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 V2.1 Installation Manual

Authors: Christian Federmann, Marc Schröder, Christian Spurk

Date: May 18, 2012

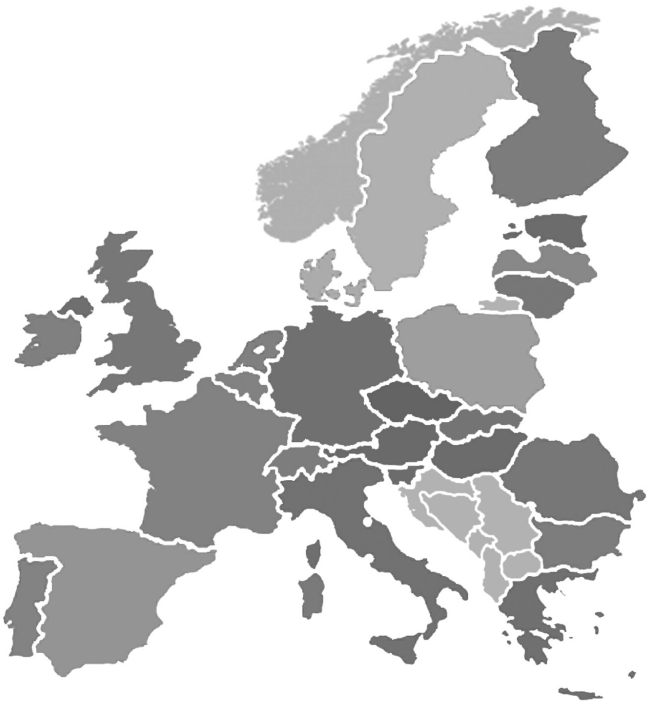

Table of Contents

1 Executive Summary 4
2 Installation Requirements 4
2.1 Software Dependencies 4
2.1.1 Web Server 4
2.1.2 Database Software 4
2.2 Python Module Dependencies 4
3 Development Server 5
3.1 Local Settings for META-SHARE Nodes 6
3.2 Public/Private Key Generation for META-SHARE Nodes 8
3.2.1 Usage Advice 9
3.2.2 Implementation Details 9
4 Deployment Server 9
5 Solr Server for Browsing and Searching 9
5.1 Installing Solr 10
5.2 Keeping the Solr Configuration Up-to-Date 10
5.3 Manually Updating the Solr Configuration 10
6 Importing and Exporting Resources 11
6.1 Importing XML Files into META-SHARE 11
6.1.1 Importing from the Command Line 11
6.1.2 Importing from the Editor 12
6.2 Exporting XML Files from META-SHARE 12
6.2.1 Exporting from the Command Line 12
6.2.2 Exporting from the Editor 12
6.3 Copying Data between META-SHARE Nodes 12
6.4 Migrating Data from a Previous META-SHARE Installation 13
6.4.1 Exporting the Metadata from META-SHARE 13
6.4.2 Upgrade the XML Files to the Latest Metadata Format 13
6.4.3 Importing the Metadata into a New META-SHARE Installation 14
7 Setting up User Accounts with Edit Permissions 14
8 Frequently Asked Questions 14
8.1 I Want to use MySQL and/or Apache 15
8.2 I Need Help Configuring lighttpd 15
8.3 I am Getting Storage Errors when Importing or Saving 15
8.4 Why Can Django not Serve the Static Files? 15
8.5 PostgreSQL Error Message 15

8.6 Problems with Importing XML Files 16

1 Executive Summary

This document is a guide for installing META-SHARE V2.1. It is intended for system administrators setting up META-SHARE nodes.

2 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.

2.1 Software Dependencies

2.1.1 Web Server

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

2.1.2 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*, which is available on the following website: http://pypi.python.org/pypi/psycopg2/2.4.2

2.2 Python Module Dependencies

All Python-related dependencies are bundled with META-SHARE v2.1. 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 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 1.3.1`: available from
   - https://www.djangoproject.com/download/

2. `setuptools 0.6c11`: available from
   - http://pypi.python.org/packages/source/s/setuptools/setuptools-0.6c11.tar.gz

3. `PyCrypto 2.3`: available from
   - http://ftp.dlitz.net/pub/dlitz/crypto/pycrypto/pycrypto-2.3.tar.gz

4. `django-countries 1.0.2`: available from
   - http://pypi.python.org/pypi/django-countries/1.0.2

5. `httplib2 0.7.1`: available from
   - http://httplib2.googlecode.com/files/httplib2-0.7.1.tar.gz

3 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-v2.1.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”.

4) Initialise database contents using `manage.py syncdb`.

   ```
   $ python manage.py syncdb
   ```

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

5) Start an Apache Solr server for the search index:

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

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

```
$ python manage.py test repository storage accounts
```

This should return "OK".

7) 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 V2.1 node in development mode. This means that all required Python/Django dependencies are functioning correctly.

3.1 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. 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 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.

- `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 on the website, for example. This may include sensitive information, so use with care, preferably only for local development servers.
- `ADMINS`
  Configure administrators for this Django project. If `DEBUG=False`, all errors will be reported as e-mails 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, so do not edit!

3.2 Public/Private Key Generation for META-SHARE Nodes

Secure transactions between META-SHARE nodes are a highly important requirement for the success 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 bytes long RSA private key and store it in 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-----
   ```

2. 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 in 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-----
```

3.2.1 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.

3.2.2 Implementation Details

Public/private key encryption for META-SHARE is implemented using `PyCrypto`, an open-source toolkit available from GitHub. 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()
```

4 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.

5 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.

5.1 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.

5.2 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.

5.3 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).

6 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.

6.1 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/v2.1/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.

6.1.1 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.

6.1.2 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.

6.2 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.

6.2.1 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 V2.1 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 installation.

6.2.2 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 installation.

6.3 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 and import.

Assume a set of language resource 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 using one of the possibilities described in Section 6.2.
2. Import the data into node B as described in Section 6.1.

6.4 Migrating Data from a Previous META-SHARE Installation

The procedure for upgrading a META-SHARE node from a previous version can be sketched as follows:

1. Export the metadata records from the previous version as XML files;
2. Upgrade the XML files to the latest version of the metadata schema;
3. Import the upgraded XML files into the latest META-SHARE software.

The following subsections provide details for each step.

6.4.1 Exporting the Metadata from META-SHARE

Section 6.2.1 explains how all language resource descriptions from a META-SHARE node can be exported.

A procedure to export data from META-SHARE V1.0 is described in the META-SHARE V1.1 installation manual. It is not repeated here because all known users have upgraded to V1.1 or newer.

6.4.2 Upgrade the XML Files to the Latest Metadata Format

META-SHARE V2.1 comes with a `metashare-resource-upgrader` tool which automates, to the extent possible, the conversion of previous versions of the META-SHARE resource XML format to the newest version. It processes all XML files in a source folder and saves into a target folder a version of each file converted to the newest format.

The tool uses a sequence of XSLT stylesheets[1] to convert metadata from previous versions. Supported data formats are META-SHARE V1.0, META-SHARE V1.1, and META-SHARE V2.0. The stylesheets automate the conversion to the extent possible, but due to differing requirements, it may be that META-SHARE V1.x resources need manual work before they can be imported into META-SHARE V2.1.

To verify if a file can be imported as is, or if manual work is required before it can be imported, `metashare-resource-upgrader` automatically performs an XML Schema validation of each target file after conversion, and prints out the result on the command line.

To use the `metashare-resource-upgrader`, proceed as follows:

1. Unpack the exported zip file into a folder, say `sourcedir`.

[1] The actual stylesheets are located in `META-SHARE/misc/tools/metashare-resource-upgrader/src/src/main/resources/metashare/upgrader/conversion`. They can also be applied manually using any XSLT transformer.

2. Run the conversion tool as follows:

```
cd META-SHARE/misc/tools/metashare-resource-upgrader
java -jar metashare-resource-upgrader-2.1.jar sourcedir targetdir
```

3. The target folder `targetdir` may exist but must not contain any files.
4. The tool will inform you for every converted resource whether it is schema-valid or not. For the files in `targetdir` that are not schema-valid, you need to manually identify the problems by means of a schema-validating parser and fix them. One option is `xmllint`:

```
xmllint --schema META-SHARE/misc/schema/v2.1/META-SHARE-Resource.xsd
targetdir/resource-1.xml
```

When all XML files in `targetdir` are schema-valid, you are ready to proceed to the import step into v2.1.

6.4.3 Importing the Metadata into a New META-SHARE Installation

Once the converted files are schema-valid, the import step should be straightforward, as described in Section 6.1.1:

```
python import_xml.py targetdir/*.xml
```

7 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 status” 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.

8 Frequently Asked Questions

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

8.1 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 2.1.

8.2 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.

8.3 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 3.1 for details.

8.4 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/dev/howto/deployment/fastcgi/#lighttpd-setup

8.5 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 2.1.2 for details.

8.6 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