This document is part of the Network of Excellence "Multilingual Europe Technology Alliance (META-NET)",
co- funded by the 7th Framework Programme of the European Commission through the T4ME grant agreement no.: 249119.

A Network of Excellence forging the
Multilingual Europe Technology Alliance

META-SHARE V3.0
Installation Manual

Authors: Christian Federmann, Marc Schröder, Christian Spurk

Date: September 24, 2012

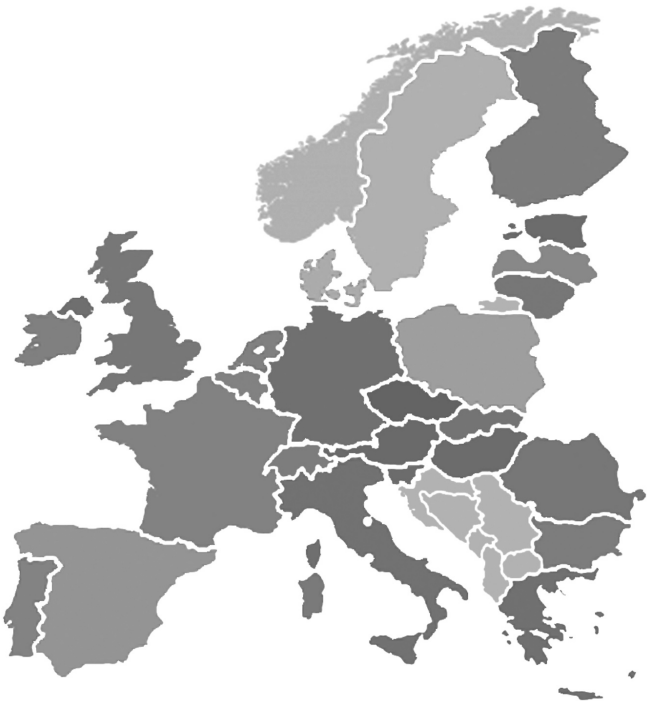

Table of Contents

1 Executive Summary

This document is a guide for installing META-SHARE V3.0. It is intended for system administrators setting up META-SHARE nodes. It also contains a section on how to migrate an existing META-SHARE V2.1.x installation to V3.0.

2 Migrating Data and User Accounts from an old META-SHARE Installation

If you already have a META-SHARE V2.1.x installation and you would like to migrate your resource descriptions, uploaded resource data, user accounts and statistics to a new META-SHARE installation, then this section is the right place to start with.

To upgrade a META-SHARE node with a software version that is older than V2.1, you first have to migrate your installation to V2.1.x as described in the Installation Manual that comes with a V2.1.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.

If you should have been one of our kind beta testers with a V2.9-beta installation currently running on your site, then you can also migrate all relevant data from this beta installation to the latest META-SHARE release. The migration steps are the same as described below, except that you use the following two scripts for export and import:

- `misc/tools/migration/export_node_from_2_9_beta_to_3_0.py`
- `misc/tools/migration/import_node_to_3_0_from_2_9_beta.py`

2.1 Migrated Data and Non-Migrated Data

META-SHARE V3.0 provides tools for the semi-automatic migration of data from a META-SHARE V2.1.x installation. The following data can be migrated with these tools:

- *resource descriptions* (metadata) – there is no need for a manual export/import cycle!
- actual *resource data* which can be directly downloaded from the node
- *user accounts* and the corresponding user profiles
- resource *ownership* information
- collected node *statistics*

This should cater for most META-SHARE V2.1.x installations. Please note, however, that the following data is *not* migrated:

- *local settings:* you have to make a new node installation before you can migrate your data, see Section 2.2.
- *any custom Django groups:* as a superuser you could create Django groups, however, on a V2.1.x node, this information would not have been used by META-SHARE directly. Therefore we assume that no such groups have been created.
- *permissions:* as a superuser you could assign Django permissions to users/groups, however, on a V2.1.x node, this information would not have been used by META-SHARE directly. Therefore we assume that no such groups have been created. An exception to this might be META-SHARE membership permissions: while these permissions influence the behaviour of the system, we assume that they have not been used as they were not documented.

Even if you should have created custom Django groups or if you should have assigned permissions manually, going through the migration steps in Section 2.2 should be worthwhile. While we recommend to use the newly introduced user rights management features (see the META-SHARE Provider Manual for more information), you can still always recreate your custom groups in the new installation again and/or reassign any permissions.

Due to bugs in V2.1.x that are now resolved, it was possible that you could create invalid resource descriptions with the metadata editor. Such resource descriptions have to be manually fixed during the migration, see Section 2.2.

2.2 Step-by-Step Migration

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. Get META-SHARE V3.0 from https://github.com/metashare/META-SHARE/downloads and install it as explained in Sections 3–6. Note: during the installation you can skip the creation of a superuser account (cf. Section 4); we will migrate your old superuser account(s) instead.

2. Copy `misc/tools/migration/export_node_from_2_1_x_to_3_0.py` to the `metashare/` folder of your old META-SHARE V2.1.x installation.

3. In the `metashare/` folder of your old META-SHARE V2.1.x installation run:
   ```
   python2.7 ./export_node_from_2_1_x_to_3_0.py /tmp/MS21_EXPORT | tee /tmp/ms21_export.log
   ```
   Note: you can use any other temporary directory as an argument to the Python script; just make sure that there is enough space available for at least your actual resource data plus about 100 megabytes more.

4. You can now remove the `export_node_from_2_1_x_to_3_0.py` Python script again from your old installation.

5. Due to some (now fixed) bugs in V2.1.x we strongly recommend to validate the resource descriptions to migrate. We therefore employ the tool `xmllint` which should be installed on your machine for this. Run the following shell command from the `metashare/` folder of your new META-SHARE V3.0 installation:
   ```
   xmllint --noout --schema ../misc/schema/v3.0/META-SHARE-Resource.xsd $(find "/tmp/MS21_EXPORT/" -name metadata.xml) 2>&1 | grep -Ev 'validates$'
   ```
   Note: make sure to use the same temporary directory as an argument after `find` that you have used above.

6. If there was any output when running the validation command from the previous step, then please make sure to manually fix the validation problem(s) directly in the specified `metadata.xml` file(s) before continuing. When done, return to the previous step in order to be on the safe side that there are no validation problems left.

7. Copy `misc/tools/migration/import_node_to_3_0_from_2_1_x.py` to the `metashare/` folder of your new META-SHARE V3.0 installation.

8. In the `metashare/` folder of your new META-SHARE V3.0 installation run the following command (which may take a while to complete):
   ```
   python2.7 ./import_node_to_3_0_from_2_1_x.py /tmp/MS21_EXPORT | tee /tmp/ms21_import.log
   ```
   Note: make sure to use the same temporary directory as an argument to the Python script that you have used above.

   Also note: as the data from your temporary export directory is copied to the new META-SHARE installation, make sure that there is again enough space available for at least a full copy of the directory!

9. If all went as expected, you should now have a new META-SHARE V3.0 installation containing your old data and user accounts.
10. You can now remove the `import_node_to_3_0_from_2_1_x.py` Python script again from your new installation.
11. You can also remove your temporary export directory and any created log files, for example:
    ```
    rm -rf /tmp/MS21_EXPORT /tmp/ms21_export.log /tmp/ms21_import.log
    ```

2.3 Post-Migration Recommendations

In META-SHARE V2.1.x, users could either be editors or not, and all editors could edit all available resources. The new META-SHARE version has a more fine-grained access rights system. After the migration, users will only be allowed to edit their own resources. We highly recommend using the new user access rights system which is described in the META-SHARE Provider Manual. As a superuser you should create one or more editor groups and assign the relevant user accounts to these groups. In addition, you should assign all resources to the relevant groups.

After a successful migration and the setup/assignment of new editor groups, you can also remove all former editor users from the legacy `globaleditors` group to get a clean new node installation.

3 Installation Requirements

This section lists all software packages that are required to install and run a META-SHARE node. We assume that you have downloaded (https://github.com/metashare/META-SHARE/downloads) and extracted the META-SHARE software package 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.

3.1 Software Dependencies

3.1.1 Web Server

META-SHARE is a web application that builds on a web server. Deployment has been tested with *lighttpd 1.4.29* 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.

3.1.2 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*. See section 11.9 for how to install it.

3.2 Python Module Dependencies

All Python-related dependencies are bundled with META-SHARE V3.0. 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 build[1] and 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 compatible 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 versions are listed here:

1) *Python 2.7.2*: available from http://www.python.org/download/releases/2.7.2/
2) *Django 1.3.3*: available from https://www.djangoproject.com/download/
3) *setuptools 0.6c11*: available from http://pypi.python.org/packages/source/s/setuptools/setuptools-0.6c11.tar.gz

[1] In order to build Python with the modules required for META-SHARE, you probably need to install some development files on your local machine first. Section 11.8 details these requirements.

4) *httplib2 0.7.1* : available from http://httplib2.googlecode.com/files/httplib2-0.7.1.tar.gz
5) *pygeoip 0.2.2*: available from http://code.google.com/p/pygeoip/downloads/
6) *Django Kronos 0.3*: available from https://github.com/jgorset/django-kronos
7) *Django Analytical 0.12.1*: available from http://packages.python.org/django-analytical/
8) *python-dateutil 1.5:* available from http://pypi.python.org/pypi/python-dateutil/
9) *django_jenkins 0.12.0*: available from https://github.com/kmmbvnr/django-jenkins
10) *django-selenium 0.8:* available from https://github.com/dragoon/django-selenium/
11) *Haystack 2.0.0-beta:* available from http://haystacksearch.org/
12) *pycountry 0.14.2:* available from http://pypi.python.org/pypi/pycountry/
13) *django-selectable 0.4.1:* available from http://pypi.python.org/pypi/django-selectable
14) *Selenium Python Client Driver 2.23:* available from http://pypi.python.org/pypi/selenium

4 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 3, “Installation Requirements”.
2. Extract the `metashare-v3.0.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`, `STORAGE_PATH`, `DEBUG`, `ADMINS`, `DATABASES`, and `EMAIL_BACKEND`. More information is available in Chapter 4.1, "Local Settings for META-SHARE Nodes".

4. Initialize database contents:
   ```
   $ python manage.py syncdb
   ```
   Answer "yes" when asked to create a superuser account and fill in the requested details.

5. Start an Apache Solr server for the search index (uses Java and Python internally, the latter should be at least Python 2.7!):
   ```
   $ ./start-solr.sh
   ```

6. Run tests to check that Django can load and serve META-SHARE:
   ```
   $ python manage.py test repository storage accounts sync stats
   ```
   This should return "OK".

   Note: This step may take a few minutes.

7. Run a Django development server:
   ```
   $ python manage.py runserver
   Validating models...
   0 errors found
   Django version 1.3.x, 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 V3.0 node in development mode. This means that all required Python and Django dependencies are functioning correctly.

4.1 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. 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.

- `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.

- `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.

- `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': '',
         # 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 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 Section 7. 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.

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

5 Deployment Server

For deployment, we assume that you have downloaded and installed the *lighttpd web server* (see also Section 11.2) 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. If the `python` binary on your `PATH` environment variable should not be at least Python 2.7, then you also have to make sure that `start_server.sh` and `stop_server.sh` are changed so that they explicitly call a Python 2.7 binary.

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 cron jobs 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.

6 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.

6.1 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 Solr server 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.

6.2 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.

6.3 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`.
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:
   `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).

7 Linking your Node with the META-SHARE Network

7.1 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. Section 7.2 details the steps that are required for this.

7.2 Step-by-Step Instructions

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

- In your `local_settings.py` file (see Section 4.1), make sure to have an entry in the `SYNC_USERS` dictionary. Remember to run the following command, whenever you change the `SYNC_USERS` setting:
  `python2.7 ./manage.py syncdb`
- Give the account credentials of your `SYNC_USERS` entry 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.dfki.de/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.

8 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.

8.1 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/v3.0/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` 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.

8.1.1 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:

```
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.

8.1.2 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.

8.2 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.

8.2.1 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 V3.0 database. It can be run as follows:

```
python export_xml.py <archive.zip>
```

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

8.2.2 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.

8.3 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.

9 Setting up Editor User Accounts

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

10Search Engine Optimization

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 any 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 at http://packages.python.org/django-analytical/install.html#enabling-the-services.

11 Frequently Asked Questions

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

11.1 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 3.1.

11.2 I Need Help Configuring lighttpd

A: 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.[2]

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.

11.3 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=e09075080027fee6a9b7ffe41433e94b19844c6038a82
5a145'
```

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 4.1 for details.

[2] https://docs.djangoproject.com/en/1.3/howto/deployment/fastcgi/

11.4 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 Django and lighttpd is available here: https://docs.djangoproject.com/en/1.3/howto/deployment/fastcgi/#lighttpd-setup

11.5 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 11.9 for how to install it.

11.6 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

11.7 Updating the GeoIP Database for Statistics Collection

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

A: 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. We are shipping a fixed version of this database. As IP address to country mappings may change over time, you might also want to update the used database for better statistics results. In order to update the current version of the database, download a new version and move the file `GeoIP.dat` into the directory `/path/to/local/MetaShareNode/metashare/stats/resources/`. New versions of the database are available here:

http://geolite.maxmind.com/download/geoip/database/GeoLiteCountry/GeoIP.dat.gz

11.8 How can I Correctly Build Python for META-SHARE?

A: Building and installing the Python version which comes with META-SHARE (see Section 3.2), has a few dependencies. You need to have *libsqlite3-dev*, *libssl-dev* und *zlib1g-dev* installed. 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`

11.9 How do I Install psycopg2 for using PostgreSQL?

A: We recommend installing *pscopg2* using the *pip* installer (`pip install psycopg2`). For this to work on Ubuntu, you have to first install the two packages *libpq-dev* and *python-dev* (`apt-get install libpq-dev python-dev`). On some versions of Ubuntu, you may also have to use an older *pscopg2* version which you can install like this: `pip install psycopg2==2.4.1`

Otherwise *pscopg2* is available on the following website:

http://pypi.python.org/pypi/psycopg2/