminate cookies round-trip.

CONTENT_LENGTH_HDR = 'Content-Length'

CONTENT_TYPE_HDR = 'Content-Type'

PROTO_RE = <_sre.SRE_Pattern object>

get(*uri*, *callback*, ***kwargs*)
Convenience function: perform a HTTP GET operation. Arguments are the same as for request.

post(*uri*, *callback*, *body=None*, *body_type=None*, *body_size=None*, *headers=None*, ***kwargs*)
Convenience function: perform a HTTP POST operation. Arguments are the same as for request.

Parameters

- **body** – Body data to be posted in this request as a string.
- **body_type** – Body MIME data type. This is a convenience for setting the Content-Type header.
- **body_size** – Length of the body to be sent. If None, the length is autodetected. Set to False to avoid this.

request(*method*, *uri*, *callback*, *body=None*, *params=None*, *headers=None*, *cookies=None*, *auth=None*, *timeout=None*, *proxies=None*, *tls_verify=None*, *tls_cert=None*, *exclude_params=None*, *exclude_headers=None*, *exclude_cookies=None*, *exclude_proxies=None*, *accept_status=None*)
Perform a request with this client. Most parameters here exist to either add to or override the defaults given by the client attributes. The parameters **exclude_**... serve to allow selective removal of defaults.

Parameters

- **method** – The HTTP method to request.
- **uri** – URL for this request. If this is a relative URL, it will be relative to the URL given by the 'uri' attribute.
- **callback** – A callback function that will be presented with the result of this request.
- **body** – An optional body for the request.
- **params** – A dictionary of key-value pairs to be passed as URI query parameters.
- **headers** – A dictionary of key-value pairs to be passed as HTTP headers.
- **cookies** – A dictionary of key-value pairs to be passed as cookies.
- **auth** – An instance of a HttpAuth object.
- **timeout** – An integer or float giving the default maximum time duration for requests before timing out.
- **proxies** – A dictionary mapping the hostname and protocol to a proxy server URI.
- **tls_verify** – For TLS servers, this determines whether the server is validated or not. It should be the path to the CA certificate file for this server, or alternatively can be set to 'True' to verify against CAs known to the client. (e.g. OS certificate store)

- **tls_cert** – For TLS servers, this specifies the certificate used by the client to authenticate itself with the server.
- **exclude_params** – If True, exclude all default parameters and use only the parameters given. Otherwise, this is an iterable of parameter names to be excluded.
- **exclude_headers** – If True, exclude all default headers and use only the headers given. Otherwise, this is an iterable of header names to be excluded.
- **exclude_cookies** – If True, exclude all default cookies and use only the cookies given. Otherwise, this is an iterable of cookie names to be excluded.
- **exclude_proxies** – If True, exclude all default proxies and use only the proxies given. Otherwise, this is an iterable of proxy names to be excluded.
- **accept_status** – If not None, this gives a list of status codes that will not raise an error, but instead be passed through for the Haystack client to handle.

silence_insecured_warnings()
Can be used to disable Insecure Requests Warnings in "controlled" environment. Use with care.

class pyhaystack.client.http.base.**HTTPResponse**(*status_code*, *headers*, *body*, *cookies=None*)
Bases: object

A class that represents the raw response from a HTTP request.

content_type
Return the content type of the body.

content_type_args
Return the content type arguments of the body.

text
Attempt to decode the raw body into text based on the encoding given.

pyhaystack.client.http.exceptions module

HTTP client exception classes.

exception pyhaystack.client.http.exceptions.**HTTPBaseError**
Bases: exceptions.IOError

Error class to represent a HTTP errors.

exception pyhaystack.client.http.exceptions.**HTTPConnectionError**
Bases: *pyhaystack.client.http.exceptions.HTTPBaseError*

Error class to represent a failed attempt to connect to a host.

exception pyhaystack.client.http.exceptions.**HTTPRedirectError**
Bases: *pyhaystack.client.http.exceptions.HTTPBaseError*

Error class to represent that the server's redirections are looping.

exception pyhaystack.client.http.exceptions.**HTTPStatusError**(*message*, *status*, *headers=None*, *body=None*)
Bases: *pyhaystack.client.http.exceptions.HTTPBaseError*

Error class to represent a returned failed status from the host.

exception pyhaystack.client.http.exceptions.**HTTPTimeoutError**
Bases: *pyhaystack.client.http.exceptions.HTTPConnectionError*

Error class to represent that a request timed out.

pyhaystack.client.http.sync module

Synchronous HTTP client using Python Requests.

class pyhaystack.client.http.sync.**SyncHttpClient**(***kwargs*)
Bases: *pyhaystack.client.http.base.HTTPClient*

Module contents

pyhaystack.client.mixins package

Subpackages

pyhaystack.client.mixins.vendor package

Subpackages

pyhaystack.client.mixins.vendor.widesky package

Submodules

pyhaystack.client.mixins.vendor.widesky.crud module

VRT Widesky low-level CRUD operation mix-in. This is a mix-in for HaystackSession that implements CRUD operations as used in VRT Widesky's implementation of Project Haystack.

At present, this has not been adopted by other implementations.

class pyhaystack.client.mixins.vendor.widesky.crud.**CRUDOpsMixin**
Bases: object

The CRUD operations mix-in implements low-level support for entity create / update / delete operation extensions to Project Haystack. (Read is not included, since that's part of the core standard.)

create(*entities*, *callback=None*)
Create the entities listed. If given a dict, we are creating a single entity, otherwise multiple entities may be given as individual dicts.

Each dict must carry an *id* which is a string giving the unique fully qualified ID of the entity. All other keys are taken to be tag values to attach to that entity, the values of which must be valid *hszinc* types.

Parameters **entities** – The entities to be inserted.

create_entity(*entities*, *single=None*, *callback=None*)
Create the entities listed, and return high-level entity instances for them. This is a high-level convenience around the above *create* method.

delete(*ids=None*, *filter_expr=None*, *callback=None*)
Delete entities matching the given criteria. Either ids or filter_expr may be given. ids may be given as a list or as a single ID string/reference.

filter_expr is given as a string. pyhaystack.util.filterbuilder may be useful for generating these programatically.

Parameters

- **ids** – IDs of many entities to retrieve as a list
- **filter_expr** – A filter expression that describes the entities of interest.

update(*entities*, *callback=None*)
Update the entities listed. If given a dict, we are creating a single entity, otherwise multiple entities may be given as individual dicts.

Each dict must carry an *id* which is a string giving the unique fully qualified ID of the entity. All other keys are taken to be tag values to update on that entity, the values of which must be valid *hszinc* types.

Parameters **entities** – The entities to be updated.

pyhaystack.client.mixins.vendor.widesky.multihis module

VRT Widesky low-level multi-hisRead/hisWrite operation mix-in. This is a mix-in for HaystackSession that implements CRUD operations as used in VRT Widesky's implementation of Project Haystack.

At present, this has not been adopted by other implementations.

class pyhaystack.client.mixins.vendor.widesky.multihis.**MultiHisOpsMixin**
Bases: object

The Multi-His operations mix-in implements low-level support for multi-point hisRead and hisWrite operations.

multi_his_read(*points*, *rng*, *callback=None*)
Read the historical data for multiple points. This processes each point given by the list points and returns the data from that point in a numbered column named idN where N starts counting from zero.

multi_his_write(*timestamp_records*, *callback=None*)
Write the historical data for multiple points.

timestamp_records may be given as: - a Pandas DataFrame object, with column names specifying IDs of points - a list of dicts with the key 'ts' mapping to a datetime object and

the remaining keys mapping point IDs to the values to be written.

- a dict of dicts, with the outer dict mapping timestamps to the inner dict mapping point IDs to values.

Module contents

Module contents

Module contents

pyhaystack.client.ops package

Subpackages

pyhaystack.client.ops.vendor package

Submodules

pyhaystack.client.ops.vendor.niagara module

Niagara AX operation implementations.

class pyhaystack.client.ops.vendor.niagara.**NiagaraAXAuthenticateOperation**(*session*, *retries=0*)

Bases: *pyhaystack.util.state.HaystackOperation*

An implementation of the log-in procedure for Niagara AX. The procedure is as follows:

1. Do a request of the log-in URL, without credentials. This sets session cookies in the client. Response should be code 200.
2. Pick up the session cookie named 'niagara_session', submit this in a GET request for the login URL with a number of other parameters. Response should NOT include the word 'login'.

Future requests should include the basic authentication credentials.

Attempt to log in to the Niagara AX server.

Parameters

- **session** – Haystack HTTP session object.
- **uri** – Possibly partial URI relative to the server base address to perform a query. No arguments shall be given here.
- **expect_format** – Request that the grid be sent in the given format.
- **args** – Dictionary of key-value pairs to be given as arguments.
- **multi_grid** – Boolean indicating if we are to expect multiple grids or not. If True, then the operation will _always_ return a list, otherwise, it will _always_ return a single grid.
- **raw_response** – Boolean indicating if we should try to parse the result. If True, then we should just pass back the raw HTTPResponse object.
- **retries** – Number of retries permitted in case of failure.

go()

Start the request.

pyhaystack.client.ops.vendor.skyspark module

Skyspark operation implementations.

class pyhaystack.client.ops.vendor.skyspark.**SkysparkAuthenticateOperation**(*session*, *retries=2*)

Bases: *pyhaystack.util.state.HaystackOperation*

An implementation of the log-in procedure for Skyspark. The procedure is as follows:

1. Retrieve the log-in URL (GET request).
2. Parse the key-values pairs returned, pick up 'username', 'userSalt' and 'nonce'.
3. **Compute** mac = Base64(HMAC_SHA1(key=password, msg="${username}:${userSalt}"))
4. **Compute** digest = Base64(SHA1("${mac}:${nonce}"))

5. **POST to log-in URL:** nonce: ${nonce} digest: ${digest}
6. Stash received cookies given in the returned body.

Future requests should the cookies returned.

Attempt to log in to the Skyspark server.

Parameters

- **session** – Haystack HTTP session object.
- **retries** – Number of retries permitted in case of failure.

go()
Start the request.

pyhaystack.client.ops.vendor.skyspark.**binary_encoding**(*string*, *encoding='utf-8'*)
This helper function will allow compatibility with Python 2 and 3

pyhaystack.client.ops.vendor.skyspark.**get_digest_info**(*param*)

pyhaystack.client.ops.vendor.widesky module

VRT WideSky operation implementations.

class pyhaystack.client.ops.vendor.widesky.**CreateEntityOperation**(*session*, *entities*, *single*)
Bases: *pyhaystack.client.ops.entity.EntityRetrieveOperation*

Operation for creating entity instances.

Parameters

- **session** – Haystack HTTP session object.
- **entities** – A list of entities to create.

go()
Start the request, preprocess and submit create request.

class pyhaystack.client.ops.vendor.widesky.**WideSkyHasFeaturesOperation**(*session*, *features*, ***kwargs*)
Bases: pyhaystack.client.ops.feature.HasFeaturesOperation

class pyhaystack.client.ops.vendor.widesky.**WideSkyPasswordChangeOperation**(*session*, *new_password*, ***kwargs*)
Bases: *pyhaystack.client.ops.grid.BaseAuthOperation*

The Password Change operation implements the logic required to change a user's password.

go()
Start the request.

class pyhaystack.client.ops.vendor.widesky.**WideskyAuthenticateOperation**(*session*, *retries=0*)
Bases: *pyhaystack.util.state.HaystackOperation*

An implementation of the log-in procedure for WideSky. WideSky uses a M2M variant of OAuth2.0 to authenticate users.

Attempt to log in to the VRT WideSky server. The procedure is as follows:

POST to auth_dir URI:

Headers: Accept: application/json Authorization: Basic [BASE64:"[ID]:[SECRET]"] Content-Type: application/json

Body: { username: "[USER]", password: "[PASSWORD]", grant_type: "password"

}

EXPECT reply:

Headers: Content-Type: application/json

Body:

{ access_token: ..., refresh_token: ..., expires_in: ..., token_type: ...

}

Parameters

- **session** – Haystack HTTP session object.
- **retries** – Number of retries permitted in case of failure.

go()
Start the request.

Module contents

Submodules

pyhaystack.client.ops.entity module

Entity state machines. These are state machines that perform CRUD operations on entities.

class pyhaystack.client.ops.entity.**EntityRetrieveOperation**(*session*, *single*)
Bases: *pyhaystack.util.state.HaystackOperation*

Base class for retrieving entity instances.

Initialise a request for the named IDs.

Parameters **session** – Haystack HTTP session object.

class pyhaystack.client.ops.entity.**FindEntityOperation**(*session*, *filter_expr*, *limit*, *single*)
Bases: *pyhaystack.client.ops.entity.EntityRetrieveOperation*

Operation for retrieving entity instances by filter. This operation peforms the following steps:

```
Issue a read instruction with the given filter:
    For each row returned in grid:
        If entity is not in cache:
            Create new Entity instances for each row returned.
        Else:
            Update existing Entity instance with new row data.
        Add the new entity instances to cache and store.
```

(continues on next page)

(continued from previous page)

```
    Return the stored entities.
    # State: done
```

Initialise a request for the named IDs.

Parameters

- **session** – Haystack HTTP session object.
- **filter_expr** – Filter expression.
- **limit** – Maximum number of entities to fetch.

go()
Start the request, check cache for existing entities.

class pyhaystack.client.ops.entity.**GetEntityOperation**(*session*, *entity_ids*, *refresh_all*, *single*)

Bases: *pyhaystack.client.ops.entity.EntityRetrieveOperation*

Operation for retrieving entity instances by ID. This operation peforms the following steps:

```
If refresh_all is False:
# State: init
    For each entity_id in entity_ids:
        If entity_id exists in cache:
            Retrieve and store entity from cache.
            Add entity_id to list got_ids.
    For each entity_id in got_ids:
        Discard entity_id from entity_ids.
If entity_ids is not empty:
# State: read
    Perform a low-level read of the IDs.
    For each row returned in grid:
        If entity is not in cache:
            Create new Entity instances for each row returned.
        Else:
            Update existing Entity instance with new row data.
    Add the new entity instances to cache and store.
Return the stored entities.
# State: done
```

Initialise a request for the named IDs.

Parameters

- **session** – Haystack HTTP session object.
- **entity_ids** – A list of IDs to request.
- **refresh_all** – Refresh all entities, ignore existing content.

go()
Start the request, check cache for existing entities.

pyhaystack.client.ops.grid module

Low-level grid state machines. These are state machines that perform GET or POST requests involving Haystack ZINC grids.

class pyhaystack.client.ops.grid.**BaseAuthOperation**(*session*, *uri*, *retries=2*, *cache=False*)

Bases: *pyhaystack.util.state.HaystackOperation*

A base class authentication operations.

Initialise a request for the authenticating with the given URI and arguments.

It also contains the state machine for reconnection if needed.

Parameters

- **session** – Haystack HTTP session object.
- **uri** – Possibly partial URI relative to the server base address to perform a query. No arguments shall be given here.
- **retries** – Number of retries permitted in case of failure.
- **cache** – Whether or not to cache this result. If True, the result is cached by the session object.

go()

Start the request.

class pyhaystack.client.ops.grid.**BaseGridOperation**(*session*, *uri*, *args=None*, *expect_format='text/zinc'*, *multi_grid=False*, *raw_response=False*, *retries=2*, *cache=False*, *cache_key=None*, *accept_status=None*, *headers=None*, *exclude_cookies=None*)

Bases: *pyhaystack.client.ops.grid.BaseAuthOperation*

A base class for GET and POST operations involving grids.

Initialise a request for the grid with the given URI and arguments.

Parameters

- **session** – Haystack HTTP session object.
- **uri** – Possibly partial URI relative to the server base address to perform a query. No arguments shall be given here.
- **expect_format** – Request that the grid be sent in the given format.
- **args** – Dictionary of key-value pairs to be given as arguments.
- **multi_grid** – Boolean indicating if we are to expect multiple grids or not. If True, then the operation will _always_ return a list, otherwise, it will _always_ return a single grid.
- **raw_response** – Boolean indicating if we should try to parse the result. If True, then we should just pass back the raw HTTPResponse object.
- **retries** – Number of retries permitted in case of failure.
- **cache** – Whether or not to cache this result. If True, the result is cached by the session object.
- **cache_key** – Name of the key to use when the object is cached.
- **accept_status** – What status codes to accept, in addition to the usual ones?

- **exclude_cookies** – If True, exclude all default cookies and use only the cookies given. Otherwise, this is an iterable of cookie names to be excluded.

class pyhaystack.client.ops.grid.**GetGridOperation**(*session*, *uri*, *args=None*, *multi_grid=False*, ***kwargs*)

Bases: *pyhaystack.client.ops.grid.BaseGridOperation*

A state machine that performs a GET operation then reads back a ZINC grid.

Initialise a GET request for the grid with the given URI and arguments.

Parameters

- **session** – Haystack HTTP session object.
- **uri** – Possibly partial URI relative to the server base address to perform a query. No arguments shall be given here.
- **args** – Dictionary of key-value pairs to be given as arguments.
- **multi_grid** – Boolean indicating if we are to expect multiple grids or not. If true, then the operation will _always_ return a list, otherwise, it will _always_ return a single grid.

class pyhaystack.client.ops.grid.**PostGridOperation**(*session*, *uri*, *grid*, *args=None*, *post_format='text/zinc'*, ***kwargs*)

Bases: *pyhaystack.client.ops.grid.BaseGridOperation*

A state machine that performs a POST operation with a ZINC grid then may read back a ZINC grid.

Initialise a POST request for the grid with the given grid, URI and arguments.

Parameters

- **session** – Haystack HTTP session object.
- **uri** – Possibly partial URI relative to the server base address to perform a query. No arguments shall be given here.
- **grid** – Grid (or grids) to be posted to the server.
- **post_format** – What format to post grids in?
- **args** – Dictionary of key-value pairs to be given as arguments.

pyhaystack.client.ops.grid.**dict_to_grid**(*d*)

pyhaystack.client.ops.his module

High-level history functions. These wrap the basic his_read function to allow some alternate representations of the historical data.

class pyhaystack.client.ops.his.**HisReadFrameOperation**(*session*, *columns*, *rng*, *tz*, *frame_format*)

Bases: *pyhaystack.util.state.HaystackOperation*

Read the series data from several 'point' entities and present them in a concise format.

Read the series data and return it.

Parameters

- **session** – Haystack HTTP session object.
- **columns** – IDs of historical point objects to read.

- **rng** – Range to read from 'point'
- **tz** – Timezone to translate timezones to. May be None.
- **frame_format** – What format to present the frame in.

FORMAT_DICT = 'dict'

FORMAT_FRAME = 'frame'

FORMAT_LIST = 'list'

go()
Start processing the operation. This is called by the caller (so after all __init__ functions have executed) in order to begin the asynchronous operation.

class pyhaystack.client.ops.his.**HisReadSeriesOperation**(*session*, *point*, *rng*, *tz*, *series_format*)
Bases: *pyhaystack.util.state.HaystackOperation*

Read the series data from a 'point' entity and present it in a concise format.

Read the series data and return it.

Parameters

- **session** – Haystack HTTP session object.
- **point** – ID of historical 'point' object to read.
- **rng** – Range to read from 'point'
- **tz** – Timezone to translate timezones to. May be None.
- **series_format** – What format to present the series in.

FORMAT_DICT = 'dict'

FORMAT_LIST = 'list'

FORMAT_SERIES = 'series'

go()
Start processing the operation. This is called by the caller (so after all __init__ functions have executed) in order to begin the asynchronous operation.

class pyhaystack.client.ops.his.**HisWriteFrameOperation**(*session*, *columns*, *frame*, *tz*)
Bases: *pyhaystack.util.state.HaystackOperation*

Write the series data to several 'point' entities.

Write the series data.

Parameters

- **session** – Haystack HTTP session object.
- **columns** – IDs of historical point objects to read.
- **frame** – Range to read from 'point'
- **tz** – Timezone to translate timezones to.

go()
Start processing the operation. This is called by the caller (so after all __init__ functions have executed) in order to begin the asynchronous operation.

class pyhaystack.client.ops.his.**HisWriteSeriesOperation**(*session*, *point*, *series*, *tz*)
Bases: *pyhaystack.util.state.HaystackOperation*

Write the series data to a 'point' entity.

Write the series data to the point.

Parameters

- **session** – Haystack HTTP session object.
- **point** – ID of historical 'point' object to write.
- **series** – Series data to be written to the point.
- **tz** – If not None, a datetime.tzinfo instance for this write.

go()
Start processing the operation. This is called by the caller (so after all __init__ functions have executed) in order to begin the asynchronous operation.

class pyhaystack.client.ops.his.**MetaDataFrame**(**args*, ***kw*)
Bases: pandas.core.frame.DataFrame

Custom Pandas Dataframe with meta data Made from MetaSeries

add_meta(*key*, *value*)

meta = {}

class pyhaystack.client.ops.his.**MetaSeries**(*data=None*, *index=None*, *dtype=None*, *name=None*, *copy=False*, *fastpath=False*)
Bases: pandas.core.series.Series

Custom Pandas Serie with meta data

add_meta(*key*, *value*)

meta = {}

Module contents

Submodules

pyhaystack.client.loader module

Haystack Implementation loader and factory. This module provides a simplified wrapper around importlib to allow implementation of a near-consistent interface for fetching session instances.

pyhaystack.client.loader.**get_implementation**(*implementation*)
Get an implementation of Project Haystack session manager based on the class name.

pyhaystack.client.loader.**get_instance**(*implementation*, **args*, ***kwargs*)
Get an instance of a Project Haystack client.

pyhaystack.client.niagara module

Tridium Niagara Client support (AX and N4)

class pyhaystack.client.niagara.**Niagara4HaystackSession**(*uri*, *username*, *password*, *grid_format='application/json'*, ***kwargs*)

Bases: *pyhaystack.client.session.HaystackSession*, pyhaystack.client.mixins.vendor.niagara.bql.BQLMixin, pyhaystack.client.mixins.vendor.niagara.encoding.EncodingMixin

The Niagara4HaystackSession class implements some base support for Niagara4. This is mainly a convenience for collecting the username and password details.

Initialise a Nagara 4 Project Haystack session handler.

Parameters

- **uri** – Base URI for the Haystack installation.
- **username** – Authentication user name.
- **password** – Authentication password.
- **grid_format** – the grid format to use in series (json, zinc)

is_logged_in
Return true if the user is logged in.

logout()

class pyhaystack.client.niagara.**NiagaraHaystackSession**(*uri*, *username*, *password*, ***kwargs*)

Bases: *pyhaystack.client.session.HaystackSession*, pyhaystack.client.mixins.vendor.niagara.bql.BQLMixin, pyhaystack.client.mixins.vendor.niagara.encoding.EncodingMixin

The NiagaraHaystackSession class implements some base support for NiagaraAX. This is mainly a convenience for collecting the username and password details.

Initialise a Nagara Project Haystack session handler.

Parameters

- **uri** – Base URI for the Haystack installation.
- **username** – Authentication user name.
- **password** – Authentication password.

is_logged_in
Return true if the user is logged in.

logout()

pyhaystack.client.session module

Core Haystack Session client object interface. This file defines an abstract interface for Project Haystack clients and is responsible for opening and maintaining a session with the server.

class pyhaystack.client.session.**HaystackSession**(*uri*, *api_dir*, *grid_format='text/zinc'*, *http_client=<class 'pyhaystack.client.http.sync.SyncHttpClient'>*, *http_args=None*, *tagging_model=<class 'pyhaystack.client.entity.models.haystack.HaystackTaggingModel'>*, *log=None*, *pint=False*, *cache_expiry=3600.0*)

Bases: object

The Haystack Session handler is responsible for presenting an API for querying and controlling a Project Haystack server.

HaystackSession itself is the base class, which is then implemented by way of HaystackOperation subclasses which are instantiated by the session object before being started and returned.

These operations by default are specified by class member references to the classes concerned.

Methods for Haystack operations return an 'Operation' object, which may be used in any of two ways:

- as a synchronous result placeholder by calling its *wait* method

followed by inspection of its *result* attribute. - as an asynchronous call manager by connecting a "slot" (*callable* that takes keyword arguments) to the *done_sig* signal.

The base class takes some arguments that control the default behaviour of the object.

Initialise a base Project Haystack session handler.

Parameters

- **uri** – Base URI for the Haystack installation.
- **api_dir** – Subdirectory relative to URI where API calls are made.
- **grid_format** – What format to use for grids in GET/POST requests?
- **http_client** – HTTP client class to use.
- **http_args** – Optional HTTP client arguments to configure.
- **tagging_model** – Entity Tagging model in use.
- **log** – Logging object for reporting messages.
- **pint** – Configure hszinc to use basic quantity or Pint Quanity
- **cache_expiry** – Number of seconds before cached data expires.

See : https://pint.readthedocs.io/ for details about pint

FEATURE_HISREAD_MULTI = 'hisRead/multi'

FEATURE_HISWRITE_MULTI = 'hisWrite/multi'

FEATURE_ID_UUID = 'id_uuid'

about(*cache=True*, *callback=None*)
Retrieve the version information of this Project Haystack server.

authenticate(*callback=None*)
Authenticate with the Project Haystack server. If an authentication attempt is in progress, we return it, otherwise we instantiate a new one.

config_pint(*value=False*)

find_entity(*filter_expr*, *limit=None*, *single=False*, *callback=None*)
Retrieve instances of entities that match a filter expression.

Parameters

- **filter_expr** – The filter expression to search for.
- **limit** – Optional limit to number of entities retrieved.
- **single** – Are we expecting a single entity? Defaults to True if *ids* is not a list.
- **callback** – Asynchronous result callback.

formats(*cache=True*, *callback=None*)
Retrieve the grid formats supported by this Project Haystack server.

get_entity(*ids*, *refresh=False*, *single=None*, *callback=None*)
Retrieve instances of entities, possibly refreshing them.

Parameters

- **ids** – A single entity ID, or a list of entity IDs.
- **refresh** – Do we refresh the tags on those entities?
- **single** – Are we expecting a single entity? Defaults to True if *ids* is not a list.
- **callback** – Asynchronous result callback.

has_features(*features*, *cache=True*, *callback=None*)
Determine if a given feature is supported. This is a helper function for determining if the server implements a given feature. The feature is given as a string in the form of "base_feature/extension".

Result is a dict of features and the states (boolean).

Parameters **features** – Features to check for.

his_read(*point*, *rng*, *callback=None*)
point is either the ID of the historical point entity, or an instance of the historical point entity to read historical from. rng is either a string describing a time range (e.g. "today", "yesterday"), a datetime.date object (providing all samples on the nominated day), a datetime.datetime (providing all samples since the nominated time) or a slice of datetime.dates or datetime.datetimes.

his_read_frame(*columns*, *rng*, *tz=None*, *frame_format=None*, *callback=None*)
Read the historical data of multiple given points and return them as a data frame.

Parameters

- **columns** – A list of Haystack 'point' instances or a dict mapping the column label to the Haystack 'point' instance.
- **rng** – Historical read range for the 'point'
- **tz** – Optional timezone to translate timestamps to
- **frame_format** – Optional desired format for the data frame

his_read_series(*point*, *rng*, *tz=None*, *series_format=None*, *callback=None*)
Read the historical data of the given point and return it as a series.

Parameters

- **point** – Haystack 'point' entity to read the data from
- **rng** – Historical read range for the 'point'
- **tz** – Optional timezone to translate timestamps to

- **series_format** – Optional desired format for the series

his_write(*point*, *timestamp_records*, *callback=None*)
point is either the ID of the writeable historical point entity, or an instance of the writeable historical point entity to write historical data to. timestamp_records should be a dict mapping timestamps (datetime.datetime) to the values to be written at those times, or a Pandas Series object.

his_write_frame(*frame*, *columns=None*, *tz=None*, *callback=None*)
Write the historical data of multiple given points.

Parameters

- **frame** – Data frame to write to. Columns either list explicit entity IDs or column aliases which are mapped in the columns parameter.
- **columns** – If frame does not list explicit IDs, this should be a dict mapping the column names to either entity IDs or entity instances.
- **tz** – Reference timestamp to use for writing, default is UTC.

his_write_series(*point*, *series*, *tz=None*, *callback=None*)
Write the historical data of the given point.

Parameters

- **point** – Haystack 'point' entity to read the data from
- **series** – Historical series data to write
- **tz** – Optional timezone to translate timestamps to

invoke_action(*entity*, *action*, *callback=None*, ***kwargs*)
entity is either the ID of the entity, or an instance of the entity to invoke the named action on. Keyword arguments give any additional parameters required for the user action.

logout()

nav(*nav_id=None*, *callback=None*)
The nav op is used navigate a project for learning and discovery. This operation allows servers to expose the database in a human-friendly tree (or graph) that can be explored.

ops(*cache=True*, *callback=None*)
Retrieve the operations supported by this Project Haystack server.

point_write(*point*, *level=None*, *val=None*, *who=None*, *duration=None*, *callback=None*)
point is either the ID of the writeable point entity, or an instance of the writeable point entity to retrieve the write status of or write a value to.

If level is None, the other parameters are required to be None too, the write status of the point is retrieved. Otherwise, a write is performed to the nominated point.

read(*ids=None*, *filter_expr=None*, *limit=None*, *callback=None*)
Retrieve information on entities matching the given criteria. Either ids or filter_expr may be given. ids may be given as a list or as a single ID string/reference.

filter_expr is given as a string. pyhaystack.util.filterbuilder may be useful for generating these programatically.

Parameters

- **id** – ID of a single entity to retrieve
- **ids** – IDs of many entities to retrieve as a list
- **filter_expr** – A filter expression that describes the entities of interest.

- **limit** – A limit on the number of entities to return.

site
This helper will return the first site found on the server. This case is typical : having one site per server.

sites
This helper will return all sites found on the server.

watch_poll(*watch*, *refresh=False*, *callback=None*)
watch is either the value of watch_id given when creating a watch, or an instance of a Watch object.

If refresh is True, then all points on the watch will be updated, not just those that have changed since the last poll.

watch_sub(*points*, *watch_id=None*, *watch_dis=None*, *lease=None*, *callback=None*)
This creates a new watch with debug string watch_dis, identifier watch_id (string) and a lease time of lease (integer) seconds. points is a list of strings, Entity objects or hszinc.Ref objects.

watch_unsub(*watch*, *points=None*, *callback=None*)
watch is either the value of watch_id given when creating a watch, or an instance of a Watch object.

If points is not None, it is a list of strings, Entity objects or hszinc.Ref objects which will be removed from the Watch object. Otherwise, it closes the Watch object.

pyhaystack.client.skyspark module

Skyspark Client support

class pyhaystack.client.skyspark.**SkysparkHaystackSession**(*uri*, *username*, *password*, *project=''*, ***kwargs*)
Bases: *pyhaystack.client.session.HaystackSession*, pyhaystack.client.mixins.vendor.skyspark.evalexpr.EvalOpsMixin

The SkysparkHaystackSession class implements some base support for Skyspark servers.

Initialise a Skyspark Project Haystack session handler.

Parameters

- **uri** – Base URI for the Haystack installation.
- **username** – Authentication user name.
- **password** – Authentication password.
- **project** – Skyspark project name

is_logged_in
Return true if the user is logged in.

class pyhaystack.client.skyspark.**SkysparkScramHaystackSession**(*uri*, *username*, *password*, *project*, *http_args=None*, ***kwargs*)
Bases: *pyhaystack.client.session.HaystackSession*, pyhaystack.client.mixins.vendor.skyspark.evalexpr.EvalOpsMixin

The SkysparkHaystackSession class implements some base support for Skyspark servers.

Initialise a Skyspark Project Haystack session handler.

Parameters

- **uri** – Base URI for the Haystack installation.
- **username** – Authentication user name.
- **password** – Authentication password.
- **project** – Skyspark project name

is_logged_in
Return true if the user is logged in.

logout()
close session when leaving context by trick given by Brian Frank

https://www.skyfoundry.com/forum/topic/5282#c1

but beware that this is not standard!

pyhaystack.client.widesky module

VRT Widesky Client support

class pyhaystack.client.widesky.**WideskyHaystackSession**(*uri*, *username*, *password*, *client_id*, *client_secret*, *api_dir='api'*, *auth_dir='oauth2/token'*, *impersonate=None*, ***kwargs*)

Bases: *pyhaystack.client.mixins.vendor.widesky.crud.CRUDOpsMixin*, *pyhaystack.client.mixins.vendor.widesky.multihis.MultiHisOpsMixin*, pyhaystack.client.mixins.vendor.widesky.password.PasswordOpsMixin, *pyhaystack.client.session.HaystackSession*

The WideskyHaystackSession class implements some base support for Widesky servers. This is mainly a convenience for collecting the username and password details.

Initialise a VRT Widesky Project Haystack session handler.

Parameters

- **uri** – Base URI for the Haystack installation.
- **username** – Authentication user name.
- **password** – Authentication password.
- **client_id** – Authentication client ID.
- **client_secret** – Authentication client secret.
- **impersonate** – A widesky user ID to impersonate (or None)

is_logged_in
Return true if the user is logged in.

Module contents

Haystack Client interface

pyhaystack.client.**get_implementation**(*implementation*)
Get an implementation of Project Haystack session manager based on the class name.

pyhaystack.client.**get_instance**(*implementation*, **args*, ***kwargs*)
Get an instance of a Project Haystack client.

pyhaystack.util package

Submodules

pyhaystack.util.asyncexc module

Asynchronous Exception Handler. This implements a small lightweight object for capturing an exception in a manner that can be passed in callback arguments then re-raised elsewhere for handling in the callback function.

Typical usage:

```
def _some_async_function(..., callback_fn):
    try:
        do some async op that may fail
        result = ...
    except: # Capture all exceptions
        result = AsynchronousException()
    callback_fn(result)
```

class pyhaystack.util.asyncexc.**AsynchronousException**
Bases: object

reraise()

pyhaystack.util.filterbuilder module

Filter abstract syntax tree builder. These classes and functions attempt to build a filter string for use with 'read' operations by combining Python's operators to trick it into producing the desired values.

Yes, we're hijacking operators to do what they weren't expected to do.

Typical usage:

```
from pyhaystack.util import filterbuilder as fb

# Get all historical points:
session.find_points(fb.Field('his'))

# All historical points in Brisbane timezone.
session.find_points(fb.Field('his') &                ( fb.Field('tz') == fb.Scalar(
↪'Brisbane') ))
```

class pyhaystack.util.filterbuilder.**And**(*x*, *y*)
Bases: *pyhaystack.util.filterbuilder.Binary*

OP = 'and'

class pyhaystack.util.filterbuilder.**Base**
Bases: object

class pyhaystack.util.filterbuilder.**Binary**(*x*, *y*)
Bases: *pyhaystack.util.filterbuilder.Base*

class pyhaystack.util.filterbuilder.**Equal**(*x*, *y*)
Bases: *pyhaystack.util.filterbuilder.Binary*

OP = '=='

class pyhaystack.util.filterbuilder.**Field**(*value*)
Bases: *pyhaystack.util.filterbuilder.Base*

class pyhaystack.util.filterbuilder.**GreaterThan**(*x*, *y*)
Bases: *pyhaystack.util.filterbuilder.Binary*

OP = '>'

class pyhaystack.util.filterbuilder.**GreaterThanOrEqual**(*x*, *y*)
Bases: *pyhaystack.util.filterbuilder.Binary*

OP = '>='

class pyhaystack.util.filterbuilder.**LessThan**(*x*, *y*)
Bases: *pyhaystack.util.filterbuilder.Binary*

OP = '<'

class pyhaystack.util.filterbuilder.**LessThanOrEqual**(*x*, *y*)
Bases: *pyhaystack.util.filterbuilder.Binary*

OP = '<='

class pyhaystack.util.filterbuilder.**Not**(*value*)
Bases: *pyhaystack.util.filterbuilder.Unary*

OP = 'not'

class pyhaystack.util.filterbuilder.**NotEqual**(*x*, *y*)
Bases: *pyhaystack.util.filterbuilder.Binary*

OP = '!='

class pyhaystack.util.filterbuilder.**Or**(*x*, *y*)
Bases: *pyhaystack.util.filterbuilder.Binary*

OP = 'or'

class pyhaystack.util.filterbuilder.**Scalar**(*value*)
Bases: *pyhaystack.util.filterbuilder.Base*

class pyhaystack.util.filterbuilder.**Unary**(*value*)
Bases: *pyhaystack.util.filterbuilder.Base*

pyhaystack.util.state module

State machine interface. This is a base class for implementing state machines.

class pyhaystack.util.state.**HaystackOperation**(*result_copy=True*, *result_deepcopy=True*)
Bases: object

A core state machine object. This implements the basic interface presented for all operations in pyhaystack.

Initialisation. This should be overridden by subclasses to accept and validate the inputs presented for the operation, raising an appropriate Exception subclass if the inputs are found to be invalid.

These should be stored here by the initialisation function as private variables in suitably sanitised form. The core state machine object shall then be created and stored before the object is returned to the caller.

go()
Start processing the operation. This is called by the caller (so after all __init__ functions have executed) in order to begin the asynchronous operation.

is_done
Return true if the operation is complete.

is_failed
Return true if the result is an Exception.

result
Return the result of the operation or raise its exception. Raises NotReadyError if not ready.

state
Return the current state machine's state.

wait(*timeout=None*)
Wait for an operation to finish. This should *NOT* be called in the same thread as the thread executing the operation as this will deadlock.

exception pyhaystack.util.state.**NotReadyError**
Bases: exceptions.Exception

Exception raised when an attempt is made to retrieve the result of an operation before it is ready.

pyhaystack.util.tools module

pyhaystack.util.tools.**isBool**(*value*)
Helper function to detect if a value is boolean

pyhaystack.util.tools.**isfloat**(*value*)
Helper function to detect if a value is a float

pyhaystack.util.tools.**prettyprint**(*jsonData*)
Pretty print json object

Module contents

2.11.2 Submodules

2.11.3 pyhaystack.exception module

pyhaystack Custom exceptions

exception pyhaystack.exception.**AuthenticationProblem**
Bases: exceptions.Exception

exception pyhaystack.exception.**HaystackError**(*message*, *traceback=None*, **args*, ***kwargs*)
Bases: exceptions.Exception

Exception thrown when an error grid is returned by the Haystack server. See http://project-haystack.org/doc/Rest#errorGrid

exception pyhaystack.exception.**NoCookieReceived**
Bases: exceptions.Exception

exception pyhaystack.exception.**NoResponseFromServer**
Bases: exceptions.Exception

exception pyhaystack.exception.**ProblemReadingCookie**
Bases: exceptions.Exception

exception pyhaystack.exception.**ProblemSendingRequestToServer**
Bases: exceptions.Exception

exception pyhaystack.exception.**UnknownHistoryType**
Bases: exceptions.Exception

2.11.4 pyhaystack.info module

File : pyhaystackTest.py (2.x) This module allow a connection to a haystack server Features provided allow user to fetch data from the server and eventually, to post to it.

See http://www.project-haystack.org for more details

Project Haystack is an open source initiative to streamline working with data from the Internet of Things. We standardize semantic data models and web services with the goal of making it easier to unlock value from the vast quantity of data being generated by the smart devices that permeate our homes, buildings, factories, and cities. Applications include automation, control, energy, HVAC, lighting, and other environmental systems.

2.11.5 Module contents

Main init file

Loading hszinc Creating "connect" shortcut for get_instance

This file will also automatically disable SubjectAltNameWarning when dealing with CA certificate. See docs for more information about that.

2.12 Indices and tables

- genindex
- modindex
- search

2.13 Plugin for Skyspark

2.13.1 Eval

Eval is used to evaluate any Axon expression. Request: a grid with one column called expr and one row with Str expression to evaluate. Response: result of the expression converted to a grid using Etc.toGrid If an error occurs an error grid is returned.

Usage

// GET request /api/demo/eval?expr=readAll(site)

// POST request ver:"2.0" expr "readAll(site)"

```
result = session.get_eval('readAll(site)').result
```

2.14 Modules documentation

2.15 pyhaystack package

2.15.1 Subpackages

2.15.2 Submodules

2.15.3 pyhaystack.exception module

pyhaystack Custom exceptions

exception pyhaystack.exception.**AuthenticationProblem**
Bases: exceptions.Exception

exception pyhaystack.exception.**HaystackError**(*message*, *traceback=None*, **args*, ***kwargs*)
Bases: exceptions.Exception

Exception thrown when an error grid is returned by the Haystack server. See http://project-haystack.org/doc/Rest#errorGrid

exception pyhaystack.exception.**NoCookieReceived**
Bases: exceptions.Exception

exception pyhaystack.exception.**NoResponseFromServer**
Bases: exceptions.Exception

exception pyhaystack.exception.**ProblemReadingCookie**
Bases: exceptions.Exception

exception pyhaystack.exception.**ProblemSendingRequestToServer**
Bases: exceptions.Exception

exception pyhaystack.exception.**UnknownHistoryType**
Bases: exceptions.Exception

2.15.4 pyhaystack.info module

File : pyhaystackTest.py (2.x) This module allow a connection to a haystack server Features provided allow user to fetch data from the server and eventually, to post to it.

See http://www.project-haystack.org for more details

Project Haystack is an open source initiative to streamline working with data from the Internet of Things. We standardize semantic data models and web services with the goal of making it easier to unlock value from the vast quantity of data being generated by the smart devices that permeate our homes, buildings, factories, and cities. Applications include automation, control, energy, HVAC, lighting, and other environmental systems.

2.15.5 Module contents

Main init file

Loading hszinc Creating "connect" shortcut for get_instance

This file will also automatically disable SubjectAltNameWarning when dealing with CA certificate. See docs for more information about that.

2.16 Indices and tables

- genindex
- modindex
- search

CHAPTER 3

Modules documentation

CHAPTER 4

pyhaystack package

4.1 Subpackages

4.2 Submodules

4.3 pyhaystack.exception module

pyhaystack Custom exceptions

exception pyhaystack.exception.**AuthenticationProblem**
Bases: exceptions.Exception

exception pyhaystack.exception.**HaystackError**(*message*, *traceback=None*, **args*, ***kwargs*)
Bases: exceptions.Exception

Exception thrown when an error grid is returned by the Haystack server. See http://project-haystack.org/doc/Rest#errorGrid

exception pyhaystack.exception.**NoCookieReceived**
Bases: exceptions.Exception

exception pyhaystack.exception.**NoResponseFromServer**
Bases: exceptions.Exception

exception pyhaystack.exception.**ProblemReadingCookie**
Bases: exceptions.Exception

exception pyhaystack.exception.**ProblemSendingRequestToServer**
Bases: exceptions.Exception

exception pyhaystack.exception.**UnknownHistoryType**
Bases: exceptions.Exception

4.4 pyhaystack.info module

File : pyhaystackTest.py (2.x) This module allow a connection to a haystack server Features provided allow user to fetch data from the server and eventually, to post to it.

See http://www.project-haystack.org for more details

Project Haystack is an open source initiative to streamline working with data from the Internet of Things. We standardize semantic data models and web services with the goal of making it easier to unlock value from the vast quantity of data being generated by the smart devices that permeate our homes, buildings, factories, and cities. Applications include automation, control, energy, HVAC, lighting, and other environmental systems.

4.5 Module contents

Main init file

Loading hszinc Creating "connect" shortcut for get_instance

This file will also automatically disable SubjectAltNameWarning when dealing with CA certificate. See docs for more information about that.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W