This document is part of the Coordination and Support Action CRACKER. This project has received funding from the European Union's Horizon 2020 program for ICT through grant agreement no.: 645357.

Deliverable D3.6

New Version of META-SHARE Software (Update)

Authors: Penny Labropoulou, Miltos Deligiannis, Juli Bakagianni, Stelios Piperidis

Dissemination Level: Public

Date: 20 December 2016

Status: Final

Grant agreement no.	645357
Project acronym	CRACKER
Project full title	Cracking the Language Barrier
Type of action	Coordination and Support Action
Coordinator	Dr. Georg Rehm (DFKI)
Start date, duration	1 January 2015, 36 months
Dissemination level	Public
Contractual date of delivery	31/12/2016
Actual date of delivery	20/12/2016
Deliverable number	D3.6
Deliverable title	New version of META-SHARE software
Type	Other (software); Report
Status and version	Final
Number of pages	20
Contributing partners	ELDA
WP leader	ATHENA RC
Task leader	ATHENA RC
Author(s)	Penny Labropoulou (ATH), Miltos Deligiannis (ATH), Juli Bakagianni (ATH), Stelios Piperidis (ATH)
Internal reviewers	Georg Rehm (DFKI)
EC project officer	Susan Fraser (M19-M36), Pierre-Paul Sondag (M01-M18)
The partners in CRACKER are:	• Deutsches Forschungszentrum für Künstliche Intelligenz GmbH (DFKI), Germany • Charles University in Prague (CUNI), Czech Republic • Evaluations and Language Resources Distribution Agency (ELDA), France • Fondazione Bruno Kessler (FBK), Italy • Athena Research and Innovation Center in Information, Communication and Knowledge Technologies (ATHENA RC), Greece • University of Edinburgh (UEDIN), UK • University of Sheffield (USFD), UK

For copies of reports, updates on project activities, and other CRACKER-related information, contact:

DFKI GmbH
CRACKER
Dr. Georg Rehm
Alt-Moabit 91c
D-10559 Berlin, Germany

georg.rehm@dfki.de
Phone: +49 (0)30 23895-1833
Fax: +49 (0)30 23895-1810

Copies of reports and other material can also be accessed via http://cracker-project.eu.

Contents

1 Introduction 4

2 Web framework upgrade 4

3 Licensing module updates 4

4 Metadata schema updates 5

5 Improvements in the search functionality 5

6 Annex 6

1 Introduction

One of the objectives of the CRACKER project is to coordinate and support the resource sharing activities underpinning high-quality multilingual technology research and development, and to build bridges with concurrently running activities preparing multilingual digital service infrastructure(s). Resource sharing builds upon, maintains and extends the existing META-SHARE resource infrastructure, which is available at http://www.meta-share.eu and http://www.meta-share.org.

The extended and improved version of the META-SHARE platform (M24) is available through Github at https://github.com/metashare/META-SHARE/tree/metashare-3.1.1, and it comes with detailed instructions on how to install, upgrade and use the software. The actual manual, as presented on github, is annexed to this document.

The current report documents changes and updates of the META-SHARE software and platform, and concerns the following aspects:

- web framework upgrade
- licensing scheme
- schema updates and improvements
- improved search functionality

During the next months, we will proceed with the migration of the META-SHARE nodes to the new version of the platform. The migration mechanism to perform the required conversion of the metadata records has been implemented and the actions needed are described in the installation manual.

2 Web framework upgrade

Since the META-SHARE infrastructure is built upon the Django Web Framework, a major upgrade has been carried out, migrating from Django V1.3, which is used in META-SHARE V3.0, to version Django V1.7.11, which solves several security issues and still has considerable community support.

The relevant unit tests of the infrastructure have been adjusted to the new framework and run in both SQlite and PostgreSQL databases.

3 Licensing module updates

An important ingredient in the META-SHARE infrastructure is the licensing module that makes clear to LR users the terms under which they can use them.

- Creative Commons licences v4.0 have been adopted for content resources http://www.meta-share.org/p/91/Licences
- Free and Open Source-Software (FOSS) licences have been adopted for tools and web services
- META-SHARE NoRedistribution Licences v.2.0 are now available http://www.meta-share.org/p/91/Licences
- META-SHARE Commons (MSCommons) licences are still available but their use is discouraged in favour of their CC or NoReD equivalents.

4 Metadata schema updates

The changes and enhancements of the new META-SHARE version regarding the metadata schema and the browser reflect requests of resource providers and points of improvement that have been observed during the operation of the infrastructure.

- Correction of spelling mistakes, mainly in values of elements
- Addition of values for elements
- Change of cardinality from single to multiple for components and elements, as long as they didn't create problems for the software operation
- Addition of elements, such as *ISLRN* which has been introduced as an identifier especially for language resources (see http://www.islrn.org)
- Replacement of free text values with controlled vocabularies for certain elements and, more specifically, *country*, *mimetype* and *language*-related elements (e.g., *documentLanguage*), following ISO standards and established practices
- Improvements for the licensing module, in two directions:
  - Update of licences (i.e., update of *licence* values and texts to the CC 4.0 and MSNoReD 2.0 versions),
  - Improvement of the presentation of licensing conditions, by adding two elements that allow LR providers to include the detailed texts of non-standard licences and terms of use/service, or point to URL links thereof;
- Improvements of the appearance of metadata records:
  - a mechanism for representing the *licence* and *conditions of use* in the form of icons on the browsing page
  - highlighting of the *attribution text* and *citation paper* for the resource (if they include such information) on a prominent place in the "view record" page.

5 Improvements in the search functionality

The existing indexing and search functionality has been improved in the following dimensions:

- the Solr indexer is now endowed with a list of stopwords for English
- the search module is enriched with a query expansion module, which takes into account relevant synonym lists, alternative forms of frequently used query terms
- the ranking of the results is now based on field boosting, i.e., it prioritises fields like *resource name*, *resource short name, resource description*, etc.

6 Annex

META-SHARE Installation Manual

Executive Summary

This document is a guide for installing META-SHARE V3.1.1. It is intended for system administrators setting up META-SHARE nodes. It also contains a section on how to upgrade an existing META-SHARE V3.0.x installation to V3.1.1.

Backing up META-SHARE

META-SHARE stores the information in two places: a database and a storage directory. Please follow these instructions to back up both things before any upgrade, to revert to the previous version in case of problems.

Before backing up, stop the server to prevent database modifications during the backup process. The server can be stopped using the `stop-server.sh` script.

The storage directory is defined in the `STORAGE_PATH` variable in the `metashare/local_settings.py` file of your current META-SHARE installation. Create a backup of that directory.

If you are using a PostgreSQL database, you can use the `pg_dump` command to create a dump of the META-SHARE database. The database name, host and port are stored in the `DATABASES` variable in `metashare/local_settings.py`.

```
pg_dump metashare_db_name > metashare_database_backup.sql
```

Make sure the database dump you have created has contents (is not empty) and no error appears. Check the PostgreSQL Documentation: SQL Dump for further details if needed.

If instead of PostgreSQL you are using the SQLite database, you can just copy the SQLite file pointed in the `DATABASES` variable. Be aware that the SQLite database is not recommended on production servers.

Upgrading META-SHARE

If you have an installation of META-SHARE previous to V3.0 and you would like to migrate your resource descriptions, uploaded resource data, user accounts and statistics to a new META-SHARE installation, then you should first upgrade to V3.0.x using the Installation Manual that comes with a V3.0.x release. When you have such an installation working (at least as a development server), you can follow the migration instructions to the most recent version in the remainder of this chapter.

Before diving into the actual migration, it should be noted that two META-SHARE installations on the same machine can interfere with each other if not configured properly. So in case you would like to upgrade an installation on a single machine,

we recommend to shut down the old installation (including the Solr server) first unless you know what you are doing.

Here are now the steps you should follow for a successful migration:

1. Make sure your META-SHARE-3.0.x instance is stopped.
2. If you have not done it already, go through Backing up META-SHARE_.
3. Follow the Installing META-SHARE_ section on a new directory.

   Note: during the installation you can skip the creation of a database and a role for PostgreSQL, as you already have one in your current installation.
4. Make sure your META-SHARE instances are stopped (no development server running).
5. Copy `/path/to/old/MetaShareNode-3.0/metashare/local_settings.py` to `/path/to/MetaShareNode-3.1.1/metashare/local_settings.py`
6. Edit the `local_settings.py`. Add a `SECRET_KEY` variable and a `ALLOWED_HOSTS` variable. See Local Settings for META-SHARE Nodes_ for information on how to generate it.
7. If you migrate from V3.0.x to V3.1.1, then you have to upgrade your metadata descriptions to the new version of the META-SHARE xsd schema, that is META-SHARE xsd schema V3.1. Hence, go to the `/path/to/local/MetaShareNode/` folder and run: :

   ```
   ./misc/tools/migration/to_3_1/migrate_to_3_1_MS_schema.sh
   ```
8. Collect static files to the `STATIC_ROOT` folder by running the command: :: source venv/bin/activate python manage.py collectstatic deactivate
9. Adapt any customization you had on the old `start-server.sh`, `stop-server.sh` scripts into the new script version.
10. Start your new META-SHARE instance using the `start-server.sh` script.

Installing META-SHARE

This section explains how to download and install META-SHARE V3.1.1 and its dependencies.

Start by downloading META-SHARE from the download page.

Extract the downloaded software into a designated META-SHARE folder, e.g., `/path/to/local/MetaShareNode/`.

Software Dependencies

Database Software

Note: if you just want to run META-SHARE in **development mode**, or if you are upgrading META-SHARE you can skip the database setup.

We currently use SQLite or PostgreSQL as our database software. SQLite comes built-in with Python 2.7. Since SQLite has a number of limitations, including missing transaction management and access permission management, the preferred database is PostgreSQL. We have tested `PostgreSQL 9.0.5`.

On Debian, Ubuntu and derivatives:

Install PostgreSQL with:

```
sudo apt-get update
sudo apt-get install postgresql postgresql-contrib
```

Create a user named `metashare_user` (choose any name you like) for META-SHARE:

```
sudo su – postgres
createuser -W metashare_user
```

Create a database `metashare_db` (or any other name), owned by the just created user `metashare_user` (or the name you chose above):

```
sudo su – postgres
createdb --owner=metashare_user metashare_db
```

Python interpreter

Note: If you are upgrading from a previous META-SHARE installation AND python-2.7 was installed during the previous installation in `/path/to/old/MetaShareNode3.0/` please make sure to remove from your `PATH` variable `/path/to/old/MetaShareNode3.0/opt/bin`. No path modifications are required anymore.

META-SHARE V3.1.1 requires Python 2.7. Most Linux/Unix distributions come already with a preinstalled version of Python. You may check the installed python version with `python2 --version`.

In case the output is something like "2.7.x", nothing else needs to be done.

If you have a previous python version, python 2.7 will be installed to `/path/to/MetaNode/opt` during the META-SHARE installation. To do so, you will need to install some dependencies, such as `libsqlite3-dev`, `libssl-dev` and `zlib1g-dev`. Please note that these packages may have different names depending on your Linux/Unix distribution.

On an older Ubuntu without Python 2.7 you might also use the following command to get all required build dependencies:

```
apt-get build-dep python2.6
```

Python Modules

Since V3.0.3, META-SHARE does not bundle anymore all the python dependencies. Instead of doing that, we follow the standard way of working with python apps, based on virtualenv and pip. Virtualenv allows us to create isolated python environments, preventing conflicts between coexisting python applications with different dependencies. Pip is the recommended tool to install python packages.

The `psycopg2` python module is used for connecting PostgreSQL to META-SHARE. In order to build this module, header files for the PostgreSQL library `libpq5` have to be installed, as well as the python headers. On Debian, Ubuntu and derivatives, this can be achieved installing the `libpq-dev` and `python-dev` packages using `apt-get install libpq-dev python-dev`.

Since V3.1. we use the `lxml` XML toolkit. lxml requires libxml2 and libxslt to be installed. To install the required development packages of these dependencies on Debian, Ubuntu and derivatives use `apt-get install libxml2-dev libxslt-dev`

Once this header files are installed, the rest of the dependencies can be installed simply by:

```
cd "/path/to/local/MetaShareNode/"
./install-dependencies.sh
```

This script will:

1. Check that Python 2.7 is installed, or download and install it to `/path/to/local/MetaShareNode/opt/bin`.
2. Download virtualenv
3. Create a virtual environment at `/path/to/local/MetaShareNode/venv`.
4. Download, build and install all META-SHARE dependencies using pip in the created virtual environment.

If everything is installed successfully the message `Installation of META-SHARE dependencies complete.` should appear in the end.

For your information, the dependencies and their respective versions are listed in the requirements.txt file.

Web Server

Note: if you just want to run META-SHARE in **development mode**, you can skip the web server setup.

META-SHARE is a web application that builds on a web server. Deployment has been tested with `lighttpd 1.4.33` via FastCGI. Other web servers can be used, but you do so on your own risk.

We strongly recommend to set up your web server so that it only serves SSL encrypted connections. We are shipping a sample configuration for `lighttpd` under `metashare/lighttpd-ssl.conf.sample` which should give you an idea on how to set this up.

Development Server

To verify that you have installed all dependencies correctly, you should first set up a development server. Proceed as follows.

1. Create `local_settings.py` for your local META-SHARE node:

   ```
   cp metashare/local_settings.sample metashare/local_settings.py
   ```

Edit at least the following constants: `DJANGO_URL`, `DJANGO_BASE`, `STORAGE_PATH`, `STATIC_ROOT`, `DEBUG`, `SECRET_KEY`, `ADMINS`, `DATABASES`, and `EMAIL_BACKEND`. More information is available in Local Settings for META-SHARE Nodes_

Note: If you are upgrading from a previous META-SHARE version, make sure to NOT use your production `STORAGE_PATH` or your production database in `local_settings.py` for testing the installation.

2. Initialize database contents:

```
source venv/bin/activate # enables META-SHARE virtual environmen
t
python manage.py migrate
deactivate  # disables META-SHARE virtual environment
```

3. Create an admin user: :

```
source venv/bin/activate
python manage.py createsuperuser
deactivate
```

4. Start an Apache Solr server for the search index (uses Java and Python internally):

```
metashare/start-solr.sh
```

5. Run tests to check that Django can load and serve META-SHARE:

```
source venv/bin/activate
python manage.py test repository storage accounts sync stats  bc
p47
deactivate
```

This should return “OK”.

Note: This step may take a few minutes.

6. Run a Django development server:

```
source venv/bin/activate
python manage.py runserver
  Validating models...
  0 errors found
  Django version 1.4.x, using settings 'metashare.settings'
  Development server is running at http://127.0.0.1:8000/
  Quit the server with CONTROL-C.
deactivate
```

Congratulations: you have successfully started a META-SHARE V3.1.1 node in development mode. This means that all required Python and Django dependencies are functioning correctly.

Local Settings for META-SHARE Nodes

Note: If you are upgrading a META-SHARE installation, you can now follow the rest of the Upgrading META-SHARE_ instructions as the `local_settings.py` file will be copied from the previous META-SHARE installation.

Django projects usually store all their configuration settings in a file named settings.py. For META-SHARE, we have split up the set of configuration parameters into two groups: local and global settings. You should never have to change the *global* settings in `settings.py` as they are neither security-critical nor node-dependant. You can and partially have to change local configuration settings, though, which are stored in their own file named `local_settings.py`.

The META-SHARE software package only contains a file named `local_settings.sample` that lists and explains all local settings available for META-SHARE nodes. You have to create a node-local copy of this sample file with the name `local_settings.py` and adapt some configuration settings.

The local settings are the following:

- `DJANGO_URL = 'http://www.example.com/path/to/metashare'`

  The URL for this META-SHARE node as it is reachable from the internet; it is important to emphasize that this must not be any internal URL which is only reachable behind some proxy server! Do not use a trailing slash (/)!You can use `http://127.0.0.1:8000` when running a development mode server.

- `DJANGO_BASE = 'path/to/metashare/'`

  The base path under which Django is deployed at `DJANGO_URL`. Use a trailing slash(/). Do not use a leading slash, though. Leave empty if META-SHARE is deployed directly under the given `DJANGO_URL`.

- `FORCE_SCRIPT_NAME = ""`

  This is required when the META-SHARE node is deployed using FastCGI and for example lighttpd. There is a known bug with FCGI hosted applications and lighttpd; it basically messes up the URL after HTTP submits.
  `FORCE_SCRIPT_NAME= ""` fixes the issue and hence is required for lighttpd use.

- `ALLOWED_HOSTS = [ 'www.example.com' ]`

  A list of strings representing the host/domain names this META-SHARE instance can be served at.

- `STORAGE_PATH = '/path/to/storage/path'`

  Absolute path to the local storage base, i.e., the folder in which your language resource data is stored. You need to supply an existing path here, even for development mode! This folder will contain data related to your language resources, so choose a suitable location that is accessible, safe and that has sufficient free space for all resource data that you would like to upload.

- `STATIC_ROOT = '/path/to/static/path'`

  Absolute path to the directory where collectstatic will collect static files for deployment.

- `DEBUG, TEMPLATE_DEBUG, DEBUG_JS`

  Debug settings: setting `DEBUG=True` will give exception stacktraces on the website, for example. This may include sensitive information, so use with care, preferably only for local development servers.

- `SECRET_KEY`

  Set this variable to a random value. This is used by django to salt the passwords stored in the database and generate tokens. See SECRET_KEY django documentation for further information. The following python code can help you to generate a random string:

```
# From: https://gist.github.com/mattseymour/9205591
import string, random
chars = ''.join([string.ascii_letters, string.digits,
string.punctuation]).replace('\'', '').replace('"',
'').replace('\\', '')
print ''.join([random.SystemRandom().choice(chars) for i in
range(50)])
```

- `ADMINS`

  Configure the administrators for this Django project. If `DEBUG=False`, all errors will be reported as e-mails to these persons. If you do not set any administrators here, you will

  (a) not get any notifications of problems with the META-SHARE site; and (b) not be able to get useful feedback from the META-SHARE technical helpdesk if you should run into internal server errors 500).

- `DATABASES`

  Configures the database settings for Django. For SQLite, use the following settings:

```
DATABASES = {
     'default': {
         'ENGINE': 'django.db.backends.sqlite3',
         'NAME': '{0}/testing.db'.format(ROOT_PATH)
     }
}
```

  For PostgreSQL, the following settings are required:

```
DATABASES = {
     'default': {
         'ENGINE': 'django.db.backends.postgresql_psycopg2',
         'NAME': 'metashare',
         'USER': 'db_user',
         'PASSWORD': 'db_password',
         'HOST': 'localhost',
         # Set to empty string for default.
         'PORT': '',
```

```
        }
    }
```

- `EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'`

  Settings for sending mail. Production servers should use the SMTP e-mail backend as indicated in the `local_settings.sample` file.

- `TIME_ZONE = 'Europe/Berlin'`

  Local time zone for this installation.

- `SYNC_USERS = {'sync-user-1': 'some_password', }`

  Credentials (user name and password) for one or more user accounts with the permission to access synchronization information on the configured META-SHARE Node. If you are no META-SHARE Managing Node, then you will only need at most sync user account here. Such an account is required for linking your node to the META-SHARE Network – see Linking your Node with the META-SHARE Network_. Essentially a sync user account is a normal user account and therefore it also lives in the same namespace. Thus, a sync user account must have a different name from any existing user accounts! You always have to run `manage.py syncdb`, whenever you change the `SYNC_USERS` setting.

See also Search Engine Optimization and Web Analytics_ for further settings that can be used in the context of web analytics.

Note: settings changes will only take effect when the Django server is restarted!

Deployment

Static files

In deployment the static files should be gathered to a single directory, i.e the directory you set in the `STATIC_ROOT` setting. To collect all the static files run the management command: :: source venv/bin/activate # enables META-SHARE virtual environment python manage.py collectstatic deactivate # disables META-SHARE virtual environment

Deployment Server

For deployment, we assume that you have downloaded and installed the lighttpd web server (see also I Want to use MySQL and/or Apache_) and a `PostgreSQL database`. You have to adapt `start_server.sh` and `stop_server.sh` with correct IP addresses and port numbers. The IP addresses should be identical to the one you added to your `lighttpd.conf`, the port number, of course, needs to be different from the web server's.

You can test your PostgreSQL database by calling `manage.py syncdb`; this will complain if it cannot properly access the database.

Once both the web server and the database are ready, use `start_server.sh` to start the threaded production server via FastCGI; don't forget to set `DEBUG=False`!

`stop_server.sh` of course stops the FastCGI server and the corresponding lighttpd process.

Note: the `start_server.sh` script automatically installs some cronjobs which are required for the automatic synchronization of linked nodes, for periodic database cleanups, etc. The `stop_server.sh` script automatically uninstalls these cron jobs again.

Solr Server for Browsing and Searching

The META-SHARE release comes with a pre-configured Solr server used to index the META-SHARE database for browsing and searching.

To start the preconfigured Solr server, go to the `metashare` folder and run:

```
./start-solr.sh
```

To stop a running Solr server, go to the metashare folder and run:

```
./stop-solr.sh
```

These commands must be run by hand for the development server; they are included in the start-server.sh and stop-server.sh scripts used for the deployment server.

This should be all you need for usual operation. The following subsections are required only for people who want to understand in depth how to operate and configure the Solr server.

Installing Solr

1. Make sure you have Java 1.6 or later (run `java -version` to check!).
2. Download the latest version of Solr from here.
3. Unzip into a folder, henceforth called `$SOLR_DIR`.
4. Go to `misc/solr-config-sample` in your local META-SHARE-Software repository and run:

   ```
   ./create_solr_config.sh "$SOLR_DIR"
   ```

   This will configure your Solr server with a sample configuration. It will overwrite the default Solr configuration. After this step you will have a Solrserver which is configured with two cores (→indexes) main and testing.
5. Change directory to $SOLR_DIR/example.
6. Run

   ```
   java -jar start.jar
   ```
7. Open a web browser and go to `http://localhost:8983/solr/main/admin/`. You should be able to see Solr's admin interface for the main core.

   For further help go to the Solr Tutorial page.

Keeping the Solr Configuration Up-to-Date

As development on the search functionality continues, you may have to occasionally recreate your Solr configuration. Before doing that you have to shut down your Solr server (`Ctrl+C`). Now you can either:

- Follow the steps in the previous section. This will erase all your index data. After that, run `python manage.py rebuild_index` to rebuild your index from the current database content.
- Or you manually update the Solr configuration by going through the following steps.

Manually Updating the Solr Configuration

1. Create Solr schema files automatically by running:

   ```
   source venv/bin/activate
   python manage.py build_solr_schema
   deactivate
   ```

   The XML output of this command should go into both `$SOLR_DIR/example/solr/main/conf/schema.xml` and `$SOLR_DIR/example/solr/testing/conf/schema.xml`.

2. If there should be any changes in the files in `misc/solr-config-sample`, then copy these files to both `$SOLR_DIR/example/solr/main/conf` and `$SOLR_DIR/example/solr/testing/conf`.
3. Restart the Solr server.
4. If you already have any data in your database, then manually build the search index once. Run:

   ```
   source venv/bin/activate
   python manage.py rebuild_index
   deactivate
   ```

   Any future changes and additions to your database should automatically be reflected in the search index. A manual rebuild should not be required anymore (except when working on the indexing itself).

Linking your Node with the META-SHARE Network

Overview

META-SHARE aims to provide an infrastructure that makes language resources available in a network of many META-SHARE Nodes, the META-SHARE Network. A number of nodes with certain technical and organizational characteristics undertake the role of META-SHARE Managing Nodes. Such nodes harvest and store metadata records from the META-SHARE Nodes of the entire META-SHARE Network. META-SHARE Managing Nodes share metadata, create, host and maintain a central inventory which includes metadata-based descriptions of all language resources available in the distributed network. Each META-SHARE Managing Node effectively hosts a copy of the central inventory.

To actually link your META-SHARE Node installation with the META-SHARE Network, your node has to be proxied by a META-SHARE Managing Node. In Step-by-Step Instructions_ are detailed the steps that are required for this.

Step-by-Step Instructions

These are the steps which are required for linking your META-SHARE node with the META-SHARE Network:

- In order to give permission to a META-SHARE managing node to harvest your records, you have to create a sync user by running the following command:

```
source venv/bin/activate
python manage.py createsyncuser
deactivate
```

With this credentials the Managing node is authenticated to request your node records, harvest them and spread them to the entire record.

- Give the account credentials of your sync user and your public node URL (e.g., `http://you.example.org/metashare`) to the system administrator of the META-SHARE Managing Node which shall proxy your META-SHARE node.
- Contact either the administrator at CNR, DFKI, ELDA, FBK or ILSP (current META-SHARE Managing Node providers); never go to more than one of these META-SHARE Managing Nodes. You can use the contact form at `<MANAGING_NODE_URL>/accounts/contact/` – for example, http://metashare.ilsp.gr:8080/accounts/contact/.
- The system administrator of the chosen META-SHARE Managing Node will set up her node as a proxy for your resource descriptions.
- If all went as expected, then the chosen META-SHARE Managing Node will automatically synchronize with your node and people will be able to see (not edit!) your resource metadata on all META-SHARE Managing Nodes of the META-SHARE Network.

Importing and Exporting Resources

Metadata descriptions of language resources can be imported into the META-SHARE software from XML files obeying the META-SHARE schema format. Likewise, the metadata descriptions in the META-SHARE database can be exported into XML files in the format defined by the META-SHARE XML schema.

Importing XML Files into META-SHARE

There are two possibilities of importing language resource XML descriptions which are outlined in the following sections.

In general, all files to import should be schema-valid according to the current META-SHARE XML schema file which is located in `misc/schema/v3.1/META-SHARE-Resource.xsd`. Please use an XML schema validator to verify that the import files are valid before trying to import them into META-SHARE. For example, you can use `xmllint` like so:

```
xmllint --schema META-SHARE-Resource.xsd data.xml
```

Schema validity is not strictly required by the importer; reasonable efforts are made to import partial or erroneous XML files. However, in order to avoid loosing data, please try to make your files schema valid.

Importing from the Command Line

META-SHARE comes with a tool called `import_xml.py` to import XML files describing language resources into the system. To import, run `import_xml.py` as follows:

```
source venv/bin/activate
cd metashare
python import_xml.py <file.xml|archive.zip> [<file.xml|archive.zip>
…]
deactivate
```

In other words, you can provide one or more individual XML files or zip files containing XML files. The script will print a summary count of successfully imported and erroneous files at the end.

Importing from the Editor

An alternative way of importing resources is provided by the “Upload” menu item of the editor. There you can also provide individual XML files or zip files containing XML files. Compared to the shell importer, the upload size is limited, though.

Exporting XML Files from META-SHARE

META-SHARE aims to be an open platform and therefore allows for the export of resources in the original XML format. As with the import, there are two possible ways for exporting, both of which are described in the following sections.

Exporting from the Command Line

The script `export_xml.py` will export all entries from the database into a zip archive containing one XML file per resource. The script requires a valid META-SHARE V3.1.1 database. It can be run as follows:

```
source venv/bin/activate
cd metashare
python export_xml.py <archive.zip>
deactivate
```

The resulting archive is suitable for import in any META-SHARE V2.1 (or later) installation.

Exporting from the Editor

As an alternative to the shell exporter you may export resource descriptions from the editor.

- A single resource XML description can be exported from the main editor page of the resource using the “Export Resource Description to XML” button at the top of the page.

- A bundle of freely selectable resources may be exported as a zip archive from the "Editable Resources" page using the "Action" menu. The resulting archive is suitable for import in any META-SHARE V2.1 or later installation.

Copying Data between META-SHARE Nodes

Since V3.0, META-SHARE supports the automatic synchronization of metadata between a configurable set of META-SHARE nodes. You should usually not manually copy resource descriptions by exporting and importing. An exception might be the case where you would like to create a brand new resource description which is very similar to an existing resource description.

Setting up Editor User Accounts

For information on how to set up and manage user accounts, please see the META-SHARE Provider Manual.

Search Engine Optimization and Web Analytics

META-SHARE integrates the most common techniques for Search Engine Optimization (SEO). In order to check whether SEO works as it should, META-SHARE also integrates "django-analytical", a package for easily integrating analytics services like Google Analytics or Clicky. If you would like to use an analytics service, then just add the corresponding configuration to your `local_settings.py` file. Valid configuration options for the supported analytics services can be found here.

Note: since META-SHARE V3.0.1 we ship with a common Google Analytics tracking code for all META-SHARE websites. The tracking code is activated by default in `metashare/templates/base.html`. If you wouldn't like your META-SHARE installation to be tracked, you can remove the Google Analytics JavaScript snippet from this template. You also have to remove the snippet if you would like to use your own Google Analytics tracking code via django-analytical!

Frequently Asked Questions

This section compiles a number of the most frequently asked questions.

I Want to use MySQL and/or Apache

It may be possible to get these to work, but we have not tested these configurations and therefore cannot provide any support for them. The recommended database and web server technologies are listed in Software Dependencies_.

I Need Help Configuring lighttpd

The release includes a sample lighttpd.conf configuration file under `metashare/lighttpd-ssl.conf.sample` (or `metashare/lighttpd-ssl.conf.sample` for the non-SSL variant) which you can use as the basis for your configuration. More information on how to properly setup lighttpd with FastCGI support can be found in the Django documentation.

Also, look at the scripts `start-server.sh` and `stop-server.sh` which should show you how to start up and shut down the production server.

I am Getting Storage Errors when Importing or Saving

```
File "/usr/local/MetaShareNode/metashare/../metashare/storage/models
.py",=  line 254, in save
mkdir(self._storage_folder()) OSError: [Errno 2] No such file or dir
ectory:
'/home/storage/b557040eff1d11= e09075080027fee6a9b7ffe41433e94b19844
c6038a825a145'
File "/usr/local/MetaShareNode/metashare/../metashare/storage/models
.py",= line 254, in save
mkdir(self._storage_folder())  OSError: [Errno 2] No such file or di
rectory:
'/home/storage/b557040eff1d11=e09075080027fee6a9b7ffe41433e94b19844c
6038a825a145'
```

The first thing to verify is whether the `STORAGE_PATH` setting in `local_settings.py` points to a valid and existing folder – see Local Settings for META-SHARE Nodes_ for details.

Why Can Django not Serve the Static Files?

While in principle, Django could also serve those static files, this is not recommended for production use – it makes a lot more sense to have a dedicated, lightweight web server handle that task. Some more information on combining Django and lighttpd is available here

PostgreSQL Error Message

```
--- File "/usr/lib/python2.7/site-packages/django/db/backends/postgr
esql_psycopg2/base.py", line 24, in <module>
raiseImproperlyConfigured("Error loading psycopg2 module: %s" % e) d
jango.core.exceptions.ImproperlyConfigured:
Error loading psycopg2 module: No module named psycopg2 ---
```

Seems like you are trying to use PostgreSQL but you have not installed the `psycopg2` dependency. See Python Modules_ for how to install it.

Problems with Importing XML Files

We are trying to use `import_xml.py` to import XML files into the database. We are using an XML file that validates against the schema, but we get the following error:

```
source venv/bin/activate
python import_xml.py  ApertiumLMFBasqueDictionary.xml
deactivate

Importing XML file: "ApertiumLMFBasqueDictionary.xml"
Could not import XML file into database!
```

If you encounter this error, please first check that the XML file is indeed schema-valid with respect to the latest schema files. If so, there might be a bug – please send us

the example file if possible so that we can reproduce and fix it: helpdesk-technical@meta-share.eu

Updating the GeoIP Database for Statistics Collection

The country-based statistics do not seem to properly work anymore.

For statistical purposes, META-SHARE collects information about the country of origin of web site visitors. In this process, the IP address of the visiting user is converted to the country using the GeoLite Country database. As IP address to country mappings may change over time, an automatically set up cron job updates the used database every month for better statistics results.

The current version of the database is downloaded into the directory `/path/to/local/MetaShareNode/metashare/stats/resources/` using the following resource file (which is configurable in `settings.py` via the `GEOIP_DATA_URL` key): http://geolite.maxmind.com/download/geoip/database/GeoLiteCountry/GeoIP.dat.gz

Funding

This work has been co-funded by the European Union's Horizon 2020 program for ICT through grant agreement no.: 645357 (Coordination and Support Action CRACKER). The initial versions of the META-SHARE software have been co-funded by the 7th Framework Programme of the European Commission through the T4ME grant agreement no.: 249119.