Atramhasis Documentation

Release 0.4.1

Flanders Heritage

March 11, 2015

Contents

1 Introduction **1**

2 Demo **3**
2.1 Running a demo site . . . 3
2.2 Running a demo site on Heroku . . . 4

3 Development **7**
3.1 Technology . . . 7
3.2 General installation . . . 7
3.3 Admin development . . . 8
3.4 Frontend development . . . 8
3.5 Contributing . . . 9
3.6 Distribution . . . 9

4 Services **11**
4.1 Read Services . . . 11
4.2 Write Services . . . 11

5 Customisation **17**
5.1 Creating your own project . . . 17
5.2 Appearance . . . 20
5.3 Security . . . 20
5.4 Foreign Keys . . . 21
5.5 Adding Google Analytics . . . 21
5.6 Adding external providers . . . 22

6 API Documentation **25**
6.1 atramhasis.data . . . 25
6.2 atramhasis.errors . . . 27
6.3 atramhasis.mappers . . . 28
6.4 atramhasis.protected_resources . . . 28
6.5 atramhasis.utils . . . 28
6.6 atramhasis.validators . . . 29
6.7 atramhasis.views . . . 31

7 History **33**
7.1 0.4.3 (11-03-2015) . . . 33
7.2 0.4.2 (11-03-2015) . . . 33
7.3 0.4.1 (04-03-2015) . . . 33

7.4 0.4.0 (23-12-2014) . . . 34
7.5 0.3.1 (05-09-2014) . . . 35
7.6 0.3.0 (15-08-2014) . . . 35
7.7 0.2.0 (16-05-2014) . . . 35
7.8 0.1.0 (22-04-2014) . . . 35

8 Glossary **37**

9 Indices and tables **39**

HTTP Routing Table **41**

Python Module Index **43**

CHAPTER 1

Introduction

Atramhasis is 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.

Atramhasis is also intended to be one of the focal points in a *Service Oriented Architecture*. It exposes as much of it's functionalities as possible through *REST* services. Both reading from and writing to concept schemes is possible with Atramhasis.

Atramhasis tries to stick as closely as possible to the *SKOS* specification. Where this was not possible, we tried to follow other standards such as *SKOS-THES*.

Atramhasis is being developed by the Flanders Heritage Agency, an agency of the Flemish Government that deals with Archaeology, Monuments and Landscapes. As such, we mainly intend to use it with vocabularies and thesauri that are related to cultural heritage. We generally construct our own thesauri, specific to our own applications, but while always keeping an eye on other thesauri in the larger field of cultural heritage such as the Art and Architecture Thesaurus (AAT).

If you have questions about the project, want to help out, want to report a bug or just want to have a conversation with us, please get in touch. For general conversations, you can use our Google Group. If you have encoutered a bug in Atramhasis or it's documenation, or if you want to ask us to consider implementing a feature, feel free to use our issue tracker.

CHAPTER 2

Demo

2.1 Running a demo site

Atramhasis comes with a demo site include. This allows you to quickly evaluate and inspect the software. To get started, just download Atramhasis from pypi and install it. We recommend doing this in a virtualenvironment.

```
$ mkvirtualenv atramhasis_demo
$ pip install -U atramhasis
```

Once Atramhasis is installed, you can call upon a pyramid scaffold to generate the demo site.

```
$ pcreate -s atramhasis_demo atramhasis_demo
$ cd atramhasis_demo
```

This creates a local demo package you can run with just a few more commands:

```
# setup
$ pip install -r requirements-dev.txt
$ python setup.py develop
# create or upgrade database
$ alembic upgrade head
# intialize sample data
$ initialize_atramhasis_db development.ini
# compile translations
$ python setup.py compile_catalog
# start server
$ pserve development.ini
```

The Atramhasis demo instance is now running on your localhost at port 6543. To reach it, open your browser and surf to the address http://localhost:6543.

You will be greeted by the Atramhasis front page. From this page you can start searching and browsing the thesauri. You can also start editing the thesauri by surfing to http://localhost:6543/admin. The demo instance requires that you login to access the admin module. We've provided a login mechanism using Mozilla Persona for the demo. If you want to run Atramhasis in a production environment, you can easily replace the security module by another one. This enables you to use the security mechanisms (eg. LDAP, Active Directory, a custom users database, ...) that your organisation requires. Please consult the documentation on *Security* customisation for further information on this topic.

2.2 Running a demo site on Heroku

This section will tell you how to deploy an Atramhasis demo (or your own implementation) in the cloud. We'll use Heroku, since this provider allows for a free Python instance (dyno) with a limited Postgresql database.

Create an account on Heroku and make sure you have Heroku Toolbelt installed. Prepare your local Heroku setup

Note: More information on running Python apps on Heroku can be found on the Heroku dev center.

2.2.1 Atramhasis scaffold

Create an Atramhasis scaffold (if you want to deploy an existing scaffold, skip this step)

```
$ mkvirtualenv atramhasis_heroku
$ pip install -U atramhasis
$ pcreate -s atramhasis_demo atramhasis_heroku
$ cd atramhasis_heroku
```

2.2.2 Git repository

Make sure your atramhasis_heroku folder is a git repository.

```
$ git init
$ git add .
$ git commit -m "initial commit"
```

2.2.3 requirements.txt

Update the requirements.txt file, make sure it contains a reference to atramhasis and to waitress.

Note: waitress has to be in the requirements.txt file for our Heroku deployment, requirements-dev.txt will be ignored.

2.2.4 Procfile

Generate `Procfile` with the following command.

```
$ echo "web: ./run" > Procfile
```

2.2.5 run file

Create `run` with the following content.

```
#!/bin/bash
set -e
python setup.py develop
python runapp.py
```

Note: Make sure to `chmod +x run` before continuing. The `develop` step is necessary because the current package must be installed before Paste can load it from the INI file.

2.2.6 runapp.py

Create a `runapp.py` file.

```
import os

from paste.deploy import loadapp
from waitress import serve

if __name__ == "__main__":
    port = int(os.environ.get("PORT", 5000))
    app = loadapp('config:production.ini', relative_to='.')

    serve(app, host='0.0.0.0', port=port)
```

Note: After creating the necessary files, commit them in your local git repository

2.2.7 Initialize the Heroku stack

```
$ heroku create
```

2.2.8 Deploy to Heroku

To deploy a new version, push it to Heroku.

```
$ git push heroku master
```

2.2.9 Postgresql

Attach an Heroku Postgres add-on to your application

```
$ heroku addons:add heroku-postgresql:hobby-dev
```

It can take a couple of minutes before your db is ready. You can wait for it to be ready using this command.

```
$ heroku pg:wait
```

When ready, check the connection url and copy paste it into your production.ini file

```
$ heroku config | grep HEROKU_POSTGRESQL
```

Also change the alembic.ini file to check your production.ini file instead of development.ini

```
ini_location = %(here)s/production.ini
```

Make sure to commit everything and push it to Heroku

```
$ git commit -a
$ git push heroku master
```

Note: More info on provisioning a database

2.2.10 Preparing the app

Open a remote console on your app

```
$ heroku run bash
```

This will start a console inside your remote Python virtualenv, so you can use all your libraries.

Run the commands to prepare your application

```
$ python setup.py develop
$ alembic upgrade head
$ initialize_atramhasis_db production.ini
$ python setup.py compile_catalog
```

Note: Close the remote console!

2.2.11 Run the app

Run your app by starting one worker

```
$ heroku scale web=1
```

Check to see if your app is running.

```
$ heroku ps
```

Take a look at the logs to debug any errors if necessary.

```
$ heroku logs -t
```

Your app should now be available on the application url.

CHAPTER 3

Development

3.1 Technology

Atramhasis is a python webapplication that is being developed within the pyramid framework. Other major technologies used are sqlalchemy as the ORM and *Jinja2* as the templating framework.

Client side the main technologies being used are Zurb Foundation and Dojo toolkit.

While Atramhasis is an editor for creating and editing *SKOS* vocabularies, it uses other libraries that are more geared towards using a vocabulary in an application.

- skosprovider: This library defines a `VocabularyProvider`. This is an abstraction of usefull functionalities an application integrating *SKOS* needs. Different libraries can implement this interface for different datasources. This allows decoupling the interface from the concrete implementation. Out of the box this comes with a simple `DictionaryProvider` that serves a vocabulary based on a simple python `dict` as datasource.
- skosprovider_sqlalchemy: An implementation of the `VocabularyProvider` interface with a SQLAlchemy backend. This allows using a RDBMS for reading, but also writing, *SKOS* concepts.
- skosprovider_rdf: An implemenation of the `VocabularyProvider` interface with an *RDF* backend. Atramhasis uses this for exporting ConceptSchemes to RDF. It can also be used to get an existing *SKOS* vocabulary defined in RDF into Atramhasis.
- pyramid_skosprovider: A library that integrates pyramid and skosprovider. This libraries creates a `skosprovider.registry.Registry` and makes it accessible through the `pyramid.request.Request`. Is also exposes a set of readonly *REST services* on the registered providers.
- skosprovider_getty: An implemenation of the `VocabularyProvider` against the Linked Open Data vocabularies published by the Getty Research Institute at http://vocab.getty.edu such as the *Art and Architecture Thesaurus (AAT)* and the *Thesaurus of Geographic Names (TGN)*.
- skosprovider_heritagedata: An implementation of the `VocabularyProvider` against the vocabularies published by EH, RCAHMS and RCAHMW at heritagedata.org.

3.2 General installation

We recommend installing Atramhasis in a virtual environment.

```
$ mkvirtualenv atramhasis_dev
```

To install a fully working development environment a pip requirements-dev.txt file is provided. By passing this file to **pip install -r** all requirements for Atramhasis and development of the software (Sphinx, py.test, tox) will be installed.

The following step will help you get the python development environment up and running. If also need to work on the javascript admin backend, please refer to the admin module documentation.

```
# Install dependencies
$ pip install -r requirements-dev.txt
# create or update database
$ alembic upgrade head
# insert sample data
$ initialize_atramhasis_db development.ini
# compile the Message Catalog Files
$ python setup.py compile_catalog
```

Once you've executed these steps, you can run a development server. This uses the standard pyramid server (Waitress) and should not be used as-is in a production environment.

```
# run a local development server
$ pserve --reload development.ini
```

3.3 Admin development

To work on the admin part, you'll need npm and bower installed. Consult your operating system documentation on how to install these. The following instructions will assume you're running a recent Debian based Linux distribution.

```
# install npm, bower and grunt-cli
$ sudo apt-get install nodejs
$ sudo apt-get install npm
$ sudo npm install -g bower grunt-cli
# install js dependencies using bower
$ cd atramhasis/static/admin
$ bower install
# install dojo build tools
$ npm install
```

These commands will install a couple of js libraries that Atramhasis uses in `/atramhasis/static/admin/src` and a set of tools to be able to generate js builds. Builds are carried out through a simple grunt file:

```
# Build a dojo distribution
$ cd atramhasis/static/admin
$ grunt -v build
```

This will create a build a place the resulting files in `atramhasis/static/admin/dist`. The web application can be told to use this build by setting *dojo.mode* in `development.ini` to *dist*.

3.4 Frontend development

When updating the frontend templates, you might want to add extra translations. This can be done by placing {% trans %} tags in the templates

```
<h2>{% trans %}welcome_to{% endtrans %}</h2>
```

To update the message catalogs, do as follows:

```
$ python setup.py extract_messages
$ python setup.py update_catalog -l fr -i atramhasis/locale/atramhasis.pot -o atramhasis/locale/fr/LC
$ python setup.py update_catalog -l nl -i atramhasis/locale/atramhasis.pot -o atramhasis/locale/nl/LC
$ python setup.py update_catalog -l en -i atramhasis/locale/atramhasis.pot -o atramhasis/locale/en/LC
```

Update the catalogs accordingly and run:

```
$ python setup.py compile_catalog
```

You might also want to add a new translation. Suppose you want to add a German translation.

```
$ python setup.py init_catalog -l de -i atramhasis/locale/atramhasis.pot -o atramhasis/locale/de/LC_M
```

Edit `atramhasis/locale/do/LC_MESSAGES/atramhasis.po` and add the necessary translations. Just as with updating the catalogs, you need to recompile them.

```
$ python setup.py compile_catalog
```

At this moment, Atramhasis will still only show the default languages in it's language switcher. If you want to add you new language, you need to edit your `development.ini` (or similar file). Look for the line that says *available_languages* and add your locale identifier.

```
available_languages = en nl fr de
```

After restarting your server you will now have the option of switching to German.

3.5 Contributing

Atramhasis is being developed as open source software by the Flanders Heritage Agency. All development is done on the agency's Github page for Atramhasis.

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.

```
# Run unit tests for all environments
$ tox
# No coverage
$ py.test
# Coverage
$ py.test --cov atramhasis --cov-report term-missing
# Only run a subset of the tests
$ py.test atramhasis/tests/test_views.py
```

Every pull request will be run through Travis-ci. When providing a pull request, please run the unit tests first and make sure they all pass. 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.

3.6 Distribution

For building a distribution use the prepare command before the distribution command. This will update the requirement files in the scaffolds.

```
$ python setup.py prepare sdist
```

CHAPTER 4

Services

Atramhasis can be used fully with a *SOA*. While we provide a public and and a administrators interface out of the box, you can also write your own client side code that interacts with the Atramhasis services, either for reading information or writing it.

4.1 Read Services

The basic read services are being provided by *Pyramid Skosprovider*. These allow you to read conceptschemes, search for concepts and collections and integrate the Atramhasis backend into other applications. Maintain you thesaurus in one database and use it in several other applications.

4.2 Write Services

The Atramhasis write services allow you to add concepts and collections, edit them and delete them.

POST /conceptschemes/{scheme_id}/c

Add a concept or collection to a conceptscheme. The response body will contain a representation of the concept or collection after is has been added to the conceptscheme.

Example request:

```
POST /conceptschemes/TREES/c HTTP/1.1
Host: demo.atramhasis.org
Accept: application/json

{
    "type": "concept",
    "broader": [],
    "narrower": [],
    "related": [],
    "labels": [
        {
            "type": "prefLabel",
            "language": "en",
            "label": "The Larch"
        }
    ],
    "notes": []
}
```

Example response:

```
HTTP/1.1 201 Created
Location: http://demo.atramhasis.org/conceptschemes/TREES/c/1
Content-Type: application/json

{
    "id": 1,
    "uri": "urn:x-atramhasis-demo:TREES:1",
    "type": "concept",
    "broader": [],
    "narrower": [],
    "related": [],
    "labels": [
        {
            "type": "prefLabel",
            "language": "en",
            "label": "The Larch"
        }
    ],
    "notes": []
}
```

Example request:

```
POST /conceptschemes/TAUNTS/c HTTP/1.1
Host: demo.atramhasis.org
Accept: application/json

{
    "type": "concept",
    "broader": [],
    "narrower": [],
    "related": [],
    "labels": [
        {
            "type": "tauntLabel",
            "language": "en-FR",
            "label": "Your mother was a Hamster!"
        }
    ],
    "notes": []
}
```

Example response:

```
HTTP/1.1 400 Bad Request
Location: http://demo.atramhasis.org/conceptschemes/TREES/c/1
Content-Type: application/json

{
    "errors": [
                {"labels": "Invalid labeltype."},
                {"labels": "Invalid language."}
              ],
    "message": "Concept could not be validated"
}
```

Parameters

- **scheme_id** – The identifier for a certain concept scheme.

Request Headers

- Accept – The response content type depends on this header. Currently only *application/json* is supported.

Response Headers

- Content-Type – This service currently always returns *application/json*
- Location – The url where the newly added concept or collection can be found.

Status Codes

- 201 Created – The concept or collection was added succesfully.
- 400 Bad Request – The concept or collection could not be added because the submitted json was invalid due to eg. validation errors.
- 404 Not Found – The conceptscheme *scheme_id* does not exist.
- 405 Method Not Allowed – The concept or collection could not be added because the conceptscheme *scheme_id* is a readonly conceptscheme.

PUT /conceptschemes/{scheme_id}/c/{c_id}

Edit the concept or collection with id *c_id*. The response body will contain a representation of the concept or collection after the modifications.

Example request:

```
PUT /conceptschemes/TREES/c/1 HTTP/1.1
Host: demo.atramhasis.org
Accept: application/json

{
    "type": "concept",
    "broader": [],
    "narrower": [],
    "related": [],
    "labels": [
        {
            "type": "prefLabel",
            "language": "en",
            "label: "The Larch"
        }, {
            "type": "prefLabel",
            "language": "nl",
            "label": "De Lariks"
        }
    ],
    "notes": []
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "uri": "urn:x-atramhasis-demo:TREES:1",
```

```
    "type": "concept",
    "broader": [],
    "narrower": [],
    "related": [],
    "labels": [
        {
            "type": "prefLabel",
            "language": "en",
            "label: "The Larch"
        }, {
            "type": "prefLabel",
            "language": "nl",
            "label": "De Lariks"
        }
    ],
    "notes": []
}
```

Parameters

- **scheme_id** – The identifier for a certain concept scheme.
- **c_id** – The identifier for a certain concept or collection.

Request Headers

- Accept – The response content type depends on this header. Currently only *application/json* is supported.

Response Headers

- Content-Type – This service currently always returns *application/json*

Status Codes

- 200 OK – The concept or collection was edited succesfully.
- 400 Bad Request – The concept or collection could not be edited because the submitted json was invalid due to eg. validation errors.
- 404 Not Found – The conceptscheme *scheme_id* or the concept or collection *c_id* does not exist.
- 405 Method Not Allowed – The concept or collection could not be edited because the conceptscheme *scheme_id* is a readonly conceptscheme.

DELETE /conceptschemes/{scheme_id}/c/{c_id}

Remove the concept with id *c_id*. The response body will contain the last representation known by the service.

Example request:

```
DELETE /conceptschemes/TREES/c/1 HTTP/1.1
Host: demo.atramhasis.org
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
```

```
    "uri": "urn:x-atramhasis-demo:TREES:1",
    "type": "concept",
    "broader": [],
    "narrower": [],
    "related": [],
    "labels": [
        {
            "type": "prefLabel",
            "language": "en",
            "label: "The Larch"
        }, {
            "type": "prefLabel",
            "language": "nl",
            "label": "De Lariks"
        }
    ],
    "notes": []
}
```

Parameters

- **scheme_id** – The identifier for a certain concept scheme.
- **c_id** – The identifier for a certain concept or collection.

Request Headers

- Accept – The response content type depends on this header. Currently only *application/json* is supported.

Response Headers

- Content-Type – This service currently always returns *application/json*

Status Codes

- 200 OK – The concept or collection was deleted succesfully.
- 400 Bad Request – The concept or collection could not be edited because the submitted json was invalid due to eg. validation errors.
- 404 Not Found – The conceptscheme *scheme_id* or the concept or collection *c_id* does not exist.
- 405 Method Not Allowed – The concept or collection could not be deleted because the conceptscheme *scheme_id* is a readonly conceptscheme.
- 409 Conflict – The concept or collection could not be deleted because Atramhasis has determined that it's still being used somewhere else. The response body will contain a message and a list of *URI*'s that are using this concept.

CHAPTER 5

Customisation

Out of the box Atramhasis tries to make as few assumptions as possible about setup. We have taken care to ensure that significant parts of the application are easy to customise and expect most installations to have custom code. We've shipped Atramhasis with sane defaults so you can get a quick feel for the capabilities of the software. However, we do not advise running a production instance with only these default settings.

5.1 Creating your own project

Whenever you want to run an instance of Atramhasis, you start by creating your own project. This is the place where you will maintain and develop your own custom templates, static assets such as stylesheets, your security implementation and other general configuration. To make it easier on you to get started, we provide a scaffold just for this. As always, we advise working in a virtual environment.

```
$ mkvirtualenv my_thesaurus
$ pip install atramhasis
$ pcreate -s atramhasis_scaffold my_thesaurus
# Install dependencies
$ pip install -r requirements-dev.txt
# compile the Message Catalog Files
$ python setup.py compile_catalog
```

This gives you a clean slate to start your customisations on. By default the scaffold comes with a simple SQLite database. This is more than enough for your first experiments and can even be used in production environment if your needs are modest. You can always instruct Atramhasis to use some other database engine, as long as SQLAlchemy supports it. Configure the *sqlalchemy.url* configuration option in `development.ini` to change the database. See the documentation of SQLAlchemy for more information about this connection url. After settings this url, run **alembic** to initialise and migrate the database to the latest version.

```
# Create or update database based on
# the configuration in development.ini
$ alembic upgrade head
```

Your custom version of Atramhasis can now be run. Run the following command and point your browser to *http://localhost:6543* to see the result.

```
$ pserve development.ini
```

Of course, this does not do very much since your Atramhasis is now running, but does not contain any ConceptSchemes. You will need to configure this by entering a database record for the ConceptScheme and writing a small piece of code.

To enter the database record, you need to enter a record in the table *conceptscheme*. In this table you need to register an id for the conceptscheme and a uri. The id is for internal database use and has no other meaning. The uri can be used externally. To register a new ConceptScheme in the sqlite database that was created:

```
$ sqlite3 my_thesaurus.sqlite

INSERT INTO conceptscheme VALUES (1, 'urn:x-my-thesaurus:stuff')
```

This take care of the first step. Now you also need to tell Atramhasis where to find your conceptscheme and how to handle it. To do this, you need to edit the file called `my_thesaurus/skos/__init__.py`. In this file you need to register `SQLAlchemyProvider` instances. First you need to tell python where to find such a provider by adding this code just below the logging configuration:

```
from skosprovider_sqlalchemy.providers import SQLAlchemyProvider
```

Then you need to instantiate such a provider within the includeme function in this file. This provider needs a few arguments: an id for the provider, an id for the conceptscheme it's working with and a function that knows how the provide a database session. The id for the provider is often a text string and will appear in certain url's and might popup in the user interface from time to time. The database sessionmaker can be found at *config.registry.dbmaker*. Finally, you need to register this provider with the `skosprovider.registry.Registry`.

```
STUFF = SQLAlchemyProvider(
    {
        'id': 'STUFF',
        'conceptscheme_id': 1
    },
    config.registry.dbmaker
)

skosregis.register_provider(STUFF)
```

After having registered your provider, the file should look more or less like this:

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

import logging
log = logging.getLogger(__name__)

from skosprovider_sqlalchemy.providers import SQLAlchemyProvider


def includeme(config):
    STUFF = SQLAlchemyProvider(
        {
            'id': 'STUFF',
            'conceptscheme_id': 1
        },
        config.registry.dbmaker
    )

    skosregis = config.get_skos_registry()

    skosregis.register_provider(STUFF)
```

Now you can restart your server and then you front page will show you a new, but empty thesaurus. You can now start creating concepts and collections by going to the admin interface at *http://localhost:6543/admin*.

You will notice that any concepts or collections you create wil get a *URI* similar to *urn:x-skosprovider:STUFF:1*. This is due to the fact that your `SQLAlchemyProvider` has a `UriGenerator` that generates uris for the provider. By

default, the provider configures a `DefaultUrnGenerator`, but it's expected that you will want to override this.

> **Warning:** The `UriGenerator` that you configure only generates URI's when creating new concepts or collections. When importing existing vocabularies, please be sure to create the URI's before or during import (possbily by using a relevant generator yourself).

Suppose you have decided that your URI's should look like this: *http://id.mydata.org/thesauri/stuff/[id]*. You can do this by registering a `UriPatternGenerator` with your provider:

```
STUFF = SQLAlchemyProvider(
    {
        'id': 'STUFF',
        'conceptscheme_id': 1
    },
    config.registry.dbmaker,
    uri_generator=UriPatternGenerator(
        'http://id.mydata.org/thesauri/stuff/%s'
    )
)
```

Don't forget to import the `UriPatternGenerator` at the top of your file:

```
from skosprovider.uri import UriPatternGenerator
```

Your final file should look similar to this:

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

import logging
log = logging.getLogger(__name__)

from skosprovider_sqlalchemy.providers import SQLAlchemyProvider
from skosprovider.uri import UriPatternGenerator


def includeme(config):
    STUFF = SQLAlchemyProvider(
        {
            'id': 'STUFF',
            'conceptscheme_id': 1
        },
        config.registry.dbmaker,
        uri_generator=UriPatternGenerator(
            'http://id.mydata.org/thesauri/stuff/%s'
        )
    )

    skosregis = config.get_skos_registry()

    skosregis.register_provider(STUFF)
```

If you need more complicated URI's, you can easily write you own generator with a small piece of python code. You just need to follow the interface provided by `skosprovider.uri.UriGenerator`.

5.2 Appearance

By implementing a few simple techniques from the *Pyramid* web framework, it's very easy to customise the look and feel of the public user interface. The default implementation is a very neutral implementation based on the basic elements in the Foundation framework. Customising and overriding this style is possible if you have a bit of knowledge about *HTML* and *CSS*.

You can also override the *HTML* templates that Atramhasis uses without needing to alter the originals so that future updates to the system will not override your modifications.

One very easy technique to use, is *Pyramid*'s *override assets mechanism*. This allows you to override a core Atramhasis template with your own template. Suppose we want to change the text on the Atramhasis homepage to welcome visitors to your instances. This text can be found in `atramhasis/templates/welcome.jinja2`.

Assuming that you created your project as *my_thesaurus*, we can now create our own template in `my_thesaurus/templates/my_welcome.jinja2`. Please consult the *Jinja2* documentation if you need help with this.

Once you've created your template file, you just need to tell your project to override the default `welcome.jinja2` with your version. To do this you need to configure the *Pyramid* config object found in `my_thesaurus.__init__.py`.

```
config.override_asset(
    to_override='atramhasis:templates/welcome.jinja2',
    override_with='templates/my_welcome.jinja2'
)
```

Note: Normally, to see the effect of the changes you made, you would need to restart your webserver. When developing, you can make use of the **pserve** command's auto-reload feature. To do this, start your server like this:

```
$ pserve --reload development.ini
```

5.3 Security

We assume that every deployment of Atramhasis has different needs when it comes to security. Some instances will run on a simple laptop for testing and evaluation purposes, others might need a simple standalone database of users and certain deployments might need to integrate with enterprise authentication systems like LDAP, Active Directory, Single Sign On, ...

Atramhasis provides authorisation hooks for security. To edit, add or delete a concept or collection, a user is required to have the 'editor' pemission. Unless no authorisation policy has been configured.

5.3.1 Sample configuration

The atramhasis_demo scaffold contains a sample security configuration, using Mozilla Persona: http://www.mozilla.org/en-US/persona/. Persona security is implemented with pyramid_persona: https://pypi.python.org/pypi/pyramid_persona

You can configure persona.secret and persona.audience in development.ini:

```
persona.secret = sosecret
persona.audiences = http://localhost:6543
```

The login and logout views, the groupfinder and rootfactory are implemented in the security.py file.

5.4 Foreign Keys

Atramhasis will often function as a central part of a *SOA* in an organisation. `Concept` and maybe `Collection` objects will be used by other applications. One of the riskier aspects of this is that someone might delete a concept in a certain scheme that is still being used by another application. Even worse, the user approving the delete might not even have a clue that the concept is being used by some external application. While in the decentralised world that is the world wide web, we can never be sure that nobody is using our concept any more, we can take some steps to at least control what happens within other applications that are within our control.

Of course, within the framework that is Atramhasis it's very difficult to know how or where your own resources might be and how they might be using concepts from Atramhasis. We have therefor provided the necessary hooks for you that can help you deal with the sort of situation. But the actual implementation is left up to you.

We have added a decorator `protected_operation()`. When you add this decorator to a view, this view will emit a `ProtectedResourceEvent`. By default we have added this decorator the `delete_concept()` view.

In you own code, you can subscribe to this `ProtectedResourceEvent` through the usual `pyramid.events.subscriber()`. In this event handler you are then free to implement whatever check you need to do. If you find that the resource in question is being used somewhere and this operation should thus not be allowed to proceed, you simply need to raise a `atramhasis.protected_resources.ProtectedResourceException`. Into this exception you can also pass a list of *URI* that might provide the user with some feedback as to where this concept might be used.

For example, a sample event handler that would make it impossible to delete concepts with a URI of less than 5 characters:

```
from pyramid.events import subscriber
from atramhasis.protected_resources import ProtectedResourceEvent

@subscriber(ProtectedResourceEvent)
def never_delete_a_short_uri(event):
    if len(event.uri) < 5:
        raise ProtectedResourceException(
            'resource {0} has a URI shorter than 5 characters, preventing this operation'.format(ever
            []
        )
```

5.5 Adding Google Analytics

Out of the box, it's very easy to add Google Analytics integration to Atramhasis. All you need to do is add you Web Property ID to `development.ini`.

```
# Enter your Google Analytics Web Property ID
ga.tracker_key = UA-12345678-9
```

This will add basic analytics to every page, using a Jinja2 macro. If you need more control over the code, you can override this macro in your own project. Suppose you always want to use SSL when sending data. First, you would create you own macro, eg. in `my_macros.jinja2` in the templates directory of your *own project*.

```
{% macro ga_tracker(ga_key) %}
    <!-- Google Analytics -->
    <script type="text/javascript">
    (function(i,s,o,g,r,a,m){i['GoogleAnalyticsObject']=r;i[r]=i[r]||function(){
    (i[r].q=i[r].q||[]).push(arguments)},i[r].l=1*new Date();a=s.createElement(o),
    m=s.getElementsByTagName(o)[0];a.async=1;a.src=g;m.parentNode.insertBefore(a,m)
    })(window,document,'script','//www.google-analytics.com/analytics.js','ga');
```

```
    ga('create', '{{ ga_key }}', 'auto');
    ga('set', 'forceSSL', true);
    ga('send', 'pageview');
    </script>
    <!-- End Google Analytics -->
{% endmacro %}
```

Once that's done, you need to override the the `ga` block in the base template. To do this, it's easiest to override Atramhasis' `base.jinja2` in your own project. To do that, add the following to your project's main function:

```
config.override_asset(
    to_override='atramhasis:templates/base.jinja2',
    override_with='templates/base.jinja2'
)
```

In this file, you can now choose what should appear within the ga block defined in `staticbase.jinja2`. Here we are just replacing one macro with another, but you are off course free to make further alterations.

```
{%- extends 'staticbase.jinja2' -%}

{% block ga %}
    {% set ga_key = ga_key|default(request.registry.settings["ga.tracker_key"]) %}
    {% from 'my_macros.jinja2' import ga_tracker %}
    {% if ga_key %}
        {{ ga_tracker(ga_key) }}
    {% endif %}
{% endblock %}
```

5.6 Adding external providers

Within your Atramhasis instance you can make use of external providers. These are other systems serving up thesauri that you can interact with. Within the admin interface you can create links to these thesauri as *SKOS* matches. This way you can state that a concept within your thesauri is the same as or similar to a concept in the external thesaurus. And, more interestingly, you can also import concepts from such a thesaurus into your own vocabulary. Importing a concept like this will automatically create a *SKOS* match for you. Once a match is in place, you can also update your local concept with information from the external concept by performing a merge.

To enable all this power, you again need to configure a provider in you application. Continuing with our *example project*, we need to go back to our `my_thesaurus/skos/__init__.py`. In this file you need to register other instances of `skosprovider.providers.VocabularyProvider`. Currently providers have already been written for Getty Vocabularies, English Heritage vocabularies and Flanders Heritage Vocabularies. Depending on the system you're trying to interact with, writing a new provider is fairly simple. For this example, we'll assume that you want to integrate the wealth of information that the *Art and Architecture Thesaurus (AAT)* vocabulary offers you.

The `AATProvider` for this (and other Getty vocabularies) is available as skosprovider_getty and is installed by default in an Atramhasis instance. All you need to do is configure it. First, we need to import the provider. Place this code at the top of `my_thesaurus/skos/__init__.py`.

```
from skosprovider_getty.providers import AATProvider
```

Once this is done, we need to instantiate the provider within the *includeme* function and register it with the `skosprovider.registry.Registry`. This is all quite similar to registering your own `skosprovider_sqlalchemy.providers.SQLAlchemyProvider`. One thing you do need to do, is tagging this provider with a subject. By adding the *external* subject to the provider, we let Atramhasis know that this is

not a regular, internal provider that can be stored in our database, but a special external one that can only be used for making matches. As such, it will not be present and visible to the public among your regular vocabularies.

```
AAT = AATProvider(
    {'id': 'AAT', 'subject': ['external']},
)
skosregis.register_provider(AAT)
```

That's all. You can do the same with the `TGNProvider` for the *Thesaurus of Geographic Names (TGN)* or any of the providers for heritagedata.org that can be found in skosprovider_heritagedata.

In the end your `my_thesaurus/skos/__init__.py` should look somewhat like this:

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

import logging
log = logging.getLogger(__name__)

from skosprovider_sqlalchemy.providers import SQLAlchemyProvider
from skosprovider_getty.providers import AATProvider
from skosprovider.uri import UriPatternGenerator


def includeme(config):
    STUFF = SQLAlchemyProvider(
        {
            'id': 'STUFF',
            'conceptscheme_id': 1
        },
        config.registry.dbmaker,
        uri_generator=UriPatternGenerator(
            'http://id.mydata.org/thesauri/stuff/%s'
        )
    )

    AAT = AATProvider(
        {
            'id': 'AAT',
            'subject': ['external']
        }
    )

    skosregis = config.get_skos_registry()

    skosregis.register_provider(STUFF)
    skosregis.register_provider(AAT)
```

Now you'll be able to import from the AAT to your heart's delight. For an extended example that adds even more providers, you could have a look at the *demo* scaffold that comes with Atramhasis.

CHAPTER 6

API Documentation

6.1 atramhasis.data

6.1.1 atramhasis.data.datamanagers

This module adds DataManagers for Atramhasis. These are service layer objects that abstract all interactions with the database away from the views.

versionadded 0.4.1

class atramhasis.data.datamanagers.**ConceptSchemeManager**(*session*)
A DataManager for ConceptSchemes <skosprovider_sqlalchemy.models.ConceptScheme>.

find(*conceptscheme_id*, *query*)
Find concepts and collections in this concept scheme.

Parameters

- **conceptscheme_id** – a concepscheme id
- **query** – A python dictionary containing query parameters.

Returns A list of skosprovider_sqlalchemy.models.Thing instances.

get(*conceptscheme_id*)

Parameters **conceptscheme_id** – a concepscheme id

Returns the concepscheme for the given id

get_all(*conceptscheme_id*)
Get all concepts and collections in this concept scheme.

Parameters **conceptscheme_id** – a concepscheme id

Returns A list of skosprovider_sqlalchemy.models.Thing instances.

get_collections_for_scheme_tree(*conceptscheme_id*)

Parameters **conceptscheme_id** – a concepscheme id

Returns all collections for the scheme_tree

get_concepts_for_scheme_tree(*conceptscheme_id*)

Parameters **conceptscheme_id** – a concepscheme id

Returns all concepts for the scheme_tree

class atramhasis.data.datamanagers.**DataManager**(*session*)
A DataManager abstracts all interactions with the database for a certain model.

class atramhasis.data.datamanagers.**LanguagesManager**(*session*)
A DataManager for Languages <skosprovider_sqlalchemy.models.Language>.

delete(*language*)

Parameters **language** – the language to delete

get_all()

Returns list of all languages

get_all_sorted(*sort_coll*, *sort_desc*)

Parameters

- **sort_coll** – sort on this column
- **sort_desc** – descending or not

Returns sorted list of languages

save(*language*)

Parameters **language** – language to save

Returns saved language

class atramhasis.data.datamanagers.**SkosManager**(*session*)
A DataManager for Concepts and Collections <skosprovider_sqlalchemy.models.Thing>.

delete_thing(*thing*)

Parameters **thing** – the thing to delete

get_by_list_type(*list_type*)

Parameters **list_type** – a specific list type

Returns all results for the specific list type

get_thing(*concept_id*, *conceptscheme_id*)

Parameters

- **concept_id** – a concept id
- **conceptscheme_id** – a conceptscheme id

Returns the selected thing (Concept or Collection)

save(*thing*)

Parameters **thing** – thing to save

Returns saved thing

atramhasis.data.datamanagers.**desc**(*column*)
Produce a descending ORDER BY clause element.

e.g.:

```
from sqlalchemy import desc

stmt = select([users_table]).order_by(desc(users_table.c.name))
```

will produce SQL as:

```
SELECT id, name FROM user ORDER BY name DESC
```

The `desc()` function is a standalone version of the `ColumnElement.desc()` method available on all SQL expressions, e.g.:

```
stmt = select([users_table]).order_by(users_table.c.name.desc())
```

Parameters **column** – A `ColumnElement` (e.g. scalar SQL expression) with which to apply the `desc()` operation.

See also:

`asc()`

`nullsfirst()`

`nullslast()`

`Select.order_by()`

6.1.2 atramhasis.data.db

Module that sets up the datamanagers and the database connections.

`atramhasis.data.db.`**`data_managers`**(*request*)
Generate a datamanager with a database session and register a cleanup handler.

Parameters **request** (*pyramid.request.Request*) – The request this db session will be tied to.

Returns A dictionary containing different `datamanagers`.

`atramhasis.data.db.`**`includeme`**(*config*)
Set up SQLAlchemy.

Parameters **config** (*pyramid.config.Configurator*) – Pyramid configuration.

6.2 atramhasis.errors

Module containing errors generated by Atramhasis.

exception `atramhasis.errors.`**`ConceptNotFoundException`**(*c_id*)
A Concept or Collection could not be found.

exception `atramhasis.errors.`**`ConceptSchemeNotFoundException`**(*scheme_id*)
A ConceptScheme could not be found.

exception `atramhasis.errors.`**`DbNotFoundException`**(*value='No database found, please check your application setup'*)
Atramhasis could not find a database.

exception `atramhasis.errors.`**`LanguageNotFoundException`**(*scheme_id*)
A Language could not be found.

exception `atramhasis.errors.`**`SkosRegistryNotFoundException`**(*value='No SKOS registry found, please check your application setup'*)
Atramhasis could not find a SKOS registry.

exception atramhasis.errors.**ValidationError**(*value*, *errors*)
Some data that was validated is invalid.

6.3 atramhasis.mappers

Module containing mapping functions used by Atramhasis.

atramhasis.mappers.**map_concept**(*concept*, *concept_json*, *skos_manager*)
Map a concept from json to the database.

Parameters

- **concept** (*skosprovider_sqlalchemy.models.Thing*) – A concept or collection as known to the database.
- **concept_json** (*dict*) – A dict representing the json sent to our REST service.
- **skos_manager** – A skos_manager to acces db operations

Returns The skosprovider_sqlalchemy.models.Thing enhanced with the information from the json object.

6.4 atramhasis.protected_resources

Thid module is used when blocking operations on a certain Concept or Collection that might be used in external applications.

versionadded 0.4.0

class atramhasis.protected_resources.**ProtectedResourceEvent**(*uri*)
Event triggered when calling a protected operation on a resource

exception atramhasis.protected_resources.**ProtectedResourceException**(*value*, *referenced_in*)
raise this exception when the resource is still used somewhere

referenced_in should contain locations where the resource is still referenced

atramhasis.protected_resources.**protected_operation**(*fn*)
use this decorator to prevent an operation from being executed when the related resource is still in use

6.5 atramhasis.utils

Module containing utility functions used by Atramhasis.

atramhasis.utils.**from_thing**(*thing*)
Map a skosprovider_sqlalchemy.models.Thing to a skosprovider.skos.Concept or a skosprovider.skos.Collection, depending on the type.

Parameters **thing** (*skosprovider_sqlalchemy.models.Thing*) – Thing to map.

Return type Concept or Collection.

atramhasis.utils.**internal_providers_only**(*fn*)
aspect oriented way to check if provider is internal when calling the decorated function

Parameters **fn** – the decorated function

Returns around advice

Raises pyramid.httpexceptions.HTTPMethodNotAllowed when provider is not internal

6.6 atramhasis.validators

Module that validates incoming JSON.

atramhasis.validators.**broader_hierarchy_build**(*skos_manager*, *conceptscheme_id*, *broader*, *broader_hierarchy*)
Builds a list of all the broader concepts of a list of concepts.

atramhasis.validators.**broader_hierarchy_rule**(*errors*, *node_location*, *skos_manager*, *conceptscheme_id*, *cstruct*)
Checks that the broader concepts of a concepts are not alreadt narrower concepts of that concept.

atramhasis.validators.**collection_members_unique_rule**(*errors*, *node_location*, *members*)
Checks that a collection has no duplicate members.

atramhasis.validators.**collection_type_rule**(*errors*, *node_location*, *skos_manager*, *conceptscheme_id*, *members*)
Checks that the targets of member_of are collections and not concepts.

atramhasis.validators.**concept_exists_andnot_different_conceptscheme_rule**(*errors*, *node_location*, *skos_manager*, *conceptscheme_id*, *members*)
Checks that the members of a collection actually exist and are within the same conceptscheme.

atramhasis.validators.**concept_matches_rule**(*errors*, *node_location*, *matches*, *concept_type*)
Checks that only concepts have matches.

atramhasis.validators.**concept_matches_unique_rule**(*errors*, *node_location*, *matches*)
Checks that a concept has not duplicate matches.

This means that a concept can only have one match (no matter what the type) with another concept. We don't allow eg. a concept that has both a broadMatch and a relatedMatch with the same concept.

atramhasis.validators.**concept_relations_rule**(*errors*, *node_location*, *relations*, *concept_type*)
Checks that only concepts have narrower, broader and related relations.

atramhasis.validators.**concept_schema_validator**(*node*, *cstruct*)
This validator validates an incoming concept or collection

This validator will run a list of rules against the concept or collection to see that there are no validation rules being broken.

Parameters

- **node** (*colander.SchemaNode*) – The schema that's being used while validating.
- **cstruct** – The concept or collection being validated.

atramhasis.validators.**concept_type_rule**(*errors*, *node_location*, *skos_manager*, *conceptscheme_id*, *items*)
Checks that the targets of narrower, broader and related are concepts and not collections.

atramhasis.validators.**label_lang_rule**(*errors*, *node*, *languages_manager*, *labels*)
Checks that languages of a label are valid.

Checks that they are valid IANA language tags. If the language tag was not already present in the database, it adds them.

atramhasis.validators.**label_type_rule**(*errors*, *node*, *skos_manager*, *labels*)
Checks that a label has the correct type.

atramhasis.validators.**languagetag_checkduplicate**(*node*, *language_tag*, *languages_manager*, *errors*)
Check that a languagetag isn't duplicated.

atramhasis.validators.**languagetag_isvalid_rule**(*node*, *language_tag*, *errors*)
Check that a languagetag is a valid IANA language tag.

atramhasis.validators.**languagetag_validator**(*node*, *cstruct*)
This validator validates a languagetag.

The validator will check if a tag is a valid IANA language tag. The the validator is informed that this should be a new language tag, it will also check if the tag doesn't already exist.

Parameters

- **node** (*colander.SchemaNode*) – The schema that's being used while validating.
- **cstruct** – The value being validated.

atramhasis.validators.**max_preflabels_rule**(*errors*, *node*, *labels*)
Checks that there's only one prefLabel for a certain language.

atramhasis.validators.**members_hierarchy_rule**(*errors*, *node_location*, *skos_manager*, *conceptscheme_id*, *cstruct*)
Checks that a collection does not have members that are in themselves already "parents" of that collection.

atramhasis.validators.**members_only_in_collection_rule**(*errors*, *node*, *concept_type*, *members*)
Checks that only collections have members.

atramhasis.validators.**min_labels_rule**(*errors*, *node*, *cstruct*)
Checks that a label or collection always has a least one label.

atramhasis.validators.**narrower_hierarchy_build**(*skos_manager*, *conceptscheme_id*, *narrower*, *narrower_hierarchy*)
Builds a list of all the narrower concepts of a list of concepts.

atramhasis.validators.**narrower_hierarchy_rule**(*errors*, *node_location*, *skos_manager*, *conceptscheme_id*, *cstruct*)
Checks that the narrower concepts of a concept are not already broader concepts of that concept.

atramhasis.validators.**subordinate_arrays_hierarchy_rule**(*errors*, *node_location*, *skos_manager*, *conceptscheme_id*, *cstruct*)
Checks that the subordinate arrays of a concept are not themselves parents of that concept.

atramhasis.validators.**subordinate_arrays_only_in_concept_rule**(*errors*, *node*, *concept_type*, *subordinate_arrays*)
Checks that only a concept has subordinate arrays.

atramhasis.validators.**subordinate_arrays_type_rule**(*errors*, *node_location*, *skos_manager*, *conceptscheme_id*, *subordinate_arrays*)
Checks that subordinate arrays are always collections.

atramhasis.validators.**superordinates_hierarchy_rule**(*errors*, *node_location*, *skos_manager*, *conceptscheme_id*, *cstruct*)
Checks that the superordinate concepts of a collection are not themselves members of that collection.

atramhasis.validators.**superordinates_only_in_concept_rule**(*errors*, *node*, *concept_type*, *superordinates*)
Checks that only collections have superordinates.

atramhasis.validators.**superordinates_type_rule**(*errors*, *node_location*, *skos_manager*, *conceptscheme_id*, *superordinates*)
Checks that superordinates are always concepts.

6.7 atramhasis.views

6.7.1 atramhasis.views.views

class atramhasis.views.views.**AtramhasisAdminView**(*request*)
This object groups HTML views part of the admin user interface.

class atramhasis.views.views.**AtramhasisListView**(*request*)
This object groups list views part for the user interface.

class atramhasis.views.views.**AtramhasisView**(*request*)
This object groups HTML views part of the public user interface.

concept_view()
This view displays the concept details

Parameters **request** – A pyramid.request.Request

conceptscheme_view()
This view displays conceptscheme details.

Parameters **request** – A pyramid.request.Request

conceptschemes_view()
This view displays a list of available conceptschemes.

Parameters **request** – A pyramid.request.Request

favicon_view()
This view returns the favicon when requested from the web root.

Parameters **request** – A pyramid.request.Request

home_view()
This view displays the homepage.

Parameters **request** – A pyramid.request.Request

search_result()
This view displays the search results

Parameters **request** – A pyramid.request.Request

set_locale_cookie()
This view will set a language cookie

Parameters **request** – A pyramid.request.Request

6.7.2 atramhasis.views.crud

Module containing views related to the REST service.

class atramhasis.views.crud.**AtramhasisCrud**(*context*, *request*)
This object groups CRUD REST views part of the private user interface.

get_concept()
Get an existing concept

Raises atramhasis.errors.ConceptNotFoundException If the concept can't be found

6.7.3 atramhasis.views.exception_views

Module containing error views.

atramhasis.views.exception_views.**data_integrity**(*exc*, *request*)
View invoked when IntegrityError was raised.

atramhasis.views.exception_views.**failed**(*exc*, *request*)
View invoked when bad data was submitted to Atramhasis.

atramhasis.views.exception_views.**failed_not_found**(*exc*, *request*)
View invoked when a resource could not be found.

atramhasis.views.exception_views.**failed_skos**(*exc*, *request*)
View invoked when Atramhasis can't find a SKOS registry.

atramhasis.views.exception_views.**failed_validation**(*exc*, *request*)
View invoked when bad data was submitted to Atramhasis.

atramhasis.views.exception_views.**protected**(*exc*, *request*)
when a protected operation is called on a resource that is still referenced

atramhasis.views.exception_views.**provider_unavailable**(*exc*, *request*)
View invoked when ProviderUnavailableException was raised.

CHAPTER 7

History

7.1 0.4.3 (11-03-2015)

We had some packaging issues with the *0.4.2* release.

7.2 0.4.2 (11-03-2015)

This release of Atramhasis is mostly a bugfix update of the *0.4.1* release.

- Fix paths of db in scaffolds
- Add more information on exceptions
- Update skosprovider_getty and skosprovider_heritagedata (fix the problems when importing external thesauri)
- Documentation update

7.3 0.4.1 (04-03-2015)

This release of Atramhasis is a minor update of the *0.4.0* release, focussing on small corrections and improvements and improving the documentation. A few interesting non-invasive features were added, mostly to the editor's admin interface and machine-readable exports of RDF data.

Upgrading from *0.4.0* should be simple and cause no or few problems.

- A conceptscheme, concept or collection can now be exported to RDF through skosprovider_rdf 0.3.1. These are individuals export endpoints that can be reached in one of two ways. Either by hitting a url like http://localhost:6543/conceptschemes/GEOGRAPHY/c/335 with a supported RDF mimetype (`application/rdf+xml`, `application/x-turtle`, `text-turle`). Or by using an RDF syntax specific suffix (.rdf or .ttl).
- When importing, allow the user to request more information on a concept or collection, before actually importing it.
- Allow merging a concept with other concepts it matches. This allows a user to compare a local concept with an external one it matches and import any notes or labels that are present in the external concept, but not the local one.
- Reworked some parts of the public interface to make everything a bit clearer and to make all pages easily reachable.

- Allow sorting the languages in the admin interface.
- Reorganised and extended the right click menu on the grid in the admin interface.
- Allow looking up a *skos:match* from within the admin interface.
- Some issues with the length of language ids were solved.
- Fixed some issues when importing a collection instead of a concept.
- Made it easy to add a Google Analytics tracker.
- Added instructions on how to deploy a demo site on heroku. These work just as well for deploying an actual production site to heroku.
- Lots of small updates and tweaks to the documentation.
- Updated some dependencies.
- Some code cleanup and reorganisation. Several smaller bugs in the admin interface were fixed.
- The data fixtures were updated with *skos:note* examples. Added a license for reuse of the fixture data.

7.4 0.4.0 (23-12-2014)

- Update to skosprovider 0.5.0. Among other things, this makes it possible to handle relations between Concepts and Collections using the *subordinate_arrays* and *superordinates* properties. Conceptschemes are now also much better integrated within the providers, thus making it possible to provider more context for a Concept. This version of skosprovider can also handle *skos:matches*.
- Add possibility to edit language tags. It's now possible to use the admin interface to add, edit and delete languages in Atramhasis.
- When the REST service receives labels or notes in currently unavailable languages, it will validate those through language_tags. It the languages are valid according to the IANA registry, they will be added to the languages available in the application.
- Default length of language id changed to 64 characters. This is not available as an alembic migration. So only effective when creating a new database. If you already have a database created from an older version of Atramhasis, please modify accordingly. Modifying column length on SQLite is not possible (see http://www.sqlite.org/omitted.html).
- Abiltity to match Concepts in an Atramhasis ConceptScheme to Concepts in external ConceptSchemes through properties such as *skos:exactMatch* and *skos:closeMatch*.
- Ability to import Concepts and Collections from external providers. This makes it possible to import Concepts from eg. the AAT (via skosprovider_getty), Flanders Heritage Thesauri (via skosprovider_oe), English Heritage Thesauri (via skosprovider_heritagedata) or any other SKOS vocabulary for which a skosprovider has been written. Currently only the concept or collection itself can be imported, without its relations to other concepts or collections.
- Add the ability to have a delete of a concept or collection fail if it is being used in other systems.
- Implement a delete permission.
- Add validation rule that a Concept must have at least one label.
- Update to skosprovider_sqlalchemy 0.4.1.
- Update to pyramid_skosprovider 0.5.0.
- Update to skosprovider_rdf 0.3.0. This update adds support for dumping ConceptScheme in an RDF file and also handles *subordinate_arrays* and *superordinates*.

- Update to language_tags 0.3.0.

7.5 0.3.1 (05-09-2014)

- Update to skosprovider_sqlalchemy 0.2.1.
- Update to skosprovider_rdf 0.1.3 This fixes an issue with RDF having some SKOS elements in the wrong namespace. Also added a missing dependency on skosprovider_rdf to setup.py
- Updated the Travis build file to run a basic dojo build and test for build failures.

7.6 0.3.0 (15-08-2014)

- Atramhasis now includes a working admin userinterface at */admin*. Still needs some polish when it comes to error handling and reporting about validation errors.
- The admin module gets run through a dojo build to minimize page loads and download times
- Added RDF/XML en RDF/Turtle downloads to the public interface. Currently only dumps a full conceptscheme, not individual concepts.
- Added more docs.

7.7 0.2.0 (16-05-2014)

- Full public userinterface
- REST CRUD service
- Security integration
- CSV export
- demo using Mozilla Persona as sample security setup

7.8 0.1.0 (22-04-2014)

- Initial version
- Setup of the project: docs, unit testing, code coverage
- Scaffolding for demo and deployment packages
- Limited public user interface
- Basis i18n abilities present
- Integration of pyramid_skosprovider
- Integration of skosprovider
- Integration of skosprovider_sqlalchemy

CHAPTER 8

Glossary

CSS Cascading Style Sheet is a style specification used to add style and presentation to webpages.

HTML HyperText Markup Language is the markup language used to create webpage.

Jinja2 Jinja2 is a python templating engine. It's used by Atramhasis for rendering *HTML* templates.

Pyramid This webframework was used to implement the server side components of Atramhasis.

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.

REST REST or *REpresentational State Transfer* is a way of data exchange that is very complimentary to the operations of the HTTP protcol.

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

SKOS-THES The ISO 25964 SKOS extension defines mappings between the ISO 25964 standard and the *SKOS* specification.

SOA Service Oriented Architecture.

URI A *Uniform Resource Identifier*.

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

CHAPTER 9

Indices and tables

- *genindex*
- *modindex*
- *search*

/conceptschemes

POST /conceptschemes/{scheme_id}/c, 11
PUT /conceptschemes/{scheme_id}/c/{c_id},
13
DELETE /conceptschemes/{scheme_id}/c/{c_id},
14

a

atramhasis.data.datamanagers, 25
atramhasis.data.db, 27
atramhasis.errors, 27
atramhasis.mappers, 28
atramhasis.protected_resources, 28
atramhasis.utils, 28
atramhasis.validators, 29
atramhasis.views.crud, 32
atramhasis.views.exception_views, 32
atramhasis.views.views, 31

A

atramhasis.data.datamanagers (module), 25
atramhasis.data.db (module), 27
atramhasis.errors (module), 27
atramhasis.mappers (module), 28
atramhasis.protected_resources (module), 28
atramhasis.utils (module), 28
atramhasis.validators (module), 29
atramhasis.views.crud (module), 32
atramhasis.views.exception_views (module), 32
atramhasis.views.views (module), 31
AtramhasisAdminView (class in atramhasis.views.views), 31
AtramhasisCrud (class in atramhasis.views.crud), 32
AtramhasisListView (class in atramhasis.views.views), 31
AtramhasisView (class in atramhasis.views.views), 31

B

broader_hierarchy_build() (in module atramhasis.validators), 29
broader_hierarchy_rule() (in module atramhasis.validators), 29

C

collection_members_unique_rule() (in module atramhasis.validators), 29
collection_type_rule() (in module atramhasis.validators), 29
concept_exists_andnot_different_conceptscheme_rule() (in module atramhasis.validators), 29
concept_matches_rule() (in module atramhasis.validators), 29
concept_matches_unique_rule() (in module atramhasis.validators), 29
concept_relations_rule() (in module atramhasis.validators), 29
concept_schema_validator() (in module atramhasis.validators), 29
concept_type_rule() (in module atramhasis.validators), 29
concept_view() (atramhasis.views.views.AtramhasisView method), 31
ConceptNotFoundException, 27
conceptscheme_view() (atramhasis.views.views.AtramhasisView method), 31
ConceptSchemeManager (class in atramhasis.data.datamanagers), 25
ConceptSchemeNotFoundException, 27
conceptschemes_view() (atramhasis.views.views.AtramhasisView method), 31
CSS, **37**

D

data_integrity() (in module atramhasis.views.exception_views), 32
data_managers() (in module atramhasis.data.db), 27
DataManager (class in atramhasis.data.datamanagers), 26
DbNotFoundException, 27
delete() (atramhasis.data.datamanagers.LanguagesManager method), 26
delete_thing() (atramhasis.data.datamanagers.SkosManager method), 26
desc() (in module atramhasis.data.datamanagers), 26

F

failed() (in module atramhasis.views.exception_views), 32
failed_not_found() (in module atramhasis.views.exception_views), 32
failed_skos() (in module atramhasis.views.exception_views), 32
failed_validation() (in module atramhasis.views.exception_views), 32
favicon_view() (atramhasis.views.views.AtramhasisView method), 31
find() (atramhasis.data.datamanagers.ConceptSchemeManager method), 25

from_thing() (in module atramhasis.utils), 28

G

get() (atramhasis.data.datamanagers.ConceptSchemeManager method), 25
get_all() (atramhasis.data.datamanagers.ConceptSchemeManager method), 25
get_all() (atramhasis.data.datamanagers.LanguagesManager method), 26
get_all_sorted() (atramhasis.data.datamanagers.LanguagesManager method), 26
get_by_list_type() (atramhasis.data.datamanagers.SkosManager method), 26
get_collections_for_scheme_tree() (atramhasis.data.datamanagers.ConceptSchemeManager method), 25
get_concept() (atramhasis.views.crud.AtramhasisCrud method), 32
get_concepts_for_scheme_tree() (atramhasis.data.datamanagers.ConceptSchemeManager method), 25
get_thing() (atramhasis.data.datamanagers.SkosManager method), 26

H

home_view() (atramhasis.views.views.AtramhasisView method), 31
HTML, **37**

I

includeme() (in module atramhasis.data.db), 27
internal_providers_only() (in module atramhasis.utils), 28

J

Jinja2, **37**

L

label_lang_rule() (in module atramhasis.validators), 29
label_type_rule() (in module atramhasis.validators), 30
LanguageNotFoundException, 27
LanguagesManager (class in atramhasis.data.datamanagers), 26
languagetag_checkduplicate() (in module atramhasis.validators), 30
languagetag_isvalid_rule() (in module atramhasis.validators), 30
languagetag_validator() (in module atramhasis.validators), 30

M

map_concept() (in module atramhasis.mappers), 28
max_preflabels_rule() (in module atramhasis.validators), 30
members_hierarchy_rule() (in module atramhasis.validators), 30
members_only_in_collection_rule() (in module atramhasis.validators), 30
min_labels_rule() (in module atramhasis.validators), 30

N

narrower_hierarchy_build() (in module atramhasis.validators), 30
narrower_hierarchy_rule() (in module atramhasis.validators), 30

P

protected() (in module atramhasis.views.exception_views), 32
protected_operation() (in module atramhasis.protected_resources), 28
ProtectedResourceEvent (class in atramhasis.protected_resources), 28
ProtectedResourceException, 28
provider_unavailable() (in module atramhasis.views.exception_views), 32
Pyramid, **37**

R

RDF, **37**
REST, **37**

S

save() (atramhasis.data.datamanagers.LanguagesManager method), 26
save() (atramhasis.data.datamanagers.SkosManager method), 26
search_result() (atramhasis.views.views.AtramhasisView method), 31
set_locale_cookie() (atramhasis.views.views.AtramhasisView method), 31
SKOS, **37**
SKOS-THES, **37**
SkosManager (class in atramhasis.data.datamanagers), 26
SkosRegistryNotFoundException, 27
SOA, **37**
subordinate_arrays_hierarchy_rule() (in module atramhasis.validators), 30
subordinate_arrays_only_in_concept_rule() (in module atramhasis.validators), 30
subordinate_arrays_type_rule() (in module atramhasis.validators), 30
superordinates_hierarchy_rule() (in module atramhasis.validators), 30

superordinates_only_in_concept_rule() (in module atramhasis.validators), 31
superordinates_type_rule() (in module atramhasis.validators), 31

U

URI, **37**
URN, **37**

V

ValidationError, 27