
RDMO Documentation Documentation

Release 0.9.2

RDMO project

Dec 07, 2017

1	Installation	3
1.1	Install prerequisites	3
1.2	Obtaining the app directory	5
1.3	Install python packages	5
1.4	Setup the application	6
2	Deployment	7
2.1	Development server	7
2.2	Apache and mod_wsgi	7
2.3	nginx and gunicorn	9
3	Configuration	11
3.1	General settings	11
3.2	Databases	12
3.3	E-Mail	14
3.4	Authentication	15
3.5	Themes	19
3.6	Export formats	19
3.7	Cache	20
3.8	Logging	20
4	Administration	23
4.1	Site configuration	23
4.2	Users and Groups	24
4.3	Social accounts	25
4.4	Auth Tokens	25
5	Management	27
5.1	Questions	28
5.2	Domain	32
5.3	Options	34
5.4	Conditions	36
5.5	Views	38
5.6	Tasks	42
5.7	Export and Import	43
6	Upgrade	45

6.1	Upgrade to version 0.9.0	45
7	Development	47
7.1	Testing	47
7.2	Internationalisation	47
7.3	Figures	48
7.4	Documentation	48

RDMO is a tool to support the systematic planning, organisation and implementation of the data management throughout the course of a research project. RDMO is funded by the Deutsche Forschungsgemeinschaft (DFG).

Home Page <https://rdmorganiser.github.io>

Source code <https://github.com/rdmorganiser/rdmo>

Documentation <http://rdmo.readthedocs.io>

Demo <https://rdmo.aip.de>

Warning: This software is currently under development and not production ready.

CHAPTER 1

Installation

For demonstration, development or testing purposes, RDMO can be installed on Linux, Windows and macOS. If you, however, to set up a production environment, serving RDMO over a Network or the Internet, we strongly suggest that you use a recent Linux distribution, namely CentOS7, Debian 8 or 9, or Ubuntu 16.04.3 LTS (Xenial Xerus).

The code is mainly written in Python and should work with a Python higher than 3.4. RDMO works also on Python 2.7. Please note that for a CentOS7/Apache setup, only Python 2.7 is possible.

An installation of RDMO contains of three parts:

1. A directory which holds all the settings and customisations, custom to your installation of RDMO. We will call this directory `rdmo-app`, but you can use any name you see fit.
2. The actual `rdmo` package, which centrally maintained by the RDMO team, and is installed as a dependency in a virtual environment.
3. A database to store the content generated by the users of your RDMO installation. Currently, we support Postgres, MySQL, and SQLite.

This chapter shows how these components are set up. Optional components can be installed afterwards and are covered under *Configuration*.

For testing and development, you can run RDMO using your regular user account. On a production system, a dedicated user account should be used. We suggest to create a user called `rdmo` with the group `rdmo` and the home directory `/srv/rdmo`. We will use this user throughout this documentation.

Do not use the `root` user to run RDMO! It is a bad idea anyway and several steps of the installation will not work. `sudo` is used in the installation when needing root-privileges to install packages.

1.1 Install prerequisites

Installing the prerequisites for RDMO differs on the different operating systems and is therefore covered in different sections.

1.1.1 Linux

We recommend to install the prerequisites using the packaging system of your distribution. On Debian/Ubuntu use:

```
sudo apt install build-essential libxml2-dev libxslt-dev zlib1g-dev \  
python3-dev python3-pip python3-venv \  
git pandoc  
  
# optional, for pdf output  
sudo apt install texlive texlive-xetex
```

on RHEL/CentOS use:

```
sudo yum install gcc gcc-c++ libxml2-devel libxslt-devel \  
python34-devel python34-pip python34-virtualenv \  
git pandoc  
  
# optional, for pdf output  
sudo yum install texlive texlive-xetex texlive-mathspec texlive-euenc \  
texlive-xetex-def texlive-xltextra
```

On Ubuntu 14.04, *python3-venv* is not available. Please use *python3.5-venv* instead.

On RHEL/CentOS *selinux* is enabled by default. This can result in unexpected errors, depending on where you store the RDMO source code on the system. While the preferable way is to configure it correctly (which is beyond the scope of this documentation), you can also set *selinux* to permissive or disabled in */etc/selinux/config* (and reboot afterwards).

If you want to use Python 2.7 instead of Python 3, please use the corresponding packages:

```
apt install python-dev python-pip python-virtualenv      # Debian/Ubuntu  
yum install python-devel python-pip python-virtualenv  # RHEL/CentOS
```

1.1.2 macOS

We recommend to install the prerequisites using *brew*:

```
brew install python3      # for python 3  
brew install python       # for python 2  
brew install git  
brew install pandoc  
  
# optional, for pdf export  
brew install texlive
```

1.1.3 Windows

On Windows, the software prerequisites need to be downloaded and installed from their particular web sites.

For python:

- download from <https://www.python.org/downloads/windows/>
- we recommend a version ≥ 3.4
- don't forget to check 'Add Python to environment variables' during setup

For git:

- download from <https://git-for-windows.github.io/>

For the Microsoft C++ Build Tools:

- download from <http://landinghub.visualstudio.com/visual-cpp-build-tools>

For pdflatex (optional, for pdf export):

- download from <http://miktex.org/>

All further steps need to be performed using the windows shell `cmd.exe`. You can open it from the Start-Menu.

1.2 Obtaining the app directory

The next step is to create the `rdmo-app` directory by clone the corresponding repository:

```
git clone https://github.com/rdmorganiser/rdmo-app
```

Note that this is not the main `rdmo`, repository, only the configuration files. Inside this directory, you will find:

- a `config` directory, containing the main settings of your RDMO installation,
- a `requirements` directory, containing shortcuts to install the different mandatory and optional dependencies, and
- a `manage.py` script, which is the main way to interact with your RDMO installation on the command line. Most of the following steps will use this script.

The `rdmo-app` directory corresponds to a `project` in Django terms.

1.3 Install python packages

After you have obtained the `rdmo-app`, you need to install the `rdmo` package and the other python dependencies.

Change to the `rdmo-app` directory and create a `virtualenv` (this is done as your user or the created `rdmo` user, not as `root`):

```
cd rdmo-app

python3 -m venv env                                # for python3
virtualenv env                                     # for python2.7

source env/bin/activate                            # on Linux or macOS
call env\Scripts\activate.bat                      # on Windows

pip install --upgrade pip setuptools               # update pip and setuptools
```

After the virtual environment is activated, the `rdmo` package can be installed using `pip`:

```
pip install rdmo
```

On windows, `pandoc` needs to be installed in an additional step:

```
# only on Windows
python -c "import py pandoc; py pandoc.download_pandoc() "
```

The virtual environment encapsulates your RDMO installation from the rest of the system. This makes it possible to run several applications with different python dependencies on one machine and to install the dependencies without root permissions.

Important: The virtual enviroment needs to be activated, using `source env/bin/activate` or call `env\Scripts\activate.bat`, everytime a new terminal is used.

1.4 Setup the application

To set up the application, create a new file `config/settings/local.py` in your cloned `rdmo-app` directory. For the example user with the home `/srv/rdmo`, this would now be `/srv/rdmo/rdmo-app/config/settings/local.py`.

You can use `config/settings/sample.local.py` as template, i.e.:

```
cp config/settings/sample.local.py config/settings/local.py    # on Linux or macOS
copy config\settings\sample.local.py config\settings\local.py  # on Windows
```

Most of the settings of your RDMO instance are specified in this file. The different settings are explained in detail *later in the documentation*. For a minimal configuration, you need to set `DEBUG = True` to see verbose error messages and serve static files, and `SECRET_KEY` to a long random string, which you will keep secret. Your database connection is configured using the `DATABASES` variable. Database configuration is covered *later in the documentation*. If no `DATABASE` setting is given `sqlite3` will be used as database backend.

Then, initialize the database of the application, using:

```
python manage.py migrate          # initializes the database
python manage.py create_groups    # creates groups with different permissions
python manage.py createsuperuser  # creates the admin user
python manage.py download_vendor_files # dowloads front-end files from the CDN
```

After these steps, RDMO can be run using Djangos intergrated development server:

```
python manage.py runserver
```

Then, RDMO is available on <http://127.0.0.1:8000> in your (local) browser. The different ways RDMO can be deployed are covered in the next chapter.

As already mentioned, RDMO can be run in two different setups:

- for *development or testing*, using the build-in Django development server.
- in production, using a web server and the `wsgi` protocol. We suggest to use one of the two following setups:
 - *Apache2 and mod_wsgi*
 - *nginx, gunicorn and systemd*

2.1 Development server

Django comes with an integrated development server. It can be started using:

```
python manage.py runserver
```

Then, RDMO is available on <http://localhost:8000> in your (local) browser. The development server is **not** suited to serve the application to the internet.

If you want the development server to be accessible from other machines you need to use:

```
python manage.py runserver 0.0.0.0:8000
```

where 8000 is the port and can be changed according to your needs. Please do not use this setup for more than testing and development, it is not secure.

More information about the development server can be found in the [Django documentation](#).

2.2 Apache and mod_wsgi

In production, you should create a dedicated user for RDMO. All steps for the installation, which do not need root access, should be done using this user. As before, we assume this user is called `rdmo` and its home is `/srv/rdmo` and therefore your `rdmo-app` is located in `/srv/rdmo/rdmo-app`.

Install the Apache server and `mod_wsgi` using:

```
# Debian/Ubuntu
sudo apt install apache2 libapache2-mod-wsgi-py3 # for python3
sudo apt install apache2 libapache2-mod-wsgi     # for python2.7

# CentOS
sudo yum install httpd mod_wsgi                  # only for python2.7
```

Next create a virtual host configuration. Sadly, the different distributions use different versions of Apache and `mod_wsgi` and therefore require a slightly different setup:

For Debian/Ubuntu use:

```
# in /etc/apache2/sites-available/000-default.conf
<VirtualHost *:80>
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

    Alias /static /srv/rdmo/rdmo-app/static_root/
    <Directory /srv/rdmo/rdmo-app/static_root/>
        Require all granted
    </Directory>

    WSGIDaemonProcess rdmo user=rdmo group=rdmo \
        home=/srv/rdmo/rdmo-app python-home=/srv/rdmo/rdmo-app/env
    WSGIProcessGroup rdmo
    WSGIScriptAlias / /srv/rdmo/rdmo-app/config/wsgi.py process-group=rdmo

    <Directory /srv/rdmo/rdmo-app/config/>
        <Files wsgi.py>
            Require all granted
        </Files>
    </Directory>
</VirtualHost>
```

for CentOS 7:

```
# in /etc/httpd/conf.d/vhosts.conf on RHEL/CentOS
<VirtualHost *:80>
    ServerAdmin webmaster@localhost

    DocumentRoot /var/www/html/

    Alias /static /srv/rdmo/rdmo-app/static_root/
    <Directory /srv/rdmo/rdmo-app/static_root/>
        Require all granted
    </Directory>

    WSGIDaemonProcess rdmo user=rdmo group=rdmo home=/srv/rdmo/rdmo-app \
        python-path=/srv/rdmo/rdmo-app:/srv/rdmo/rdmo-app/env/lib/python2.7/site-
↪ packages
    WSGIProcessGroup rdmo
    WSGIScriptAlias / /srv/rdmo/rdmo-app/config/wsgi.py process-group=rdmo

    <Directory /srv/rdmo/rdmo-app/config/>
```

```
<Files wsgi.py>
    Require all granted
</Files>
</Directory>
</VirtualHost>
```

Restart the Apache server. RDMO should now be available on `YOURDOMAIN`. Note that the Apache user needs to have access to `/srv/rdmo/rdmo-app/static_root/`.

As you can see from the virtual host configurations, the static assets, like CSS and JavaScript files are served independent from the WSGI-python script. In order to do so they need to be gathered in the `static_root` directory. This can be archived by running:

```
python manage.py collectstatic
```

in your virtual environment.

In order to apply changes to the RDMO code (e.g. after an *upgrade*) the webserver needs to be reloaded or the `config/wsgi.py` file needs to appear modified. This can be done using the `touch` command:

```
touch config/wsgi.py
```

Also, the `collectstatic` command has to be executed again. Both can be achieved using:

```
python manage.py deploy
```

in your virtual environment.

2.3 nginx and gunicorn

As mentioned several times, you should create a dedicated user for RDMO. All steps for the installation, which do not need root access, should be done using this user. Here we assume this user is called `rdmo` and it's home is `/srv/rdmo` and therefore your `rdmo-app` is located in `/srv/rdmo/rdmo-app`.

First install gunicorn inside your virtual environment:

```
pip install -r requirements/gunicorn.txt
```

Then, test gunicorn using:

```
gunicorn --bind 0.0.0.0:8000 config.wsgi:application
```

This should serve the application like `runserver`, but without the static assets, like CSS files and images. After the test kill the gunicorn process again.

Now, create a systemd service file for RDMO. Systemd will launch the gunicorn process on startup and keep running. Create a new file in `/etc/systemd/system/rdmo.service` and enter (you will need root/sudo permissions for that):

```
[Unit]
Description=RDMO gunicorn daemon
After=network.target

[Service]
User=rdmo
Group=rdmo
WorkingDirectory=/srv/rdmo/rdmo-app
```

```
ExecStart=/srv/rdmo/rdmo-app/env/bin/gunicorn --bind unix:/srv/rdmo/rdmo.sock config.  
↳wsgi:application  
  
[Install]  
WantedBy=multi-user.target
```

This service needs to be started and enabled like any other service:

```
sudo systemctl start rdmo  
sudo systemctl enable rdmo
```

Next, install nginx

```
sudo apt install nginx # on Debian/Ubuntu  
sudo yum install nginx # on RHEL/CentOS
```

Edit the nginx configuration as follows (again with root/sudo permissions):

```
# in /etc/nginx/sites-available/default on Debian/Ubuntu  
# in /etc/nginx/conf.d/vhost.conf      on RHEL/CentOS  
server {  
    listen 80;  
    server_name YOURDOMAIN;  
  
    location / {  
        proxy_pass http://unix:/srv/rdmo/rdmo.sock;  
    }  
    location /static/ {  
        alias /srv/rdmo/rdmo-app/static_root/;  
    }  
}
```

Restart nginx. RDMO should now be available on YOURDOMAIN. Note that the unix socket `/srv/rdmo/rdmo.sock` needs to be accessible by nginx.

As you can see from the virtual host configurations, the static assets, like CSS and JavaScript files are served independent from the reverse proxy to the gunicorn process. In order to do so they need to be gathered in the `static_root` directory. This can be achieved by running:

```
python manage.py collectstatic
```

in your virtual environment.

In order to apply changes to the RDMO code (e.g. after an *upgrade*) the gunicorn process need to be restarted:

```
sudo systemctl restart rdmo
```

The RDMO application uses the [Django settings](#) module for its configuration. To separate the base configuration and your local adjustments and secret information (e.g. database connections), RDMO splits the settings into two files:

- `config/settings/base.py`, which is part of the git repository and maintained by the RDMO development team.
- `config/settings/local.py`, which is ignored by git and should be edited by you.

As part of the installation `config/settings/local.py` should be created from the template `config/settings/sample.local.py`.

While technically the local settings file `config/settings/local.py` can be used to override all of the settings in `config/settings/sample.local.py`, it should be used to customize the settings already available in `config/settings/sample.local.py`.

This comprises *general settings*, *database connections*, how to send *emails*, the different *authentication methods*, the usage of *themes*, and *caches*.

3.1 General settings

A few general setting should be included in your `config/settings/local.py`. The first, and probably most important one is if you run RDMO in [debug mode](#) or not:

```
DEBUG = True
```

In debug mode, verbose error pages are shown in the case something goes wrong and static assets like CSS and JavaScript files are found by the development server automatically. The debug mode *must not* be enabled when running RDMO in production connected to the internet.

Django needs a [secret key](#), which “should be set to a unique, unpredictable value”:

```
SECRET_KEY = 'this is not a very secret key'
```

This key must be kept secret since otherwise many of Django's security protections fail.

In production, Django only [allows requests to certain urls](#), which you need to specify:

```
ALLOWED_HOSTS = ['localhost', 'rdmo.example.com']
```

If you want to run RDMO under an alias like <http://example.com/rdmo> you need to set the base URL:

```
BASE_URL = '/rdmo'
```

Furthermore, you might want to choose the main language for RDMO and the timezone:

```
LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'Europe/Berlin'
```

3.2 Databases

RDMO can be used with all database supported by the Django framework. The particular database connection is defined using the setting `DATABASE`. An overview about the Django database settings is given [here](#). In the following, we show the settings for PostgreSQL, MySQL, and SQLite.

3.2.1 PostgreSQL

PostgreSQL can be installed using:

```
# Debian/Ubuntu
sudo apt install postgresql

# CentOS
sudo yum install postgresql-server postgresql-contrib
sudo postgresql-setup initdb
sudo systemctl start postgresql
sudo systemctl enable postgresql
```

To use PostgreSQL as your database backend install `psycopg2` in your virtual environment:

```
pip install -r requirements/postgres.txt
```

Then, add the following to your `config/settings/local.py`:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'rdmo',
        'USER': 'rdmo',
        'PASSWORD': '',
        'HOST': '',
        'PORT': '',
    }
}
```

where `Name` is the name of the database, `USER` the PostgreSQL user, `PASSWORD` her password, `HOST` the database host, and `PORT` the port PostgreSQL is listening on. Note that, depending on your setup, not all settings are needed. If you are using the peer authentication methods you only need the `NAME` and `ENGINE` settings. The user and the database can be created using:


```
sudo su - postgres
createuser rdmo
createdb rdmo -O rdmo
```

This assumes peer authentication for the rdmo user.

The command

```
python manage.py migrate
```

should now create the RDMO database tables on PostgreSQL.

3.2.2 MySQL

MySQL (or community-developed fork MariaDB) can be installed using:

```
# Debian/Ubuntu
sudo apt install mysql-client mysql-server libmysqlclient-dev      # for MySQL
sudo apt install mariadb-client mariadb-server libmariadbclient-dev # for MariaDB

# CentOS
sudo yum install -y mysql mysql-server mysql-devel                # for MySQL
↪MySQL
sudo yum install -y mariadb mariadb-server mariadb-devel          # for MariaDB
↪MariaDB
sudo systemctl enable mariadb
sudo systemctl start mariadb
sudo mysql_secure_installation
```

To use MySQL as your database backend install `mysqlclient` in your virtual environment:

```
pip install -r requirements/mysql.txt
```

Then, add the following to your `config/settings/local.py`:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'rdmo',
        'USER': 'rdmo',
        'PASSWORD': 'not a good password',
        'HOST': '',
        'PORT': '',
        'OPTIONS': {
            'unix_socket': '',
        }
    }
}
```

to your `config/settings/local.py`. Here, Name is the name of the database, USER the MySQL user, PASSWORD her password, HOST the database host, and PORT the port MySQL is listening on. If you don't use `/tmp/mysql.sock`, you can use `unix_socket` to specify its path. The user and the database can be created using:

```
CREATE USER 'rdmo'@'localhost' identified by 'not a good password';
GRANT ALL ON `rdmo`.* to 'rdmo'@'localhost';
CREATE DATABASE `rdmo`;
```

on the MySQL-shell.

The command

```
python manage.py migrate
```

should now create the RDMO database tables on MySQL.

3.2.3 SQLite

SQLite ist the default option in RDMO and configured in `config/settings/base.py`. We recommend it only for a development/testing setup. It can be configured in `config/settings/local.py` by adding:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': '',
    }
}
```

where Name is the name of database file.

The command

```
python manage.py migrate
```

should now create RDMO database tables in the specified database file.

3.3 E-Mail

RDMO needs to send E-Mails to its users. The connection to the SMTP server is configured several settings in your `config/settings/local.py`:

```
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'localhost'
EMAIL_PORT = '25'
EMAIL_HOST_USER = ''
EMAIL_HOST_PASSWORD = ''
EMAIL_USE_TLS = False
EMAIL_USE_SSL = False

DEFAULT_FROM_EMAIL = ''
```

Here, `EMAIL_HOST` is the URL or IP of the SMTP server, `EMAIL_PORT` is the port it is listening on (usually 25, 465, or 587), and `EMAIL_HOST_USER` and `EMAIL_HOST_PASSWORD` are credentials if the SMTP server needs authentication.

For a STARTTLS connection (usually on port 587) `EMAIL_USE_TLS` needs to be set to `True`, while `EMAIL_USE_SSL` needs to be set to `True` for an implicit TLS/SSL connection (usually on port 465).

`DEFAULT_FROM_EMAIL` sets the FROM field for the emails send to the users.

For a development/testing setup a simple e-mail backend, which only displays the mail on the terminal can be used:

```
EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
EMAIL_FROM = 'info@example.com'
```

This is also the default backend, if no email settings are added to `config/settings/local.py`.

3.4 Authentication

RDMO has three main modes for Authentication:

- Regular user accounts with registration using the *django-allauth* library.
- Using a (read-only) connection to a *LDAP server*
- Installing a *Shibboleth* service provider next to RDMO and connect to an identity provider or even a whole Shibboleth federation.

Important: These modes are only tested individually and may be treated **mutually exclusive** (unless proven otherwise).

If none of the modes is enabled, only a very basic login will be available and users need to be created using the Django Admin Interface.

3.4.1 django-allauth

RDMO uses the excellent *django-allauth* as its main authorization library. It enables workflows for user registration and password retrieval, as well as authentication from 3rd party sites using OAUTH2.

Accounts

To enable regular accounts in RDMO add:

```
from rdmocore.settings import INSTALLED_APPS, AUTHENTICATION_BACKENDS

ACCOUNT = True
ACCOUNT_SIGNUP = True

INSTALLED_APPS += [
    'allauth',
    'allauth.account',
]

AUTHENTICATION_BACKENDS.append('allauth.account.auth_backends.AuthenticationBackend')
```

to your `config/settings/local.py`. The setting `ACCOUNT = True` enables the general *django-allauth* features in RDMO, while `ACCOUNT_SIGNUP = True` enables new users to register with your RDMO instance. The last lines enable *django-allauth* to be used by RDMO.

The behavior of *django-allauth* can be further configured by the settings documented in the *django-allauth documentation*. RDMO sets a few default which can be found in `config/settings/base.py`.

Social accounts

In order to use 3rd party accounts (facebook, github, etc.) with RDMO add:

```
from rdmo.core.settings import INSTALLED_APPS, AUTHENTICATION_BACKENDS

ACCOUNT = True
ACCOUNT_SIGNUP = True
SOCIALACCOUNT = True

INSTALLED_APPS += [
    'allauth',
    'allauth.account',
    'allauth.socialaccount'
    'allauth.socialaccount.providers.facebook',
    'allauth.socialaccount.providers.github',
    'allauth.socialaccount.providers.google',
    'allauth.socialaccount.providers.orcid',
    'allauth.socialaccount.providers.twitter',
    ...
]

AUTHENTICATION_BACKENDS.append('allauth.account.auth_backends.AuthenticationBackend')
```

to your `config/settings/local.py`. The setting `SOCIALACCOUNT = True` is used by RDMO to show certain parts of the user interface connected to 3rd party accounts, while as before, the lines after `INSTALLED_APPS` enable the feature to be used by RDMO. Each provider has a separate app you need to add to `INSTALLED_APPS`. A list of all providers supported by `django-allauth` can be found [here](#).

Once the installation is complete, the credentials of your OAUTH provider need to be entered in the admin interface. This is covered in the [administration chapter](#) of this documentation.

3.4.2 LDAP

In order to use a LDAP backend with RDMO you need to install some prerequisites. On Debian/Ubuntu you can install them using:

```
sudo apt-get install libsasl2-dev python-dev libldap2-dev libssl-dev
```

On the python side, we use `django-auth-ldap` to connect to the LDAP server. As before, it should be installed inside the virtual environment created for RDMO using:

```
pip install -r requirements/ldap.txt
```

LDAP installations can be very different and we only discuss one particular example. We assume that the LDAP service is running on `ldap.example.com`. RDMO needs a *System Account*. In order to create it, run:

```
ldapmodify -x -D 'cn=Directory Manager' -W
```

on the machine running the LDAP server and type in:

```
dn: uid=system,cn=sysaccounts,cn=etc,dc=example,dc=com
changetype: add
objectclass: account
objectclass: simplesecurityobject
uid: rdmo
userPassword: YOURPASSWORD
passwordExpirationTime: 20380119031407Z
nsIdleTimeout: 0
```

and end with a blank line followed by `ctrl-d`.

Then, in your `config/settings/local.py` add or uncomment:

```
import ldap
from django_auth_ldap.config import LDAPSearch
from rdm.core.settings import AUTHENTICATION_BACKENDS

PROFILE_UPDATE = False

AUTH_LDAP_SERVER_URI = "ldap://ldap.example.com"
AUTH_LDAP_BIND_DN = "cn=rdmo,dc=ldap,dc=example,dc=com"
AUTH_LDAP_BIND_PASSWORD = "YOURPASSWORD"
AUTH_LDAP_USER_SEARCH = LDAPSearch("dc=ldap,dc=example,dc=com", ldap.SCOPE_SUBTREE,
↪ "(uid=%(user)s)")

AUTH_LDAP_USER_ATTR_MAP = {
    "first_name": "givenName",
    "last_name": "sn",
    'email': 'mail'
}

AUTHENTICATION_BACKENDS.insert(
    AUTHENTICATION_BACKENDS.index('django.contrib.auth.backends.ModelBackend'),
    'django_auth_ldap.backend.LDAPBackend'
)
```

The setting `PROFILE_UPDATE = False` tells RDMO to disable the update form for the user profile so that users cannot update their credentials anymore. The other settings are needed by `django-auth-ldap` and are described in the [django-auth-ldap documentation](#).

3.4.3 Shibboleth

In order to use Shibboleth with RDMO it needs to be deployed in a production environment using Apache2. The Setup is documented [here](#).

Next, install the Shibboleth Apache module for service providers from your distribution repository, e.g. for Debian/Ubuntu:

```
sudo apt-get install libapache2-mod-shib2
```

In addition, `django-shibboleth-remoteuser` needs to be installed in your RDMO virtual environment:

```
pip install -r requirements/shibboleth.txt
```

Configure your Shibboleth service provider using the files in `/etc/shibboleth/`. This may vary depending on your Identity Provider. RDMO needs the `REMOTE_SERVER` to be set and 4 attributes from your identity provider:

- a username (usually `eppn`)
- an email address (usually `mail` or `email`)
- a first name (usually `givenName`)
- a last name (usually `sn`)

In our test environent this is accomplished by editing `/etc/shibboleth/shibboleth2.xml`:

```
<ApplicationDefaults entityID="https://sp.vbox/shibboleth"
    REMOTE_USER="uid eppn persistent-id targeted-id">
```

and `/etc/shibboleth/attribute-map.xml`:

```
<Attribute name="urn:oid:0.9.2342.19200300.100.1.1" id="uid"/>
<Attribute name="urn:oid:2.5.4.4" id="sn"/>
<Attribute name="urn:oid:2.5.4.42" id="givenName"/>
<Attribute name="urn:oid:0.9.2342.19200300.100.1.3" id="mail"/>
```

Restart the Shibboleth service provider demon.

```
service shibd restart
```

In your Apache2 virtual host configuration, add:

```
<Location /Shibboleth.sso>
    SetHandler shib
</Location>
<LocationMatch / (account|domain|options|projects|questions|tasks|conditions|views)>
    AuthType shibboleth
    require shibboleth
    ShibRequireSession On
    ShibUseHeaders On
</LocationMatch>
```

In your `config/settings/local.py` add or uncomment:

```
from rdmocore.settings import INSTALLED_APPS, AUTHENTICATION_BACKENDS, MIDDLEWARE_
    CLASSES

SHIBBOLETH = True
PROFILE_UPDATE = False

INSTALLED_APPS += ['shibboleth']

AUTHENTICATION_BACKENDS.append('shibboleth.backends.ShibbolethRemoteUserBackend')
MIDDLEWARE_CLASSES.insert(
    MIDDLEWARE_CLASSES.index('django.contrib.auth.middleware.AuthenticationMiddleware')
    + 1,
    'shibboleth.middleware.ShibbolethRemoteUserMiddleware'
)

SHIBBOLETH_ATTRIBUTE_MAP = {
    'uid': (True, 'username'),
    'givenName': (True, 'first_name'),
    'sn': (True, 'last_name'),
    'mail': (True, 'email'),
}

LOGIN_URL = '/Shibboleth.sso/Login?target=/projects'
LOGOUT_URL = '/Shibboleth.sso/Logout'
```

where the keys of `SHIBBOLETH_ATTRIBUTE_MAP`, `LOGIN_URL`, and `LOGOUT_URL` need to be modified according to your setup. The setting `SHIBBOLETH = True` disables the regular login form in RDMO, and tells RDMO to disable the update form for the user profile so that users cannot update their credentials anymore. The `INSTALLED_APPS`, `AUTHENTICATION_BACKENDS`, and `MIDDLEWARE_CLASSES` settings enable `django-shibboleth-remoteuser` to be used with RDMO.

Restart the webserver.

```
service apache2 restart
```

3.5 Themes

RDMO allows for a high level of customization by modifying the Django *templates* as well as the static assets (CSS file, images, etc.). Django which RDMO is based on offers a powerful method for this. Inside your `rdmo-app` directory you can create a `theme` folder with a `static` and a `templates` directory inside:

```
mkdir theme
mkdir theme/static
mkdir theme/templates
```

Then add:

```
THEME_DIR = os.path.join(BASE_DIR, 'theme')
```

to your `config/settings/local.py`.

Templates and static files in the `theme` directory override files from RDMO as long as they have the same relative path, e.g. the file `theme/templates/core/base_navigation.html` overrides `rdmo/core/templates/core/base_navigation.html`.

Some files you might want to override are:

SASS variables `rdmo/core/static/core/css/variables.scss` can be copied to `theme/static/css/variables.scss` and be used to customize colors.

Navigation bar `rdmo/core/templates/core/base_navigation.html` can be copied to `theme/templates/core/base_navigation.html` and be used to customize the navbar.

Home page text `rdmo/core/templates/core/home_text_en.html` and `rdmo/core/templates/core/home_text_de.html` can be copied to `theme/templates/core/home_text_en.html` and `theme/templates/core/home_text_de.html` and be used to customize text on the home page.

Note that updates to the RDMO package might render your theme incompatible to the RDMO code and cause errors. In this case the files in `theme` need to be adjusted to match their RDMO counterparts in functionality.

3.6 Export formats

RDMO supports exports to certain formats using the excellent [pandoc](#) converter. The list of formats to select can be customized by changing the `EXPORT_FORMATS` setting in your `config/settings/local.py`.

```
EXPORT_FORMATS = (
    ('pdf', _('PDF')),
    ('rtf', _('Rich Text Format')),
    ('odt', _('Open Office')),
    ('docx', _('Microsoft Office')),
    ('html', _('HTML')),
    ('markdown', _('Markdown')),
    ('mediawiki', _('mediawiki')),
    ('tex', _('LaTeX'))
)
```

The different formats supported by pando can be found [on the pandoc homepage](#).

3.7 Cache

RDMO uses a cache for some of it's pages. In the development setup, this is done using local-memory caching. In production, we suggest using [memcached](#). Memcached can be installed on Debian/Ubuntu using:

```
sudo apt install memcached
```

On RHEL/CentOS a few more steps are needed. First install the package using:

```
sudo yum install memcached
```

Then edit the settings file to prevent external connections:

```
# in /etc/sysconfig/memcached
PORT="11211"
USER="memcached"
MAXCONN="1024"
CACHESIZE="64"
OPTIONS="-l 127.0.0.1"
```

Then start the service:

```
systemctl start memcached
systemctl enable memcached
```

Back in your virtual enviroment, you need to install python-memcached:

```
pip install -r requirements/memcached.txt
```

and add the following to your `config/settings/local.py`:

```
CACHES = {
    {
        'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',
        'LOCATION': '127.0.0.1:11211',
        'KEY_PREFIX': 'rdmo_default'
    },
    'api': {
        'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',
        'LOCATION': '127.0.0.1:11211',
        'KEY_PREFIX': 'rdmo_api'
    }
}
```

3.8 Logging

Logging in Django can be very complex and is covered extensively in the [Django documentation](#). For a suitable logging of RDMO you can add the following to your `config/settings/local.py`:

```
import os
from . import BASE_DIR
```



```

LOGGING_DIR = '/var/log/rdmo/'
LOGGING = {
    'version': 1,
    'disable_existing_loggers': True,
    'filters': {
        'require_debug_false': {
            '()': 'django.utils.log.RequireDebugFalse'
        },
        'require_debug_true': {
            '()': 'django.utils.log.RequireDebugTrue'
        }
    },
    'formatters': {
        'default': {
            'format': '[%(asctime)s] %(levelname)s: %(message)s'
        },
        'name': {
            'format': '[%(asctime)s] %(levelname)s %(name)s: %(message)s'
        },
        'console': {
            'format': '[%(asctime)s] %(message)s'
        }
    },
    'handlers': {
        'mail_admins': {
            'level': 'ERROR',
            'filters': ['require_debug_false'],
            'class': 'django.utils.log.AdminEmailHandler'
        },
        'error_log': {
            'level': 'ERROR',
            'class': 'logging.FileHandler',
            'filename': os.path.join(LOGGING_DIR, 'error.log'),
            'formatter': 'default'
        },
        'rdmo_log': {
            'level': 'DEBUG',
            'class': 'logging.FileHandler',
            'filename': os.path.join(LOGGING_DIR, 'rdmo.log'),
            'formatter': 'name'
        },
        'console': {
            'level': 'DEBUG',
            'filters': ['require_debug_true'],
            'class': 'logging.StreamHandler',
            'formatter': 'console'
        }
    },
    'loggers': {
        'django': {
            'handlers': ['console'],
            'level': 'INFO',
        },
        'django.request': {
            'handlers': ['mail_admins', 'error_log'],
            'level': 'ERROR',
            'propagate': True
        }
    }
}

```

```
    },
    'rdmo': {
        'handlers': ['rdmo_log'],
        'level': 'DEBUG',
        'propagate': False
    }
}
```

This produces two logs:

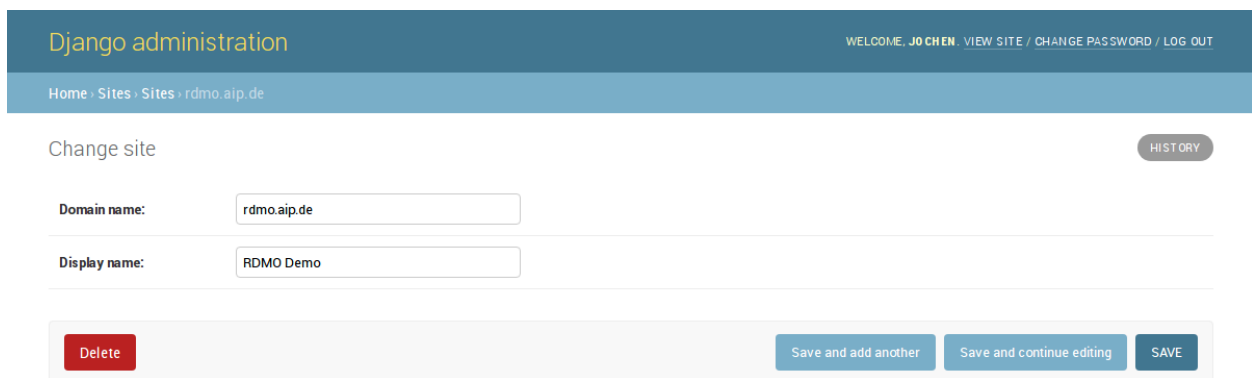
- `/var/log/rdmo/error.log` will contain exception messages from application errors (status code: 500). The messages is the same that ist shown when `DEBUG = True`, which should not be the case in a production environment. In addition to the log entry, an email is send to all admins specified in the `ADMINS` setting.
- `/var/log/rdmo/rdmo.log` will contain additional logging information from the RDMO code.

The Django framework offers a rich administration (or short admin) interface, which allows you to directly manipulate most of the entries in the database directly. Obviously, only users with the correct permissions are allowed to use this interface. The user created during the installation process using `./manage.py createsuperuser` has this *superuser* status.

The admin interface is available under the link *Admin* in the navigation bar. It will only be needed on rare occasion, since most the configuration of the questionnaire and the other functions of RDMO can be done using the more user-friendly Management interface described *in the following chapter of this documentation*.

That being said, the admin interface is needed, especially after installation, to set the title and URL of the *site*, to configure *users and groups*, to configure the connection to *OAuth providers*, and to create *tokens* to be used with the API.

4.1 Site configuration



The screenshot shows the Django administration interface for site configuration. At the top, there's a header bar with "Django administration" on the left and "WELCOME, JO CHEN. VIEW SITE / CHANGE PASSWORD / LOG OUT" on the right. Below the header, a breadcrumb trail reads "Home > Sites > Sites > rdm0.aip.de". The main content area is titled "Change site" and includes a "HISTORY" button. There are two input fields: "Domain name:" with the value "rdmo.aip.de" and "Display name:" with the value "RDMO Demo". At the bottom, there are four buttons: "Delete" (red), "Save and add another" (blue), "Save and continue editing" (blue), and "SAVE" (blue).

Fig. 4.1: Screenshot of the site admin interface.

RDMO used Django's [sites framework](#). It is therefore necessary to configure the **Domain name** and the **Display name** of your RDMO installation. This can be done in the admin interface under *SITE*. To configure your site:

1. Click on the already configured domain name *example.com*.
2. Enter the URL of your RDMO installation as **Domain name** (e.g. *rdmo.aip.de*).
3. Enter title of your RDMO installation as **Display name** (e.g. *RDMO Demo*).
4. Click **Save** to save the site.

4.2 Users and Groups

The users and groups of your RDMO instance can be managed under **AUTHENTICATION AND AUTHORIZATION**. You can create and update users and set their password directly, but most of the time this will be done by the users themselves using the account menu.

The user created in the installation process can access all features of RDMO. In order to allow other users to access the management or the admin interface, they need to have the needed permissions assigned to them. This can be done in two ways: through groups or using the superuser flag.

4.2.1 Groups

During the installation, the `./manage create-groups` command created 3 groups:

editor Users of the group editor can access the [management interface](#) and can edit all elements of the data model, except the user data entered through the structured interview.

reviewer Users of the group reviewer can access the [management interface](#), like editors, but are not allowed to change them (Save will not work). This group can be used to demonstrate the management backend of RDMO to certain users.

api Users of the group api can use the programmable API to access all elements of the data model. They will need a [token](#) to use an api client.

Existing users can be assigned to these groups to gain access to these functions:

1. Click **Users** under **AUTHENTICATION AND AUTHORIZATION** in the admin interface.
2. Click on the user to be changed.
3. Click on the group to be added to the user in the **Available groups** field.
4. Click on the little arrow to move the group to the **Chosen groups** field.
5. Save the user.

4.2.2 Superuser

Superusers have all permissions available and all permission checks will return positive for them. This does not only allow them to access the management and admin interfaces, but also **access all data from other user** (including the project pages).

To make a user superuser:

1. Click **Users** under **AUTHENTICATION AND AUTHORIZATION** in the admin interface.
2. Click on the user to be changed.
3. Tick the box **Superuser status**.

4. Save the user.

4.3 Social accounts

If you use allauth as you mode of authentication and configured RDMO to use one or more OAUTH provider (as described in the *Configuration chapter*), you need to register your RDMO site with theses services. This process is different from provider to provider. Usually, you need to provide a set of information about your site. Always included is a redirect or callback url. In the following we will use <http://127.0.0.1:8000> as an example (which will work on the development server) and you will need to replace that with the correct url of your RDMO application in production.

ORCID Login into <https://orcid.org> and go to the developer tools page at <https://orcid.org/developer-tools>. Create an app with the Redirect URI

```
http://127.0.0.1:8000/account/orcid/login/callback/
```

github Login into github and go to <https://github.com/settings/applications/new> and create a new app. Use

```
http://127.0.0.1:8000/account/github/login/callback/
```

facebook Login into facebook and go to <https://developers.facebook.com/>. Click on the top right menu *My Apps* and choose *Add a new app*. Create a new app. In the following screen choose Facebook login -> Getting started and choose *Web* as the platform. Put in a URL under which your application is accessible (Note: 127.0.0.1 will not work here.). Back on the dashboard, go to Settings -> Basic and copy the *App ID* and the *App Secret*.

twitter Login into twitter and go to <https://apps.twitter.com/app/new> and create a new app. Use

```
http://127.0.0.1:8000/account/facebook/login/callback/
```

as the Authorized redirect URI. Copy the Client-ID and the Client key.

Google Login into google and go to <https://console.developers.google.com>. Create a new project. After the project is created go to Credentials on the left side and configure the OAuth Authorization screen (second tab). Then create the credentials (first tab), more precisely a OAuth Client-ID. Use

```
http://127.0.0.1:8000/account/google/login/callback/
```

as the Authorized redirect URI. Copy the Client-ID and the Client key.

Once the credentials are obtained you need to enter them in the admin interface. To this purpose, go to **Social applications** under **SOCIAL ACCOUNTS** and click on **Add social application**. Then:

1. Select the corresponding **provider**
2. Enter a **Name** of your choice
3. Enter the **Client id** (or App ID) and the **Secret key** (or Client secret, Client key, App Secret)
4. Add your site to the chosen sites.
5. Click save.

4.4 Auth Tokens

In order to access the data entered into RDMO through the programmable API, a user needs to have a token associated with it. This is done under **AUTH TOKEN / Tokens**. To create a token, click **Add token** on the button at the right and:

1. Select the **user** for the new token.
2. Save the token.

This token can now be used instead of the username and the password when making HTTP requests from a non-browser client. To this purpose, a HTTP-Header of the form

Authorization: Token 9944b09199c62bcf9418ad846dd0e4bbdfc6ee4b

needs to be provided.

A freshly installed instance of RDMO is not very useful without a questionnaire to fill out by the user and a set of DMP templates later to be populated by the given answers. The main idea of RDMO is that every question and every output can be customized by you. This, however, introduces a certain level of complexity. RDMO employs a datamodel organized along different Django apps and models (representing database tables). A graphical overview is given in the figure below:

Fig. 5.1: Overview of the RDMO data model.

A full representation is shown [on a different page](#). Here, we explain the different parts of the data model. Each section has a link to a more detailed explanation how to create and edit the relevant elements.

For most users, the structured interview will be the most visible part of RDMO. It is configured using **catalogs**, **sections**, **subsections**, **questionsets**, and **questions**. A single installation of RDMO can have several catalogs. When creating a new project, a user can select one of these catalogs to be used with this project. A catalog has a number of sections, which themselves have subsections. Questions can be directly added to subsections, which will result in a single question on a single page of the interview. Alternatively, they can be organized into question sets. A question has a text, which will be shown in bold to the user and an optional help text. It also has a widget type, which determines which interface widget is presented to the user (e.g. text field, select field, radio buttons). The questionnaire is configured under `/questions` available in the management menu. More documentation about the questions management can be found [here](#).

The **domain model** is the central part of the data model and connects the questions from the questionnaire with the user input. It is organized as a tree-like structure. Every piece of information about a user's project is represented by an **attribute**. In this sense these attributes can be compared to a variable in source code. Attributes are the leaves of the domain model tree and can be organized into **entities**, much like files are organized along directories on a disk. Every question must have an attribute and every question set must have an entity connected to it. An example would be the attribute with the path `project/schedule/project_start` for the start date of the project. The attribute itself has the key `project_start` and resides in the entity `schedule`, which itself is located in the entity `project`.

Attributes can be marked as a collection to allow users to give several answers for the question connected to it. A question connected to this Attribute will show a button to add a new item in a new line. An example would be several keywords for a project. Questions with check box widgets also need collection attributes. Entities can also be marked

as collections. In this case users can enter sets of answers for all attributes below the entity in the tree. A question set connected to this entity will show interface elements to create new sets of answers. This can be used to ask the same set questions for different datasets or partner institutions. All entities in the tree below a collection entity adopt this behavior, so that questions about the same set can be spread over several question sets on separate pages of the interview.

Attributes have a value type which describes if the information is a piece of text, a number, or a date and can have a unit. Attributes can also have a **range**, which is used for slider widgets. Both attributes and entities can have a **verbose name**. In case of a collection this verbose name will be show to user in interface elements instead of “Add item” or “Add set”. Attributes and Entities are configured under `/domain` available in the management menu. More documentation about the options management can be found [here](#).

Attributes can further be connected to **option sets** consisting of **options**. This allows for the use of a controlled vocabulary for the user’s answers. If an attribute has one or more option sets (and the value type “Options”), the user can choose his/her answer from the different options of these option sets. Select, radio or check box widgets are used in the interview for this feature. Option sets and Options are configured under `/options` available in the management menu. More documentation about the options management can be found [here](#).

Conditions can be connected to attributes and entities and control if they are valid in the current context. If an attribute is not valid, a question connected to this attribute will not be shown to the user. Similarly, if an entity is not valid, the connected question set is not shown. Conditions are also needed to disable/enable option sets, tasks and can be used in views. Conditions are configured with a source attribute which will be evaluated, a relation like “equal” or “greater than” and a target. The target is a text string or an option. As an example, if the source is the attribute `project/legal_aspects/ipr/yesno`, the relations is “equal to”, and the target text is “1”, the condition will be true for a project where the answer to the question connected to the attribute `project/legal_aspects/ipr/yesno` is “1” (or “yes” for a yesno widget). Conditions configured under `/conditions` available in the management menu. More documentation about the conditions management can be found [here](#).

Views allow for custom DMP templates in RDMO. To this purpose every view has a template which can be edited using the Django template syntax, which is based on HTML. Views have also a title and a help text to be shown in the project overview. Views are configured under `/views` available in the management menu. More documentation about editing views can be found [here](#).

After filling out the interview the user will be presented with follow up **tasks** based on his/her answers. A task has a title and a text. **Time frames** can be added to tasks, which themselves are evaluating attributes of the value type “datetime”, to use answers like the beginning or the end of a project to compute meaningful tasks. Most of the time tasks will have a condition connected to them, to determine if this task is needed for a particular project or not. Tasks configured under `/tasks` are available in the management menu. More documentation about editing views can be found [here](#).

The different elements of the RDMO datamodel have various parameters, which control their behavior in RDMO and can be configured using the different management pages, which are decribed on the following pages. In addition, all elements contain a set of common parameters:

- An URI Prefix to identify the entity who created this element.
- A key which is the internal identifier for this element.
- An internal comment to share information to be seen by users with access to the management backend.

The key is used as an internal identifier and determines, together with the URI Prefix, the URI of the element. This URI is used as a global identifier for the export/import functionality.

5.1 Questions

The questions management is available under *Questions* in the management menu in the navigation bar. The link in the navbar opens the first catalog. Other catalogs can be selected in the sidebar afterwards.

RDMO DemoManagementAdminLanguageJochen Klar

Questions

RDMO

Section Generalrdmo/general

Subsection Topicrdmo/general/topic

Question setrdmo/general/topic/research_question → project/research_question

Questionrdmo/general/topic/research_question/title → project/research_question/title
What is the main research question of the project?

Questionrdmo/general/topic/research_question/keywords → project/research_question/keywords
Please give some keywords describing the research question.

Questionrdmo/general/topic/research_field → project/research_field/title
Which research field(s) does this project belong to?

Subsection Project schedulerdmo/general/project-schedule

Question setrdmo/general/project-schedule/schedule → project/schedule

Questionrdmo/general/project-schedule/schedule/project_start → project/schedule/project_start
When does the project start?

Questionrdmo/general/project-schedule/schedule/project_end → project/schedule/project_end
When does the project end?

Catalog

RDMO

Filter

Options

Update catalog details

Delete catalog

Create new catalog

Create new section

Create new subsection

Create new questionset

Create new question

Export

PDF

Rich Text Format

Open Office

Microsoft Office

HTML

Markdown

mediawiki

LaTeX





XML

Fig. 5.2: Screenshot of the questions management interface.

5.1. Questions

29

On the left is the main display of sections, subsections, and questions for the current catalog. For sections and subsections the title and the key is shown. For questions and question set the key and the key of the attribute or entity they are connected with is shown. The order of the different elements is the same as in the structured interview shown to the user. On the left side of each elements panel, icons indicate ways to interact the element. The following options are available:

- **Add** () a new subsection to a section, a new question or question set to a subsection or a new question to a questionset.
- **Update** () an element to change its properties.
- **Copy** () a question or questionset. This will open the same modal as update. You can the change some of the properties and save the element as a new one. This can save time when creating several similar questions.
- **Delete** () an element and all of it's decendents (e.g. a subsection and all the questions and question sets it contains). **This action cannot be undone!**

The sidebar on the right shows additional interface items:

- **Catalog** switches the view to a different Catalog.
- **Filter** filters the view according to a user given string. Only elements containing this string in their `path` will be shown.
- **Options** offers additional operations:
 - Update the details of the current catalog
 - Delete the current catalog
 - Create a new (empty) catalog
 - Create a new (empty) section
 - Create a new (empty) subsection
 - Create a new (empty) question set
 - Create a new (empty) question
- **Export** exports the current catalog to one of the displayed formats. While the text based formats are mainly for showing the full catalog, the XML export can be used to transfer this catalog to a different installation of RDMO.

The different elements of the questionnaire have different properties to control their behavior. As described in [the introduction](#), all elements have an URI Prefix, a key, and an internal comment only to be seen by other managers of the RDMO installation. In addition, you can edit the parameters below:

5.1.1 Catalog

Order Controls the position of the catalog in lists or in the interview.

Title (en) The English title for the catalog to be displayed to the user.

Title (de) The German title for the catalog to be displayed to the user.

5.1.2 Section

Catalog The catalog this section belongs to. Changing the catalog will move the section to a different catalog. Therefore it will not be visible in the current view anymore.

Order Controls the position of the section in lists or in the interview.

Title (en) The English title for the section to be displayed to the user.

Title (de) The German title for the section to be displayed to the user.

5.1.3 Subsection

Section The section this subsection belongs to. Changing the section will move the subsection into another section.

Order Controls the position of the subsection in lists or in the interview.

Title (en) The English title for the subsection to be displayed to the user.

Title (de) The German title for the subsection to be displayed to the user.

5.1.4 Question sets

Subsection The subsection this question set belongs to. Changing the subsection will move the question set into another section.

Order Controls the position of the subsection in lists or in the interview.

Entity The entity from the domain model this question set is connected to. Note that the way the question set is presented to the user is partly determined by the entity. A question connected to a collection entity will allow for answers for different sets.

Title (en) The English title for the subsection to be displayed to the user.

Title (de) The German title for the subsection to be displayed to the user.

5.1.5 Questions

Subsection The subsection this question belongs to. Changing the subsection will move the question set into another section.

Parent The question set this question belongs to. This should be “- - -” for a question added directly to a subsection and not to a question set.

Order Controls the position of the subsection in lists or in the interview.

Attribute The attribute from the domain model this question is connected to. Note that the way the question is presented to the user is partly determined by the entity. A question connected to an collection entity will allow for more than one answer and shows an “Add item” button.

Widget type The type of widget for the question. The following widgets can be selected:

- **Text** (a one line text field)
- **Textarea** (a multy line text field)
- **Yes/No** (a set of radio buttons for “Yes” and “No”)
- **Checkboxes** (a set of check boxes, the connected attribute needs to be a collection)
- **Radio Buttons** (a set of radio buttons, the connected attribute needs to have and option set)

- **Select drop down** (a drop down menu, the connected attribute needs to have an option set)
- **Range slider** (a horizontal slider, the connected attribute needs to have a range)
- **Date picker** (a drop down element with a calendar to select a date, the connected attribute needs to have the value type datetime)

Text (en) The English text for the question. The text will be shown in bold face to the user.

Title (de) The German text for the question. The text will be shown in bold face to the user.

Help (en) The English help text for the question. The help text will be shown in grey to the user.

Help (de) The German help text for the question. The help text will be shown in grey to the user.

5.2 Domain

The domain model can be managed under *Domain* in the management menu in the navigation bar.

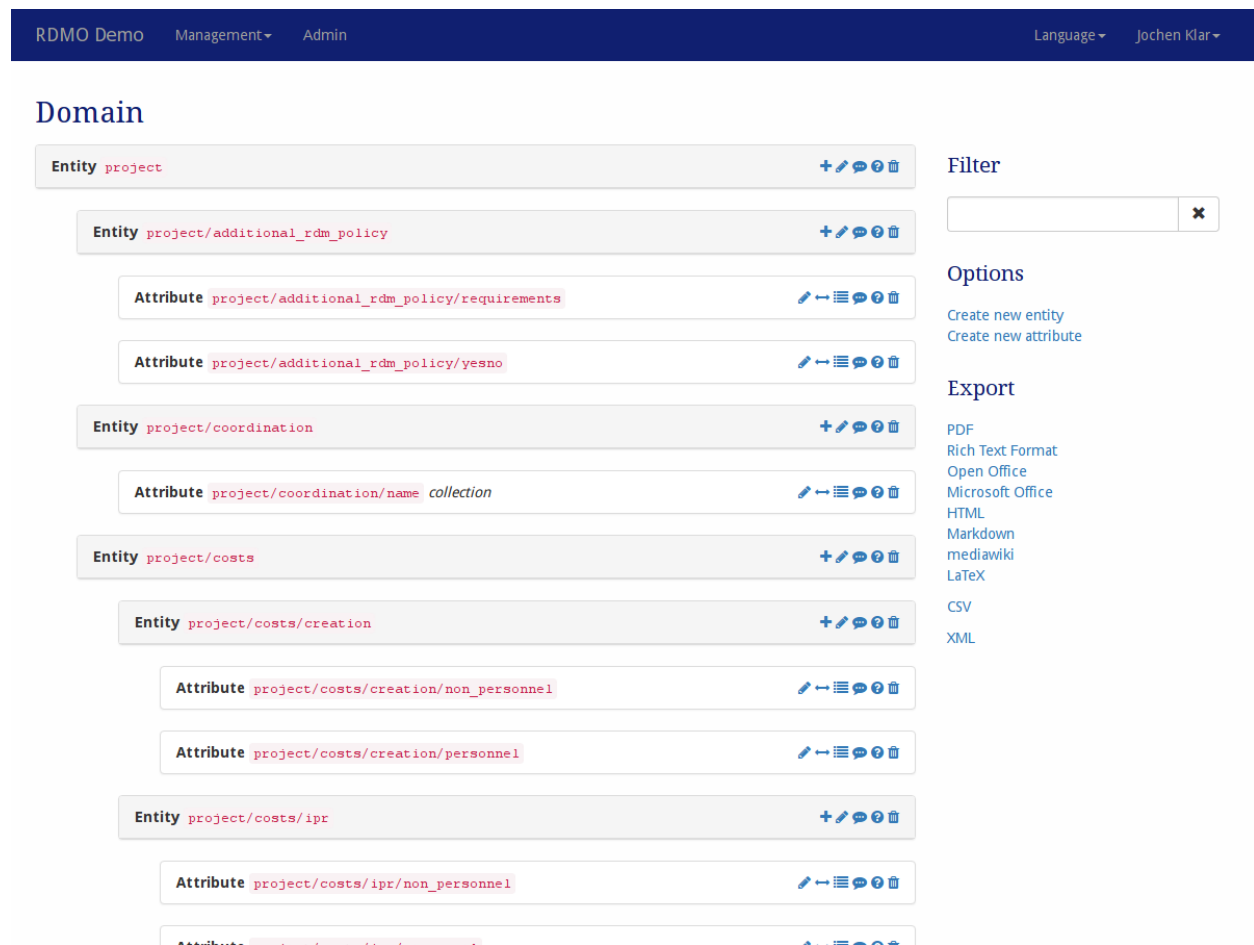









Fig. 5.3: Screenshot of the domain management interface.

On the left is the main display of all the entities and attributes available in this installation of RDMO. The entities and attributes show their path and if they are configured to be a collection. On the left side of each element's panel, icons indicate ways to interact with the element. The following options are available:

- **Add** () a new attribute or entity to an entity.
- **Update** () an entity or attribute to change its properties.
- **Update range** () of an attribute. The range is only needed if the attribute is connected to a question using the slider widget.
- **Update option sets** () of an attribute. Option sets determine the choices when the attribute is connected to a question using a select, radio button oder check boxes widget. The option sets themselves are configured in the *the options management*.
- **Update verbose name** () of an entity or attribute. For an entity, the verbose name is displayed to the user when adding sets to a question (instead of “Add set”, e.g. “Add dataset”), while for an attribute the verbose name is displayed when adding items to a question with multiple answers instead (instead of “Add item”, e.g. “Add keyword”).
- **Update conditions** () of an entity or attribute. A question connected to an attribute with one or more conditions will be skipped automatically in the questionnaire, when the condition is evaluated to be false. The same holds for question sets connected to an entity with a condition. The conditions themselves are configured in the *the conditions management*.
- **Delete** () an entity or attribute and all of it’s decendents (e.g. an entity and all the entities and attributes below in the domain model tree). **This action cannot be undone!**

The sidebar on the left shows additional interface items:

- **Filter** filters the view according to a user given string. Only elements containing this string in their path will be shown.
- **Options** offers additional operations:
 - Create a new (empty) entity
 - Create a new (empty) attribute
- **Export** exports the current catalog to one of the displayed formats. While the textual formats are mainly for presentation purposes, the XML export can be used to transfer the domain model to a different installation of RDMO.

The different elements of the domain model have different properties to control their behavior. As described in *the introduction*, all elements have an URI Prefix, a key, and an internal comment only to be seen by other managers of the RDMO installation. In addition, you can edit the parameters below:

5.2.1 Entity

Parent entity Parent entity in the domain model. Changing the parent entity will move the entity and all of it’s decendents to a differen branch of the domain model tree.

is collection Designates whether this entity can have several sets of values. A question set connected to this entity will show interface elements to create new sets of answers. All entities in the tree below a collection entity adopt this behavior, so that questions about the same set can be spread over several question sets on separate pages of the interview.

If an attribute `id` with a value type *Text* is added to the entity, users will be able to give a title to individual sets (like “Dataset A” or “Funder X”), otherwise the sets will be named #1, #2, etc.

5.2.2 Attribute

Value type Type of value for this attribute. The following types can be selected:

- **Text**
- **URL**
- **Integer**
- **Float**
- **Boolean**
- **Datetime**
- **Options**

As of now only datetime and options offer a different behavior. This will change when validation in the interview will be implemented into RDMO.

Unit Unit for this attribute. The unit will be displayed in the different output features.

Parent entity Parent entity in the domain model. Changing the parent entity will move the attribute to a different branch of the domain model tree.

is collection Designates whether this attribute can have several sets of values. A question connected to this attribute will allow the user to give several answers for a question connected to it. The question will show a button to add a new item in a new line. An example would be several keywords for a project. Questions with check box widgets also need collection attributes.

5.2.3 Range

The range is used if an attribute is connected to a question using the slider widget.

Minimum Minimal value for this attribute.

Maximum Maximum value for this attribute.

Step Step in which this attribute can be incremented/decremented.

5.2.4 Verbose name

The verbose name is configured in singular and plural in German and English and is shown on buttons and in the automatically generated help text.

Name (en) The English name displayed for this attribute/entity (e.g. project).

Plural name (en) The English plural name displayed for this attribute/entity (e.g. projects).

Name (de) The German name displayed for this attribute/entity (e.g. Projekt).

Plural name (de) The German plural name displayed for this attribute/entity (e.g. Projekte).

5.3 Options

Options and option sets can be managed under *Options* in the management menu in the navigation bar.

RDMO Demo
Management
Admin
Language
Jochen Klar

Options

Option set	<code>data_protection_laws</code>	+ / ✎ / 🗑
Option	<code>data_protection_laws/21</code>	Bundesdatenschutzgesetz (BDSG, Federal Data Protection Act) ✎ / 🗑
Option	<code>data_protection_laws/22</code>	Landesdatenschutzgesetz Baden-Württemberg (State Data Protection Act of Baden-Württemberg) ✎ / 🗑
Option	<code>data_protection_laws/23</code>	Landesdatenschutzgesetz Bayern (State Data Protection Act of Bavaria) ✎ / 🗑
Option	<code>data_protection_laws/24</code>	Landesdatenschutzgesetz Berlin (State Data Protection Act of Berlin) ✎ / 🗑
Option	<code>data_protection_laws/25</code>	Landesdatenschutzgesetz Bremen (State Data Protection Act of Bremen) ✎ / 🗑
Option	<code>data_protection_laws/26</code>	Landesdatenschutzgesetz Brandenburg (State Data Protection Act of Brandenburg) ✎ / 🗑
Option	<code>data_protection_laws/27</code>	Landesdatenschutzgesetz Hamburg (State Data Protection Act of Hamburg) ✎ / 🗑
Option	<code>data_protection_laws/28</code>	Landesdatenschutzgesetz Mecklenburg-Vorpommern (State Data Protection Act of Mecklenburg-Vorpommern) ✎ / 🗑
Option	<code>data_protection_laws/29</code>	Landesdatenschutzgesetz Hessen (State Data Protection Act of Hesse) ✎ / 🗑
Option	<code>data_protection_laws/30</code>	Landesdatenschutzgesetz Nordrhein-Westfalen (State Data Protection Act of North Rhine-Westphalia) ✎ / 🗑
Option	<code>data_protection_laws/31</code>	Landesdatenschutzgesetz Rheinland-Pfalz (State Data Protection Act of Rhineland-Palatinate) ✎ / 🗑
Option	<code>data_protection_laws/32</code>	Landesdatenschutzgesetz Niedersachsen (State Data Protection Act of Lower Saxony) ✎ / 🗑
Option	<code>data_protection_laws/33</code>	Landesdatenschutzgesetz Saarland (State Data Protection Act of Saarland) ✎ / 🗑
Option	<code>data_protection_laws/34</code>	Landesdatenschutzgesetz Sachsen (State Data Protection Act of Saxony) ✎ / 🗑
Option	<code>data_protection_laws/35</code>	Landesdatenschutzgesetz Sachsen-Anhalt (State Data Protection Act of Saxony-Anhalt) ✎ / 🗑





Filter
 ✕

Options
[Create new option set](#)
[Create new option](#)

Export
[PDF](#)
[Rich Text Format](#)
[Open Office](#)
[Microsoft Office](#)
[HTML](#)
[Markdown](#)
[mediawiki](#)
[LaTeX](#)
[XML](#)

Fig. 5.4: Screenshot of the options management interface.

On the left is the main display of all the option sets and options available in this installation of RDMO. Option sets show their key while options show their path and their text. On the left side of each elements panel, icons indicate ways to interact the element. The following options are available:

- **Add** () a new option to an option set.
- **Update** () an option set or option to change its properties.
- **Update conditions** () of an option set. An question connected to an attribute which is itself connected to an option set with one or more conditions will not show the options of the set in the questionnaire, when the condition is evaluated to be false. The conditions themselves are configured in the [the conditions management](#).
- **Delete** () an option set or option and, in the case of an option set all of it's options. **This action cannot be undone!**

The sidebar on the left shows additional interface items:

- **Filter** filters the view according to a user given string. Only elements containg this string in their path will be shown.
- **Options** offers additional operations:
 - Create a new (empty) option set
 - Create a new (empty) option
- **Export** exports the options sets to one of the displayed formats. While the textual formats are mainly for presentation purposes, the XML export can be used to transfer the options sets to a different installation of RDMO.

Option sets and options model have different properties to control their behavior. As described in [the introduction](#), all elements have an URI Prefix, a key, and an internal comment only to be seen by other managers of the RDMO installation. In addition, you can edit the parameters below:

5.3.1 Option set

Order Controls the position of the option set in lists or in the interview (if an attribute has more than one option set).

5.3.2 Option

Option set The option set this option belongs to. Changing the option set will move the option to a different option set.

Order Controls the position of the option in lists or in the interview.

Text (en) The English text for the option to be displayed to the user.

Text (de) The German text for the option to be displayed to the user.

Additional input Designates whether an additional input is possible for this option. In this case a text box is displayed to the radio button or check box. Usually this is used for an option “Other”.

5.4 Conditions

Conditions can be created and managed under *Conditions* in the management menu in the navigation bar. They are later connected to Entities, Attributes, Option sets, or Tasks.

RDMO DemoManagementAdminLanguageJochen Klar

Conditions

Condition additional_rdm_policy

project/additional_rdm_policy/yesno is equal to (==) "Yes"

Condition data_sharing

project/dataset/sharing/yesno is equal to (==) "Yes, externally for everyone"

Condition infrastructure_resources

project/dataset/usage_infrastructure is equal to (==) "The following infrastructure resources are needed"

Condition intellectual_property_rights

project/legal_aspects/ipr/yesno is equal to (==) "1"

Condition international

project/legal_aspects/international_yesno is equal to (==) "1"

Condition law_international

project/legal_aspects/international_yesno is equal to (==) "1"

Condition other_sensitive_data

project/dataset/sensitive_data/other/yesno is equal to (==) "1"

Condition personal_data

Filter

Options

Create new condition

Export



PDF
Rich Text Format
Open Office
Microsoft Office
HTML
Markdown
mediawiki
LaTeX
XML

Fig. 5.5: Screenshot of the conditions management interface.

5.4. Conditions

37

On the left is the main display of all the conditions available in this installation of RDMO. Conditions show their key and a textual representation of what they evaluate. On the left side of each conditions panel, icons indicate ways to interact the element. The following options are available:

- **Update** () a condition to change its properties.
- **Delete** () a condition. **This action cannot be undone!**

The sidebar on the left shows additional interface items:

- **Filter** filters the view according to a user given string. Only conditions containing this string in their path will be shown.
- **Options** offers additional operations:
 - Create a new condition
- **Export** exports the conditions to one of the displayed formats. While the textual formats are mainly for presentation purposes, the XML export can be used to transfer the conditions to a different installation of RDMO.

Conditions have different properties to control their behavior. As described in [the introduction](#), all elements have an URI Prefix, a key, and an internal comment only to be seen by other managers of the RDMO installation. In addition, you can edit the parameters below:

5.4.1 Condition

Conditions are configured with a source attribute which will be evaluated, a relation like “equal” or “greater than” and a target. The target is a text string or an option. As an example, if the source is the attribute `project/legal_aspects/ipr/yesno`, the relation is “equal to”, and the target text is “1”, the condition will be true for a project where the answer to the question connected to the attribute `project/legal_aspects/ipr/yesno` is “1” (or “yes” for a yesno widget).

Source The Attribute this condition is evaluating.

Relation The Relation this condition is using.




Target (Text) If using a regular attribute, the text value this condition is checking against.

Target (Option) If using an options attribute, the option this condition is checking against.

5.5 Views

Views can be configured under *Views* in the management menu in the navigation bar.

On the left is the main display of all the views available in this installation of RDMO. Views show their key, title and description. On the left side of each views panel, icons indicate ways to interact the element. The following options are available:

- **Update** () a view to change its properties.
- **Edit the template** () of a view.
- **Delete** () a view. **This action cannot be undone!**
















The sidebar on the right shows additional interface items:

- **Filter** filters the view according to a user given string. Only views containing this string in their path will be shown.

RDMO DemoManagementAdmin

LanguageJochen Klar

Views

View bielefeld Bielefeld	  
DMP template from the University of Bielefeld.	
View citec CITEC DMP	  
DMP template from the University of Bielefeld for CITEC funded research projects.	
View dmponline DMPonline template	  
Template from DMPonline, online https://dmponline.dcc.ac.uk	
View dmptool DMPTool template	  
Template from DMPTool, based on "NSF-GEN: Generic", online: https://dmptool.org	
View horizon2020 Horizon 2020 FAIR Data Management Plan template	  
Template for Horizon 2020, from "Guidelines on FAIR Data Management in Horizon 2020", online: http://ec.europa.eu/research/participants/data/ref/h2020/grants_manual/hi/oa_pilot/h2020-hi-oa-data-mgt_en.pdf	

Filter

×

Options

[Create new view](#)

Export

[PDF](#)
[Rich Text Format](#)
[Open Office](#)
[Microsoft Office](#)
[HTML](#)
[Markdown](#)
[mediawiki](#)
[LaTeX](#)
[XML](#)

Project RDMO

Contact

Impressum

Fig. 5.6: Screenshot of the views management interface.

- **Options** offers additional operations:
 - Create a new view
- **Export** exports the conditions to one of the displayed formats. While the textual formats are mainly for presentation purposes, the XML export can be used to transfer the views to a different installation of RDMO.

Views have different properties to control their behavior. As described in [the introduction](#), all elements have an URI Prefix, a key, and an internal comment only to be seen by other managers of the RDMO installation. In addition, you can edit the parameters below:

5.5.1 View

Title (en) The English title for the view. The title will be shown in the projects overview.

Title (de) The German title for the view. The title will be shown in the projects overview.

Help (en) The English help text for the view. The help text will be shown in the projects overview.

Help (de) The German help text for the view. The help text will be shown in the projects overview.

5.5.2 Template

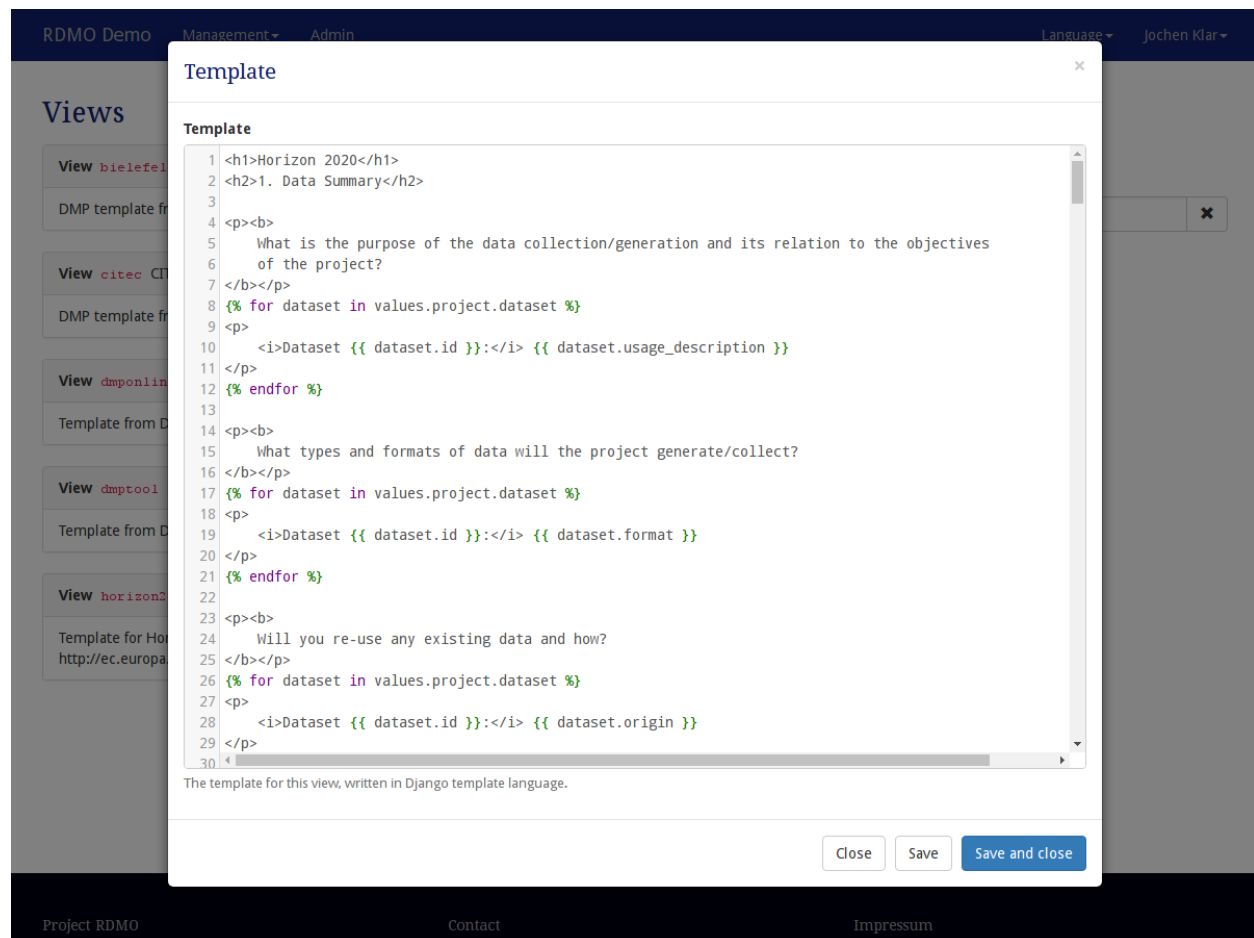


Fig. 5.7: Screenshot of the template modal.

Each view has a template, which determines how the answers given by the user are mapped to a textual document. The template is composed using the [Django template](#) syntax, which is a combination of regular HTML, variables, which get replaced with values when the template is evaluated (`{{ a_variable }}`), and tags, which control the logic of the template (`{% a_tag %}`).

Two variables can be used when used in RDMO templates:

- `values`, which contains nested dictionary mapping the users answers to their attributes.
- `conditions`, which is a dictionary mapping the keys of the conditions to the evaluated conditions according to the current project (i.e. True or False).

Consider an attribute `project/research_question/title` (more specific an attribute `title` in the entity `research_question` in the entity `project`) and a user, who answered the question connected to this attribute with “To boldly go where no man has gone before.”. The attribute would be available in the template as `values.project.research_question.title` (note the `.` instead of `/`). Used in the template using the syntax for a variable:

```
The main research question of the project is: {{ values.project.research_question.  
→title }}
```

would, when evaluated in the context by a user in his/her project, render:

```
The main research question of the project is: To boldly go where no man has gone_  
→before.
```

Collections can be rendered using the `for` tag of the Django template syntax.

```
<ul>  
{% for keyword in project.research_question.keywords %}  
  <li>{{ keyword }}</li>  
{% endfor %}  
</ul>
```

The usual filters of the Django syntax can also be used, e.g.

```
<p>  
  {{ values.project.research_question.keywords | join:', ' }}  
</p>
```

For collection entities, you can use:

```
{% for dataset in values.project.dataset %}  
<p>  
  <i>Dataset {{ dataset.id }}:</i> {{ dataset.usage_description }}  
</p>  
{% endfor %}
```

Conditions can be used using the `if` tag:

```
{% if conditions.personal_data %}  
This will be only rendered if personal_data resolves to be true.  
{% endif %}
```

Please consult the documentation of the Django template syntax for all the available tags and filters: <https://docs.djangoproject.com/en/1.11/ref/templates/language>.

5.6 Tasks

Tasks can be configured under *Tasks* in the management menu in the navigation bar.

The screenshot shows the RDMO Tasks management interface. The top navigation bar includes 'RDMO Demo', 'Management', 'Admin', 'Language', and 'Jochen Klar'. The main content area is titled 'Tasks' and displays a list of tasks. Each task entry includes a key, a title, a description, and a set of action icons (edit, conditions, time frame, delete). The sidebar on the right contains a 'Filter' input field, 'Options' with a 'Create new task' link, and 'Export' options including PDF, Rich Text Format, Open Office, Microsoft Office, HTML, Markdown, mediawiki, LaTeX, and XML.

Task	Key	Title	Description	Actions
Task	clarify_consequences_international_law	Clarify consequences of different international jurisdictions	Please get in touch with the legal department or a respective contact person at your institution to clarify if this has consequences for the project and its data management and if yes, what consequences these are.	[Edit] [Conditions] [Time Frame] [Delete]
Task	contact_data_security_officer	Contact data security officer about special requirements	At least one of the datasets contains particularly sensitive personal information according to BDSG § 3, 9. Please get in touch with the data protection officer of your institution to check which additional protection measures are necessary.	[Edit] [Conditions] [Time Frame] [Delete]
Task	contact_IT_department	Contact IT department to acquire infrastructure resources	Please get in touch with your IT department to arrange that the required infrastructure resources are provided.	[Edit] [Conditions] [Time Frame] [Delete]
Task	contact_IT_or_data_management_expert	Contact IT or data management expert	Please contact an IT or data management expert to get support in terms of data usage.	[Edit] [Conditions] [Time Frame] [Delete]
Task	contact_rights_owner	Contact rights owner	Clarify, if and how this restricts the possibility of long-term preservation and re-use and if the rights owner is willing to grant the necessary rights.	[Edit] [Conditions] [Time Frame] [Delete]
Task	create_naming_policy	Create data naming policy	Create a guideline for a consistent naming of the data within the project.	[Edit] [Conditions] [Time Frame] [Delete]
Task	create_data_organisation_policy	Create data organisation policy	Create a policy for a consistent data organisation in the project.	[Edit] [Conditions] [Time Frame] [Delete]

Fig. 5.8: Screenshot of the tasks management interface.

On the left is the main display of all the tasks available in this installation of RDMO. Tasks show their key, title and the text describing the task. On the left side of each task's panel, icons indicate ways to interact with the element. The following options are available:

- **Update** (🔧) a task to change its properties.
- **Update conditions** (❓) of a task. A task will only be shown to the user if all of its conditions are evaluated True. The conditions themselves are configured in the *the conditions management*.
- **Update the time frame** (🕒) of a task. The time frame is constructed from one or two dates from the user's answers. This allows for tasks about a specific deadline or special period.
- **Delete** (🗑️) a task. **This action cannot be undone!**

The sidebar on the left shows additional interface items:

- **Filter** filters the view according to a user given string. Only tasks containing this string in their path will be shown.

- **Options** offers additional operations:
 - Create a new task
- **Export** exports the conditions to one of the displayed formats. While the textual formats are mainly for presentation purposes, the XML export can be used to transfer the tasks to a different installation of RDMO.

Tasks have different properties to control their behavior. As described in [the introduction](#), all elements have an URI Prefix, a key, and an internal comment only to be seen by other managers of the RDMO installation. In addition, you can edit the parameters below:

5.6.1 Task

Title (en) The English title for the view. The title will be shown in the projects overview.

Title (de) The German title for the view. The title will be shown in the projects overview.

Text (en) The English text for the view. The text will be shown in the projects overview.

Text (de) The German text for the view. The text will be shown in the projects overview.

5.6.2 Time frame

Start date attribute The attribute that is setting the start date for this task. The attribute needs to be of value type *datetime*.

End date attribute The Attribute that is setting the end date for this task (optional, if no end date attribute is given, the start date attribute sets also the end date). The attribute needs to be of value type *datetime*.

Days before Additional days before the start date.

Days after Additional days after the end date.

5.7 Export and Import

Upgrade

The `rdmo` package can be conveniently upgraded using the `pip` command. However, before you perform any changes to your installation, please backup the important components to a save location.

A PostgreSQL or MySQL database can be database can be dumped into a file using:

```
pg_dump [DBNAME] > rdmo.sql # PostgreSQL  
mysqldump -uroot -p [DBNAME] > rdmo.sql
```

Your `rdmo-app` directory (including any `sqlite3` database) can be copied using the usual commands. Note that your virtual environment will not work after being moved to a different path.

In order to upgrade your RDMO installation go to your `rdmo-app` directory, activate your virtual environment, and upgrade the `rdmo` package using `pip`:

```
pip install --upgrade rdmo
```

In order to install a specific version (e.g. 0.9.0) of RDMO use:

```
pip install --upgrade rdmo==0.9.0
```

After the upgrade a database migration might be necessary:

```
python manage.py migrate
```

Please check the release notes if this, or other, steps are necessary.

6.1 Upgrade to version 0.9.0

With version 0.9.0 we introduced the split into the `rdmo-app` and the centrally maintained `rdmo` package. Therefore a few additional steps are needed to upgrade any earlier version to 0.9.0 or beyond:

1. In any case perform a backup of your `rdmo` directory and your database as described above.

2. Perform the steps *Obtaining the app directory* and *Install python packages* as if you would install a new instance of RDMO.
3. Copy your old configuration from `/path/to/old/rdmo/rdmo/settings/local.py` to `/path/to/new/rdmo-app/config/settings/local.py`. The new config directory replaces the old rdmo directory.
4. If you already have a theme directory, copy it into the new `rdmo-app` folder.
5. Run a database migration (Unless you skipped several versions, the output should be No migrations to apply.):

```
python manage.py migrate
```

6. Download the front-end files from the CDN. We don't use bower and npm anymore.

```
python manage.py download_vendor_files
```

7. Update the path to the `wsgi.py` script in your Apache or nginx configuration. It is now under `/path/to/new/rdmo-app/config/wsgi.py`.
8. Redeploy RDMO as described under `/deployment/redeploy`.

If you have trouble with the upgrade process, don't hesitate to contact the RDMO team for support.

7.1 Testing

For running the test suite use:

```
./manage.py test
```

For a coverage report use:

```
./manage.py coverage  
./manage.py coverage --html # for an HTML coverage report
```

The HTML report can be viewed by opening `htmlcov/index.html` in a browser.

7.2 Internationalisation

To update the locale files automatically run:

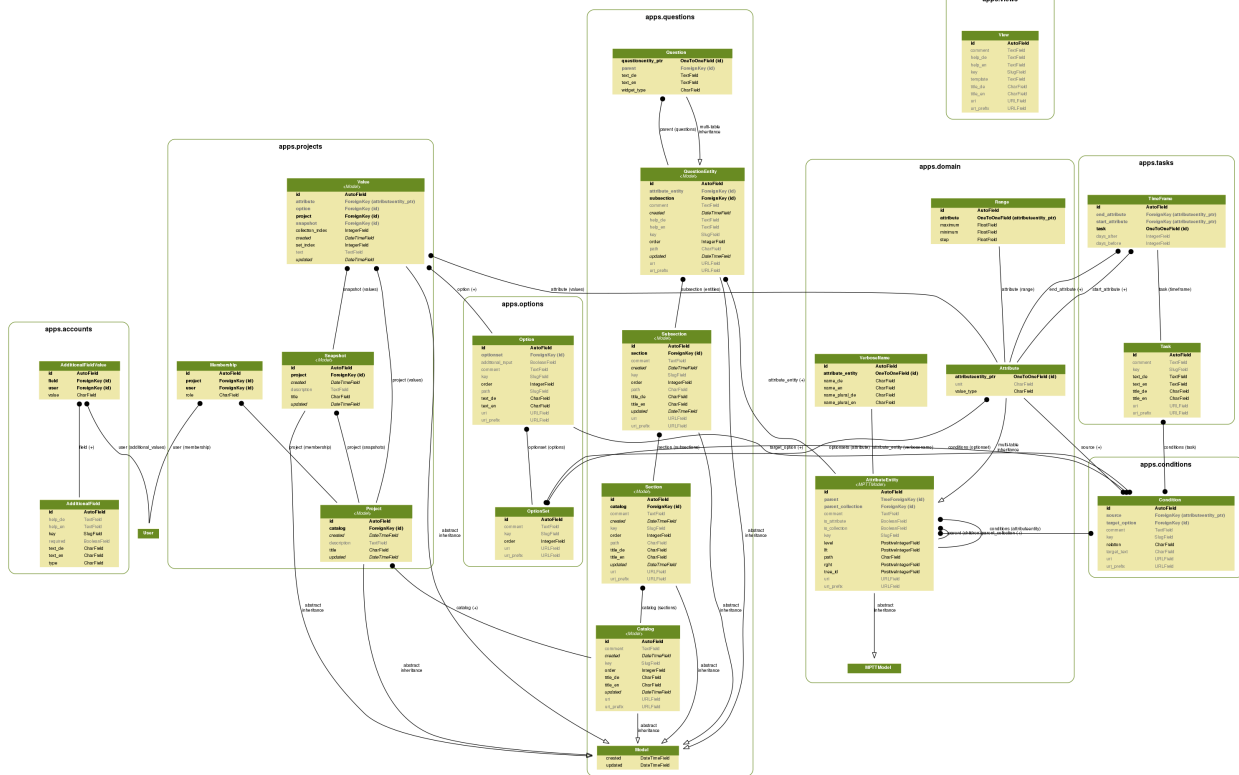
```
python manage.py makemessages -a --ignore=env/* --ignore=htmlcov/*
```

Then, edit the `.po` files in the `locale` directory. Afterwards run

```
python manage.py compilemessages
```

7.3 Figures

Below is a graphical representation of the different models in rdm, created with graphviz:



To create/update the figure, install *graphviz*:

```
sudo apt install graphviz-dev # Debian/Ubuntu
sudo yum install graphviz-devel # RHEL/CentOS
```

Then, in your virtual environment install *pygraphviz*:

```
pip install pygraphviz
```

Then create the image using:

```
./manage.py graph_models \
    accounts conditions domain options questions projects tasks views \
    -g > docs/_static/img/models.dot

dot -Tsvg -o docs/_static/img/models.svg docs/_static/img/models.dot
dot -Tpdf -o docs/_static/img/models.pdf docs/_static/img/models.dot
dot -Tpng -o docs/_static/img/models.png docs/_static/img/models.dot
```

7.4 Documentation

In order to build the documentation, additional dependencies must be installed in your virtual enviroment:

```
pip install sphinx sphinx-autobuild sphinx_rtd_theme
```

Then change to the `docs` directory and run `make live` to live-edit the documentation.