

---

# **skosprovider\_rdf Documentation**

***Release 0.8.1***

**Flanders Heritage Agency**

**Oct 01, 2020**



---

## Contents

---

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Installation . . . . .	3
1.2	Usage . . . . .	3
<b>2</b>	<b>Development</b>	<b>7</b>
<b>3</b>	<b>API Documentation</b>	<b>9</b>
3.1	Providers module . . . . .	9
3.2	Utils module . . . . .	9
<b>4</b>	<b>History</b>	<b>11</b>
4.1	0.8.1 (27-07-2020) . . . . .	11
4.2	0.8.0 (08-06-2020) . . . . .	11
4.3	0.7.0 (12-02-2020) . . . . .	11
4.4	0.6.0 (16-07-2017) . . . . .	11
4.5	0.5.0 (11-08-2016) . . . . .	12
4.6	0.4.1 (17-07-2015) . . . . .	12
4.7	0.4.0 (03-03-2015) . . . . .	12
4.8	0.3.0 (19-12-2014) . . . . .	12
4.9	0.2.0 (14-10-2014) . . . . .	12
4.10	0.1.3 (02-09-2014) . . . . .	12
4.11	0.1.2 (31-07-2014) . . . . .	13
4.12	0.1.1 (20-05-2014) . . . . .	13
4.13	0.1.0 . . . . .	13
<b>5</b>	<b>Glossary</b>	<b>15</b>
<b>6</b>	<b>Indices and tables</b>	<b>17</b>
	<b>Python Module Index</b>	<b>19</b>
	<b>Index</b>	<b>21</b>



This library offers an implementation of the `skosprovider.providers.VocabularyProvider` interface that uses an `RDFlib` graph as input.



### 1.1 Installation

To be able to use this library you need to have a modern version of Python installed.

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

```
$ pip install skosprovider_rdf
```

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

### 1.2 Usage

This library offers an implementation of the `skosprovider.providers.VocabularyProvider` interface that uses an `rdflib.graph.Graph` as input. This provider can be used to add a *SKOS* vocabulary contained in an *RDF* file to your application. The provider itself does not read the *SKOS* file, but expects to be passed a *Graph*. So any type of RDF serialisation that can be read by `rdflib`, can be used with this provider.

```
# -*- coding: utf-8 -*-

import os

from rdflib import Graph

from skosprovider_rdf.providers import RDFProvider

graph = Graph()

file = os.path.join(os.path.dirname(__file__), '..', 'tests', 'data', 'simple_turtle_
↳ products')
graph.parse(file, format="turtle")
```

(continues on next page)

(continued from previous page)

```

provider = RDFProvider(
    {'id': 'PRODUCTS'},
    graph
)

print("provider.get_all()")
print("-----")
print(provider.get_all())
print("")

print("provider.find({'label': 'jewelry'})")
print("-----")
print(provider.find({'label': 'jewelry'}))
print("")

print("provider.get_by_id('http://www.products.com/Jewellery')")
print("-----")
print(provider.get_by_id('http://www.products.com/Jewellery'))
print("")

print("provider.get_by_uri('http://www.products.com/Jewellery')")
print("-----")
print(provider.get_by_uri('http://www.products.com/Jewellery'))
print("")

```

Out of the box `skosprovider_rdf` assumes your *RDF* file contains exactly one conceptscheme. If no conceptscheme is found in the file and you did not pass one to the provider through the `concept_scheme` parameter, a new conceptscheme is automatically created. If more than one conceptscheme is present in the file, you can again specify the conceptscheme through the `concept_scheme` parameter (passing a `skosprovider.skos.ConceptScheme`) or you can pass the uri of one of the conceptschemes present in the `concept_scheme_uri` parameter. When you specify a conceptscheme like this, only the concepts linked to this scheme through *skos:inScheme* statements will be loaded.

```

# -*- coding: utf-8 -*-
'''
This examples fetches only one conceptscheme from a turtle file containing two.
'''

import os

from rdflib import Graph

from skosprovider_rdf.providers import RDFProvider

graph = Graph()

file = os.path.join(os.path.dirname(__file__), '..', 'tests', 'data', 'waarde_en_
↳besluit_types.ttl')
graph.parse(file, format="turtle")

provider = RDFProvider(
    {'id': 'WAARDETYPES'},
    graph,
    concept_scheme_uri = 'https://id.erfgoed.net/thesauri/waardetypes'
)

```

(continues on next page)



(continued from previous page)

```

print("provider.get_all()")
print("-----")
print(provider.get_all())
print("")

print("provider.find({'label': 'esthetische waarde'})")
print("-----")
print(provider.find({'label': 'esthetische waarde'}))
print("")

print("provider.get_by_id(46)")
print("-----")
print(provider.get_by_id('46'))
print("")

print("provider.get_by_uri('https://id.erfgoed.net/thesauri/waardetypes/46')")
print("-----")
print(provider.get_by_uri('https://id.erfgoed.net/thesauri/waardetypes/46'))
print("")

```

It also provides a utility function to dump any implementation of `skosprovider.providers.VocabularyProvider` to a `rdflib.graph.Graph`. Again, since the provider only deals with the `Graph` object, it's possible to serialise a `VocabularyProvider` to whatever RDF serialisations `rdflib` allows.

```

# -*- coding: utf-8 -*-
'''
This script demonstrates dumping a
:class:`skosprovider.providers.SimpleCsvProvider` as a RDF Graph. In this
case, `n3` serialisation is used, other serialisations are available through
:mod:`rdflib`.
'''

import os
import csv

from skosprovider.providers import SimpleCsvProvider
from skosprovider.uri import UriPatternGenerator
from skosprovider.skos import ConceptScheme, Label, Note, Source
from skosprovider_rdf.utils import rdf_dumper

ifile = open(
    os.path.join(os.path.dirname(__file__), 'data', 'menu.csv'),
    "r"
)

reader = csv.reader(ifile)

csvprovider = SimpleCsvProvider(
    {'id': 'MENU'},
    reader,
    uri_generator=UriPatternGenerator('http://id.python.org/menu/%s'),
    concept_scheme=ConceptScheme(

```

(continues on next page)

(continued from previous page)

```
uri='http://id.python.org/menu',
labels=[
    Label(type='prefLabel', language='en', label='A pythonesque menu.')
],
notes=[
    Note(
        type='changeNote',
        language='en',
        note="<strong>We didn't need no change notes when I was younger.</
↪strong>",
        markup='HTML'
    )
],
sources=[
    Source("Monthy Python's Flying Circus, 1970. Spam.")
]
)
)

graph = rdf_dumper(csvprovider)

print(graph.serialize(format='n3'))
```

## CHAPTER 2

---

### Development

---

skosprovider\_rdf is being developed by the [Flanders Heritage Agency](#).

Since we place a lot of importance of 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_rdf --cov-report term-missing tests
# Only run a subset of the tests
$ py.test skosprovider_rdf/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.



## 3.1 Providers module

This module contains an `RDFProvider`, an implementation of the `skosprovider.providers.VocabularyProvider` interface that uses a `rdflib.graph.Graph` as input.

**class** `skosprovider_rdf.providers.RDFProvider(metadata, graph, **kwargs)`

Should the provider only take concepts into account explicitly linked to the conceptscheme?

**check\_in\_scheme = False**

A simple vocabulary provider that use an `rdflib.graph.Graph` as input. The provider expects a RDF graph with elements that represent the SKOS concepts and collections.

Please be aware that this provider needs to load the entire graph in memory.

**to\_text** (*data*)

data of binary type or literal type that needs to be converted to text. :param data :return: text representation of the data

## 3.2 Utils module

This module contains utility functions for dealing with skos providers.

`skosprovider_rdf.utils.rdf_c_dumper(provider, c)`

Dump one concept or collection from a provider to a format that can be passed to a `skosprovider.providers.RDFProvider`.

**Parameters**

- **provider** (`skosprovider.providers.VocabularyProvider`) – The provider that wil be turned into an `rdflib.graph.Graph`.
- **c** (*String*) – identifier

**Return type** `rdflib.graph.Graph`

`skosprovider_rdf.utils.rdf_conceptscheme_dumper(provider)`

Dump all information of the conceptscheme of a provider to a format that can be passed to a `skosprovider.providers.RDFProvider`.

**Parameters** `provider` (`skosprovider.providers.VocabularyProvider`) – The provider that will be turned into an `rdflib.graph.Graph`.

**Return type** `rdflib.graph.Graph`

`skosprovider_rdf.utils.rdf_dumper(provider)`

Dump a provider to a format that can be passed to a `skosprovider.providers.RDFProvider`.

**Parameters** `provider` (`skosprovider.providers.VocabularyProvider`) – The provider that will be turned into an `rdflib.graph.Graph`.

**Return type** `rdflib.graph.Graph`

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

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

#### 4.1 0.8.1 (27-07-2020)

- Cleaner handling of *infer\_concept\_relations*. When exporting through `skosprovider_rdf` this attribute will determine if broader/narrower relations between concepts are generated when there's a collection between them, as is the case when a concept has a guide term dividing the underlying concepts. When reading from an RDF file, the *infer\_concept\_relations* attribute will be set to `True` if at least one concept in a collection under a concept has a broader relation with said concept. (#73)
- Prevent the `_add_labels` method from generating an error. (#80)

#### 4.2 0.8.0 (08-06-2020)

- Update to `RDFlib` 0.5.0 (#74)

#### 4.3 0.7.0 (12-02-2020)

- Compatible with `SkosProvider` 0.7.0.
- Make it possible to read an RDF file containing more than one conceptscheme. (#35)
- Drop support for Python 3.3, 3.4 and 3.5. This is the last version that will support Python 2. (#63)

#### 4.4 0.6.0 (16-07-2017)

- Compatible with `SkosProvider` 0.6.1.
- Add information about the `void.Dataset` when dumping to RDF.

## 4.5 0.5.0 (11-08-2016)

- Compatible with [SkosProvider 0.6.0](#).
- Add official python 3.5 compatibility.
- Add support for sources when dumping to RDF and reading from RDF. (#17)
- Add support for languages to conceptschemes when dumping to and reading from RDF. (#16)
- Add support for HTML in SKOS notes and sources. (#15, #20)

## 4.6 0.4.1 (17-07-2015)

- RDF dump: Add the top concepts and the conceptscheme identifier in the full RDF dump (equal to the RDF conceptscheme dump).
- RDF provider: literal and binary type to text when parsing the graph to a list.

## 4.7 0.4.0 (03-03-2015)

- Allow dumping a single conceptscheme to RDF. This does not dump the entire conceptscheme with all it's concepts or collections, just information on the conceptscheme itself and it's top concepts.
- Allow dumping a single concept or collection to RDF, and not just an entire conceptscheme with all concepts or collections.
- Add skos:inScheme information to RDF dumps.
- Better handling of dc(t):identifier. When reading an RDF file both dcterms:identifier and dc:identifier are considered when analysing the identifier. During dumping, we also dump to dcterms:identifier.

## 4.8 0.3.0 (19-12-2014)

- Compatible with [SkosProvider 0.5.0](#).
- Dumping to an RDF file now also dumps information on the Conceptscheme.
- Dumping to an RDF file now also adds notes to a Collection, not just to a Concept.
- Now handles subordinate\_array and superordinate concept.

## 4.9 0.2.0 (14-10-2014)

- Add support for Dublin Core identifier (#5)

## 4.10 0.1.3 (02-09-2014)

- Fix a namespace error for SKOS Notes. (#2)



## **4.11 0.1.2 (31-07-2014)**

- Documentation fixes and cleanup
- Removed RDFlib artefacts from output.

## **4.12 0.1.1 (20-05-2014)**

- Bugfixing
- encoding/decoding problems
- casting rdf subjects and objects to rdflib URI's
- Added tests

## **4.13 0.1.0**

- Initial version



**RDF** *Resource Description Framework*. A very flexible model for data definition organised around *triples*. These triples forms a directed, labeled graph, where the edges represent the named link between two resources, represented by the graph nodes.

**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 6

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



### S

`skosprovider_rdf.providers`, 9  
`skosprovider_rdf.utils`, 9





## C

`check_in_scheme(skosprovider_rdf.providers.RDFProvider  
attribute)`, 9

## R

RDF, 15

`rdf_c_dumper()` (*in module skosprovider\_rdf.utils*), 9

`rdf_conceptscheme_dumper()` (*in module skosprovider\_rdf.utils*), 9

`rdf_dumper()` (*in module skosprovider\_rdf.utils*), 10

`RDFProvider` (*class in skosprovider\_rdf.providers*), 9

## S

SKOS, 15

`skosprovider_rdf.providers` (*module*), 9

`skosprovider_rdf.utils` (*module*), 9

## T

`text_()` (*in module skosprovider\_rdf.utils*), 10

`to_text()` (*skosprovider\_rdf.providers.RDFProvider*  
*method*), 9

## U

URI, 15

URN, 15