skosprovider_atramhasis Documentation

Release 1.0.0

Flanders Heritage Agency

Aug 11, 2022

CONTENTS

CHAPTER

ONE

INTRODUCTION

This library offers an implementation of the `skosprovider.providers.VocabularyProvider` interface against an Atramhasis backend. This allows you to use an Atramhasis instance as a central SKOS repository that can easily be called by other applications. Either through using this Skosprovider, or by using the services provided by Atramhasis directly.

While this library works with Atramhasis, the services that Atramhasis exposes are in fact provided by pyramid_skosprovider. So, if you're not using Atramhasis, but do have a backend that implements the services offered by pyramid_skosprovider, you can use `skosprovider_atramhasis` as well.

Finally, if you have a backend that does not implement `pyramid_skosprovider`, but implements exactly the same services, you could use that as well.

1.1 Installation

To be able to use this library you need to have a modern version of Python installed. Currently we're supporting the last 3 versions of Python 3.

This easiest way to install this library is through **`pip`** or **`easy install`**:

```
$ pip install skosprovider_atramhasis
```

This will download and install `skosprovider_atramhasis` and a few libraries it depends on.

CHAPTER

TWO

USING THE PROVIDERS

For demonstration purposes of `AtramhasisProvider`, the following examples use the Flanders Heritage thesauri hosted at it's own thesaurus website. You can adapt the examples with your own instance of Atramhasis, by changing the *base_url* and *scheme_id*.

2.1 Using AtramhasisProvider

The `AtramhasisProvider` is a general provider for the Atramhasis vocabularies. It's use is identical to all other SKOSProviders. A base_url of the Atramhasis instance and a scheme_id are required to indicate the vocabulary to be used. Please consult *Supported thesauri* for a complete list.

For better perfomance in an environment with a `skosprovider.registry.Registry` we recommend manually providing the *uri* parameter as metadata. This can cut out the need for a call to the backend upon initialisation.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
'''
This script demonstrates using the AtramhasisProvider to get the concept of
'water tricks'.
'''

from skosprovider_atramhasis.providers import AtramhasisProvider

def main():
    # you can adapt this example by using the base_url of another
    # atramhasis-instance and provide an available scheme_id and
    # id within this conceptscheme
    provider = AtramhasisProvider(
        {'id': 'vioe-erfgoedtypes)'},
        base_url='https://thesaurus.onroerenderfgoed.be',
        scheme_id='ERFGOEDTYPES')
    id = 1524

    result = provider.get_by_id(id)


    print('Labels')
    print('------')
    for l in result.labels:
       print(l.language + ': ' + l.label + ' [' + l.type + ']')
```

(continues on next page)

(continued from previous page)

```
    print('Notes')
    print('-----')
    for n in result.notes:
        print(n.language + ': ' + n.note + ' [' + n.type + ']')


if __name__ == "__main__":
    main()
```

2.2 Finding concepts

See the *skosprovider_atramhasis.providers.AtramhasisProvider.find()* method for a detailed description of how this works.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
'''
This script demonstrates using the AtramhasisProvider to find concepts with
'kerk' in their label
'''

from skosprovider_atramhasis.providers import AtramhasisProvider

def main():
    # you can adapt this example by using the base_url of another
    # atramhasis-instance and provide an available scheme_id and
    # the keyword to search for
    provider = AtramhasisProvider(
        {'id': 'vioe-erfgoedtypes)'},
        base_url='https://thesaurus.onroerenderfgoed.be',
        scheme_id='ERFGOEDTYPES')

    keyword = 'kerk'

    results = provider.find({
        'label': keyword,
        'type': 'concept'
    })


    print('Results')
    print('------')
    for result in results:
        print(result)

if __name__ == "__main__":
    main()
```

2.3 Using expand()

The expand method return the id's of all the concepts that are narrower concepts of a certain concept or collection.

See the *skosprovider_atramhasis.providers.AtramhasisProvider.expand()* method for a detailed description of how this works.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
'''
This script demonstrates using the AtramhasisProvider to expand a concept
'''

from skosprovider_atramhasis.providers import AtramhasisProvider


def main():
    # you can adapt this example by using the base_url of another
    # atramhasis-instance and provide an available scheme_id and the id
    # within this concept scheme to expand
    provider = AtramhasisProvider(
        {'id': 'vioe-erfgoedtypes)'},
        base_url='https://thesaurus.onroerenderfgoed.be',
        scheme_id='ERFGOEDTYPES')
    id = 63 # Id for auxiliary buildings

    results = provider.expand(id)

    print('Results')
    print('------')
    for result in results:
        print(result)


if __name__ == "__main__":
    main()
```

2.4 Adding a cache

Skosprovider_atramhasis has to do a lot of HTTP requests. Depending on your use case, this might cause a lot of overhead. Therefor we've made it possible to add a *Dogpile cache <https://dogpilecache.sqlalchemy.org/en/latest/>_*. To configure the cache, simple add a *cache_config* key when instantiating the provider. This config will de passed to the Dogpile CacheRegion's configure_from_config method.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
'''
This script demonstrates using the AtramhasisProvider to get concepts with or
without cache.
'''
```

(continues on next page)

(continued from previous page)

```
from skosprovider_atramhasis.providers import AtramhasisProvider
import timeit

def main():
    # you can adapt this example by using the base_url of another
    # atramhasis-instance and provide an available scheme_id and
    # id within this conceptscheme
    id = 1524

    number = 50

    # No caching
    print('Fetching without cache')
    print('----------------------')

    provider = AtramhasisProvider(
        {'id': 'vioe-erfgoedtypes)'},
        base_url='https://thesaurus.onroerenderfgoed.be',
        scheme_id='ERFGOEDTYPES'
    )

    print('%d times: %.5f\n' % (number,
        timeit.timeit(lambda: provider.get_by_id(id),number=number)))


    # Only caching during the script
    print('Fetching with in memory cache')
    print('-----------------------------')

    provider = AtramhasisProvider(
        {'id': 'vioe-erfgoedtypes)'},
        base_url='https://thesaurus.onroerenderfgoed.be',
        scheme_id='ERFGOEDTYPES',
        cache_config={
            'cache.backend' : 'dogpile.cache.memory'
        }
    )

    print('%d times: %.5f\n' % (number,
        timeit.timeit(lambda: provider.get_by_id(id),number=number)))


    # Keep cache in between runs of the script
    # Value is considered valid for 1 day
    print('Fetching with file cache')
    print('------------------------')

    provider = AtramhasisProvider(
        {'id': 'vioe-erfgoedtypes)'},
        base_url='https://thesaurus.onroerenderfgoed.be',
        scheme_id='ERFGOEDTYPES',
        cache_config={
```

(continues on next page)

(continued from previous page)

```
            'cache.backend': 'dogpile.cache.dbm',
            'cache.expiration_time': 60 * 60 * 24,
            'cache.arguments.filename': 'erfgoedtypes.dbm'
        }
    )

    print('%d times: %.5f\n' % (number,
        timeit.timeit(lambda: provider.get_by_id(id),number=number)))

if __name__ == "__main__":
    main()
```

2.5 Supported thesauri

Currently the only known publically available backed is Flanders Heritage Ageny's own thesaurus website. It offers a range of thesauri dealing with cultural heritage in general and specifically in Flanders. Examples are a thesaurus of heritage types, a thesaurus of cultures and periods, a thesaurus of heritage even types, …

CHAPTER

THREE

DEVELOPMENT

Skosprovider_atramhasis is being developed by the Flanders Heritage Agency.

Since we place a lot of importance on code quality, we expect to have a good amount of code coverage present and run frequent unit tests. All commits and pull requests will be tested with Travis-ci. Code coverage is being monitored with Coveralls.

Locally you can run unit tests by using pytest or tox. Running pytest manually is good for running a distinct set of unit tests. For a full test run, tox is preferred since this can run the unit tests against multiple versions of python.

```
# Setup for development
$ python setup.py develop
# Run unit tests for all environments
$ tox
# No coverage
$ py.test
# Coverage
$ py.test --cov skosprovider_atramhasis --cov-report term-missing tests
# Only run a subset of the tests
$ py.test skosprovider_atramhasis/tests/test_providers.py
```

Please provide new unit tests to maintain 100% coverage. If you send us a pull request and this build doesn't function, please correct the issue at hand or let us know why it's not working.

CHAPTER

FOUR

API DOCUMENTATION

4.1 Providers module

This module implements a `skosprovider.providers.VocabularyProvider` for Atramhasis

class skosprovider_atramhasis.providers.**AtramhasisProvider**(*metadata*, ***kwargs*)

A provider that can work with the Atramhasis REST services

base_url = None

Base URL of an Atramhasis instance.

expand(*id_*)

Expand a concept or collection to all it's narrower concepts.

This method should recurse and also return narrower concepts of narrower concepts.

If the id passed belongs to a `skosprovider.skos.Concept`, the id of the concept itself should be include in the return value.

If the id passed belongs to a `skosprovider.skos.Collection`, the id of the collection itself must not be present in the return value In this case the return value includes all the member concepts and their narrower concepts.

Parameters **id** – A concept or collection id.

Return type A list of id's or *False* if the concept or collection doesn't exist.

find(*query*, ***kwargs*)

Find concepts that match a certain query.

Currently query is expected to be a dict, so that complex queries can be passed. You can use this dict to search for concepts or collections with a certain label, with a certain type and for concepts that belong to a certain collection.

```
# Find anything that has a label of church.
provider.find({'label': 'church'})

# Find all concepts that are a part of collection 5.
provider.find({'type': 'concept', 'collection': {'id': 5}})

# Find all concepts, collections or children of these
# that belong to collection 5.
provider.find({'collection': {'id': 5, 'depth': 'all'}})

# Find anything that has a label of church.
```

(continues on next page)

(continued from previous page)

```
# Preferentially display a label in Dutch.
provider.find({'label': 'church'}, language='nl')

# Find anything that has a match with an external concept
# Preferentially display a label in Dutch.
provider.find({
    'matches': {
        'uri': 'http://id.python.org/different/types/of/trees/nr/1/the/larch'
    }}, language='nl')

# Find anything that has a label of lariks with a close match to an external␣
↪concept
# Preferentially display a label in Dutch.
provider.find({
    'label': 'lariks',
    'matches': {
        'type': 'close',
        'uri': 'http://id.python.org/different/types/of/trees/nr/1/the/larch'
    }}, language='nl')
```

Parameters

- **query** – A dict that can be used to express a query. The following keys are permitted:
  - *label*: Search for something with this label value. An empty label is equal to searching for all concepts.
  - *type*: Limit the search to certain SKOS elements. If not present or *None*, *all* is assumed:
    - *concept*: Only return `skosprovider.skos.Concept` instances.
    - *collection*: Only return `skosprovider.skos.Collection` instances.
    - *all*: Return both `skosprovider.skos.Concept` and `skosprovider.skos.Collection` instances.
  - *collection*: Search only for concepts belonging to a certain collection. This argument should be a dict with two keys:
    - *id*: The id of a collection. Required.
    - *depth*: Can be *members* or *all*. Optional. If not present, *members* is assumed, meaning only concepts or collections that are a direct member of the collection should be considered. When set to *all*, this method should return concepts and collections that are a member of the collection or are a narrower concept of a member of the collection.
  - ***matches*: Search only for concepts having a match to a certain** external concept. Since collections can't have matches, this automatically excludes collections. The argument with two keys:
    - *uri*: The uri of the concept to match. Required.
    - *type*: The type of match, see `matchtypes` for the full list of options.
- **language** (*string*) – Optional. If present, it should be a language-tag. This language-tag is passed on to the underlying providers and used when selecting the label to display for each concept.

- **sort** (*string*) – Optional. If present, it should either be *id*, *label* or *sortlabel*. The *sortlabel* option means the providers should take into account any *sortLabel* if present, if not it will fallback to a regular label to sort on.
- **sort_order** (*string*) – Optional. What order to sort in: *asc* or *desc*. Defaults to *asc*

Returns

A `lst` of concepts and collections. Each of these is a dict with the following keys:

- id: id within the conceptscheme
- uri: *URI* of the concept or collection
- type: concept or collection
- label: A label to represent the concept or collection. It is determined by looking at the *language* parameter, the default language of the provider and finally falls back to *en*.

get_all(***kwargs*)

Returns all concepts and collections in this provider.

Parameters

- **language** (*string*) – Optional. If present, it should be a language-tag. This language-tag is passed on to the underlying providers and used when selecting the label to display for each concept.
- **sort** (*string*) – Optional. If present, it should either be *id*, *label* or *sortlabel*. The *sortlabel* option means the providers should take into account any *sortLabel* if present, if not it will fallback to a regular label to sort on.
- **sort_order** (*string*) – Optional. What order to sort in: *asc* or *desc*. Defaults to *asc*

Returns

A `lst` of concepts and collections. Each of these is a dict with the following keys:

- id: id within the conceptscheme
- uri: *URI* of the concept or collection
- type: concept or collection
- label: A label to represent the concept or collection. It is determined by looking at the *language* parameter, the default language of the provider and finally falls back to *en*.

get_by_id(*id_*)

Get all information on a concept or collection, based on id.

Providers should assume that all id's passed are strings. If a provider knows that internally it uses numeric identifiers, it's up to the provider to do the typecasting. Generally, this should not be done by changing the id's themselves (eg. from int to str), but by doing the id comparisons in a type agnostic way.

Since this method could be used to find both concepts and collections, it's assumed that there are no id collisions between concepts and collections.

Return type `skosprovider.skos.Concept` or `skosprovider.skos.Collection` or *False* if the concept or collection is unknown to the provider.

get_by_uri(*uri*)

Get all information on a concept or collection, based on a *URI*.

Return type `skosprovider.skos.Concept` or `skosprovider.skos.Collection` or *False* if the concept or collection is unknown to the provider.

`get_children_display`(*id_*, ***kwargs*)

Return a list of concepts or collections that should be displayed under this concept or collection.

Parameters

- **`language`** (*`string`*) – Optional. If present, it should be a language-tag. This language-tag is passed on to the underlying providers and used when selecting the label to display for each concept.
- **`sort`** (*`string`*) – Optional. If present, it should either be *id*, *label* or *sortlabel*. The *sortlabel* option means the providers should take into account any *sortLabel* if present, if not it will fallback to a regular label to sort on.
- **`sort_order`** (*`string`*) – Optional. What order to sort in: *asc* or *desc*. Defaults to *asc*
- **`id`** (*`str`*) – A concept or collection id.

Returns

A `lst` of concepts and collections. Each of these is a dict with the following keys:

- id: id within the conceptscheme
- uri: *URI* of the concept or collection
- type: concept or collection
- label: A label to represent the concept or collection. It is determined by looking at the *language* parameter, the default language of the provider and finally falls back to *en*.

`get_top_concepts`(***kwargs*)

Returns all top-level concepts in this provider.

Top-level concepts are concepts that have no broader concepts themselves. They might have narrower concepts, but this is not mandatory.

Parameters

- **`language`** (*`string`*) – Optional. If present, it should be a language-tag. This language-tag is passed on to the underlying providers and used when selecting the label to display for each concept.
- **`sort`** (*`string`*) – Optional. If present, it should either be *id*, *label* or *sortlabel*. The *sortlabel* option means the providers should take into account any *sortLabel* if present, if not it will fallback to a regular label to sort on.
- **`sort_order`** (*`string`*) – Optional. What order to sort in: *asc* or *desc*. Defaults to *asc*

Returns

A `lst` of concepts, NOT collections. Each of these is a dict with the following keys:

- id: id within the conceptscheme
- uri: *URI* of the concept or collection
- type: concept or collection

- label: A label to represent the concept or collection. It is determined by looking at the *language* parameter, the default language of the provider and finally falls back to *en*.

get_top_display(***kwargs*)

Returns all concepts or collections that form the top-level of a display hierarchy.

As opposed to the `get_top_concepts()`, this method can possibly return both concepts and collections.

Parameters

- **language** (*string*) – Optional. If present, it should be a language-tag. This language-tag is passed on to the underlying providers and used when selecting the label to display for each concept.
- **sort** (*string*) – Optional. If present, it should either be *id*, *label* or *sortlabel*. The *sortlabel* option means the providers should take into account any *sortLabel* if present, if not it will fallback to a regular label to sort on.
- **sort_order** (*string*) – Optional. What order to sort in: *asc* or *desc*. Defaults to *asc*

Returns

A `lst` of concepts and collections. Each of these is a dict with the following keys:

- id: id within the conceptscheme
- uri: *URI* of the concept or collection
- type: concept or collection
- label: A label to represent the concept or collection. It is determined by looking at the *language* parameter, the default language of the provider and finally falls back to *en*.

scheme_id = None

Identifier of the ConceptScheme this provider is managing.

session = None

The `requests.Session` being used to make HTTP requests.

4.2 Utils module

Utility functions for `skosprovider_atramhasis`.

`skosprovider_atramhasis.utils.`**dict_to_thing**(*data_dict*)

Transform a dict into a `skosprovider.skos.Concept` or `skosprovider.skos.Collection` .

If the argument passed is already a `skosprovider.skos.Concept` or `skosprovider.skos.Collection`, this method just returns the argument.

`skosprovider_atramhasis.utils.`**text_**(*s*, *encoding='latin-1'*, *errors='strict'*)

If `s` is an instance of `binary_type`, return `s.decode(encoding, errors)`, otherwise return `s`

4.3 Cache_utils module

Utility functions for chaching in `skosprovider_atramhasis`.

CHAPTER

FIVE

HISTORY

5.1 next version (2022-??-??)

- Update documentation (#131)

5.2 1.0.0 (2021-12-17)

- Upgrade requirements. Including skosprovider 1.1.0 (#116)
- Retry GET calls within _request method 5 times before giving up (#120)

5.3 0.4.1 (2021-02-10)

- The variabele scheme_id in AtramhasisProvider is used before the variabele is defined. (#106)

5.4 0.4.0 (2021-02-09)

- Upgrade libraries, fe dogpile.cache (#95)
- Drop support for python 2 (#96)
- Only build property concept_scheme once (#97)
- Lazy access concept_scheme.uri (#98)

5.5 0.3.1 (2020-02-19)

- Fix a bug in *dict_to_thing* that resulted in the provider exposing objects instead of id's in relations. (#80)

5.6 0.3.0 (2020-01-24)

- Update to Skosprovider 0.7.0.
- Add caching support to the provider. (#75)
- Don't load concepscheme at startup, since this prevents the provider from instantiating when an Atramhasis instance is down. (#66)
- This is the last version to support Python 2.7. Updated Python 3 support to 3.6, 3.7 and 3.8.

5.7 0.2.1 (2019-11-13)

- Update supported Python version (#61)
- Add a long_description_content_type (#55)
- Correct handling of get_by_uri (#69)

5.8 0.2.0 (2017-08-24)

- Update to skosprovider 0.6.1
- Allow Notes with HTML (#17)
- Add languages to Conceptscheme (#18)
- Add sources to Conceptscheme, Collection and Concept (#19)
- Add sorting to finders (#20)

5.9 0.1.1 (2016-03-23)

- Official support for Py35
- Allow configuring a requests session. (#14)

5.10 0.1.0 (2015-03-19)

- Initial version
- Compatible with SkosProvider 0.5.x.

CHAPTER

SIX

GLOSSARY

Atramhasis an online SKOS editor. It allows a user to create and edit an online thesaurus or vocabulary adhering to the SKOS specification through a simple web interface. This allows any user with access to a web browser to consult the thesauri and if so wanted, to edit them.

SKOS Simple Knowledge Organization System. An general specification for Knowledge Organisation Systems (thesauri, word lists, authority files, ...) that is commonly serialised as RDF.

URI A *Uniform Resource Identifier*.

URN A URN is a specific form of a *URI*.

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX