

**A Network of Excellence forging the**

**Multilingual Europe Technology Alliance**

**META-SHARE V 2 Installation Manual**

**Authors:** Christian Federmann, Marc Schröder

**Date:** 6 March, 2012



**Table of Contents**

1 Executive Summary 3

2 Installation Requirements 3

2.1 Software Dependencies 3

2.1.1 Web Server 3

2.1.2 Database Software 3

2.2 Python Module Dependencies 3

3 Development Server 4

3.1 Local Settings for META-SHARE Nodes 5

3.2 Public/Private Key Generation for META-SHARE Nodes 7

3.2.1 Usage Advice 8

3.2.2 Implementation Details 8

4 Deployment Server 8

5 Importing and exporting Resources 8

5.1 Importing XML files into META-SHARE 9

5.2 Exporting XML files from META-SHARE 9

5.3 Copying data between META-SHARE nodes 10

5.4 Migrating data from a META-SHARE 1.0 installation 10

6 Frequently asked questions 11

6.1 'NoneType' object has no attribute 'identifier' 11

6.2 I want to use MySQL and/or Apache 11

6.3 I need help configuring lighttpd 11

6.4 The editor gives me error messages when saving 11

6.5 I am getting storage errors when importing or saving 12

6.6 Why can Django not serve the static files? 12

6.7 PostgreSQL error message 12

6.8 No module named metashare.storage 12

6.9 Problems with importing XML files 13

6.10 How to set up user accounts with edit permissions who are not staff users? 13

# Executive Summary

This document is a guide for installing META-SHARE V2. It is intended for system administrators setting up META-SHARE nodes.

# Installation Requirements

This section lists all software packages that are required to install and run a META-SHARE node. We assume that all software is installed into a designated META-SHARE folder, e.g. /path/to/local/MetaShareNode/.

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

## Software Dependencies

### Web Server

META-SHARE is a web application that builds on a web server. Deployment has been tested with *lighttpd 1.4.29*. Other web servers can be used, but you do so on your own risk.

### Database Software

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*. To connect PostgreSQL and your Django project, you need to install *psycopg2*, which is available from under the following URL: <http://pypi.python.org/pypi/psycopg2/2.4.2>

## Python Module Dependencies

All python-related dependencies are bundled with META-SHARE v2. A Linux / Unix / Mac install script is provided as /path/to/local/MetaShareNode/install-dependencies.sh. Run this script once after unpacking the bundle. If you don’t have a local python installed, or it does not have the right version, the install script will install the required python version locally, below /path/to/local/MetaShareNode/opt/. If you use this version of python, make sure to prepend /path/to/local/MetaShareNode/opt/bin to your PATH variable.

The compilable python module dependencies are compiled and installed into /path/to/local/MetaShareNode/lib/python2.7/site-packages.

This location is automatically added to the PYTHONPATH in the django server script /path/to/local/MetaShareNode/metashare/manage.py.

For information, the dependencies and their version are listed here:

1. Python 2.7.2: available from

- <http://www.python.org/download/releases/2.7.2/>

1. Django 1.3.1: available from

- <https://www.djangoproject.com/download/>

1. setuptools 0.6c11: available from

- <http://pypi.python.org/packages/source/s/setuptools/setuptools-0.6c11.tar.gz>

1. PyCrypto 2.3: available from

- <http://ftp.dlitz.net/pub/dlitz/crypto/pycrypto/pycrypto-2.3.tar.gz>

1. django-countries 1.0.2: available from

- <http://pypi.python.org/pypi/django-countries/1.0.2>

1. httplib2 0.7.1: available from

- <http://httplib2.googlecode.com/files/httplib2-0.7.1.tar.gz>

# Development Server

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

1. Install all required software as described in Chapter 2 “Installation Requirements”.
2. Extract the metashare-v2.tar.gz release package into a local folder MetaShareNode.
3. Create local\_settings.py for your local META-SHARE node:

$ cd MetaShareNode/metashare

$ cp local\_settings.sample local\_settings.py

Edit at least the following constants: DJANGO\_URL, DJANGO\_BASE, SECRET\_KEY, PRIVATE\_KEY\_PATH, STORAGE\_PATH, DEBUG, ADMINS, DATABASES, and EMAIL\_BACKEND. More information is available in Chapter 3.1 “Local Settings for META-SHARE Nodes”.

1. Initialise database contents using manage.py syncdb.

$ python manage.py syncdb

Answer “yes” when asked to create a superuser account and fill in details.

1. Start a SOLR server for the search index:

$ cd metashare

$ ./start-solr.sh

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

$ python manage.py test repo2 storage accounts

This should return "OK".

1. Run Django development server using manage.py runserver.

$ python manage.py runserver

Validating models...

0 errors found

Django version 1.3, using settings 'metashare.settings' Development server is running at http://127.0.0.1:8000/

Quit the server with CONTROL-C.

...

**Congratulations:** you have successfully started a META-SHARE v2 node in development mode. This means that all required Python/Django dependencies are functioning correctly.

## Local Settings for META-SHARE Nodes

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. While *global* settings can be stored within the Git repository (as they are neither security-critical nor node-dependant) we cannot do so for the *local* parameters. Hence, these are stored in an own file named local\_settings.py, which is automatically imported by settings.py. The repository contains a file named local\_settings.sample that lists and explains all *local* settings available for META-SHARE nodes. The local settings are the following.

* DJANGO\_URL = 'http://www.example.com/path/to/metashare'

The URL for this META-SHARE node Django application. Do not use 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 trailing /. Do not use leading / 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 e.g. 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.

* SECRET\_KEY = '7h$+o^h4f%q#d … +u5p\*+p\*lpz++z^q^9^+a5p--'

Make sure that the secret keys are unique, and don't share them with anybody. You can generate a new SECRET\_KEY using the enclosed helper script create\_secret\_key.py.

* PRIVATE\_KEY\_PATH = '/path/to/private/key'

Absolute path to the local private key used for decryption of content. Can be ignored for local development servers. Generation of public/private key pairs is described in the following section.

* STORAGE\_PATH = '/path/to/storage/path'

Absolute path to the local *storage base*, i.e. the folder in which local StorageObject instances can store their attachments. 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 but safe.

* DEBUG, TEMPLATE\_DEBUG, DEBUG\_JS

Debug settings, setting DEBUG=True will give exception stacktraces.

* ADMINS

Configure administrators for this Django project. If DEBUG=False, all errors will be reported as emails to these persons.

* 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': '',

# This is required to make import more robust.

'OPTIONS': {

'autocommit': True,

}

}

}

* EMAIL\_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'

Settings for sending mail. Production servers should use the SMTP email backend.

* TIME\_ZONE = 'Europe/Berlin'

Local time zone for this installation.

* SSO\_SECRET\_KEY = '6736d82b807811e0a1e5109 … cb1aa160fbd198e2a'

SSO\_SECRET\_KEYs are 64-character hexadecimal Strings. This might change in a future release depending on further development of the SSO architecture.

* STATS\_SERVER\_URL = <http://metastats.fbk.eu/>

The URL for the META-SHARE statistics server. Currently fixed, hence do not edit!

## Public/Private Key Generation for META-SHARE Nodes

Secure transactions between META-SHARE nodes are a highly important requirement for a successful release of the software. A key pair is created as follows.

1. Create a private key for your META-SHARE node:

$ openssl genrsa -out my\_private\_key.pem 2048

This will generate a 2048 byte-long RSA private key and store it inside a new file named my\_private\_key.pem. The file will look similar (except for contents, of course) to this:

-----BEGIN RSA PRIVATE KEY-----

MIIEpAIBAAKCAQEA5BUA1yNpne5N7d4dlLoFcUYOzhyBh6x5vcxQv+rrxabF3rDr

...

b4KeO+XXDGK4vUiq0UoHHMgKk/RLeLgNCYV3pH5O0REXOCbWaZMQcQ==

-----END RSA PRIVATE KEY-----

1. Given the private key, create a public key:

$ openssl rsa -pubout -in my\_private\_key.pem -out my\_public\_key.pem

This will derive a public key from the given private key file and store it inside a new file named my\_public\_key. The format is similar to the one from the private key:

-----BEGIN PUBLIC KEY-----MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA5BUA1yN

...

SGk38ou1Fo63d6FrPw8eKPLVpQIDAQAB

-----END PUBLIC KEY-----

### Usage Advice

Never, which really means **NEVER**, disclose your private key outside your institution. The only thing you are to export is (as the name suggests) the public key.

### Implementation Details

Public/private key encryption for META-SHARE is implemented using PyCrypto, an open-source toolkit available from [GitHub](https://github.com/dlitz/pycrypto). Given a private/public key file in .pem format, we can instantiate corresponding key objects like this:

from Crypto.PublicKey import RSA

with open('my\_private\_key.pem', 'r') as pem:

my\_private\_key = pem.read()

with open('my\_public\_key.pem', 'r') as pem:

my\_public\_key = pem.read()

You can check the integrity of the key instances like this:

# This should return True

my\_public\_key == my\_private\_key.publickey()

# Deployment Server

For deployment, we assume that you have downloaded and installed the *lighttpd web server* 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.

# 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.

## Initializing the Database

Without having the Haystack dependency installed, you may not have been able to initialize your database. Now that Haystack is in place, you should be able to safely run the following in your META-SHARE-Software directory:

metashare/manage.py syncdb

## 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](http://www.apache.org/dyn/closer.cgi/lucene/solr/).
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 Solr server which is configured with two cores (→ indexes) main and testing.

1. Change directory to $SOLR\_DIR/example.
2. Run

java -jar start.jar

1. 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](http://lucene.apache.org/solr/tutorial.html) 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:

python manage.py build\_solr\_schema

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.

1. 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
2. Restart the Solr server.
3. If you already have any data in your database, then manually build the search index once. Run:

python manage.py rebuild\_index

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).

## Searching in META-SHARE-Software

We now have a search page using Haystack/Solr for META-SHARE. Open a web browser and navigate to $MY\_SERVER\_NAME$/repo2/search2. You should see a search field with a “Search” button.

# 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 schema.

## Importing XML files into META-SHARE

META-SHARE comes with a tool called import\_xml.py to import XML files describing language resources into the system. The input files should be Schema-valid according to the current META-SHARE Schema file which are located in folder misc/schema/v2.0/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:

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

Schema validity is not strictly required by the import script; 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.

To import, run import\_xml.py as follows:

python import\_xml.py <file.xml|archive.zip> [<file.xml|archive.zip> ...]

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.

## 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. The script export\_xml.py will export all entries in the database into a zip archive containing one xml file per resource. The script requires a valid META-SHARE 2 database. It can be run as follows:

python export\_xml.py <archive.zip>

The resulting archive is suitable for import in any META-SHARE 2 installation.

## Copying data between META-SHARE nodes

Future versions of META-SHARE will support automatic synchronization of the data between a configurable set of META-SHARE nodes. In the meantime, the data provided by multiple nodes can be manually synchronized by means of XML export-import.

Assume a set of LR descriptions have been authored on node A using the META-SHARE metadata editor. These descriptions are to be made available on node B. The sequence of steps is the following.

1. Export the data from node A as described in Section 6.2.
2. If node A also contained entries that should not be copied, repackage the resulting archive to only contain the relevant entries.
3. Import the data into node B as described in Section 6.1.

## Migrating data from a META-SHARE 1.1 installation

There is not yet an automatic tool for migrating data. Please contact the technical helpdesk to discuss how to proceed.

# Setting up user accounts with edit permissions

During database creation you have set up one superuser. As this superuser, log into the system and then access the user management page:

DJANGO\_URL/admin/auth/user/

Use the “add user” button in the upper right to create a new user; after the initial save, you will see a rich “change form”. You need to make the following two changes:

1. Make sure the checkbox next to “staff user” is checked;
2. At the bottom of the page under “Groups”, make sure the group “globaleditors” is selected.

To test, log in as the newly created user and go to the front page. In the upper right, you should see the “Editor” button; clicking it should open the editor start page.

# Frequently asked questions

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

## I want to use MySQL and/or Apache

A: 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 section 2.1.

## I need help configuring lighttpd

A: The release includes a sample lighttpd.conf configuration file which you can use as the basis for your configuration.

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 directory: '/home/storage/b557040eff1d11= e09075080027fee6a9b7ffe41433e94b19844c6038a825a145' File "*/usr/local/MetaShareNode/metashare/*../metashare/storage/models.py",= line 254, in save    mkdir(self.\_storage\_folder()) OSError: [Errno 2] No such file or directory: '/home/storage/b557040eff1d11=e09075080027fee6a9b7ffe41433e94b19844c6038a825a145'

A: The first thing to verify is whether the STORAGE\_PATH setting in local\_settings.py points to a valid and existing folder – see section 3.1 for details.

## Why can Django not serve the static files?

A: 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 Djangp+lighttpd is available here: - <https://docs.djangoproject.com/en/dev/howto/deployment/fastcgi/#lighttpd-setup>

## PostgreSQL error message

--- File "/usr/lib/python2.7/site-packages/django/db/backends/postgresql\_psycopg2/base.py", line 24, in <module> raise ImproperlyConfigured("Error loading psycopg2 module: %s" % e) django.core.exceptions.ImproperlyConfigured: Error loading psycopg2 module: No module named psycopg2 ---

A: Seems like you are trying to use PostgreSQL but you have not installed the psycopg2 dependency. See Section 2.1 for details.

## Problems with importing XML files

Q: 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:

$ /usr2/MetaShareNode/software/bin/python import\_xml.py  ApertiumLMFBasqueDictionary.xml   
              
Importing XML file: "ApertiumLMFBasqueDictionary.xml"  
Could not import XML file into database!

…

A: 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](mailto:helpdesk-technical@meta-share.eu)