pyhaystack Documentation

Release 0.9

Christian Tremblay, P.Eng.

July 15, 2016

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 :

- NiagaraAX by Tridium
- Widesky by VRT
- Skyspark by SkyFoundry

Connection to Niagara AX requires the nHaystack module by J2 Innovations

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
```

1.4 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

1.5 New implementation

We've been reworking everything from the early version. Now, pyhaystack is more robust and will be ready for asynchronous development.

This new 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.

1.6 Dependency

Pyhaystack highly depend 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

2.1.1 Using Niagara AX

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

Note that pint parameter is optionnal and default to False.

2.1.2 Using Widesky

```
from pyhaystack.client.widesky import WideskyHaystackSession
session = SkysparkHaystackSession(uri='http://ip:port',
                                 username='user',
                                 password='my_password',
                                 client_id = 'my_id',
                                 client_secret = 'my_secret'
                                 *pint=True)
```

2.1.3 Using Skyspark

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

2.1.4 On-Demand connection

Once the session is initialized, it won't connect until it needs to. Pyhaystack will benefit from the asynchronous framework and connect on demand. The session will be connected and the request will be sent to the server.

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

See next section to know more about requests.

2.2 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

```
about = session.about()
```

The output of "about" would print

```
<GetGridOperation done: <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', se
</Grid>>
```

2.2.1 Session.nav()

Session.nav() let you see what's under the server. Result of the request coud look like that

```
session.nav()

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

2.2.2 Site

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

```
site = session.find_entity(filter_expr='site')
```

and get something like

```
<FindEntityOperation done: {'S.SERVISYS': <@S.SERVISYS: {area=Quantity(0.0, 'ft²'), axSlotPath='slot
```

2.3 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...

2.3.1 Finding sensors

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

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

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.result
```

2.3.2 Exploring points and tags

This will return a dict that can be used the way you want. As the filter may include a lot of points, you will need to choose the one you're interested in.

```
my_office = znt['S.SERVISYS.Salle-Conf~e9rence.ZN~2dT']
# You wil then get acces to the tags of that point
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

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

2.4 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.

2.4.1 Range

rng (range) can be one of the following [ref : his_read]

- "today"
- "yesterday"
- "{date}"
- "{date},{date}"
- "{dateTime},{dateTime}"
- "{dateTime}" // anything after given timestamp

2.4.2 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

Retrieve simple grid

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

Retrieve a Pandas Series

For more details about Pandas : pandas_datastructure

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

Retrieve a Pandas Dataframe

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

```
znt = session.find_entity(filter_expr='sensor and zone and temp').result
b = session.his_read_frame(znt, rng= 'today').result
b
```

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

2.5 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.

2.5.1 How to configure

You choose between Quantities when defining the session.

```
session = NiagaraHaystackSession(uri='http://server', username='user', password='myComplicatedPasswo
```

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

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

2.5.2 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.

2.5.3 Unit database

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

2.5.4 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})>
```

2.6 Using pyhaystack in a synchronous way

2.6.1 Declaring session

```
from pyhaystack.client.niagara import NiagaraHaystackSession
import logging
logging.root.setLevel(logging.DEBUG)
session = NiagaraHaystackSession(uri='http://server', username='user', password='myComplicatedPasswo
```

2.6.2 Browsing a site

Let's have a look to the site

```
site = session.find_entity(filter_expr='site')
site
```

```
<FindEntityOperation done: {'S.SERVISYS': <@S.SERVISYS: {area=Quantity(0.0, 'ft²'), axSlotPath='slot
```

This print shows us the `__repr__()` function return value as a FindEntityOperation. If we were using this asynchronously, and let say the operation would not be finished, we would be noticed about the fact that it's not done.

Actually, we know the operation succeeded.

But site is not an object we can use easily. To retrive something useful, we need to call the result property.

```
site = site.result
site
```

```
{'S.SERVISYS': <@S.SERVISYS: {area=Quantity(0.0, 'ft²'), axSlotPath='slot:/site', axType='nhaystack:
```

Now we have a dict that we can use to retrieve the entity which is a pyhaystack object

```
type(site['S.SERVISYS'])
```

```
pyhaystack.client.entity.model.SiteTzEntity
```

Most common properties of entities are “dis” and “tags”

```
site['S.SERVISYS'].dis
```

```
'SERVISYS'
```

```
site['S.SERVISYS'].tags
```

```
{area=Quantity(0.0, 'ft²'), axSlotPath='slot:/site', axType='nhaystack:HSite', dis='SERVISYS', geoAdd
```

```
site['S.SERVISYS'].tags['tz']
```

```
'Montreal'
```

2.6.3 Wrap up

We created a request to find something on the server (the site). Pyhaystack gave us in return an operation. This operation runs in the background (if you’re using an asynchronous call or a thread...) The operation tells you when it’s done.

When the operation is done, you can retrieve the “result” using the property named “result”.

Typically, result will give a dict that contains the information you need.

In our case, the result was a pyhaystack entity that contained tags.

Tags are also a dict that can be browsed using square brackets.

2.6.4 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.

Find histories

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').resu
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
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})>
```

Describe

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
```

CHAPTER 3

Modules documentation

CHAPTER 4

pyhaystack package

4.1 Subpackages

4.1.1 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 equip. This is a convenience around the session find_entity method.

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

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

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

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

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.

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.

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.**HTTPClient**(*uri=None*, *params=None*, *headers=None*, *cookies=None*, *auth=None*, *timeout=None*, *proxies=None*, *tls_verify=None*, *tls_cert=None*, *log=None*)

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.

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*)

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.

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

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

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/Niagara 4 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.**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.
    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.**BaseGridOperation**(*session*, *uri*, *args=None*, *expect_format='zinc'*, *multi_grid=False*, *raw_response=False*, *retries=2*)
Bases: *pyhaystack.util.state.HaystackOperation*

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.

go()
Start the request.

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='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.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()

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()

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()

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()

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*, ***kwargs*)
Get an instance of a Project Haystack client.

pyhaystack.client.niagara module

NiagaraAX Client support

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

Bases: *pyhaystack.client.session.HaystackSession*

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.

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='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*)

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

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

about(*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(*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.

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.

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(*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.

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*

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.

pyhaystack.client.widesky module

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*, ***kwargs*)
Get an instance of a Project Haystack client.

4.1.2 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

4.2 Submodules

4.3 pyhaystack.exception module

This module allow a connection to a haystack server Feautures 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.

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 Feautures 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

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

p

A

B

C

D

E

F

L

M

N

O

P