Pyramid Skosprovider Documentation

Release 0.9.2

Koen Van Daele

Jan 21, 2021

CONTENTS

This library helps to integrate skosprovider in a Pyramid application.

CHAPTER

ONE

SUPPORT

If you have questions regarding Pyramid_Skosprovider, feel free to contact us. Any bugs you find or feature requests you have, you can add to our issue tracker. If you're unsure if something is a bug or intentional, or you just want to have a chat about this library or SKOS in general, feel free to join the Atramhasis discussion forum. While these are separate software projects, they are being run by the same people and they integrate rather tightly.

1.1 Installation

To install *pyramid_skosprovider*, use pip

```
pip install pyramid_skosprovider
```

To activate *pyramid_skosprovider*, you need to include it and configure your `skosprovider.registry.Registry`. Older versions, before *0.9.0*, attached the *skosregistry* to the Pyramid application registry. Starting with *0.9.0* it's possible and recommended to attach the skos registry to a request. The old way is still allowed for backward compatibility, but should only be used with local providers that store all their data in memory, such as a `skosprovider.providers.DictionaryProvider`.

To maintain the old way and enable the new one, two new settings were added in *pyramid_skosprovider 0.9.0*:

skosprovider.skosregistry_location deteremines where your registry lives in the application. Currently two options are supported: *registry* or *request*. The first attaches the `skosprovider.registry.Registry` to the Pyramid application registry as the Pyramid application is started. The second option attaches the skos registry to the Pyramid request. This ensures every request has it's own registry.

skosprovider.skosregistry_factory allows you to specify a factory function for instantiating the `skosprovider.registry.Registry`. This function will receive the Pyramid request as a first argument if you are using a registry attached to a request. If you do not specify a factory, an empty skos registry will be created and used.

Please be aware that attaching a registry to the Pyramid application registry was the only option before *0.9.0*. It is still supported because it makes sense for a registry that only contains providers that load all their data in memory on initialisation. Providers that require a database connection should always be attached to a request.

Supposing you want to attach your registry to requests, you would write some configuration. A possible configuration for a *myskos* application would be:

```
skosprovider.skosregistry_location: request
skosprovider.skosregistry_factory: myskos.skos.build_registry
```

To actually use *pyramid_skosprovider*, you still need to include it in your pyramid application and write the factory function. In your Pyramid startup function:

```
config = Configurator()
config.include('pyramid_skosprovider')
```

This will add some views and configuration. Every request will now contain a *skos_registry* attribute. The first time this attribute is accessed, the SKOS registy will be build for you, using the specified factory function.

Your *myskos.skos* python module should contain this factory function. In our example config, we called it *build_registry*. This is a function that receives a Pyramid request, creates the skos registry and returns it:

```
from skosprovider.registry import Registry
from skosprovider.providers import DictionaryProvider

def build_registry(request):

    r = Registry(
        instance_scope='threaded_thread'
    )

    dictprovider = DictionaryProvider(
        {
            'id': 'TREES',
            'default_language': 'nl',
            'subject': ['biology'],
            'dataset': {
                'uri': 'http://id.trees.org/dataset'
            }
        },
        [],
        uri_generator=UriPatternGenerator('http://id.trees.org/types/%s'),
        concept_scheme=ConceptScheme('http://id.trees.org')
    )
    r.register_provider(dictprovider)

    return r
```

This is a very simple example. A typical real-life application would have several providers. Some of them might be DictionaryProviders, others might reaf from rdf files and still others might read from a SQL Databases. If you're using the *skosprovider_sqlalchemy* provider, you would attach your database session maker to the request and then pass it on to the SQLAlchemy provider in your factory function.

If you want to attach the SKOS registry to the Pyramid registry, and not the request, you would have the following config:

```
skosprovider.skosregistry_location: registry
skosprovider.skosregistry_factory: myskos.skos.build_registry
```

The *build_registry* factory would be very similar, but it does not have acces to the request. This makes it a bad fit for threaded web-servers and leads to bugs. But something like a `skosprovider.providers.DictionaryProvider` wpuld be fine. The factory function is almost identical, but we would also set the Registry instance_scope to *threaded_global*. This can alert providers that register with the registry that they might not be compatible. `

```
from skosprovider.registry import Registry
from skosprovider.providers import DictionaryProvider

def build_registry():
```

(continues on next page)

(continued from previous page)

```
    r = Registry(
        instance_scope='threaded_global'
    )

    dictprovider = DictionaryProvider(
        {
            'id': 'TREES',
            'default_language': 'nl',
            'subject': ['biology'],
            'dataset': {
                'uri': 'http://id.trees.org/dataset'
            }
        },
        [],
        uri_generator=UriPatternGenerator('http://id.trees.org/types/%s'),
        concept_scheme=ConceptScheme('http://id.trees.org')
    )
    r.register_provider(dictprovider)

    return r
```

1.2 Usage

To get a `skosprovider.registry.Registry` instance that was configured globally, call *`pyramid_skosprovider.get_skos_registry()`* with the current application registry.

Eg. in a view:

```
from pyramid_skosprovider import get_skos_registry

def my_view(request):
    skos = get_skos_registry(request.registry)
    providers = skos.get_providers()
    # ...
```

Since this only works for globally configured registries, it's not the preferred way. Alternatively you can get the registry as an attribute of a pyramid request:

```
def my_view(request):
    skos = request.skos_registry
    providers = skos.get_providers()
    # ...
```

For a real-world example of an integration of *pyramid_skosprovider* in a *Pyramid* application, have a look at Atramhasis, a SKOS vocabulary editor partially built upon this library.

1.3 Service Documentation

This library takes your skosproviders and makes them available as REST services. The pyramid_skosprovider serves JSON as a REST service so it can be used easily inside a AJAX webbrowser call or by an external program.

The following API can be used by clients:

GET /uris

Find more information on a certain URI. This can map to eiter a concept, collection or conceptscheme that is known by the current SKOS registry.

Example request:

```
GET /uris?uri=urn:x-skosprovider:trees HTTP/1.1
Host: localhost:6543
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Content-Type:  application/json; charset=UTF-8

{
    "id": "TREES",
    "uri": "urn:x-skosprovider:trees",
    "type": "concept_scheme"
}
```

Example request:

```
GET /uris/?uri=http://python.com/trees/larch HTTP/1.1
Host: localhost:6543
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Content-Type:  application/json; charset=UTF-8

{
    "id": "1",
    "uri": "http://python.com/trees/larch",
    "type": "concept",
    "concept_scheme": {
        "id": "TREES",
        "uri": "urn:x-skosprovider:trees"
    }
}
```

Query Parameters

- **uri** – The URI to search for.

Status Codes

- 200 OK – The URI maps to something known by pyramid_skosprovider, either a conceptscheme, a concept or collection.
- 404 Not Found – The URI can't be found by pyramid_skosprovider.

GET /c

Search for concepts or collections, no matter what scheme they're a part of.

Although it is possible to search a single conceptscheme with just this endpoint, for performance reasons it is advised to use *GET /conceptschemes/{scheme_id}/c*.

Example request:

```
GET /c HTTP/1.1
Host: localhost:6543
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Content-Range:  items 0-2/232
Content-Type:  application/json; charset=UTF-8

[
    {
        "id": "1",
        "uri": "urn:x-skosprovider:TREES:1",
        "type": "concept",
        "label": "De Lariks"
    }, {
        "id": "2",
        "uri": "urn:x-skosprovider:TREES:2",
        "type": "concept",
        "label": "De Paardekastanje"
    }, {
        "id": 3,
        "uri": "urn:x-skosprovider:TREES:3",
        "type": "collection",
        "label": "Bomen per soort"
    }
]
```

Example request:

```
GET /c?type=concept&providers.subject=external&sort=uri HTTP/1.1
Host: localhost:6543
Accept: application/json
```

Query Parameters

- **type** – Define if you want to show concepts or collections. Leave blank to show both.
- **mode** – Allows for special processing mode for dijitFilteringSelect. Makes it possible to use wildcards in the label parameter.
- **label** – Shows all concepts and collections that have this search string in one of their labels.
- **language** – Returns the label with the corresponding language-tag if present. If the language is not present for this concept/collection, it falls back to 1) the default language of the provider. 2) 'en' 3) any label. Eg. `?language=nl` to show the dutch labels of the concepts/collections.
- **sort** – Define if you want to sort the results by a given field. Otherwise items are returned in an indeterminate order. Prefix with '+' to sort ascending, '-' to sort descending. eg.

`?sort=-label` to sort all results descending by label.

- **match** – A URI for an external concept. Searches if any of the providers have a matching concept.
- **match_type** – A type of match: exact, close, related, broader, narrower. Only used if a match URI is present as well.
- **providers.ids** – A comma separated list of concept scheme id's. The query will only be passed to the providers with these id's. eg. `?providers.ids=TREES, PARROTS` will only list concepts from these two providers.
- **providers.subject** – A subject can be registered with a skosprovider in the registry. Adding this search parameter means that the query will only be passed on to providers that have been tagged with this subject. Eg. `?providers.subject=external` to only query the providers that have been marked with the subject *external*.

Request Headers

- Range – Can be used to request a certain set of results. eg. `items=0-24` requests the first 25 results.

Response Headers

- Content-Range – Tells the client what set of results is being returned eg. `items=0-24/306` means the first 25 out of 306 results are being returned.

Status Codes

- 200 OK – The concepts in this conceptscheme were found.

GET /conceptschemes
Get all registered conceptschemes.

Example request:

```
GET /conceptschemes HTTP/1.1
Host: localhost:6543
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Content-Type:  application/json; charset=UTF-8
Date:  Mon, 14 Apr 2014 14:42:34 GMT

[
    {
        "id": "TREES",
        "uri": "urn:x-skosprovider:trees",
        "label": "Different types of trees."
    }
]
```

Status Codes

- 200 OK – The list of conceptschemes was found.

GET /conceptschemes/{scheme_id}
Get information about a concept scheme.

Example request:

```
GET /conceptschemes/TREES HTTP/1.1
Host: localhost:6543
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Content-Length:  15
Content-Type:  application/json; charset=UTF-8
Date:  Mon, 14 Apr 2014 14:45:37 GMT
Server:  waitress

{
    "id": "TREES",
    "uri": "urn:x-skosprovider:trees",
    "label": "Different types of trees.",
    "labels": [
        {"type": "prefLabel", "language": "en", "label": "Different types of␣
↪trees."},
        {"type": "prefLabel", "language": "nl", "label": "Verschillende soorten␣
↪bomen."}
    ]
}
```

Example request:

-.. sourcecode:: http

GET /conceptschemes/PLANTS HTTP/1.1 Host: localhost:6543 Accept: application/json

Example response:

```
HTTP/1.1 404 Not Found
Content-Length:  775
Content-Type:  text/html; charset=UTF-8
Date:  Tue, 15 Apr 2014 20:32:52 GMT
Server:  waitress
```

Status Codes

- 200 OK – The conceptscheme was found.
- 404 Not Found – The conceptscheme was not found.

GET /conceptschemes/{scheme_id}/topconcepts

Get all top concepts in a certain conceptscheme. These are all the concepts in the conceptscheme that have no broader concept.

Example request:

```
GET /conceptschemes/TREES/topconcepts HTTP/1.1
Host: localhost:6543
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Content-Type:  application/json; charset=UTF-8
```

(continues on next page)

(continued from previous page)

```
Date:  Mon, 14 Apr 2014 14:47:33 GMT
Server:  waitress

[
    {
        "id": "1",
        "uri": "urn:x-skosprovider:TREES:1",
        "type": "concept",
        "label": "De Lariks"
    }, {
        "id": "2",
        "uri": "urn:x-skosprovider:TREES:2",
        "type": "concept",
        "label": "De Paardekastanje"
    }
]
```

Query Parameters

- **language** – Returns the label with the corresponding language-tag if present. If the language is not present for this concept/collection, it falls back to 1) the default language of the provider. 2) 'en' 3) any label. Eg. `?language=nl` to show the dutch labels of the concepts/collections.

Status Codes

- 200 OK – The topconcepts in this conceptscheme were found.
- 404 Not Found – The conceptscheme was not found.

GET /conceptschemes/{scheme_id}/displaytop

Get the top of a display hierarchy. Depending on the underlying provider this will be a list of Concepts and Collections.

Example request:

```
GET /conceptschemes/TREES/displaytop HTTP/1.1
Host: localhost:6543
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Content-Type:  application/json; charset=UTF-8
Date:  Mon, 14 Apr 2014 14:47:33 GMT
Server:  waitress

[
    {
        "id": "1",
        "uri": "urn:x-skosprovider:TREES:1",
        "type": "concept",
        "label": "De Lariks"
    }, {
        "id": "2",
        "uri": "urn:x-skosprovider:TREES:2",
        "type": "concept",
```

(continues on next page)

(continued from previous page)

```
        "label": "De Paardekastanje"
    }
]
```

Query Parameters

- **language** – Returns the label with the corresponding language-tag if present. If the language is not present for this concept/collection, it falls back to 1) the default language of the provider. 2) 'en' 3) any label. Eg. `?language=nl` to show the dutch labels of the concepts/collections.

Status Codes

- 200 OK – The concepts and collections were found.
- 404 Not Found – The conceptscheme was not found.

GET /conceptschemes/{scheme_id}/c

Search for concepts or collections in a scheme.

Example request:

```
GET /conceptschemes/TREES/c HTTP/1.1
Host: localhost:6543
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Content-Length:  117
Content-Range:  items 0-2/3
Content-Type:  application/json; charset=UTF-8
Date:  Mon, 14 Apr 2014 14:47:33 GMT
Server:  waitress

[
    {
        "id": "1",
        "uri": "urn:x-skosprovider:TREES:1",
        "type": "concept",
        "label": "De Lariks"
    }, {
        "id": "2",
        "uri": "urn:x-skosprovider:TREES:2",
        "type": "concept",
        "label": "De Paardekastanje"
    }, {
        "id": 3,
        "uri": "urn:x-skosprovider:TREES:3",
        "type": "collection",
        "label": "Bomen per soort"
    }
]
```

Example request:

```
GET /conceptschemes/PLANTS/c HTTP/1.1
Host: localhost:6543
Accept: application/json
```

Example response:

```
HTTP/1.1 404 Not Found
Content-Length:  775
Content-Type:  text/html; charset=UTF-8
Date:  Tue, 15 Apr 2014 20:32:52 GMT
Server:  waitress
```

Query Parameters

- **type** – Define if you want to show concepts or collections. Leave blank to show both.
- **mode** – Allows for special processing mode for dijitFilteringSelect. Makes it possible to use wildcards in the label parameter.
- **label** – Shows all concepts and collections that have this search string in one of their labels.
- **collection** – Get information about the content of a collection. Expects to be passed an id of a collection in this scheme. Will restrict the search to concepts or collections that are a member of this collection or a narrower concept of a member.
- **match** – A URI for an external concept. Searches if any of the providers have a matching concept.
- **match_type** – A type of match: exact, close, related, broader, narrower. Only used if a match URI is present as well.
- **language** – Returns the label with the corresponding language-tag if present. If the language is not present for this concept/collection, it falls back to 1) the default language of the provider. 2) 'en' 3) any label. Eg. `?language=nl` to show the dutch labels of the concepts/collections.
- **sort** – Define if you want to sort the results by a given field. Otherwise items are returned in an indeterminate order. Prefix with '+' to sort ascending, '-' to sort descending. eg. `?sort=-label` to sort all results descending by label.

Request Headers

- Range – Can be used to request a certain set of results. eg. `items=0-24` requests the first 25 results.

Response Headers

- Content-Range – Tells the client was set of results is being returned eg. `items=0-24/306` means the first 25 out of 306 results are being returned.

Status Codes

- 200 OK – The concepts in this conceptscheme were found.
- 404 Not Found – The conceptscheme was not found.

GET /conceptschemes/{scheme_id}/c/{c_id}
Get information about a concept or collection.

Example request:

```
GET /conceptschemes/TREES/c/1 HTTP/1.1
Host: localhost:6543
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Date: Mon, 14 Apr 2014 14:49:27 GMT
Server: waitress

{
    "broader": [],
    "narrower": [],
    "notes": [
        {"note": "A type of tree.", "type": "definition", "language": "en"}
    ],
    "labels": [
        {"type": "prefLabel", "language": "en", "label": "The Larch"},
        {"type": "prefLabel", "language": "nl", "label": "De Lariks"}
    ],
    "type": "concept",
    "id": "1",
    "uri": "urn:x-skosprovider:TREES:1",
    "related": [],
    "label": "The Larch",
    "matches": {
        "close": [
            "http://id.python.org/different/types/of/trees/nr/1/the/larch"
        ]
    },
    "concept_scheme": {
        "uri": "urn:x-foo:bar"
    }
}
```

Example request:

```
GET /conceptschemes/TREES/c/4 HTTP/1.1
Host: localhost:6543
Accept: application/json
```

Example response:

```
HTTP/1.1 404 Not Found
Content-Length:  775
Content-Type:  text/html; charset=UTF-8
Date:  Tue, 15 Apr 2014 20:06:12 GMT
Server:  waitress
```

Status Codes

- 200 OK – The concept was found in the conceptscheme.
- 404 Not Found – The concept was not found in the conceptscheme or the conceptscheme was not found.

GET /conceptschemes/{scheme_id}/c/{c_id}/displaychildren
Get a list of Collections and Concepts that should be displayed as children of this Concept or Collection.

Example request:

```
GET /conceptschemes/TREES/c/3/displaychildren HTTP/1.1
Host: localhost:6543
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Content-Type:  application/json; charset=UTF-8
Date:  Mon, 14 Apr 2014 14:49:27 GMT
Server:  waitress

[
    {
        "id": "1",
        "uri": "urn:x-skosprovider:TREES:1",
        "type": "concept",
        "label": "De Lariks"
    }, {
        "id": "2",
        "uri": "urn:x-skosprovider:TREES:2",
        "type": "concept",
        "label": "De Paardekastanje"
    }
]
```

Query Parameters

- **language** – Returns the label with the corresponding language-tag if present. If the language is not present for this concept/collection, it falls back to 1) the default language of the provider. 2) 'en' 3) any label. Eg. `?language=nl` to show the dutch labels of the concepts/collections.

Status Codes

- 200 OK – The concept was found in the conceptscheme.
- 404 Not Found – The concept was not found in the conceptscheme or the conceptscheme was not found.

GET /conceptschemes/{scheme_id}/c/{c_id}/expand
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.

Returns A list of id's or `HTTPNotFound` if the concept or collection doesn't exist.

Example request:

```
GET /conceptschemes/TREES/c/3/expand HTTP/1.1
Host: localhost:6543
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Content-Type:  application/json; charset=UTF-8
Date:  Mon, 14 Apr 2014 14:49:27 GMT
Server:  waitress

[1 , 2]
```

Status Codes

- 200 OK – The concept/collection was found in the conceptscheme.
- 404 Not Found – The concept/collection was not found in the conceptscheme or the conceptscheme was not found.

1.4 API Documentation

1.4.1 General

pyramid_skosprovider.**get_skos_registry**(*registry*)

Get the skosprovider.registry.Registry attached to this pyramid application.

Parameters **registry** – A Pyramid registry, request or config.

Return type skosprovider.registry.Registry

1.4.2 Utils

This module contains a few utility functions.

pyramid_skosprovider.utils.**parse_range_header**(*range*)

Parse a range header as used by the dojo Json Rest store.

Parameters **range** (*str*) – The content of the range header to be parsed. eg. *items=0-9*

Returns A dict with keys start, finish and number or *False* if the range is invalid.

1.4.3 Renderers

This module contains function for rendering SKOS objects to JSON.

pyramid_skosprovider.renderers.**collection_adapter**(*obj*, *request*)

Adapter for rendering a skosprovider.skos.Collection to json.

Parameters **obj** (*skosprovider.skos.Collection*) – The collection to be rendered.

Return type dict

pyramid_skosprovider.renderers.**collection_ld_adapter**(*obj*, *request*)

Adapter for rendering a skosprovider.skos.Concept to jsonld.

Parameters **obj** (*skosprovider.skos.Concept*) – The concept to be rendered.

Return type dict

pyramid_skosprovider.renderers.**concept_adapter**(*obj*, *request*)
Adapter for rendering a skosprovider.skos.Concept to json.

Parameters **obj** (*skosprovider.skos.Concept*) – The concept to be rendered.

Return type dict

pyramid_skosprovider.renderers.**concept_ld_adapter**(*obj*, *request*)
Adapter for rendering a skosprovider.skos.Concept to jsonld.

Parameters **obj** (*skosprovider.skos.Concept*) – The concept to be rendered.

Return type dict

pyramid_skosprovider.renderers.**conceptscheme_ld_adapter**(*obj*, *request*)
Adapter for rendering a skosprovider.skos.ConceptScheme to jsonld.

Parameters **obj** (*skosprovider.skos.ConceptScheme*) – The conceptscheme to be rendered.

Return type dict

pyramid_skosprovider.renderers.**label_adapter**(*obj*, *request*)
Adapter for rendering a skosprovider.skos.Label to json.

Parameters **obj** (*skosprovider.skos.Label*) – The label to be rendered.

Return type dict

pyramid_skosprovider.renderers.**note_adapter**(*obj*, *request*)
Adapter for rendering a skosprovider.skos.Note to json.

Parameters **obj** (*skosprovider.skos.Note*) – The note to be rendered.

Return type dict

pyramid_skosprovider.renderers.**source_adapter**(*obj*, *request*)
Adapter for rendering a skosprovider.skos.Source to json.

Parameters **obj** (*skosprovider.skos.Source*) – The source to be rendered.

Return type dict

1.4.4 Views

This module contains the pyramid views that expose services.

class pyramid_skosprovider.views.**ProviderView**(*request*)
A set of views that expose information from a certain provider.

1.5 History

1.5.1 0.9.2 (2021-01-21)

- Fix an issue with case insensitive search containing a wildcard. (#82)

1.5.2 0.9.1 (2020-10-19)

- Add download links to JSON-LD version of concept and conceptscheme to improve user experience. (#78)
- Remove pyup. (#79)
- Update soms development dependencies.

1.5.3 0.9.0 (2020-08-06)

- Support running a registry per request, as opposed to per application as before. (#44)
- Add the *infer_concept_relations* attribute to the collection renderer. (#73)
- Add JSON-LD output to the REST service. (#63)
- Add support for match and match_type search parameters to search for concepts that match a certain URI and optionally have a certain type. (#68)
- Drop support for Python 3.4, add support for 3.7 and 3.8. This is the last version that will support Python 2. (#66)
- Remove the JSON renderers from the utils module.

1.5.4 0.8.0 (2017-07-12)

- Return an HTTP 404 response when a conceptscheme could not be found. (#24)
- Add universal wheel distribution. (#23)
- Add support for sorting on a SortLabel. This means a client can now ask to sort the results either on *id*, *label* or *sortlabel*. See the *skosprovider* docs for more on the *sortlabel*. This basically allows for arbitrary sorting per language so it's possible to eg. sort Historical periods chronologically. (#26) [cahytinne]

1.5.5 0.7.0 (2016-08-11)

- Sort case insensitive when sorting by label. This is a BC break, although to most users it might actually be a bug fix. (#16) [TalissaJoly]
- Add the markup attribute to Note json representations. This is a new addition to skosprovider 0.6.0 that allows marking that a note contains some markup (currently only HTML).
- Looking for a certain URI is now done with a query parameter in stead of in the path of a resource. So, */uris/urn:x-skosprovider:trees* should now be called as */uris?uri=urn:x-skosprovider:trees*. The old way is deprecated. It will still function under version *0.7.0*, but will be removed in a future version. (#19)
- Add support for the sources attribute, a new feature in skosprovider 0.6.0
- Add support for languages to Conceptschemes, a new feature in skosprovider 0.6.0 that allows detailing what languages a conceptscheme uses.

- Move JSON renderers to their own file and fix some language handling issues. (#22)
- Add support for Python 3.5

1.5.6 0.6.0 (2015-03-02)

- Allow the client to specify in which language labels should preferentially be returned. This can be chosen by adding a `language` parameter to certain query strings. If not present, pyramid_skosprovider falls back on pyramid's locale negotiation. (#10) (#14) [dieuska]
- Expose a provider's expand method. This returns the narrower transitive closure for a certain concept or collection. (#11) [dieuska]
- Some documentation updates.

1.5.7 0.5.0 (2014-12-19)

- Conceptschemes expose information on the subject they're tagged with. [BartSaelen]
- A new search endpoint for searching across conceptschemes was added. Search syntax is the same as for searching within a single scheme, but the collection parameter is not accepted. Two extra parameters were added for limiting the search to a subset of available conceptschemes. (#8)
- A new endpoint for looking up a certain URI was added. This endpoint does not redirect to an external URI, but lets a client know where more information about this URI can be found (eg. in which conceptscheme a concept lives). (#7)

1.5.8 0.4.0 (2014-10-23)

- Compatibility with skosprovider 0.4.0
- Drop support for Python 2.6 and Python 3.2.
- Expose notes on collections.
- Expose matches on concepts (collections don't have matches).
- Expose subordinate_arrays on concepts and superordinates on collections.
- Integrate concept scheme information. Concepts and collections passed through the service now contain the uri of the concept scheme they belong to. The concept scheme endpoint now also exposes information like a uri, a list of labels and notes.

1.5.9 0.3.0 (2014-06-24)

- Expose information about top concepts.
- Expose information about display top and display children.
- Fix a bug with returning concepts and collections not on the first page of data through the Range header. (#3)
- Added support for sorting. (#4, #5) [cedrikv]

1.5.10 0.2.0 (2014-05-14)

- Compatibility with skosprovider 0.3.0
- Added service documentation (#1)

1.5.11 0.1.1 (2014-04-10)

- Code coverage by coveralls.
- Removed unit tests from resulting package.
- Moved documentation to Sphinx.
- Reorganisation of tests.
- Changed to py.test as testrunner.
- Some Flake8 fixes.

1.5.12 0.1.0 (2013-05-16)

- Initial version
- Includes json views based on the interfaces skosprovider offers.
- Adds a skosprovider registry to the pyramid request.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

HTTP ROUTING TABLE

/c

/conceptschemes

/uris

INDEX