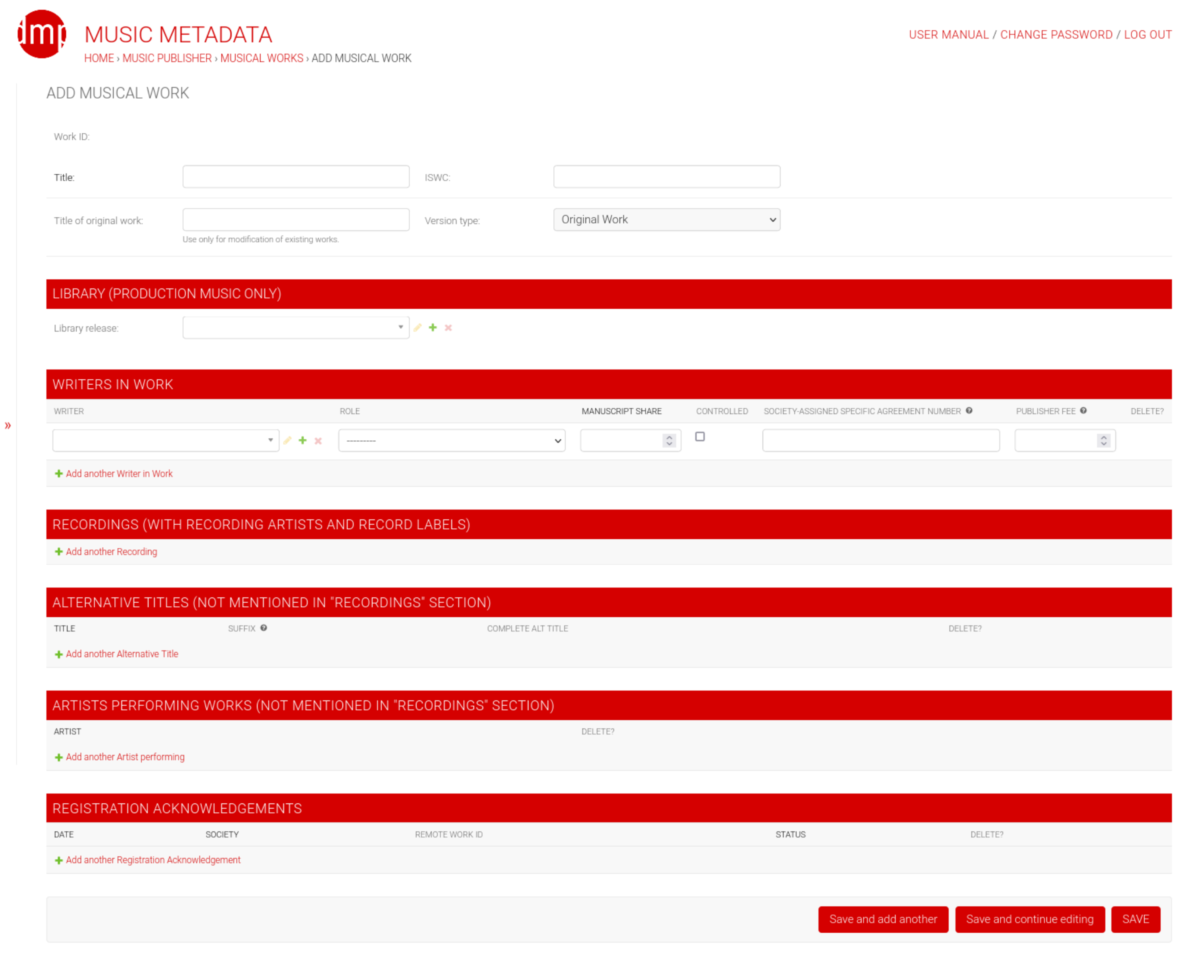
DMP Documentation

Release 23.4

Matija Kolarić

Apr 12, 2023

Contents



MUSIC METADATA

HOME › MUSIC PUBLISHER › MUSICAL WORKS › ADD MUSICAL WORK

USER MANUAL / CHANGE PASSWORD / LOG OUT

ADD MUSICAL WORK

Work ID:			
Title:		ISWC:	
Title of original work:	Use only for modification of existing works.	Version type:	Original Work

LIBRARY (PRODUCTION MUSIC ONLY)

Library release:

WRITERS IN WORK

WRITER	ROLE	MANUSCRIPT SHARE	CONTROLLED	SOCIETY-ASSIGNED SPECIFIC AGREEMENT NUMBER	PUBLISHER FEE	DELETE?
	---------		☐			

+ Add another Writer in Work

RECORDINGS (WITH RECORDING ARTISTS AND RECORD LABELS)

+ Add another Recording

ALTERNATIVE TITLES (NOT MENTIONED IN "RECORDINGS" SECTION)

TITLE	SUFFIX	COMPLETE ALT TITLE	DELETE?

+ Add another Alternative Title

ARTISTS PERFORMING WORKS (NOT MENTIONED IN "RECORDINGS" SECTION)

ARTIST	DELETE?

+ Add another Artist performing

REGISTRATION ACKNOWLEDGEMENTS

DATE	SOCIETY	REMOTE WORK ID	STATUS	DELETE?

+ Add another Registration Acknowledgement

Save and add another | Save and continue editing | SAVE

DMP (Django-Music-Publisher) is open-source music catalogue management software for

- **management of music metadata**,
- **registration of musical works**,
- **royalty statement processing**, and
- **basic data distribution**.

DMP will work for most *small* publishers world-wide, but it does not try to solve first-world problems (e.g. US PRO rules) for free. See *Features and Limitations*.

Based on Django web framework, DMP is primarily designed to be deployed to a cloud, but it can be installed to a personal computer. (Linux, Mac OS or Windows). See *Installation* for details.

CHAPTER 1

Introduction

1.1 Music Metadata Management

DMP (Django-Music-Publisher) is free, open-source software for **managing music metadata**:

- musical works and recordings (with audio files),
- writers, artists and labels (with photos/logos),
- releases/albums (with cover art), and
- music libraries.

Fig. 1: Simplified Class Diagram

1.2 Common Works Registration (CWR)

It implements CWR protocol for **batch registration of musical works** with Collective Management Organizations (CMOs) and Digital Service Providers (DSPs).

Fig. 2: Sequence diagram: Work registration and incoming royalty statements

1.3 Royalty Management

Simple **royalty processing** calculations can split received royalties among controlled writers and calculate publisher fees.

Incoming data is accepted as a CSV file. If registrations are done using CWR, work matching is fully automatic. Output is a similar CSV file with additional rows and columns.

Fig. 3: Sequence diagram: Processing incoming royalty statements

This file can be then imported into Excel and turned into individual outgoing statements and accounting data using pivot tables. This process can be automated using Excel templates and simple scripts.

1.4 Data Distribution

Besides the aforementioned CWR protocol, music metadata can be exported in several other formats, or be accessed through the read-only *REST API*.

CHAPTER 2

Features and Limitations

Key features and limitations of DMP are listed below. DMP works for most publishers world-wide, but not nearly for all.

By reading this section, you may save a lot of your time.

DMP is open-source, you are free to modify it to suit your needs. This may require serious software development skills.

If you need additional features, That Green Thing *(available as Software-as-a-Service) is the recommended solution. Notes about differences between it and DMP are in green.*

2.1 Music metadata management

DMP can store detailed metadata for musical works and recordings, data about writers, recording and performing artists, releases (albums), labels and music libraries, as well as CWR exports and acknowledgements.

2.2 Total data validation

All entered data is validated for CWR and DDEX compatibility on field-, record-, and transaction-level.

The flip side of the coin is that you can not enter incomplete data into DMP and hope to fix it later. The author does not believe in fixing in post.

2.3 Single controlled original publisher

DMP supports only a single original controlled publisher (single publishing entity), entered through settings.

It will **not work** for publishers with multiple entities, most notably US publishers affiliated with multiple PROs.

That Green Thing *fully supports multiple controlled publishers, administration, global sub-publishing, etc.*

2.4 No other publishers

DMP holds **no data** about **other/uncontrolled publishers**. Uncontrolled writers appear as unpublished in CWR files.

That Green Thing *holds data about uncontrolled original publishers and administrators.*

2.5 Co-publishing

With DMP, **co-publishing deals** are possible, with each publisher registering their own shares. In this case, the other publisher appears as unknown in CWR files.

That Green Thing *has full support for co-publishing deals.*

2.6 Manuscript shares

DMP uses a **single manuscript share** model, splits between writers (prior to publishing) are same for performance, mechanical and synchronisation rights.

That Green Thing *has share fields for performance, mechanical and sync shares for both writers and original publishers. Manuscript shares are calculated back if required.*

2.7 Original publishing agreement data

Basic **original publishing agreement** data can be entered, sufficient for registrations in societies that require **society-assigned agreement numbers**.

2.8 Share transfer

Share transfer from a controlled writer to the publisher can be configured, in accordance with national regulations and customs. There is only a **single setting for all controlled writers**.

Default is *London rule* (50% performance, 100% mechanical), but this can be reconfigured.

That Green Thing *uses explicitly entered shares after original agreements. By default, they remain the same in case of sub-publishing, but this can be overridden on per-work basis.*

2.9 Publisher fees

Publisher fees are customisable per-writer, or even per-writer-per-work.

2.10 No support for composite works

Composite musical works, as well as recordings based on multiple musical works (e.g. medleys), are not supported.

2.11 Registrations

Registrations can be exported as **CWR** files. Supported versions are: 2.1, 2.2, 3.0, and 3.1.

Acknowledgement files in CWR format can be imported.

CWR preview features syntax highlighting with additional data on mouse-over, for both CWR files generated by DMP and imported acknowledgements.

2.12 Defaults when creating CWR files

When creating CWR, many fields are left with blank/zero values. When the fields are required in CWR, it uses reasonable defaults, e.g.:

- *Musical Work Distribution* is set to *Unclassified*,
- *Recorded indicator* is set to *Yes* or *Unknown*, depending if recording metadatawas entered, and
- *Work for Hire*, *Grand Rights Indicator*, *Reversionary Indicator*, and *First Recording Refusal Indicator* are set to No.

2.13 Royalty management

Incoming royalty statements in CSV format can be processed, resulting in CSV statements containing data for distribution between controlled interested parties. Statement processing is extremely fast.

Actual outgoing statements must be created in Excel using pivot tables. For experienced Excel user, this takes about 10 minutes for the first statement and then about 30 seconds per statement for all others. This process can be fully automated by using scripts.

That Green Thing *can be configured with templates, so that outgoing statements come in any format you desire.*

2.14 Data imports and exports

Data about works can be imported from *CSV* files.

Data for selected works can be exported as *JSON* (complete) or *CSV* (partial).

That Green Thing *can import multiple formats, including EBR and CWR.*

2.15 Audio files and images

If persistent file storage is available, images can be uploaded (photos for writers and artists, logos for labels, cover arts for releases), as well as audio files.

2.16 Sharable playlists

Playlists can be created and shared, protected only by secret URLs.

2.17 REST API

Read-only REST API, with basic HTTP authentication, is available. It can be used for:

- Complete data export
- Metadata exchange
- Content exchange

CHAPTER 3

Support

No individual support is available for DMP, which is usual for open-source projects. Because most music professionals are unfamiliar with this concept, an explanation is due here.

Creator of this software is not the vendor for your instance of DMP. You are.

If you run into an issue with any third party, most notably a CMO or their administrative agency, and they tell you to ask your software vendor for support, that would be you.

If you forget your password and have to reset it, there is no one who can do it but you.

If you have any issues whatsoever, solving them is solely your responsibility.

Having said that, here is what you can try.

3.1 Documentation

The first step in resolving any issues is reading the relevant parts of this documentation, most notably the *User Manual*. If you are not sure what is relevant, use search. If it does not help, read everything.

3.2 Videos

Go to *Videos* and watch the videos. If you are not familiar with **all** terms from Music Metadata Basics series, watch the relevant videos. Then watch the whole DMP series.

3.3 Questions and Discussions

If you still don't know how to resolve the issue, you should try asking in the Facebook Group Music Publisher Support. Before you do, search the group for similar questions.

Alternatively, you can do it in Discussions within the code repository.

3.4 Bug and Feature Requests

If you believe you have encountered a bug in DMP, you can search through issues, or raise a new one.

How can you be sure if it is a bug? Here are some rules.

3.4.1 Bug

If you repeatedly encounter *500 Server Error*, it is a bug. Please report it. All such bugs are usually fixed within a week. Make sure you are following the thread, so you know when the bug is fixed. You still need to update your DMP instance yourself.

3.4.2 Not a bug

If you see errors, that are incomprehensible to you, during data imports or acknowledgement imports, this is not a bug. You can still report it as issue, but that will not help you in short term. Creator of DMP might make the error more comprehensible in one of the next versions, but don't count on it.

3.4.3 Unlikely a bug

If things work, but not the way you would like, that is probably not a bug. You can still raise an issue, and it will be investigated. However, if it is not a bug, it might be rejected without an explanation.

3.4.4 Upgrade to Commercial

That Green Thing *is the commercial upgrade to DMP. It has many more features and comes with professional support. Migration from an unmodified DMP instance is included in the price.*

CHAPTER 4

Quality Assurance

This project is now close to five years old and, as any continuously developed project, has legacy issues. If anyone tells you that their project has no legacy issues, they are either ignorant or lying. Probably both.

Here is how issues are reduced, caught and fixed in this project.

4.1 Test coverage, Continuous Integration, Continuous Deployment

4.1.1 Test Coverage

For years, test coverage was around 99% (mostly functional tests), and the goal is to keep it over 99.5% (rounded to 100%) for major releases.

4.1.2 Continuous Integration

These tests are run on every push to the code repository, (together with code style validation).

4.1.3 Manual Testing

Before each major release, all functionality is manually tested.

Of course, there is a small chance that some edge case is not covered, and that someone will hit a bug in production, but it is reduced to the minimum.

4.2 Code Style, Complexity and Maintainability

Some of these issues can be detected and/or measured, sometimes even fixed, with standard tools. Code style, complexity and maintainability are good examples.

4.2.1 Code Style

Code style in this project is current Black, with line length of 79 characters. This is validated on every push.

4.2.2 Code Complexity

Recently, code complexity has been improved. No code block has nor should have complexity over `C (20)`. Average should remain around `A (3.0)`.

4.2.3 Code Maintainability

Code maintainability is to be improved, currently 2 files have dead low index, due to their size. The goal is to have `A` across all files for the next major release (in 2024).

CHAPTER 5

Release History

Django-Music-Publisher was originally released in July 2018, and for the rest of 2018, development was very rapid, with major improvements being released in August, September and November.

From January 2019 to January 2022, major versions were released twice per year.

Minor versions, with bug fixes and security updates, are released when required. They are not mentioned in this document.

5.1 Major Release History

5.1.1 18.7 - 18.11

Initial release in July 2018 had a very simple data structure. It used external API for CWR generation. The code was open-source, but it was dependant on a free tier of a commercial service.

5.1.2 19.1 Epiphany

This version was focused on making DMP completely independent of any software not available as open-source and compatible with the MIT license.

CWR generation and complete data validation was added to the open-source code. Full support for modified works was added, as well as basic co-publishing support. Data export in JSON format was added.

5.1.3 19.7 Metanoia

This version was about making DMP compatible with both current and future requirements within the precisely defined scope: **single publisher**, **single manuscript share**. (This scope has not changed since, nor will in the future.)

Most notably, support for multiple recordings per work and CWR 3.0 (labeled as "experimental") were added. CWR preview, for both versions, received basic syntax highlighting. Since this version, CWR files are zipped.

5.1.4 20 Twenty

Twenty-twenty was primarily about simplified deployment. Since this version, DMP can be deployed to the Free Heroku dyno (container) by non-techies.

Note: This free service was cancelled in late 2022. See "Rubicon" below.

Support for custom global share splits was added. MR/SR affiliations for writers were also added. Syntax highlighting for CWR acknowledgements was added, to simplify dealing with conflicts and other registration-related issues.

5.1.5 20.7 Endemic

This version added a lot of new features!

Processing of royalty statements is the most important new feature since the initial release. It can import statements in practically **any** CSV format. Processing is extremely fast.

Basic CSV imports and exports for musical works, and JSON exports for releases were added.

ISWCs can now be imported from CWR acknowledgements. Controlled writers with no society affiliation are now fully supported.

Index (home) page became clearer due to grouping of views. User manual was reorganised to follow the same structure. `User manual` links now lead to the relevant page in the user manual.

5.1.6 21.1 Victor

This version was focused on improving and extending existing features.

Support for CWR was extended to include latest revisions:

- CWR 2.1 Revision 8,
- CWR 2.2 Revision 2 (includes cross-references),
- CWR 3.0 Revision 0 (includes cross-references, experimental), and
- CWR 3.1 DRAFT (includes cross-references, experimental).

CWR Syntax highlighting was improved and now includes all fields DMP generates from data, with more detailed descriptions on mouse-over, for all supported CWR versions.

A side menu was added to all add/change/view pages, making navigation faster.

5.1.7 21.5 Mayday

The version focuses on improving data exchange with other solutions, most notably That Green Thing.

- Support for writers with IPI numbers, but without CMO affiliations was improved
- Internal notes for writers, artists and labels were added
- More data is available in CSV exports:
  - separate manuscript, performance, mechanical and sync shares for writers,
  - data about an original publisher, with performance, mechanical and sync shares,
  - data about recordings, including recording ID, record labels and recording artists, and
  - society Work IDs.
- More data is available in CSV imports:

    - data about recordings: ISRC, duration, release date, and
    - society work IDs.
- Improved support for ISWC imports and duplicate handling.
- Interface now also available in *dark* mode

5.1.8 22.1 Exofile

With very little to do in the realm of music publishing, within the defined scope, DMP has moved towards supporting music companies who are **both publishers and labels**.

This version added support for file uploads, either locally (for traditional installations) or to S3 storage (for containers). Please consult *Installation* for instructions how to enable and configure file storage.

Writers, artists, labels and releases received `image` and `description` fields, to be used in front-end representations. Recordings received an `audio_file` field.

Read-only REST API endpoints are available for releases and recording artists, enabling integration with websites.

Playlists can now be created, either by manually adding recordings, or by using batch actions in various list views, and shared using secret URLs.

Full metadata backup can be download using REST API endpoint.

5.1.9 23.4 Rubicon

As the release name suggests, this release is a game changer. Not necessarily in a good way for small music publishers without development/IT skills.

Since version *20 Twenty*, it was possible for anyone to deploy DMP to a free cloud account using a wizard. The free cloud service no longer exists, so the wizard was removed.

Deploying to Heroku and Digital Ocean is still possible for those who can read and follow installation instructions.

`Account #` field was added to the `Writer` model. This field can be used for linking royalty statement data with accounting. This is the only visible change to an end user within DMP.

Several important projects based on TGT were released in the previous 3 years, not only targeting music publishers, but also CMOs (societies). That is what open source projects are really about, and DMP will in the future be more focused on providing the core for such projects. Optionally combined with consulting by the author and the team.

Source code has been reviewed and partly cleaned up, with average complexity reduced to `A` and no block more complex than `C`. Code style is now validated with Black.

Introduction chapter of this documentation was extended with graphs, and split into two separate documents. Several external articles were linked to improve clarity.

5.2 Future open-source features

Nothing is planned for the foreseeable future. Unless there is a significant change in the industry, the next major release will be out in 2024. Bugfix and security releases will be coming out when required.

CHAPTER 6

Related Videos

Note: All videos listed here were released before Heroku cancelled their free tier. Any information about DMP installation (deployment) is outdated and probably wrong. Otherwise, all videos are still completely accurate.

6.1 DMP

2022 video series about DMP.

Total playlist duration is around 30 minutes.

6.2 Music Publishing 101

Music Publishing 101 videos are good, technically oriented introduction into music publishing with practical software examples. DMP is used in videos 2-8, serving as video tutorials for DMP.

Total playlist duration is around 1 hour.

6.3 Music Royalty Management

Music Royalty Management videos cover royalty management, using DMP for examples in several episodes.

Total playlist duration is around 20 minutes.

CHAPTER 7

Installation

Code repository for DMP can be found at https://github.com/matijakolaric-com/django-music-publisher.

7.1 Installation to a cloud

DMP (Django-Music-Publisher) is based on Django Web Framework (https://djangoproject.org), and requires Python 3 (https://python.org). It can be installed to a PC, but installing it into a cloud is highly recommended.

Digital Ocean is the recommended provider.

7.1.1 Digital Ocean

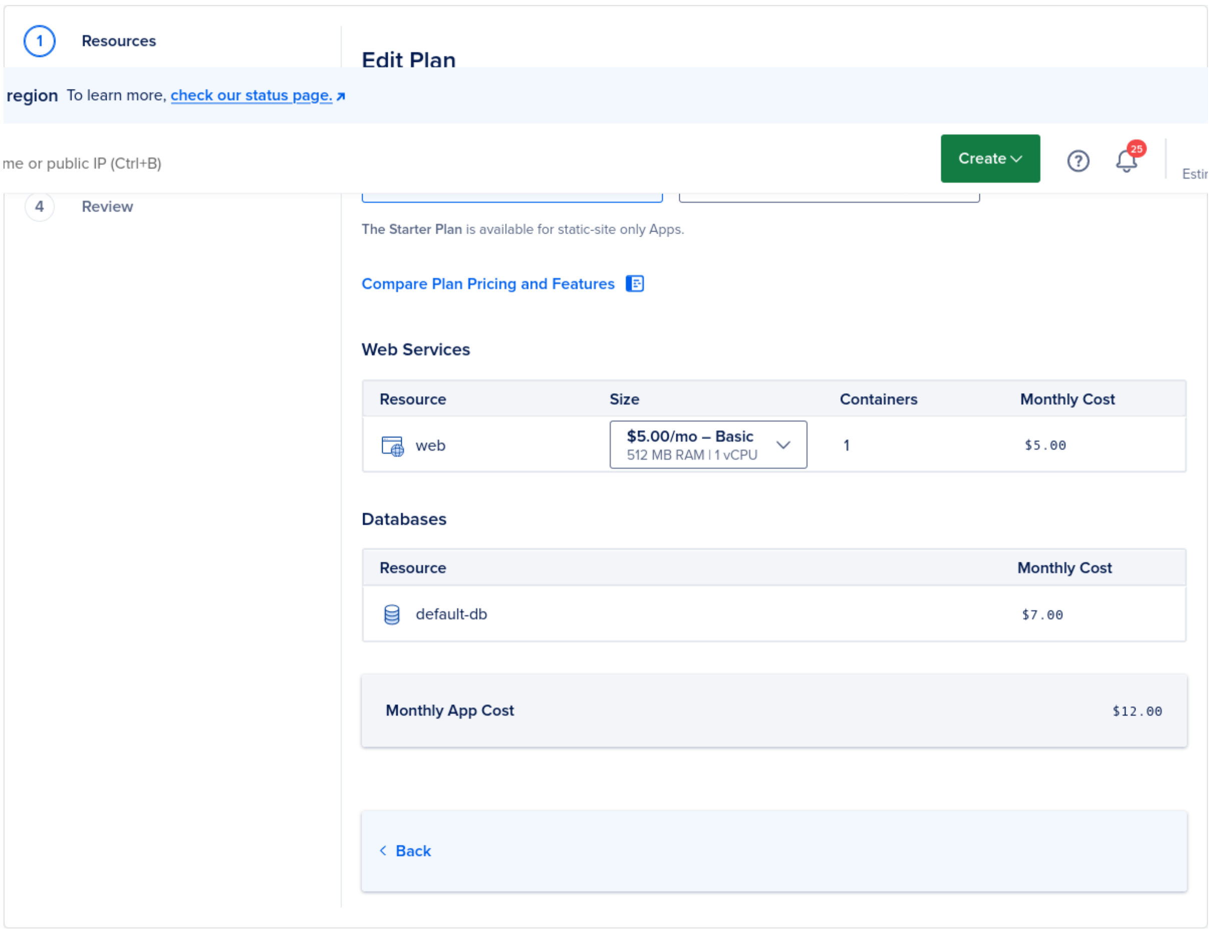
Minimal monthly cost is $5 for the application, $7 for the database, so $12 in total. Optional $5 for file storage is only required for experimental features.

They usually give free credits that must be used within 60 days.

1. Click on the button below. (This is an affiliate link, providing you with free credits.)
2. Wizard

Once you have registered, click on the next button to start the installation wizard.

2.1. In the first step, edit the plan and select Basic, then the cheapest plan, this is enough for publishers with up to several thousand works.


1 Resources

Edit Plan

region To learn more, check our status page.

me or public IP (Ctrl+B)

Create

Estir

4 Review

The Starter Plan is available for static-site only Apps.

Compare Plan Pricing and Features

Web Services

Resource	Size	Containers	Monthly Cost
web	$5.00/mo – Basic 512 MB RAM | 1 vCPU	1	$5.00

Databases

Resource	Monthly Cost
default-db	$7.00

Monthly App Cost $12.00

Back

2.2 Edit `web` environment variables. See *settings* for details. Click on **SAVE**!!

2.3 Select region closest to you.

2.4 Review and click on “create resources”.

3. Installation takes several minutes. Once it is done, click on the `console` tab and enter:

```
python manage.py migrate
python manage.py createsuperuser
```

Then enter your user name and password (twice). You can leave e-mail empty, it is not used.

If you forget your login/password, you can use the console for adding a new superuser or change the password with:

```
python manage.py changepassword
```

7.1.2 Heroku

This is another provider with semi-automated deployment. The deployment to Heroku using the button below is NOT tested, and issues with deployment will not be tested nor fixed.

7.2 Custom installation

For everything else, basic programming and/or system administration skills are required.

Start with Deploying Django documentation.

If you plan to use Django-Music-Publisher as one of the apps in your Django project, there is nothing special about it:

```
pip install --upgrade django_music_publisher
```

Add `music_publisher.apps.MusicPublisherConfig` to `INSTALLED_APPS`. Almost everything goes through the Django Admin. The only exception is royalty calculation, which has to be added to `urls.py`

```
from music_publisher.royalty_calculation import RoyaltyCalculationView

urlpatterns = [
    ...
    path('royalty_calculation/', RoyaltyCalculationView.as_view(), name='royalty_
↪calculation'),
]
```

Experimental features (involving file system) may require additional work.

Good luck!

7.3 Settings

There are several environment variables that need to be set, and several optional ones. Note that if invalid data is entered or required data is not entered, deployment may fail and/or application may break down.

7.3.1 Secret key

Django requires `SECRET_KEY` to be set. It can be any random string. You can use https://miniwebtool.com/django-secret-key-generator/ to generate one, but do change it somewhat after pasting for complete security.

7.3.2 Publisher-related settings

- `PUBLISHER_NAME` - Name of the publisher using Django-Music-Publisher, **required**
- `PUBLISHER_IPI_NAME` - Publisher's IPI *Name* Number, **required**
- `PUBLISHER_CODE` - Publisher's CWR Delivery code, defaults to `000`, which is not accepted by CMOs, but may be accepted by (sub-)publishers.
- `PUBLISHER_SOCIETY_PR` - Publisher's performance collecting society (PRO) numeric code, required. See *Collective management organisations*.
- `PUBLISHER_IPI_BASE` - Publisher's IPI *Base* Number, rarely used
- `PUBLISHER_SOCIETY_MR` - Publisher's mechanical collecting society (MRO) numeric code
- `PUBLISHER_SOCIETY_SR` - Publisher's synchronization collecting society numeric code, rarely used

For the list of codes, please have a look at societies.csv file in the music_publisher folder of the code repository.

7.3.3 Agreement-related settings

These settings define the percentage of the manuscript share transferred to the publisher. The default is "London Split", where 50% of performance and 100% of mechanical and sync rights are transferred.

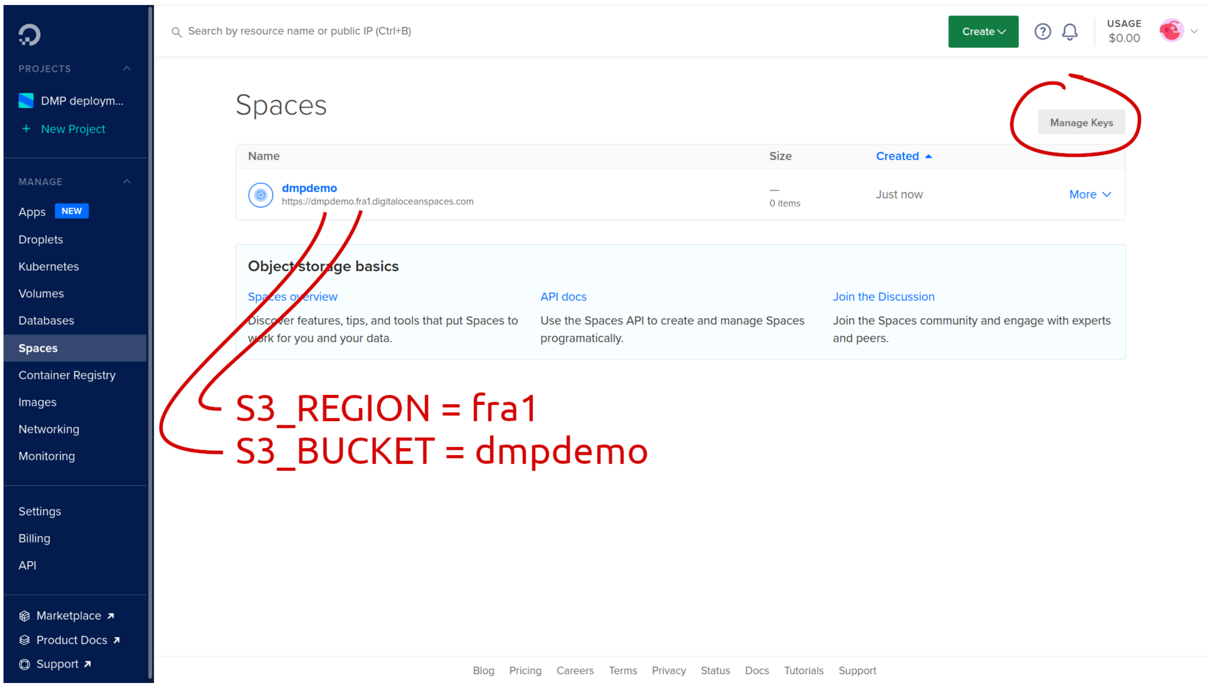
- `PUBLISHING_AGREEMENT_PUBLISHER_PR` - Performance share transferred to the publisher, default is '0.5' (50%)

- `PUBLISHING_AGREEMENT_PUBLISHER_MR` - Mechanical share transferred to the publisher, default is '1.0' (100%)
- `PUBLISHING_AGREEMENT_PUBLISHER_SR` - Synchronization share transferred to the publisher, default is '1.0' (100%)

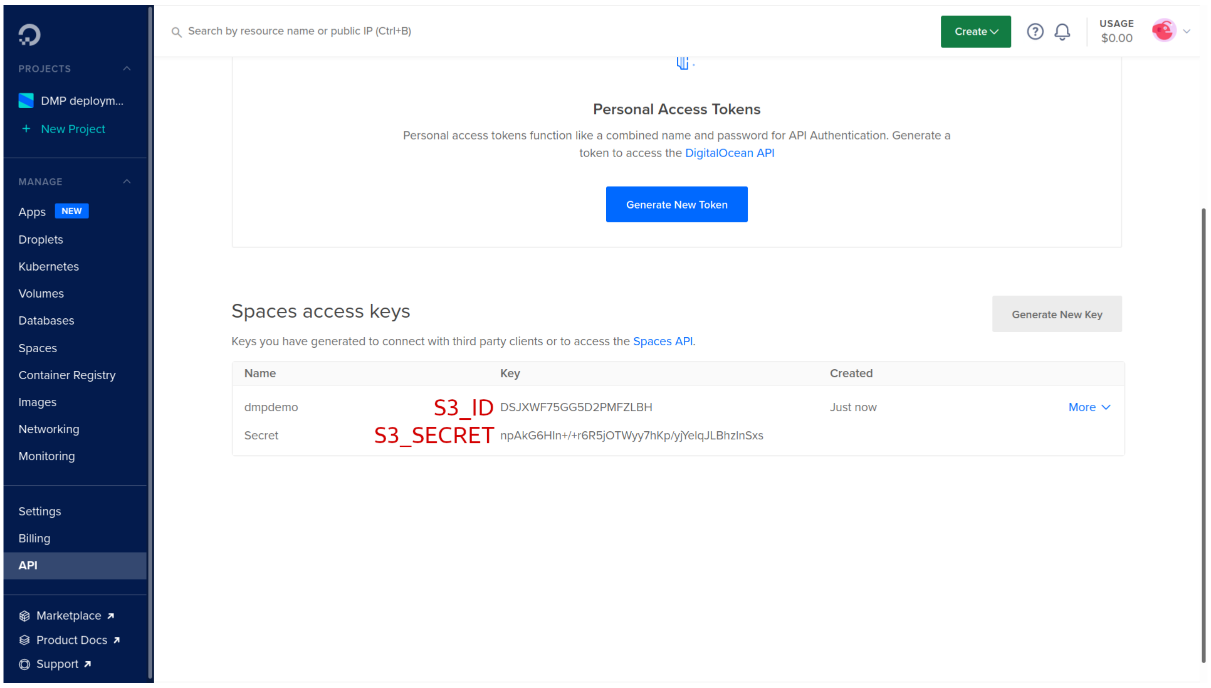
Enter `1.0` for 100%, `0.5` for 50%, `0.3333` for 33.33%, etc.

7.3.4 S3 storage

For Digital Ocean Spaces, you need to set up only four config (environment) variables. AWS and other S3 providers will also work.



- `S3_REGION` (alias for `AWS_S3_REGION_NAME`) and `S3_BUCKET` (alias for `AWS_STORAGE_BUCKET_NAME`), you get them when you set up your *Spaces*, and



- `S3_ID` (alias for `AWS_ACCESS_KEY_ID`) and `S3_SECRET` (alias for `AWS_SECRET_ACCESS_KEY`), you get them when you generate your *Spaces* API key.

If you want to use AWS or some other S3 provider, the full list of settings is available here.

7.3.5 Other options

- `OPTION_FORCE_CASE` - available options are `upper`, `title` and `smart`, converting nearly all strings to UPPER CASE or Title Case or just UPPERCASE fields to Title Case, respectively. If unset, everything is left as entered.
- `OPTION_FILES` - enables support for file uploads (audio files and images), using local file storage (PC & VPS)

7.4 Collective management organisations

Following list contains official CWR codes for CMOs, to be entered in `PUBLISHER_SOCIETY_PR`, `PUBLISHER_SOCIETY_MR` and rarely `PUBLISHER_SOCIETY_SR` environment variables.

1	ACUM	ISRAEL
2	ADDAF	BRAZIL
3	AEPI	GREECE
4	AGADU	URUGUAY
5	AKM	AUSTRIA
6	BUCADA	CENTRAL AFRICAN REPUBLIC
7	APDAYC	PERU
8	APRA	AUSTRALIA
9	ARTISJUS	HUNGARY
10	ASCAP	UNITED STATES
11	AUSTRO-MECHANA (AUME)	AUSTRIA
12	AMCOS	AUSTRALIA
14	ARGENTORES	ARGENTINA
15	APA	PARAGUAY
16	BUMDA	MALI
17	AMRA	UNITED STATES
18	BGDA	GUINEA
19	BMDA	MOROCCO
20	SODRAC	CANADA
21	BMI	UNITED STATES
22	MCSN	NIGERIA
23	BUMA	NETHERLANDS
24	BURIDA	COTE D'IVOIRE
25	SODAV	SENEGAL
26	CASH	HONG KONG
28	LITA	SLOVAKIA
29	SCD	CHILE
30	AMAR	BRAZIL
31	DILIA	CZECH REPUBLIC
32	FILSCAP	PHILIPPINES
33	OMDA	MADAGASCAR
34	HFA	UNITED STATES
35	GEMA	GERMANY
36	IPRS	INDIA
37	BUBEDRA	BENIN

Continued on next page

Table 1 – continued from previous page

38	JASRAC	JAPAN
39	MUSICAUTOR	BULGARIA
40	KODA	DENMARK
41	LITERAR-MECHANA	AUSTRIA
43	MCSK	KENYA
44	MCPS	UNITED KINGDOM
45	BBDA	BURKINA FASO
47	BCDA	CONGO
48	NCB	DENMARK
49	ONDA	ALGERIA
50	OSA	CZECH REPUBLIC
51	PROLITTERIS	SWITZERLAND
52	PRS	UNITED KINGDOM
54	ALCS	UNITED KINGDOM
55	SABAM	BELGIUM
56	SACD	FRANCE
57	SACERAU	EGYPT
58	SACEM	FRANCE
59	SACM	MEXICO
60	SACVEN	VENEZUELA
61	SADAIC	ARGENTINA
62	SADEMBRA	BRAZIL
63	SAMRO	SOUTH AFRICA
64	SOKOJ	SERBIA AND MONTENEGRO
65	SAYCE	ECUADOR
66	SBACEM	BRAZIL
67	SBAT	BRAZIL
68	SDRM	FRANCE
69	SPA	PORTUGAL
70	SOGEM	MEXICO
71	SESAC Inc.	UNITED STATES
72	SGAE	SPAIN
73	SCAM	FRANCE
74	SIAE	ITALY
75	SUISSIMAGE	SWITZERLAND
76	ACEMLA	PUERTO RICO
77	STEF	ICELAND
78	STEMRA	NETHERLANDS
79	STIM	SWEDEN
80	SUISA	SWITZERLAND
82	OTPDA	TUNISIA
84	SAYCO	COLOMBIA
85	SOZA	SLOVAKIA
86	SICAM	BRAZIL
87	SPACEM	FRANCE
88	CMRRA	CANADA
89	TEOSTO	FINLAND
90	TONO	NORWAY
91	SSA	SWITZERLAND
93	UBC	BRAZIL
94	RAO	RUSSIAN FEDERATION
95	VG WORT	GERMANY
96	COTT	TRINIDAD AND TOBAGO
97	ZAIKS	POLAND

Continued on next page

Table 1 – continued from previous page

98	ZIMURA	ZIMBABWE
101	SOCAN	CANADA
102	NASCAM	NAMIBIA
103	ACDAM	CUBA
104	MACP	MALAYSIA
105	MASA (RMS)	MAURITIUS
106	COMPASS	SINGAPORE
107	ACAM	COSTA RICA
108	CHA	TAIWAN, CHINESE TAIPEI
109	KCI	INDONESIA
110	LATGA-A	LITHUANIA
111	HDS-ZAMP	CROATIA
112	SAZAS	SLOVENIA
115	UCMR-ADA	ROMANIA
116	EAU	ESTONIA
117	MESAM	TURKEY
118	KOMCA	KOREA, REPUBLIC OF
119	MCSC	CHINA
120	LIRA	NETHERLANDS
121	VDFS	AUSTRIA
122	AKKA-LAA	LATVIA
124	COSOMA	MALAWI
125	BNDA	NIGER
126	MCT	THAILAND
127	ALBAUTOR	ALBANIA
128	IMRO	IRELAND
129	SOBODAYCOM	BOLIVIA
130	BUTODRA	TOGO
131	SADA	GREECE
132	BILD-KUNST	GERMANY
133	ZAMCOPS	ZAMBIA
134	SLPRS	SRI LANKA
135	SADH	GREECE
136	ZAMP - Macédoine	MACEDONIA
137	SOFAM	BELGIUM
138	KOPIOSTO	FINLAND
139	VISDA	DENMARK
140	UACRR	UKRAINE
141	ATN	CHILE
142	DALRO	SOUTH AFRICA
143	TEATERAUTOR	BULGARIA
144	HAA	CROATIA
145	DIRECTORS UK	UNITED KINGDOM
146	SPAC	PANAMA
147	FILMAUTOR	BULGARIA
148	ADAGP	FRANCE
149	ARS	UNITED STATES
151	BONO	NORWAY
152	Bildupphovsrätt (Visual Copyright Society)	SWEDEN
153	DACS	UNITED KINGDOM
154	HUNGART	HUNGARY
155	SOMAAP	MEXICO
156	VAGA	UNITED STATES
157	BILDRECHT GmbH	AUSTRIA

Continued on next page

Table 1 – continued from previous page

158	VEGAP	SPAIN
159	VISCOPY	AUSTRALIA
160	NCIP	BELARUS
161	MÜST	TAIWAN, CHINESE TAIPEI
162	AMPAL	AUSTRALIA
163	APG-Japan	JAPAN
164	APSAV	PERU
166	AUTORARTE	VENEZUELA
168	CA	AUSTRALIA
169	COSCAP	BARBADOS
170	CPSN	NEPAL
171	CREAIMAGEN	CHILE
172	DGA	UNITED STATES
173	DIRECTORES	MEXICO
174	FILMJUS	HUNGARY
175	CopyRo	ROMANIA
176	JACAP	JAMAICA
177	KazAK	KAZAKSTAN
178	KOSA	KOREA, REPUBLIC OF
179	KUVASTO	FINLAND
181	NMPA	UNITED STATES
182	PAPPRI	INDONESIA
183	SACK	KOREA, REPUBLIC OF
184	SARTEC	CANADA
186	SGDL	FRANCE
187	SNAC	FRANCE
189	SOCINPRO	BRAZIL
190	SOPE	GREECE
191	SPACQ	CANADA
192	SFF	SWEDEN
193	The Society of Authors (SOA)	UNITED KINGDOM
194	UFFICIO GIURIDICO	HOLY SEE (VATICAN CITY STATE)
195	VEVAM	NETHERLANDS
196	WGA	UNITED STATES
197	WGJ	JAPAN
198	ZAMP Association of Slovenia	SLOVENIA
199	SFP-ZAPA	POLAND
200	MSG	TURKEY
201	ABRAMUS	BRAZIL
202	AsDAC	MOLDOVA, REPUBLIC OF
203	AWGACS	AUSTRALIA
204	GCA (former SSA)	GEORGIA
206	UFW	FINLAND
207	The Author’s Registry Inc.	UNITED STATES
208	SGA	GUINEA-BISSAU
209	ARMAUTHOR NGO	ARMENIA
210	ACCESS COPYRIGHT	CANADA
212	CSCS	CANADA
213	DRCC	CANADA
214	ECCO	SAINT LUCIA
215	Kyrgyzpatent	KYRGYZSTAN
216	SQN	BOSNIA AND HERZEGOVINA
217	ABRAC	BRAZIL
218	ANACIM	BRAZIL

Continued on next page

Table 1 – continued from previous page

219	ASSIM	BRAZIL
220	ATIDA	BRAZIL
221	SABEM	BRAZIL
222	FONOPERU	PERU
223	COSOTA	TANZANIA, UNITED REPUBLIC OF
224	SOMAS	MOZAMBIQUE
225	SAIF	FRANCE
226	AACIMH	HONDURAS
227	SGACEDOM	DOMINICAN REPUBLIC
228	ROMS	RUSSIAN FEDERATION
229	ICG	UNITED STATES
230	ADAVIS	CUBA
231	AUTVIS	BRAZIL
232	GESTOR	CZECH REPUBLIC
233	SACEMLUXEMBOURG	LUXEMBOURG
234	UPRS	UGANDA
235	SACENC	FRANCE
236	ARTEGESTION	ECUADOR
237	TALI	ISRAEL
238	BSCAP	BELIZE
239	CMC	CAMEROON
240	DAMA	SPAIN
241	NICAUTOR	NICARAGUA
242	SACIM	EL SALVADOR
243	SADIA	ANGOLA
244	SASUR	SURINAME
245	SETEM	TURKEY
246	VCPMC	VIET NAM
247	IVARO	IRELAND
248	DAC	ARGENTINA
249	PAM CG	MONTENEGRO
250	AEI-GUATEMALA	GUATEMALA
251	ASDACS	AUSTRALIA
252	COLCCMA	TAIWAN, CHINESE TAIPEI
253	AAS	AZERBAIJAN
254	SOCILADRA	CAMEROON
256	PICTORIGHT	NETHERLANDS
257	SAVA	ARGENTINA
258	MRCSN	NEPAL
259	SDCSI	IRELAND
260	ACS	UNITED KINGDOM
261	GAI Uz	UZBEKISTAN
262	SINEBIR	TURKEY
263	SACS	SEYCHELLES
264	CARCC	CANADA
265	MACA	MACAU
266	BeAT	BRUNEI DARUSSALAM
267	UPRAVIS	RUSSIAN FEDERATION
268	COSON	NIGERIA
269	WAMI	INDONESIA
270	JASPAR	JAPAN
271	DHFR	CROATIA
272	MOSCAP	MONGOLIA
273	AMUS	BOSNIA AND HERZEGOVINA

Continued on next page

Table 1 – continued from previous page

274	AuPO CINEMA	UKRAINE
275	AUTODIAHIRISI	GREECE
276	DASC	COLOMBIA
277	RSAU	RWANDA
278	RUR	RUSSIA
279	SDADV	ANDORRA
280	SANASTO	FINLAND
281	ATHINA- SADA - S.A.D.A.	GREECE
282	UNAC-SA	ANGOLA
283	CAPASSO	SOUTH AFRICA
284	COSOZA	ZANZIBAR
285	GHAMRO	GHANA
286	ODDA	DJIBOUTI
287	KORRA	KOREA
288	ABYROY	KAZAKHSTAN
289	AIPA	SLOVENIA
290	AZDG	AZERBAIJAN
291	OFA	SERBIA
292	ZPAP	POLAND
293	DBCA	BRAZIL
294	REDES	COLOMBIA
295	SAGCRYT	MEXICO
296	DACIN-SARA	ROMANIA
297	GEDAR	BRAZIL
298	OOA-S	CZECH REPUBLIC
299	SCM-COOPERATIVA	CAPE VERDE
300	WID Centre	UNITED STATES
301	GESAC	BELGIUM
302	LATINAUTOR	URUGUAY
303	NORD-DOC	SWEDEN
304	SONGCODE	UNITED STATES
306	ACCS	TRINIDAD AND TOBAGO
307	MIS@ASIA	SINGAPORE
308	ECAD	BRAZIL
309	LatinNet	SPAIN
310	DIVA	HONG KONG
311	MCPS-PRS Alliance	UNITED KINGDOM
312	CISAC	FRANCE
313	FastTrack DCN	FRANCE
314	IDA	FRANCE
315	CSI	FRANCE
316	CIS-Net AVI	FRANCE
317	INTL-REP	FRANCE
318	SGS	
319	ICE Services AB	SWEDEN
320	ARMONIA	FRANCE
321	PUBLISHERS	
322	EVA	BELGIUM
635	GEMA-US	Additional CIS-Net Node
658	SACEM-US	Additional CIS-Net Node
672	SGAE-NY	Additional CIS-Net Node
707	MusicMark	USA
758	SACEM-LIBAN	Additional CIS-Net Node
775	Solar EMI	GERMANY/UK

Continued on next page

Table 1 – continued from previous page

776	Solar Sony	GERMANY/UK
777	CELAS	GERMANY/UK
778	GMR	
779	Polaris Nordic	SCANDINAVIA
780	UNISON	Spain
781	SOUNDREEF	ENGLAND and WALES
782	NexTone	JAPAN
888	PAECOL	Additional CIS-Net Node

CHAPTER 8

MIT License

CHAPTER 9

User Manual

Note: This user manual applies to version 23.4 Rubicon.

9.1 Basics

This section explains the very basics, logging in, home view and general overview of model views.

9.1.1 Login

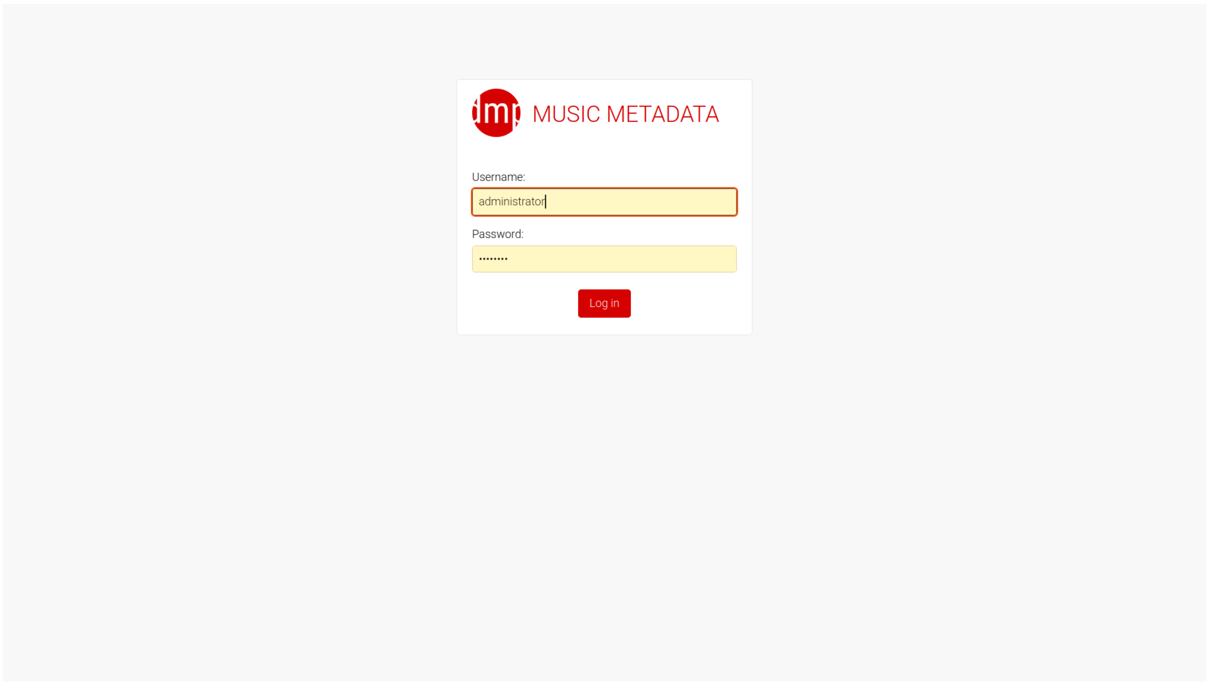
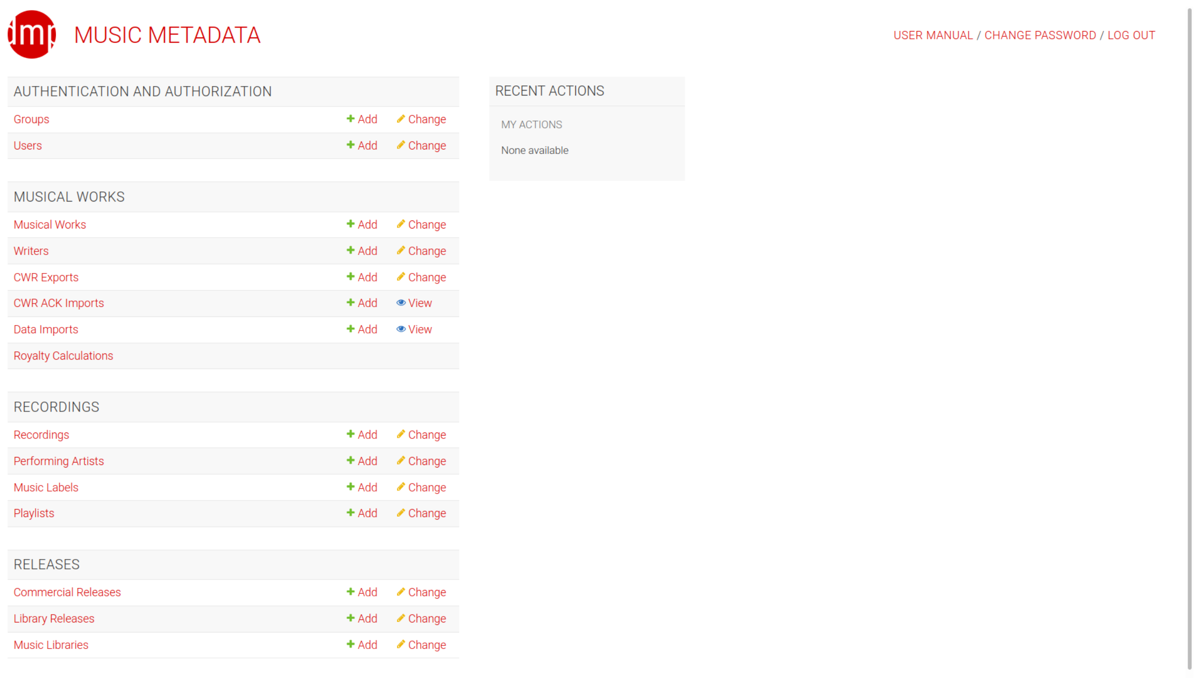


Fig. 1: Default log-in view

The first screen that appears is the log-in screen. Please log in with your credentials.

9.1.2 Home view



MUSIC METADATA

USER MANUAL / CHANGE PASSWORD / LOG OUT

AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change

MUSICAL WORKS		
Musical Works	+ Add	Change
Writers	+ Add	Change
CWR Exports	+ Add	Change
CWR ACK Imports	+ Add	View
Data Imports	+ Add	View
Royalty Calculations		

RECORDINGS		
Recordings	+ Add	Change
Performing Artists	+ Add	Change
Music Labels	+ Add	Change
Playlists	+ Add	Change

RELEASES		
Commercial Releases	+ Add	Change
Library Releases	+ Add	Change
Music Libraries	+ Add	Change

RECENT ACTIONS

MY ACTIONS

None available

Fig. 2: Home view for superusers

The `home view` will show up after a successfull login. It changes, based on user permissions. In this example, it is the view superusers see – everything.

In the header, the left part shows the name of the publisher and the link to the maintainer's website. The right shows links to this user manual, for changing the password and logging out. This header is present in all views.

We have two columns, the left one shows sections of models, with links to change and add views. The right column shows up to 10 latest actions of the current user.

9.1.3 Model Views

Every model has at least 4 views:

- `List` - view listing objects, includes search, filtering and batch actions
- `Add` - view for adding new objects
- `Change` - view for changing an object, includes delete button
- `History` - view where changes to an object are shown, accessible from `change` view

`Add` and `change` are usually very similar. They often contain forms for editing related models. E.g. in `add musical work`, one can also add alternate titles, recordings, etc.

The views are explained in detail in *Musical Works*.

9.2 User Administration

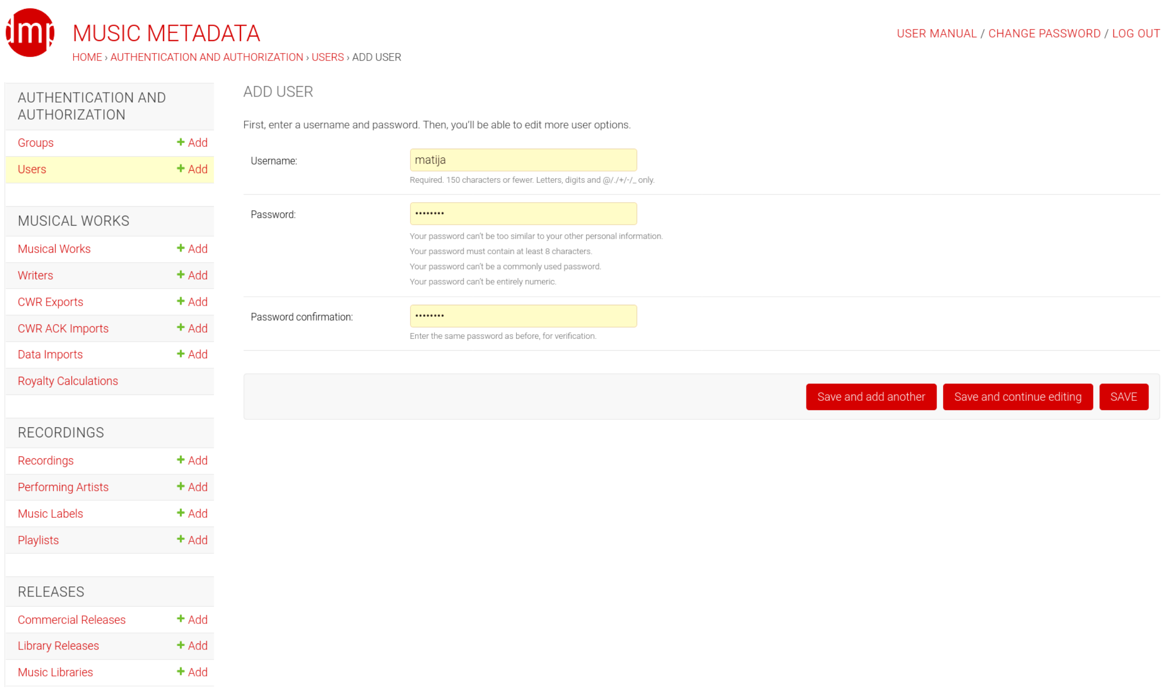
This section is covering user administration.

Note: If you don't have the permission to manage other users, you don't see the `Authentication and Authorization` Section.

Warning: If you have deployed DMP to Heroku, the password you used for the superuser account was written in plain text to the config variables. It is strongly recommended that you change the password upon the first login.

Warning: Superusers should not do everyday tasks. Create staff users.

You add users by clicking on `+ add` link for the `users` in the `Authentication and Authorization`. The following view is shown:



ADD USER

First, enter a username and password. Then, you'll be able to edit more user options.

Username: matija

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Password: ••••••••

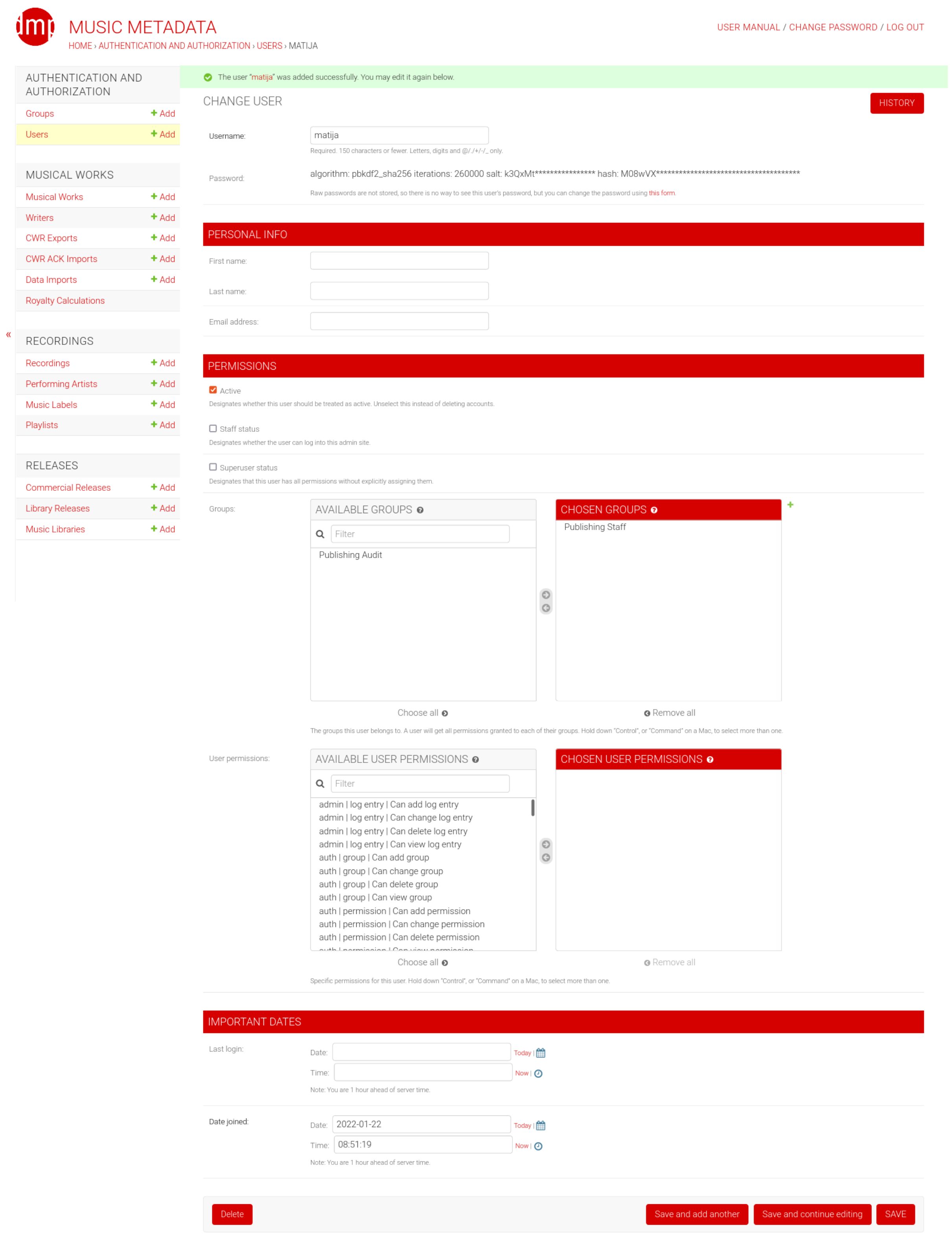
Your password can't be too similar to your other personal information.
Your password must contain at least 8 characters.
Your password can't be a commonly used password.
Your password can't be entirely numeric.

Password confirmation: ••••••••

Enter the same password as before, for verification.

Fig. 3: Add User view

Add a username and a password twice and click on `Save and continue editing`. Then, in the next view, add additional data.



The user "matija" was added successfully. You may edit it again below.

CHANGE USER

Username: matija

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Password: algorithm: pbkdf2_sha256 iterations: 260000 salt: k3QxMt**************** hash: M08wVX***************************************

Raw passwords are not stored, so there is no way to see this user's password, but you can change the password using this form.

PERSONAL INFO

First name:

Last name:

Email address:

PERMISSIONS

☑ Active

Designates whether this user should be treated as active. Unselect this instead of deleting accounts.

☐ Staff status

Designates whether the user can log into this admin site.

☐ Superuser status

Designates that this user has all permissions without explicitly assigning them.

Groups:

AVAILABLE GROUPS

Filter

Publishing Audit

CHOSEN GROUPS

Publishing Staff

The groups this user belongs to. A user will get all permissions granted to each of their groups. Hold down "Control", or "Command" on a Mac, to select more than one.

User permissions:

AVAILABLE USER PERMISSIONS

Filter

admin | log entry | Can add log entry
admin | log entry | Can change log entry
admin | log entry | Can delete log entry
admin | log entry | Can view log entry
auth | group | Can add group
auth | group | Can change group
auth | group | Can delete group
auth | group | Can view group
auth | permission | Can add permission
auth | permission | Can change permission
auth | permission | Can delete permission
auth | permission | Can view permission

CHOSEN USER PERMISSIONS

Specific permissions for this user. Hold down "Control", or "Command" on a Mac, to select more than one.

IMPORTANT DATES

Last login: Date: Today | Time: Now |

Note: You are 1 hour ahead of server time.

Date joined: Date: 2022-01-22 Today | Time: 08:51:19 Now |

Note: You are 1 hour ahead of server time.

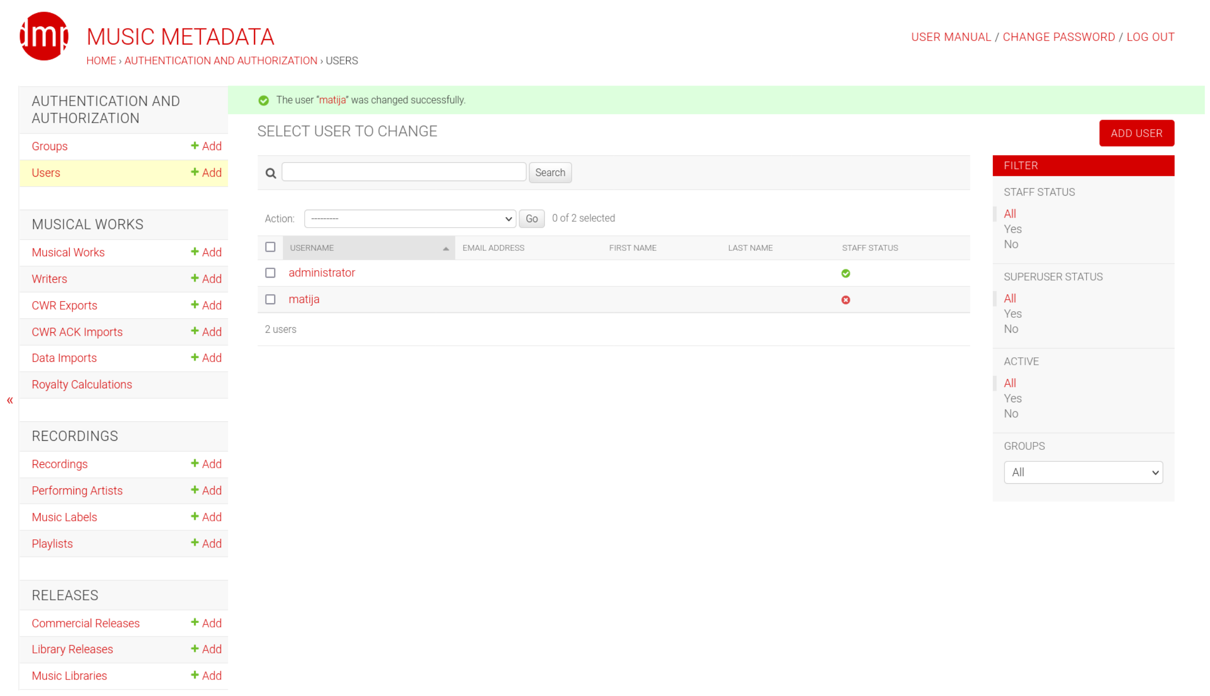
Fig. 4: Change User view

Note: Passwords are not visible, and not saved in plaintext. To change a password for another user, use `this form` link.

`Staff status` has to be set for all users of Django-Music-Publisher, and they have to be assigned to an appropriate permission group. Two permission groups are set during installation:

- `Publishing staff` gives all permissions required for everyday publishing work
- `Publishing audit` gives read-only permissions to all data in Music Publisher module

Select one of them and click on the icon that will move it to `chosen groups`. Then you can click on `save`.



MUSIC METADATA

HOME › AUTHENTICATION AND AUTHORIZATION › USERS

USER MANUAL / CHANGE PASSWORD / LOG OUT

AUTHENTICATION AND AUTHORIZATION
- Groups + Add
- Users + Add

MUSICAL WORKS
- Musical Works + Add
- Writers + Add
- CWR Exports + Add
- CWR ACK Imports + Add
- Data Imports + Add
- Royalty Calculations

RECORDINGS
- Recordings + Add
- Performing Artists + Add
- Music Labels + Add
- Playlists + Add

RELEASES
- Commercial Releases + Add
- Library Releases + Add
- Music Libraries + Add

The user "matija" was changed successfully.

SELECT USER TO CHANGE

ADD USER

Search

Action: --------- Go 0 of 2 selected

	USERNAME	EMAIL ADDRESS	FIRST NAME	LAST NAME	STAFF STATUS
☐	administrator				✓
☐	matija				✗

2 users

FILTER

STAFF STATUS: All, Yes, No

SUPERUSER STATUS: All, Yes, No

ACTIVE: All, Yes, No

GROUPS: All

Fig. 5: User list view

You will be taken to the `user list` view. All users are shown here. Just as the add and change views, list views are quite standard. They will be covered a bit later.

Now you can log out, and log in as the newly added staff user. The `home view` is a bit different, according to the assigned permissions.

9.3 Section: Musical Works

Models are divided into sections for more intuitive navigation.

This section contains all models and actions closely related to managing musical works, including `Musical Works` model, the workhorse in this software.

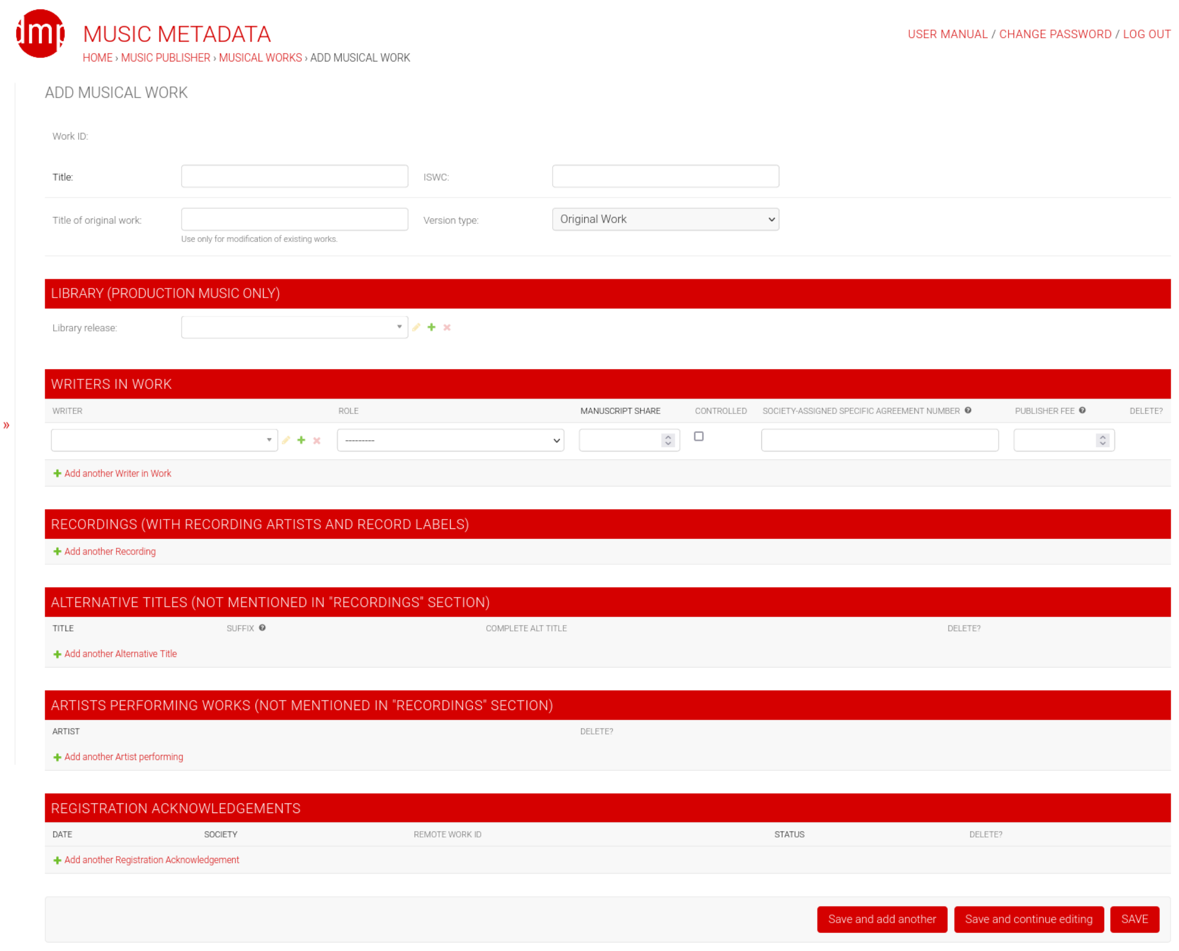
Note: CWR exports and CWR Acknowledgement imports will not work unless `PUBLISHER_CODE` is defined in the settings, regardless of user permissions.

Note: Data imports require additional permissions, not given to staff users by default. Use the superuser account for importing data.

9.3.1 Musical Works

This part explains views for Musical Work model specifically, but much of it applies to views of other models as well.

Add/Change View



MUSIC METADATA

HOME › MUSIC PUBLISHER › MUSICAL WORKS › ADD MUSICAL WORK

USER MANUAL / CHANGE PASSWORD / LOG OUT

ADD MUSICAL WORK

Work ID:

Title: ISWC:

Title of original work: Version type: Original Work

Use only for modification of existing works.

LIBRARY (PRODUCTION MUSIC ONLY)

Library release:

WRITERS IN WORK

WRITER	ROLE	MANUSCRIPT SHARE	CONTROLLED	SOCIETY-ASSIGNED SPECIFIC AGREEMENT NUMBER	PUBLISHER FEE	DELETE?
	---------		☐			

Add another Writer in Work

RECORDINGS (WITH RECORDING ARTISTS AND RECORD LABELS)

Add another Recording

ALTERNATIVE TITLES (NOT MENTIONED IN "RECORDINGS" SECTION)

TITLE	SUFFIX	COMPLETE ALT TITLE	DELETE?

Add another Alternative Title

ARTISTS PERFORMING WORKS (NOT MENTIONED IN "RECORDINGS" SECTION)

ARTIST	DELETE?

Add another Artist performing

REGISTRATION ACKNOWLEDGEMENTS

DATE	SOCIETY	REMOTE WORK ID	STATUS	DELETE?

Add another Registration Acknowledgement

Fig. 6: Add work view

The view for adding and changing works is shown in this screenshot. It is the most complex view in Django-Music-Publisher (DMP). It has several parts, which will be covered one by one.

General

This fieldset contains basic fields.

Field `work ID` is not editable in this view.

Note: `Work ID` is set by DMP, but it can also be imported. See *Importing Data* for details.

Work title is entered into `title` field.

`ISWC` (International Standard Musical Work Code) is a unique identifier assigned to works by a central authority through collecting societies. It can be edited manually or imported either through *data imports* or *CWR acknowledgements*.

Fields `title of original work` and `version type`, with only the former being editable, are used for modifications. By filling out `title of original work` field, the `version type` will be set to `modification` and a more complex set of validation rules will apply.

Library

DMP has support for music libraries. If a work is part of a music library, then a `Library release` must be set here. Details can be found in *library release*.

Writers in Work

This is where you put in the information about writers (composers and lyricists) of the work. At least one record is required, to add more, click on `add another writer in work`.

Each column in this table is described next.

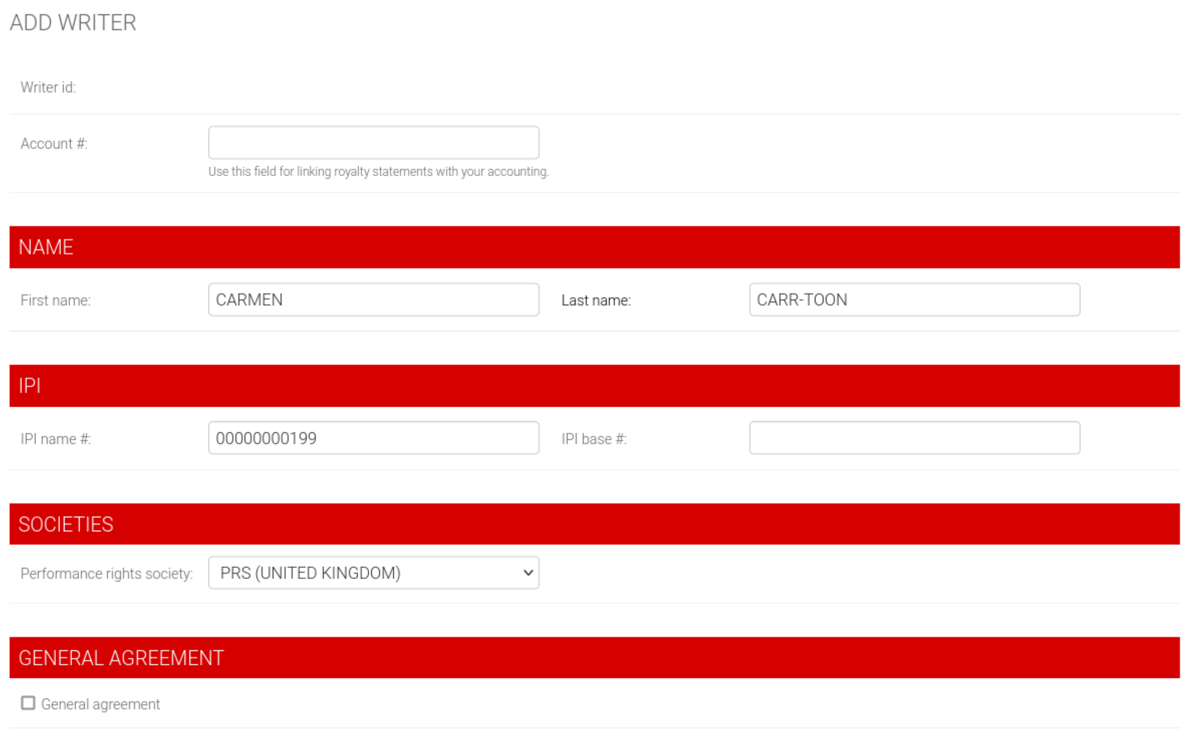
Writer

This is where you select a writer.

This field is conditionally required for controlled writers, and at least one writer in work must be controlled.

Like many other fields, this field is searchable. You can search by writer's `last name` or `ipi name number`. Click on the desired writer to select them. To unselect a writer, click the black **x** icon **in the box**.

To add a new writer, click the green plus sign next to it. To edit the selected writer, click the yellow pencil icon. To delete the selected writer, click the red **X** icon **outside the box**. For all three cases, a pop-up window will appear.


ADD WRITER

Writer id:			
Account #:	Use this field for linking royalty statements with your accounting.		

NAME

First name:	CARMEN	Last name:	CARR-TOON

IPI

IPI name #:	00000000199	IPI base #:	

SOCIETIES

Performance rights society:	PRS (UNITED KINGDOM)

GENERAL AGREEMENT

☐ General agreement

Fig. 7: Add writer pop-up view

The details about the fields in the pop-up window are covered in *writer*.

Note: If `writer` field is left empty, it means that the writer is unknown. This is often used with modifications of traditional musical works.

Role

This is where you select how this writer contributed to the work. This field is required for controlled writers.

At least one of the writers should be a `composer` or a `composer and lyricist`.

Options for original works are `composer`, `lyricist` and `composer and lyricist`.

Roles `arranger`, `adaptor` or `translator` can only be used in modifications.

For modifications, at least two rows are required, one being (original) `composer` or a `composer and lyricist`, and one being `arranger`, `adaptor` or `translator`.

For modifications of traditional works, set the capacity of the unknown writer to `composer and lyricist` or `composer`, depending on whether the original work has lyrics or not.

Manuscript Share

Django-Music-Publisher (DMP) uses a very simple single-field share model.

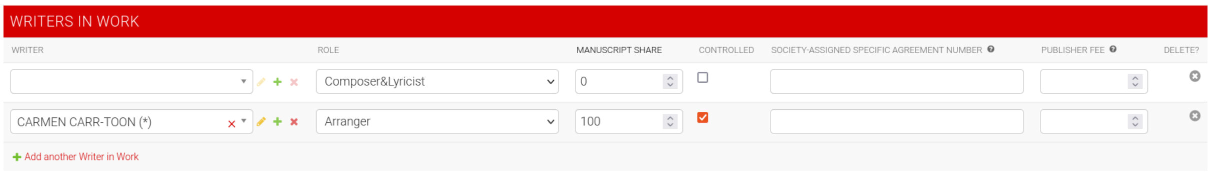
Writers create a work and decide how they want to split the shares among themselves. This is referred to as `manuscript share`.

Each of the writers may choose a publisher and transfer part of their manuscript shares to the publisher, according to their publishing agreement. This does not influence other writers.

In DMP, publishing agreements between all controlled writers and you as the original publisher have same splits, globally defined in settings.

Note: The sum of relative shares in a work must be 100%.

Note: For a musical work that is a modification of a work in public domain, set the share of original writers (`composer`, `lyricist`, `composer and lyricist`) to 0.



WRITERS IN WORK

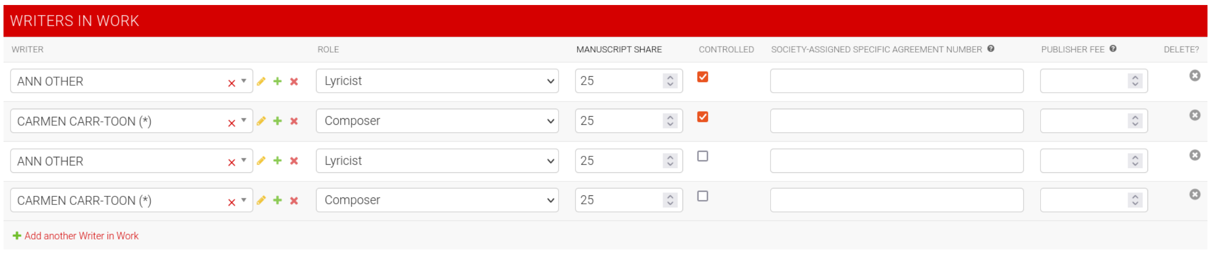
WRITER	ROLE	MANUSCRIPT SHARE	CONTROLLED	SOCIETY-ASSIGNED SPECIFIC AGREEMENT NUMBER	PUBLISHER FEE	DELETE?
	Composer&Lyricist	0	☐			
CARMEN CARR-TOON (*)	Arranger	100	✓			

Fig. 8: Writers in work for a work that is a modification of a work in public domain

Controlled

This is where you select whether you control the writer or not. Select it for at least one `writer in work` row.

A writer can be entered in two rows, once as controlled, once as not. This allows for co-publishing deals. If there is more than one other publisher per writer, add their shares to a single row.



WRITERS IN WORK

WRITER	ROLE	MANUSCRIPT SHARE	CONTROLLED	SOCIETY-ASSIGNED SPECIFIC AGREEMENT NUMBER	PUBLISHER FEE	DELETE?
ANN OTHER	Lyricist	25	✓			
CARMEN CARR-TOON (*)	Composer	25	✓			
ANN OTHER	Lyricist	25	☐			
CARMEN CARR-TOON (*)	Composer	25	☐			

Add another Writer in Work

Fig. 9: Writers in work for a co-published work

Society-assigned agreement number

In this field, society-assigned agreement numbers for **specific agreements** are entered. For **general agreements**, they are set when defining the *writer*. If both exist and are different, the **specific** one is used.

Note: This field is required for controlled writers in some collecting societies, while not used in most.

Publisher fee

This is the fee kept by the publisher when royalties are paid and distributed.

Note: This field is not used in registrations. It is used only for *royalty statement processing*. Details are explained in that section.

Recordings (With Recording Artists and Record Labels)

This is where the details about a recording based on this musical works are added. There is a separate set of views for *recordings*, fields are explained there.

Alternative Titles

Alternative titles section is for alternative titles. There is no need to enter the recording or version titles already entered in the recordings section.

Field `alternative title` is where you enter the title, or it's suffix, based on the field `suffix`. If the latter is checked, then the suffix will be appended to the work `title`. The actual alternative title is always shown in the read-only field `complete alt title`.

Artists Performing Works

Here you list the artists who are performing the work, there is no need to repeat the artists already set as `recording artists` in the `recordings` section.

The field `artist` behaves similarly to the field *Writer*.

Registration Acknowledgements

This is where the work registration acknowledgements are recorded.

Note: In the default configuration, only superusers can modify this section, as it is automatically filled out from *uploaded acknowledgement files*.

Saving and Deleting

At the bottom, there is a delete button and three save buttons.

`Delete` button starts the deletion of the work and all related objects. A confirmation screen shows all objects being deleted.

Note: Deleting a work is not always allowed, regardless of user permissions. E.g. if a *CWR acknowledgement* for this work exists. If you are sure you want to delete a work, a superuser must delete such linked objects first.

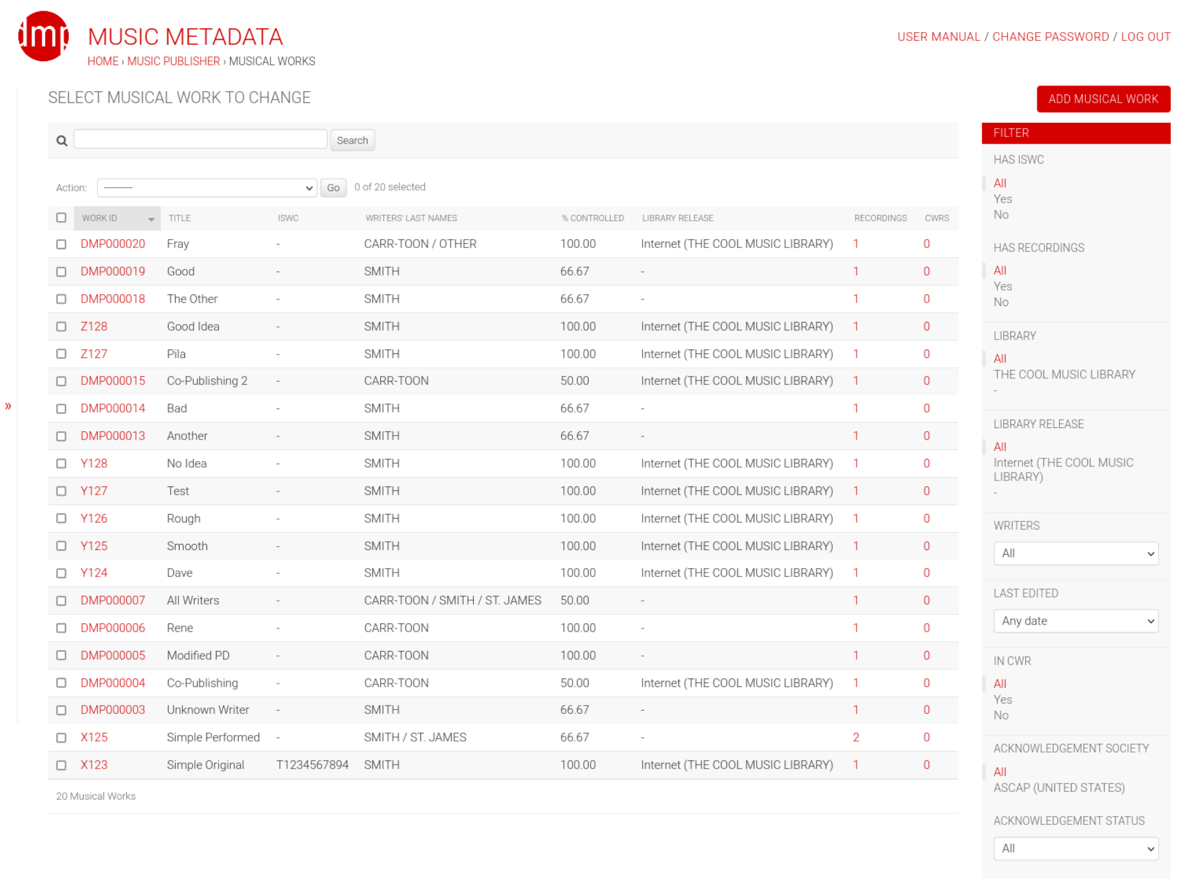
The save buttons do following:

- `Save and add another` (when adding new work) saves the work and then opens a new, empty form for the next one.
- `Save as new` (when editing existing work) saves this data as a new work (with a different work ID). Note that you must change all unique fields as well, e.g. ISWC.
- `Save and continue editing` saves the work and then opens the same work for further editing.
- `SAVE` saves the work and returns to the `list view`, covered next.

The combination is extremely powerful, especially when the changes between works is small.

Enter the first work, using suffixes as much as possible, click on `save and continue editing`. If successful, then data make the changes for the next work, and click on `save as new`, and this new work is saved.

List View



MUSIC METADATA

HOME › MUSIC PUBLISHER › MUSICAL WORKS

USER MANUAL / CHANGE PASSWORD / LOG OUT

SELECT MUSICAL WORK TO CHANGE

ADD MUSICAL WORK

Search

Action: --------- Go 0 of 20 selected

WORK ID	TITLE	ISWC	WRITERS' LAST NAMES	% CONTROLLED	LIBRARY RELEASE	RECORDINGS	CWRS
DMP000020	Fray	-	CARR-TOON / OTHER	100.00	Internet (THE COOL MUSIC LIBRARY)	1	0
DMP000019	Good	-	SMITH	66.67	-	1	0
DMP000018	The Other	-	SMITH	66.67	-	1	0
Z128	Good Idea	-	SMITH	100.00	Internet (THE COOL MUSIC LIBRARY)	1	0
Z127	Pila	-	SMITH	100.00	Internet (THE COOL MUSIC LIBRARY)	1	0
DMP000015	Co-Publishing 2	-	CARR-TOON	50.00	Internet (THE COOL MUSIC LIBRARY)	1	0
DMP000014	Bad	-	SMITH	66.67	-	1	0
DMP000013	Another	-	SMITH	66.67	-	1	0
Y128	No Idea	-	SMITH	100.00	Internet (THE COOL MUSIC LIBRARY)	1	0
Y127	Test	-	SMITH	100.00	Internet (THE COOL MUSIC LIBRARY)	1	0
Y126	Rough	-	SMITH	100.00	Internet (THE COOL MUSIC LIBRARY)	1	0
Y125	Smooth	-	SMITH	100.00	Internet (THE COOL MUSIC LIBRARY)	1	0
Y124	Dave	-	SMITH	100.00	Internet (THE COOL MUSIC LIBRARY)	1	0
DMP000007	All Writers	-	CARR-TOON / SMITH / ST. JAMES	50.00	-	1	0
DMP000006	Rene	-	CARR-TOON	100.00	-	1	0
DMP000005	Modified PD	-	CARR-TOON	100.00	-	1	0
DMP000004	Co-Publishing	-	CARR-TOON	50.00	Internet (THE COOL MUSIC LIBRARY)	1	0
DMP000003	Unknown Writer	-	SMITH	66.67	-	1	0
X125	Simple Performed	-	SMITH / ST. JAMES	66.67	-	2	0
X123	Simple Original	T1234567894	SMITH	100.00	Internet (THE COOL MUSIC LIBRARY)	1	0

20 Musical Works

FILTER

HAS ISWC
All
Yes
No

HAS RECORDINGS
All
Yes
No

LIBRARY
All
THE COOL MUSIC LIBRARY
-

LIBRARY RELEASE
All
Internet (THE COOL MUSIC LIBRARY)
-

WRITERS
All

LAST EDITED
Any date

IN CWR
All
Yes
No

ACKNOWLEDGEMENT SOCIETY
All
ASCAP (UNITED STATES)

ACKNOWLEDGEMENT STATUS
All

Fig. 10: Work list view

The `work list` view, just as all other list views, has a `search field`, an `action bar`, a table with works and, once there are over 100 works, pagination, all on the left side.

Search looks for titles, writer's last names, ISWCs, ISRCs (in related recordings) and work IDs.

Data table can be sorted by almost any column or combination of the columns.

Counts of related objects are also links to *recording* and *CWR export* `list` views, filtered for this work.

On the right side, there is the `add musical work` button, which takes you to the appropriate view, and the set of `filters`.

Filters change, based on the number of options. For four options or less, they are simple links, and for more, they turn into a pull-down menus.

`Has ISWC` will show only works with ISWCs or only works without them.

`Has recordings` will show only works with recordings or only works without them.

`Library` will list only works in a particular *library*.

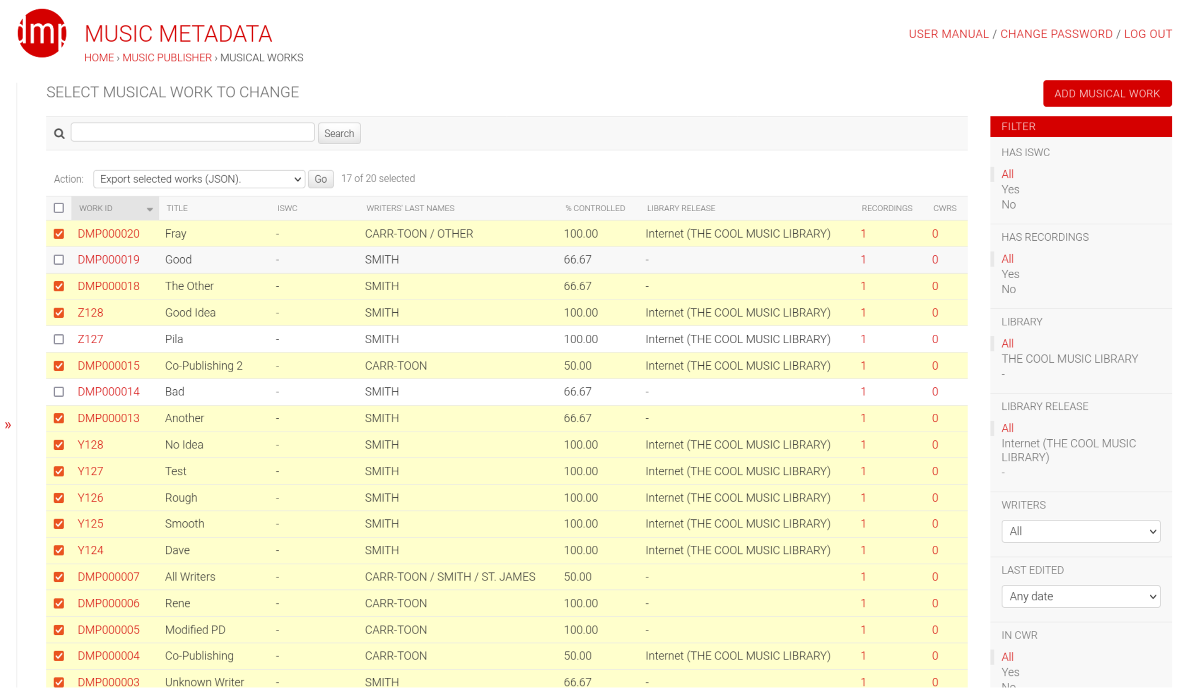
`Library Release` will list only works in a particular *library release*.

`Writers` will list only works by a particular *writer*.

`Last edited` filter allows you to find all works that have changed recently.

Filters and search can be combined. Only works fulfilling all the criteria will be shown.

Exporting JSON


MUSIC METADATA

HOME › MUSIC PUBLISHER › MUSICAL WORKS

USER MANUAL / CHANGE PASSWORD / LOG OUT

SELECT MUSICAL WORK TO CHANGE

ADD MUSICAL WORK

Action: Export selected works (JSON). Go 17 of 20 selected

	WORK ID	TITLE	ISWC	WRITERS' LAST NAMES	% CONTROLLED	LIBRARY RELEASE	RECORDINGS	CWRS
✓	DMP000020	Fray	-	CARR-TOON / OTHER	100.00	Internet (THE COOL MUSIC LIBRARY)	1	0
☐	DMP000019	Good	-	SMITH	66.67	-	1	0
✓	DMP000018	The Other	-	SMITH	66.67	-	1	0
✓	Z128	Good Idea	-	SMITH	100.00	Internet (THE COOL MUSIC LIBRARY)	1	0
☐	Z127	Pila	-	SMITH	100.00	Internet (THE COOL MUSIC LIBRARY)	1	0
✓	DMP000015	Co-Publishing 2	-	CARR-TOON	50.00	Internet (THE COOL MUSIC LIBRARY)	1	0
☐	DMP000014	Bad	-	SMITH	66.67	-	1	0
✓	DMP000013	Another	-	SMITH	66.67	-	1	0
✓	Y128	No Idea	-	SMITH	100.00	Internet (THE COOL MUSIC LIBRARY)	1	0
✓	Y127	Test	-	SMITH	100.00	Internet (THE COOL MUSIC LIBRARY)	1	0
✓	Y126	Rough	-	SMITH	100.00	Internet (THE COOL MUSIC LIBRARY)	1	0
✓	Y125	Smooth	-	SMITH	100.00	Internet (THE COOL MUSIC LIBRARY)	1	0
✓	Y124	Dave	-	SMITH	100.00	Internet (THE COOL MUSIC LIBRARY)	1	0
✓	DMP000007	All Writers	-	CARR-TOON / SMITH / ST. JAMES	50.00	-	1	0
✓	DMP000006	Rene	-	CARR-TOON	100.00	-	1	0
✓	DMP000005	Modified PD	-	CARR-TOON	100.00	-	1	0
✓	DMP000004	Co-Publishing	-	CARR-TOON	50.00	Internet (THE COOL MUSIC LIBRARY)	1	0
✓	DMP000003	Unknown Writer	-	SMITH	66.67	-	1	0

Fig. 11: Exporting musical works in JSON format.

Select several (or all) works in the `musical work list` view, select the `Export selected works (JSON)` action and click `Go`. A JSON file will be downloaded, containing **all** the information about your works.

Exporting CSV

Select several (or all) works in the `musical work list` view, select the `Export selected works (CSV)` action and click `Go`. A CSV file will be downloaded, containing **most** information about your works.

This CSV format is similar to the one used for *Importing data*.

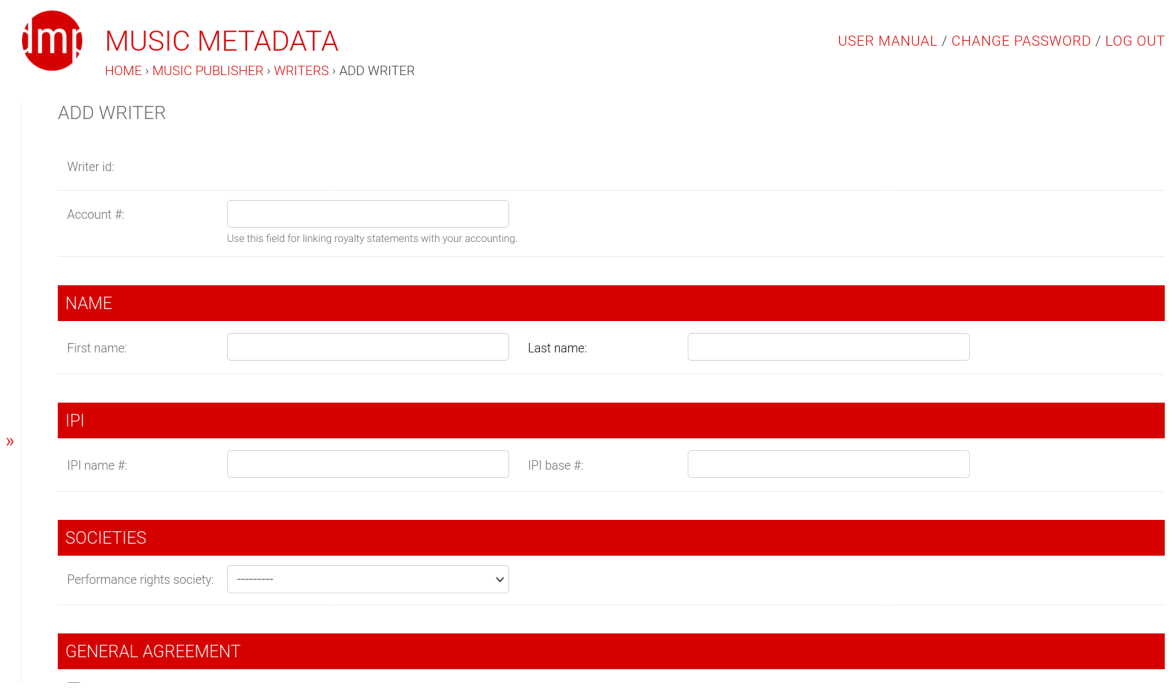
CWR Exporting Wizard

Currently, the only other available action is to `create CWR from selected works`. Once you run it, you will be taken to *CWR Export* view with your work selection.

Note: `Create CWR from selected works` action is only visible if `PUBLISHER_CODE` is defined in settings.

9.3.2 Writers

Add/Change View


MUSIC METADATA
USER MANUAL / CHANGE PASSWORD / LOG OUT
HOME › MUSIC PUBLISHER › WRITERS › ADD WRITER
»

ADD WRITER

Writer id:

Account #:

Use this field for linking royalty statements with your accounting.

NAME

First name: Last name:

IPI

IPI name #: IPI base #:

SOCIETIES

Performance rights society: ---------

GENERAL AGREEMENT

Fig. 12: Add writer view

`Add` and `change` views for writers have several fieldsets.

Writer ID and Account Number

At the top, before the first fieldset are two fields, `Writer ID`, assigned by the system and not editable, and `Account #`, used for linking data from DMP with your accounting, when processing royalty statements.

Name

`Last name` and `first name` fields in the first, quite self-explanatory. Only last name is required.

IPI

`IPI name #` and `IPI Base #` in the second. If you are unfamiliar with these identifiers, see IPI name and base numbers.

Societies

`Performance Rights Society` in the third. In most cases, writers are only affiliated with performance rights societies. Depending on settings, fields for mechanical and even sync affiliation might be visible.

General Agreement

In the last group, we have three fields:

- `General agreement` to mark that there is an original general agreement with this writer. This means that this writer must be controlled in all works.
- `Society-assigned agreement number` for the original general agreement between you and this writer (required in some societies)
- `Publisher fee` is the fee kept by the publisher when royalties are paid and distributed.

Note: `Publisher fee` is not used in registrations. It is used only for *royalty statement processing*. Details are explained in that section.

Public

Note: This section is only visible if file uploads are configured.

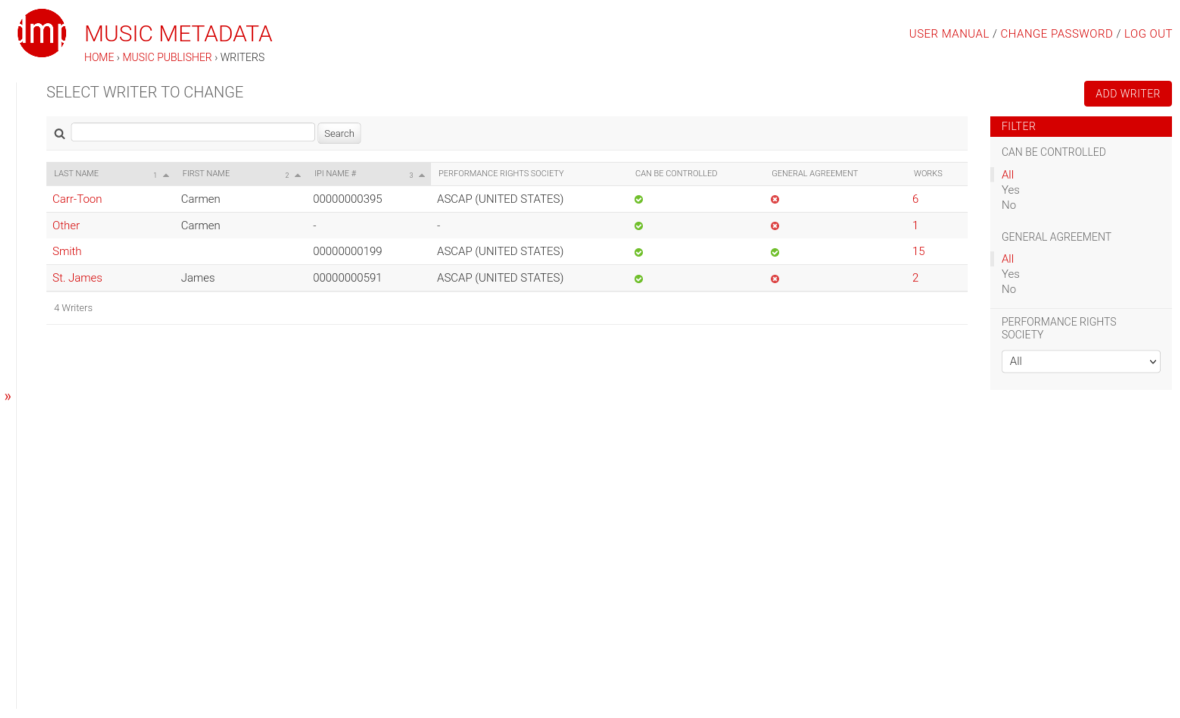
This section has two fields:

- `Image` - for uploading an image of the writer
- `Description` - for public description

Internal

This section has only a single field `Notes`. You can use it in any way you like.

List View



LAST NAME 1 ▲	FIRST NAME 2 ▲	IPI NAME # 3 ▲	PERFORMANCE RIGHTS SOCIETY	CAN BE CONTROLLED	GENERAL AGREEMENT	WORKS
Carr-Toon	Carmen	00000000395	ASCAP (UNITED STATES)	✓	✗	6
Other	Carmen	-	-	✓	✗	1
Smith		00000000199	ASCAP (UNITED STATES)	✓	✓	15
St. James	James	00000000591	ASCAP (UNITED STATES)	✓	✗	2

4 Writers

Fig. 13: List writers view

The last column is both a work counter and link to the list of *works* by this writer.

`Can be controlled` column requires an explanation.

For writers who are controlled (whose works are published by you), more data is required than for those who are not. This column shows if data is sufficient for the writer to be marked as controlled.

Controlled writers without affiliation and/or IPI name number

In very rare cases, writers choose not to affiliate with any society and even get an IPI name number. And consequently not getting paid.

If you control such a writer, you can still enter them. If they don't have an IPI name number, you can enter `00000000000`. If they are not affiliated with any performance rights society, there is a `NO SOCIETY` option at the bottom of the list.

This has to be manually re-entered on *every* save of the writer form. It is a feature, not a bug. In almost all cases, both IPI name number and PR affiliations should be entered for controlled writers. Entering edge case exceptions should not be simple.

Other writers

For writers you do not control, you should still provide as much data as possible.

Note: Only if ALL writers are identified with their IPI numbers, the work can receive an International Standard Musical Work Code (ISWC).

9.3.3 Common Works Registration Exports

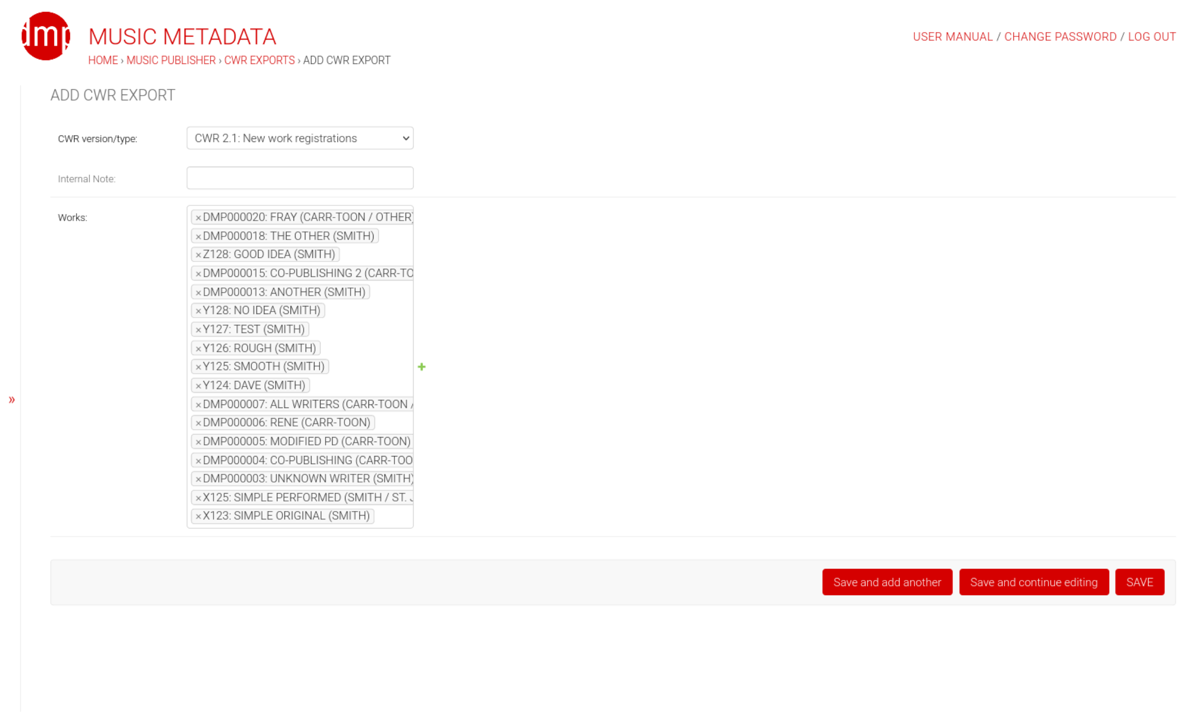
Common Works Registration (CWR) is a protocol and a file format for batch registrations of musical works with collecting societies worldwide. Publishers send registrations and societies reply with acknowledgement files. Registrations in this formats are usually called CWRs and acknowledgement ACKs.

Unofficially, CWRs are also used for data exchange among publishers.

CWR is an extremely complex topic. Only technical aspects of creating CWR files and *importing acknowledgements* are covered in this manual.

Note: Collecting societies and other receivers of CWR files may, if issues arise, refer you to the software **vendor** for support. According to the *MIT license*, that is you, not the **creator** of this software.

Add View


MUSIC METADATA

HOME › MUSIC PUBLISHER › CWR EXPORTS › ADD CWR EXPORT

USER MANUAL / CHANGE PASSWORD / LOG OUT

ADD CWR EXPORT

CWR version/type: CWR 2.1: New work registrations

Internal Note:

Works:
- ×DMP000020: FRAY (CARR-TOON / OTHER)
- ×DMP000018: THE OTHER (SMITH)
- ×Z128: GOOD IDEA (SMITH)
- ×DMP000015: CO-PUBLISHING 2 (CARR-TC
- ×DMP000013: ANOTHER (SMITH)
- ×Y128: NO IDEA (SMITH)
- ×Y127: TEST (SMITH)
- ×Y126: ROUGH (SMITH)
- ×Y125: SMOOTH (SMITH)
- ×Y124: DAVE (SMITH)
- ×DMP000007: ALL WRITERS (CARR-TOON /
- ×DMP000006: RENE (CARR-TOON)
- ×DMP000005: MODIFIED PD (CARR-TOON)
- ×DMP000004: CO-PUBLISHING (CARR-TOO
- ×DMP000003: UNKNOWN WRITER (SMITH)
- ×X125: SIMPLE PERFORMED (SMITH / ST. .
- ×X123: SIMPLE ORIGINAL (SMITH)

Fig. 14: Add CWR export view

Note: If *CWR delivery code* is not entered as `PUBLISHER_CODE` in settings, `000` will be used. Such CWR files will not be accepted by most CMOs, but may be accepted by (sub-)publishers.

Warning: Do NOT use an arbitraty CWR delivery code for creating CWR exports.

There are several ways to get to `Add CWR Export` view:

- by clicking `Add CWR Export` button or
- by using `Create CWR from selected works` batch action in *Musical Works*.

There are only three fields:

- `CWR version/type` is where you select the version of CWR and transaction type. Here are current options:
  - CWR 2.1: New work registrations
  - CWR 2.1: Revisions of registered works
  - CWR 2.2: New work registrations
  - CWR 2.2: Revisions of registered works
  - CWR 3.0: Work registration
  - CWR 3.0: ISWC request (EDI)
  - CWR 3.1 DRAFT: Work registration

Note: Consult with the receiver which version they can process. If they can process multiple versions, choose the highest.

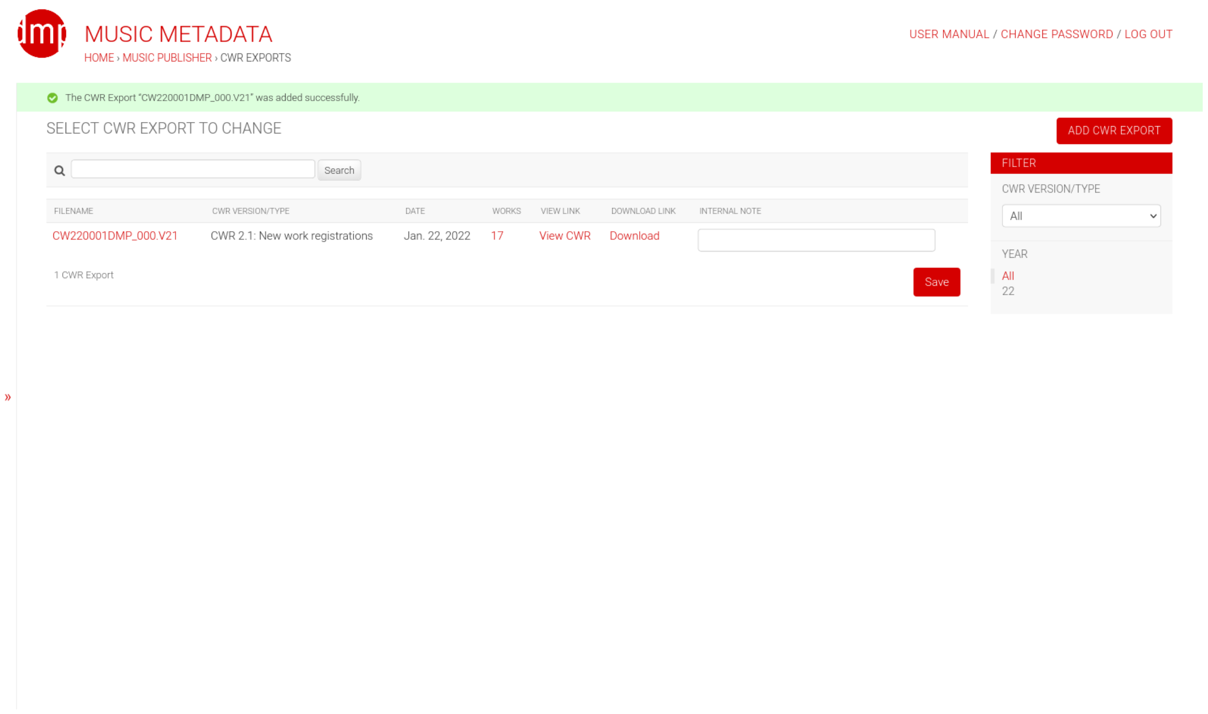
- `Internal note` is a field where you can put a meaningful description of the export.

Warning: File naming is part of the CWR specifications. CWR file names should NOT be changed.

- `Works` is a multi-select field for works to be included in CWR exports.

CWR Export model does not have `change view`, nor `delete` button. CWR files once created should NOT be deleted, although they may not be used. Use *internal note* to mark a CWR file as not sent.

List View


MUSIC METADATA
USER MANUAL / CHANGE PASSWORD / LOG OUT
HOME › MUSIC PUBLISHER › CWR EXPORTS
ADD CWR EXPORT
Search
FILTER
Save

The CWR Export "CW220001DMP_000.V21" was added successfully.

SELECT CWR EXPORT TO CHANGE

FILENAME	CWR VERSION/TYPE	DATE	WORKS	VIEW LINK	DOWNLOAD LINK	INTERNAL NOTE
CW220001DMP_000.V21	CWR 2.1: New work registrations	Jan. 22, 2022	17	View CWR	Download	

1 CWR Export

CWR VERSION/TYPE: All

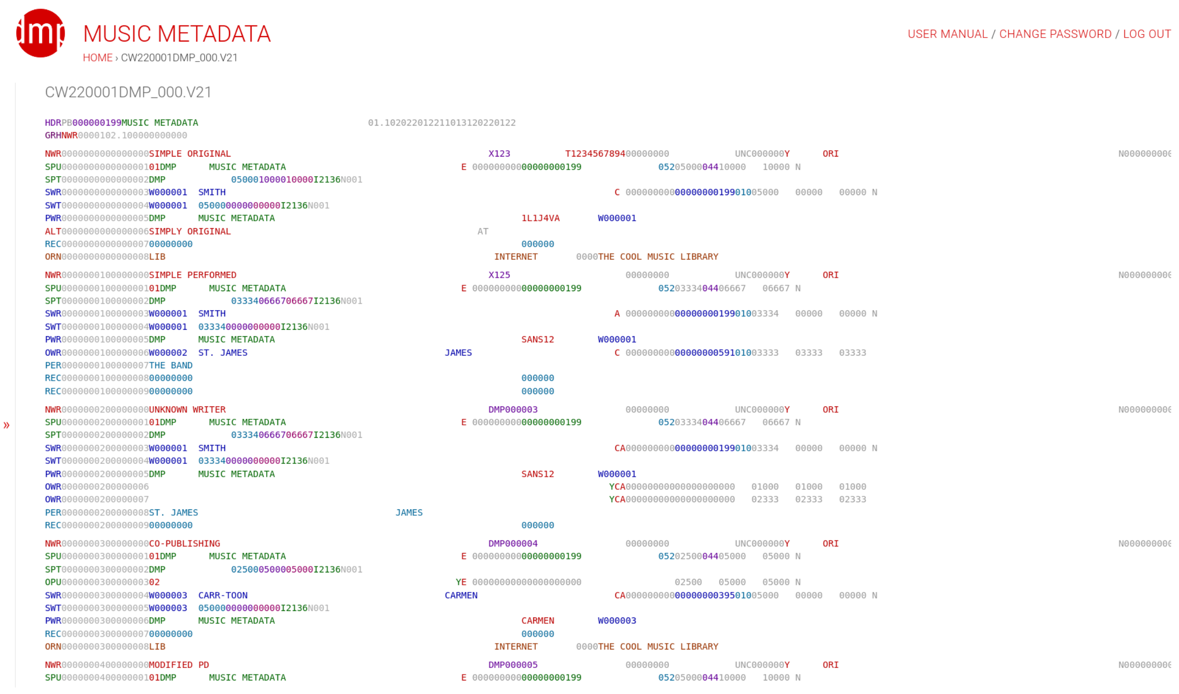
YEAR: All, 22

Fig. 15: List CWR export view

`CWR export list` view. Besides the link in the first column with the file name, which opens a view with additional information, and the counter that opens the list of works in this file, it has two additional links in each row: `View CWR` and `Download`.

The latter downloads the zipped CWR file, and the former opens the CWR file for viewing.

View CWR


MUSIC METADATA
HOME › CW220001DMP_000.V21
USER MANUAL / CHANGE PASSWORD / LOG OUT
»

CW220001DMP_000.V21

```
HDRPB000000199MUSIC METADATA                               01.102022012211013120220122
GRHNWR0000102.100000000000
NWR0000000000000000SIMPLE ORIGINAL                                               X123          T123456789400000000            UNC000000Y      ORI                                                   N00000000
SPU000000000000000101DMP      MUSIC METADATA                                E 00000000000000000199              0520500004410000   10000 N
SPT0000000000000002DMP            050001000010000I2136N001
SWR0000000000000003W000001  SMITH                                                                       C 0000000000000000019901005000   00000   00000 N
SWT0000000000000004W000001  050000000000000I2136N001
PWR0000000000000005DMP      MUSIC METADATA                                             1L1J4VA       W000001
ALT0000000000000006SIMPLY ORIGINAL                                             AT
REC000000000000000700000000                                                            000000
ORN0000000000000008LIB                                                            INTERNET       0000THE COOL MUSIC LIBRARY
NWR0000000100000000SIMPLE PERFORMED                                              X125                     00000000            UNC000000Y      ORI                                                   N00000000
SPU000000010000000101DMP      MUSIC METADATA                                E 00000000000000000199              0520333404406667   06667 N
SPT0000000100000002DMP            033340666706667I2136N001
SWR0000000100000003W000001  SMITH                                                                       A 0000000000000000019901003334   00000   00000 N
SWT0000000100000004W000001  033340000000000I2136N001
PWR0000000100000005DMP      MUSIC METADATA                                             SANS12        W000001
OWR0000000100000006W000002  ST. JAMES                                    JAMES                          C 0000000000000000059101003333   03333   03333
PER0000000100000007THE BAND
REC000000010000000800000000                                                            000000
REC000000010000000900000000                                                            000000
NWR0000000200000000UNKNOWN WRITER                                                DMP000003                00000000            UNC000000Y      ORI                                                   N00000000
SPU000000020000000101DMP      MUSIC METADATA                                E 00000000000000000199              0520333404406667   06667 N
SPT0000000200000002DMP            033340666706667I2136N001
SWR0000000200000003W000001  SMITH                                                                       CA0000000000000000019901003334   00000   00000 N
SWT0000000200000004W000001  033340000000000I2136N001
PWR0000000200000005DMP      MUSIC METADATA                                             SANS12        W000001
OWR0000000200000006                                                                                    YCA00000000000000000000   01000   01000   01000
OWR0000000200000007                                                                                    YCA00000000000000000000   02333   02333   02333
PER0000000200000008ST. JAMES                                    JAMES
REC000000020000000900000000                                                            000000
NWR0000000300000000CO-PUBLISHING                                                 DMP000004                00000000            UNC000000Y      ORI                                                   N00000000
SPU000000030000000101DMP      MUSIC METADATA                                E 00000000000000000199              0520250004405000   05000 N
SPT0000000300000002DMP            025000500005000I2136N001
OPU000000030000000302                                                      YE 00000000000000000000                 02500   05000   05000 N
SWR0000000300000004W000003  CARR-TOON                                    CARMEN                         CA0000000000000000039501005000   00000   00000 N
SWT0000000300000005W000003  050000000000000I2136N001
PWR0000000300000006DMP      MUSIC METADATA                                             CARMEN        W000003
REC000000030000000700000000                                                            000000
ORN0000000300000008LIB                                                            INTERNET       0000THE COOL MUSIC LIBRARY
NWR0000000400000000MODIFIED PD                                                   DMP000005                00000000            UNC000000Y      ORI                                                   N00000000
SPU000000040000000101DMP      MUSIC METADATA                                E 00000000000000000199              0520500004410000   10000 N
```

Fig. 16: CWR 2.1 NWR (work registration) file with basic syntax highlighting

The example shown above shows the CWR file with basic syntax highlighting. When you hover over the fields with your cursor, additional information is shown.

9.3.4 Importing CWR Acknowledgements

Societies and administrative agencies (that handle CWR registrations for some societies) send CWR acknowledgement files in response to publishers' registrations. They are also in CWR format. You may receive more than one CWR acknowledgement file for every CWR file you delivered.

Note: **CWR acknowledgement file** means group of transactions of type **ACK** in a CWR file. **Work registration acknowledgement** means one of these transactions.

Django-Music-Publisher can import basic information from CWR acknowledgements sent in response to your CWR registrations:

- Date of the CWR acknowledgement file
- Sender of the CWR acknowledgement file
- Remote Work ID (work ID assigned by the sender of the CWR acknowledgement file)
- Status of the work registration
- ISWCs (optional)

Only CWR 2.1 acknowledgement files are fully supported, with an experimental support for CWR 3.0.

Add view

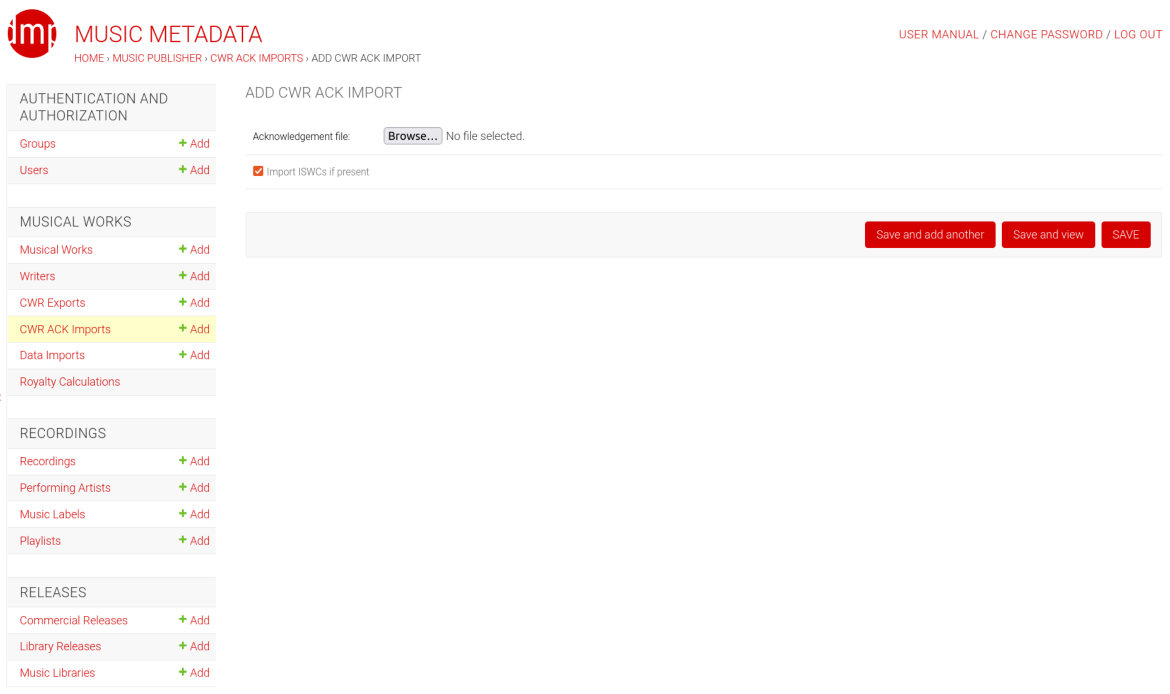


Fig. 17: Add view

This view only has two fields:

- `Acknowledgement file` is where you select the file from your file system
- `Import ISWCs` selects whether to import ISWCs or not.

Once you click on `Save` (any of them), the file is processed.

A brief report is created, with links to all works that received work acknowledgements, work titles and statuses. It can also hold detailed information about encountered issues. All issues are also reported as messages.

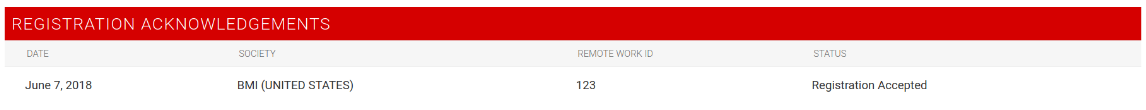
Note: Only works present in at least one of *CWR exports* are matched.

Actual work acknowledgements are shown in the last section of the `change work view`, described *below*.

List view

List view is very simple and self-explanatory. Just as with `CWR exports`, the file name is a link to a page with slightly more information, and the last one opens the CWR file with **syntax highlighting**. See *CWR exports* for more information.

Work registration acknowledgements



REGISTRATION ACKNOWLEDGEMENTS

DATE	SOCIETY	REMOTE WORK ID	STATUS
June 7, 2018	BMI (UNITED STATES)	123	Registration Accepted

Fig. 18: Work registration acknowledgement

They show the aforementioned information, with the exception of imported ISWCs, that go into the ISWC field at the top of the *change work view*. Column `status` is the most important one.

The registration process should end with `Registration accepted`.

`Registration accepted with changes` is usually also OK.

`Transaction accepted` is sent by societies with a two-step process of importing CWR files. This means that the first step for this work was succesfull, and the second step is pending.

Any other status requires investigation. That is far beyond the scope of this user manual. Or any manual. Syntax highlighting of CWR acknowledgement files, mentioned above, may help in the process. Consult the official CWR documentation as well as inquiry with your society.

Note: If you are instructed to contact the software **vendor**, according to the *MIT license*, it is you, not the **creator** of this software.

9.3.5 Importing Data

Note: Default *Publishing Staff* permission group does not include data imports because importing data is not everyday routine.

Musical works metadata can be imported from CSV files.

Warning: There is no way to undo a successful import other than by restoring your database from a backup. If you don't know how to back up and restore your database, do not import data!

What is being imported?

The import process will *add* works, including alternative titles, writers, recordings (partial), performing artists, libraries, library releases and society work references.

No data is ever *modified*, with only one exception. A general agreement for an existing writer may be *set* and a society-assigned agreement number may be *added*.

Why are errors reported?

If data in the file is incomplete or conflicting with data in the database (or other data in the same file), an error will be thrown. Not all errors shown in a user-friendly way.

Note: When an error is thrown, no changes to the database occur.

Work IDs

The template contains `Work ID` column. If you never assigned IDs to works, leave this blank. The system will generate work IDs. Note that this is *not* the ID given by your society or any third party.

On the other hand, work IDs must be maintained when moving from one software to another. Failing to do so may overwrite your existing registrations at collecting societies or create duplicates.

Warning: Not assigning work IDs when required will lead to double registrations and other issues.

Warning: Assigning wrong work IDs will lead to registrations cancelling each other.

How to import?

Obtaining and extending the template

Download the CSV template from the *Add Data Import* view. You can edit it in Excel or another spreadsheet tool.

Alternativelly, you can go to CWR Tools - CSV to CWR, and download the template in Excel format. You still need to save it as CSV before uploading to DMP.

It contains 6 columns for alternative titles, as well as 6 column sets for writers, recordings and artists.

For another writer column set, add all of: `Writer 7 Last`, `Writer 7 First`, `Writer 7 IPI`, `Writer 7 PRO`, `Writer 7 Role`, `Writer 7 Manuscript Share`, `Writer 7 Controlled`, `Writer 7 SAAN`.

You can add as many writer-, recording-, artist- and alternative-title-sets as you require. Just keep incrementing the counter.

Note that this file has a subset of columns described in *Exporting CSV*.

Filling out the template

Fill out the template. Make sure to save as CSV.

Values in `Writer PRO`, `Writer Role` and `Writer Controlled` columns must start with correct codes.

`Writer PRO` must start with society code without the leading zero. `10`, `10 ASCAP`, `10 - ASCAP` or `10 - BMI` will all resolve as ASCAP. `ASCAP` without the code will throw an error.

`Writer Role` must start with one of `C`, `A`, `CA`, `AR`, `TR` or `AD`, e.g. `C - Composer`.

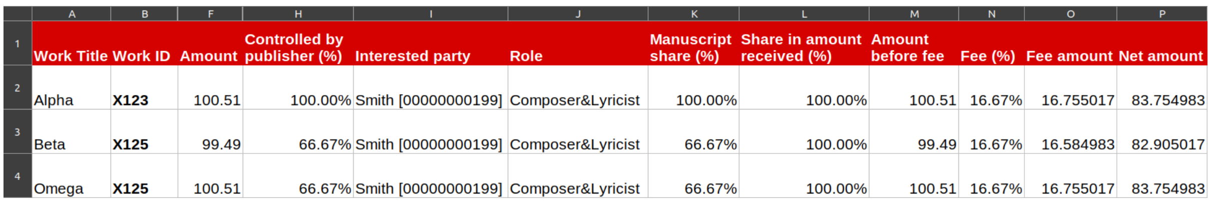
`Writer Controlled` should be set to `No`, `Yes` or `General` (see *Writer* for details).

Data upload

Upload the CSV file through the data import form. If all goes well, the import report will show links to imported works.

9.3.6 Royalty Calculations

If you are interested in the complete Royalty Management process, please read the articles about Royalty Management with DMP, or watch the relevant videos from *Related Videos*. This document describes only a single step in this process.



	A	B	F	H	I	J	K	L	M	N	O	P
1	Work Title	Work ID	Amount	Controlled by publisher (%)	Interested party	Role	Manuscript share (%)	Share in amount received (%)	Amount before fee	Fee (%)	Fee amount	Net amount
2	Alpha	**X123**	100.51	100.00%	Smith [00000000199]	Composer&Lyricist	100.00%	100.00%	100.51	16.67%	16.755017	83.754983
3	Beta	**X125**	99.49	66.67%	Smith [00000000199]	Composer&Lyricist	66.67%	100.00%	99.49	16.67%	16.584983	82.905017
4	Omega	**X125**	100.51	66.67%	Smith [00000000199]	Composer&Lyricist	66.67%	100.00%	100.51	16.67%	16.755017	83.754983

Fig. 19: Outgoing royalty statement

DMP is extremely fast in calculating royalty distributions. Incoming royalty statements in almost any CSV format can be processed. Output will be in a similar CSV format, with several additional columns.

Incoming formats

Incoming statement must be a CSV file with a header row. It can have any number of columns, in any order, as long as it has:

- a column with one of these identifiers:
  - internal work ID
  - sender's work ID, imported through work acknowledgements
  - ISWC
  - ISRC
- a column with amount to be distributed, values must be numeric

Note: Matching by internal work ID only works for musical works that have been exported at least once (as CSV, CWR or JSON).

Values for these columns must be present in all rows.

In most cases, no pre-processing is required. Most of societies and other senders of royalty statements have an option of sending them in CSV format.

Outgoing formats

Outgoing format is a CSV file. It has all the columns of the incoming file. Each incoming row will be copied for every participant who shares in distribution. Additional data will be provided in additional columns at the end.

If no matching work is found, the original row is still copied, and an error is shown in `Interested party` column.

Additional columns depend on the used algorithm.

Algorithms

DMP has two different algorithms for calculating royalty distributions.

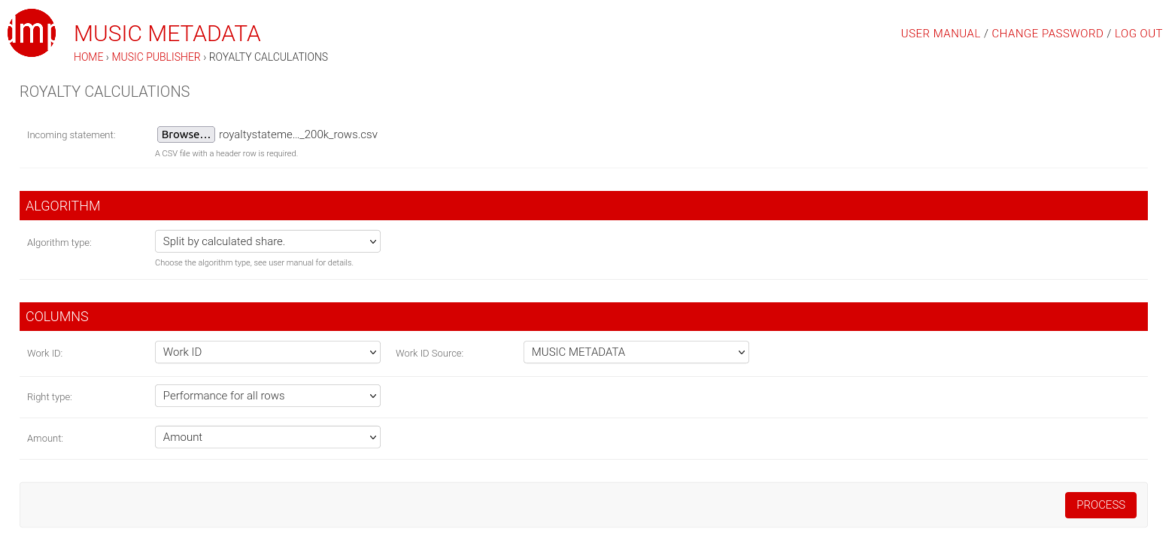
In both algorithms, user has to select:

- column containing the identifier
- type of identifier
- column containing the amount

Both algorithms add these columns:

- `Controlled by publisher (%)`
- `Interested party`
- `IP Account Number`
- `Role`
- `Share in amount received (%)`
- `Net amount`

Split by calculated share


MUSIC METADATA
HOME › MUSIC PUBLISHER › ROYALTY CALCULATIONS
USER MANUAL / CHANGE PASSWORD / LOG OUT
PROCESS

ROYALTY CALCULATIONS

Incoming statement:	Browse... royaltystateme..._200k_rows.csv A CSV file with a header row is required.

ALGORITHM

Algorithm type:	Split by calculated share. Choose the algorithm type, see user manual for details.

COLUMNS

Work ID:	Work ID	Work ID Source:	MUSIC METADATA
Right type:	Performance for all rows		
Amount:	Amount		

Fig. 20: Royalty calculation form: Split by calculated share

In this algorithm, one additional information is required:

- **column** containing the type of right (performance, mechanical, sync) or **the type of right** applicable to the whole file.

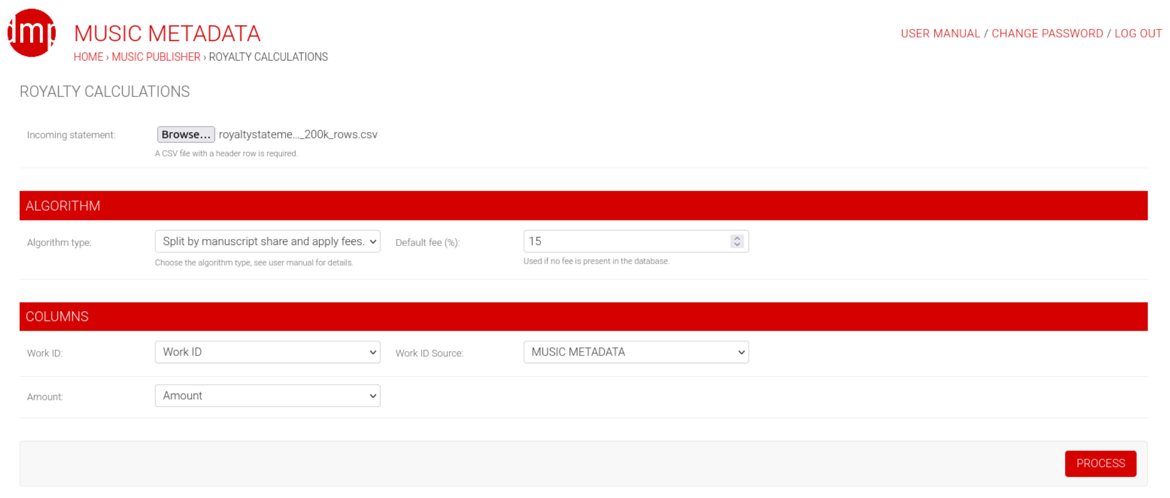
The amount in each row is split between controlled writers and the publisher, using the publishing agreement shares from the settings and manuscript shares.

Outgoing rows are generated for each controlled `writer in work` and the publisher.

In addition to columns added by both algorithms, this one also adds:

- `Right type`
- `Owned Share (%)`

Split by manuscript share and apply fees


MUSIC METADATA
HOME › MUSIC PUBLISHER › ROYALTY CALCULATIONS
USER MANUAL / CHANGE PASSWORD / LOG OUT
PROCESS

ROYALTY CALCULATIONS

Incoming statement: Browse… royaltystateme..._200k_rows.csv
A CSV file with a header row is required.

ALGORITHM

Algorithm type: Split by manuscript share and apply fees.
Choose the algorithm type, see user manual for details.
Default fee (%): 15
Used if no fee is present in the database.

COLUMNS

Work ID: Work ID
Work ID Source: MUSIC METADATA
Amount: Amount

Fig. 21: Royalty calculation form: Split by manuscript share and apply fees

This is default algorithm.

One additional information is required:

- **default publisher fee**, to be used when the fee is set neither in the `writer in work`, nor in the `writer`.

For each incoming row, each controlled writer in work receives one row in the output file. The amount is split among controlled writers, based on their relative manuscript shares. The fee is deducted from this gross amount, resulting in net amount to be paid to the writer.

Publisher fee is taken from the first available of:

- `writer in work`
- `writer` (for general agreements only)
- `default publisher fee` from this form

Note: If publisher fee is empty, it is not used, and the next option is taken. If it has value 0, then no fee is applied (zero fee), and next option is not considered.

In addition to columns added by both algorithms, this one also adds:

- `Manuscript share (%)`
- `Amount before fee`
- `Fee (%)`
- `Fee amount`

Post-processing

Excel or an alternative is the best tool for post-processing, especially creating outgoing statements.

Outgoing royalty statements

For creating outgoing statement, use pivot tables, filtering by `Interested party` column. You can design outgoing statements however you wish.

Note: If no matching work was found, there will be a row with an error message in `Interested party` column. Use the same filter to make a statement with unmatched rows.

Foreign currencies

All amounts calculated by DMP are in the same currency as the incoming data. Use a dedicated exchange rate table and VLOOKUP function for conversions.

Precision

For calculations, precision exceeds the number of decimal places in any currency. You are advised to round up only the totals, not the amounts in rows.

9.4 Section: Recordings

This section contains the model `Recordings` and closely related models `Performing Artists` and `Music Labels`.

9.4.1 Recordings

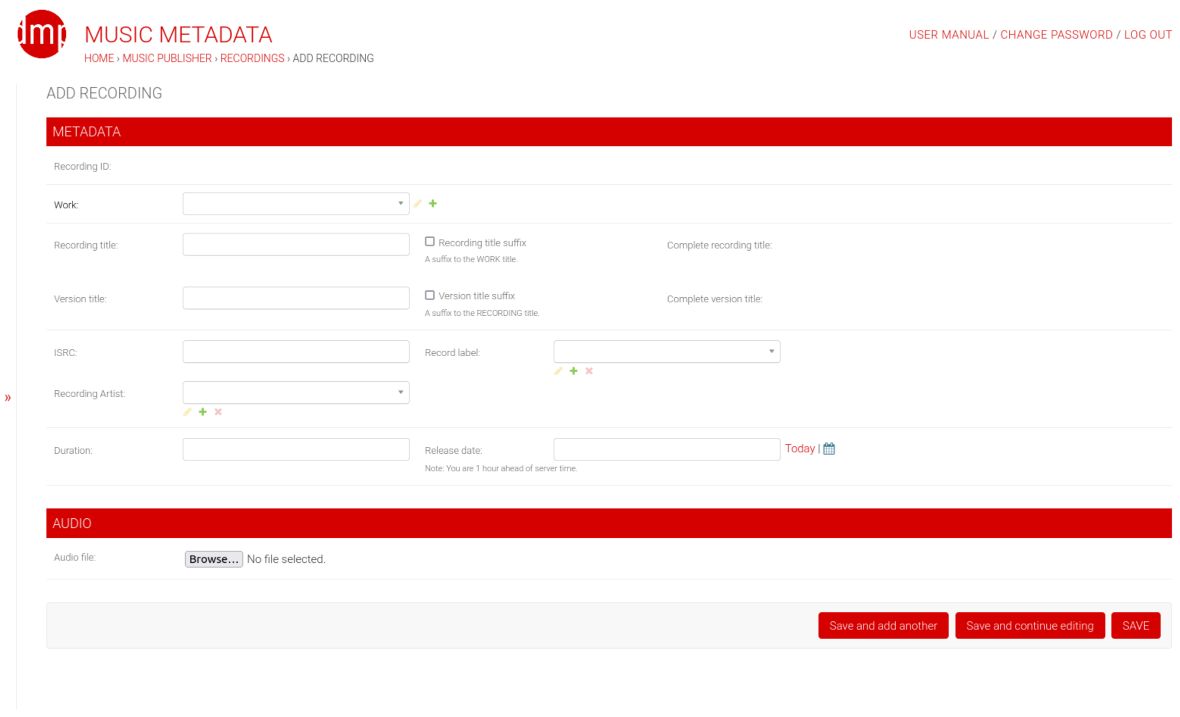
Note: Django-Music-**Publisher** is primarily software for music publishers. It can store metadata about recordings, but not audio files.

Add/Change view

There are three ways to add or edit recordings in DMP, in order of importance:

- in `add/change view` of *musical works*, in section `Recordings`
- in `add/change view` of releases (*commercial* and *library*), through pop-ups in `tracks`
- in `add/change view` of recordings (described here)

The first exists because that is the most natural way for publishers to add them. The second exists because recordings are released on releases (albums, products) as `tracks`. The last, for consistent user experience.


MUSIC METADATA

HOME › MUSIC PUBLISHER › RECORDINGS › ADD RECORDING

USER MANUAL / CHANGE PASSWORD / LOG OUT

ADD RECORDING

METADATA

Recording ID:

Work:

Recording title: ☐ Recording title suffix
A suffix to the WORK title.
Complete recording title:

Version title: ☐ Version title suffix
A suffix to the RECORDING title.
Complete version title:

ISRC: Record label:

Recording Artist:

Duration: Release date: Today |
Note: You are 1 hour ahead of server time.

AUDIO

Audio file: Browse… No file selected.

Fig. 22: Add recording

Compared to the `Recordings` section in `add work` view, there is only one additional field at the top, where the work can be chosen or added through a popup.

Note: DMP only supports recordings based on a single musical works. The link between a recording and the underlying musical work is required.

Metadata

`Recording title` should only be used if the title is different than the work title. `Version title` should only be used if different from the `recording title`. The use of suffixes is explained in *works*, section `Alternative titles`. section.

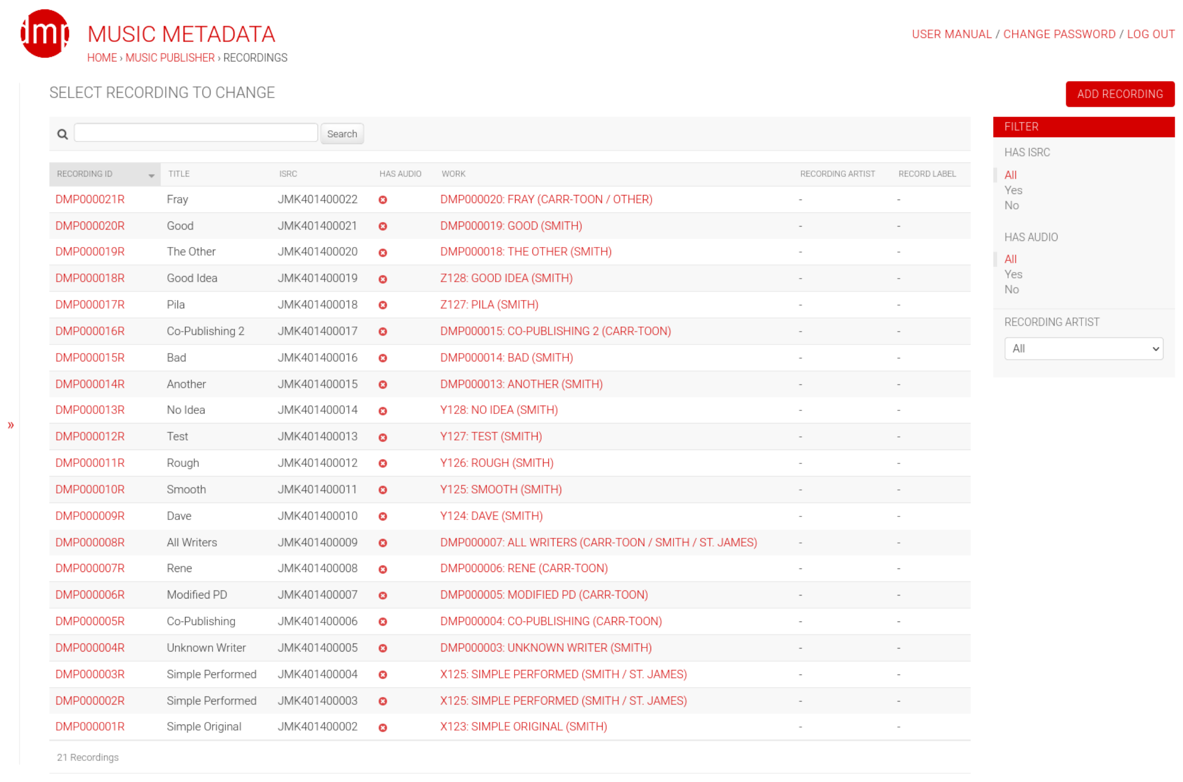
`ISRC` is International Standard Recording Code.

`Record label`, `recording artist`, `duration` and `release date` are obvious. `Duration` can be entered in seconds or in `HH:MM:SS` format. It will always be shown in the latter format.

Audio

`Audio` field is for uploading audio files. DMP currently only supports MP3 files.

List view



MUSIC METADATA

HOME › MUSIC PUBLISHER › RECORDINGS

USER MANUAL / CHANGE PASSWORD / LOG OUT

SELECT RECORDING TO CHANGE

ADD RECORDING

RECORDING ID	TITLE	ISRC	HAS AUDIO	WORK	RECORDING ARTIST	RECORD LABEL
DMP000021R	Fray	JMK401400022	✗	DMP000020: FRAY (CARR-TOON / OTHER)	-	-
DMP000020R	Good	JMK401400021	✗	DMP000019: GOOD (SMITH)	-	-
DMP000019R	The Other	JMK401400020	✗	DMP000018: THE OTHER (SMITH)	-	-
DMP000018R	Good Idea	JMK401400019	✗	Z128: GOOD IDEA (SMITH)	-	-
DMP000017R	Pila	JMK401400018	✗	Z127: PILA (SMITH)	-	-
DMP000016R	Co-Publishing 2	JMK401400017	✗	DMP000015: CO-PUBLISHING 2 (CARR-TOON)	-	-
DMP000015R	Bad	JMK401400016	✗	DMP000014: BAD (SMITH)	-	-
DMP000014R	Another	JMK401400015	✗	DMP000013: ANOTHER (SMITH)	-	-
DMP000013R	No Idea	JMK401400014	✗	Y128: NO IDEA (SMITH)	-	-
DMP000012R	Test	JMK401400013	✗	Y127: TEST (SMITH)	-	-
DMP000011R	Rough	JMK401400012	✗	Y126: ROUGH (SMITH)	-	-
DMP000010R	Smooth	JMK401400011	✗	Y125: SMOOTH (SMITH)	-	-
DMP000009R	Dave	JMK401400010	✗	Y124: DAVE (SMITH)	-	-
DMP000008R	All Writers	JMK401400009	✗	DMP000007: ALL WRITERS (CARR-TOON / SMITH / ST. JAMES)	-	-
DMP000007R	Rene	JMK401400008	✗	DMP000006: RENE (CARR-TOON)	-	-
DMP000006R	Modified PD	JMK401400007	✗	DMP000005: MODIFIED PD (CARR-TOON)	-	-
DMP000005R	Co-Publishing	JMK401400006	✗	DMP000004: CO-PUBLISHING (CARR-TOON)	-	-
DMP000004R	Unknown Writer	JMK401400005	✗	DMP000003: UNKNOWN WRITER (SMITH)	-	-
DMP000003R	Simple Performed	JMK401400004	✗	X125: SIMPLE PERFORMED (SMITH / ST. JAMES)	-	-
DMP000002R	Simple Performed	JMK401400003	✗	X125: SIMPLE PERFORMED (SMITH / ST. JAMES)	-	-
DMP000001R	Simple Original	JMK401400002	✗	X123: SIMPLE ORIGINAL (SMITH)	-	-

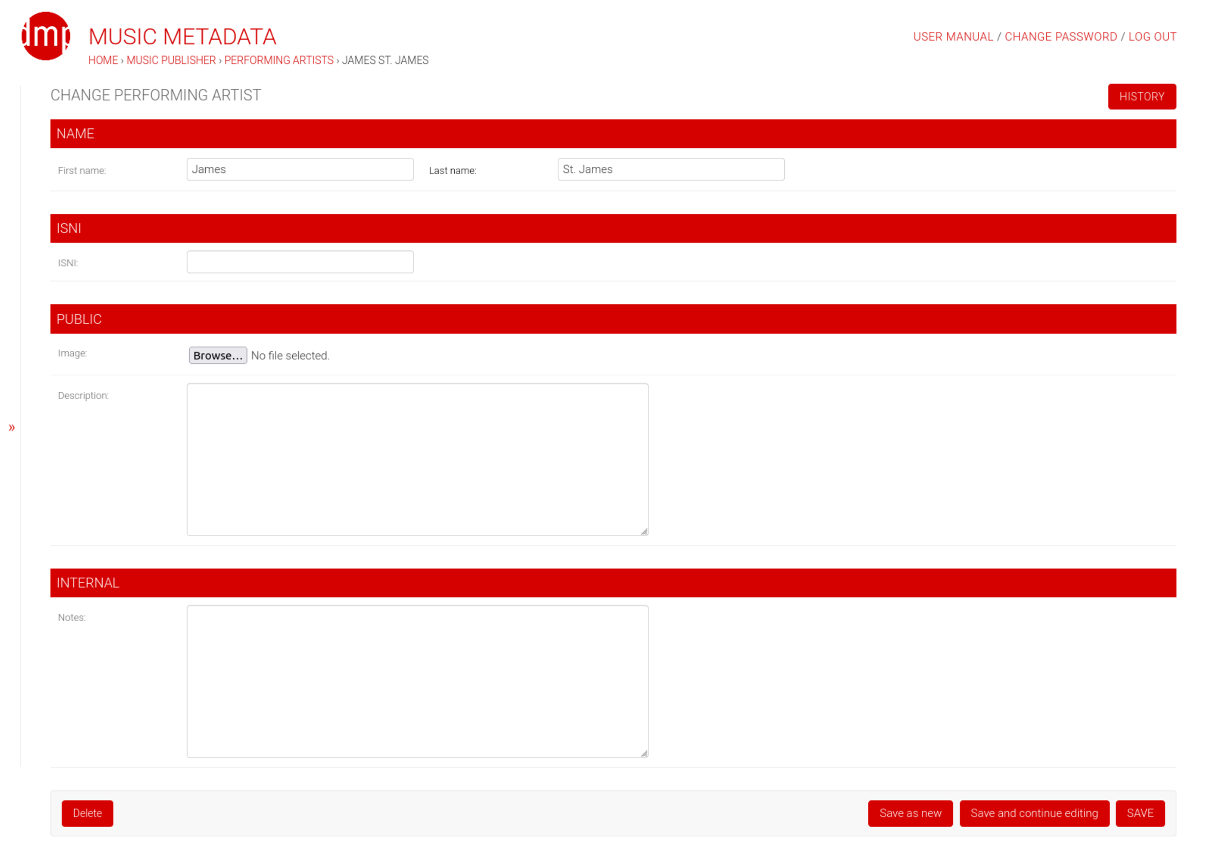
21 Recordings

Fig. 23: Recording list view

`Recording list` view provides a nice overview, with search and filter capabilities and links for *work*, *recording artist* and *record label*.

9.4.2 Performing Artists

Add View



MUSIC METADATA

HOME › MUSIC PUBLISHER › PERFORMING ARTISTS › JAMES ST. JAMES

USER MANUAL / CHANGE PASSWORD / LOG OUT

CHANGE PERFORMING ARTIST

HISTORY

NAME

First name:	James	Last name:	St. James

ISNI

ISNI:

PUBLIC

Image: Browse... No file selected.

Description:

INTERNAL

Notes:

Delete | Save as new | Save and continue editing | SAVE

Fig. 24: Add view

Add and change views for writers have four fieldsets.

Name

Last name and first name fields in the first, quite self-explanatory. Only last name is required. For bands, band name goes into last name field.

ISNI

ISNI is a unique and unambiguous identifier for performing artists.

Public

Note: This section is only visible if file uploads are configured.

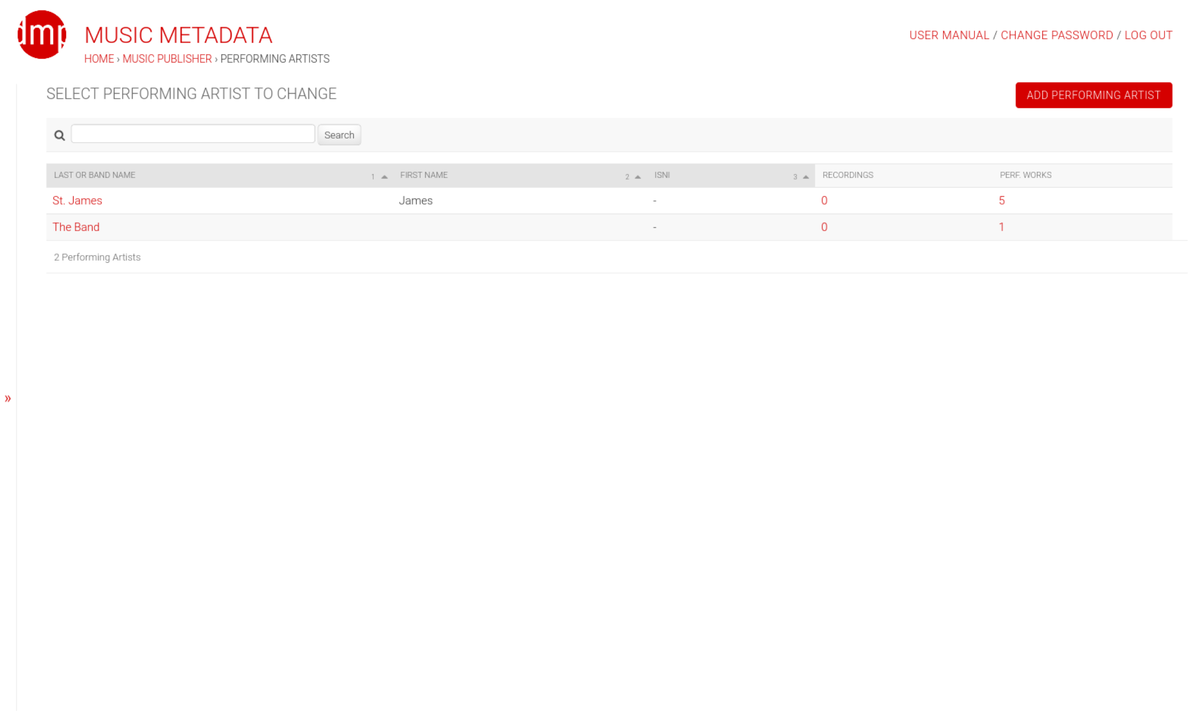
This section has two fields:

- Image - for uploading an image of the artist
- Description - for public description

Internal

This section has only a single field `Notes`. You can use it in any way you like.

List View


MUSIC METADATA
HOME › MUSIC PUBLISHER › PERFORMING ARTISTS
USER MANUAL / CHANGE PASSWORD / LOG OUT
SELECT PERFORMING ARTIST TO CHANGE
ADD PERFORMING ARTIST
Search

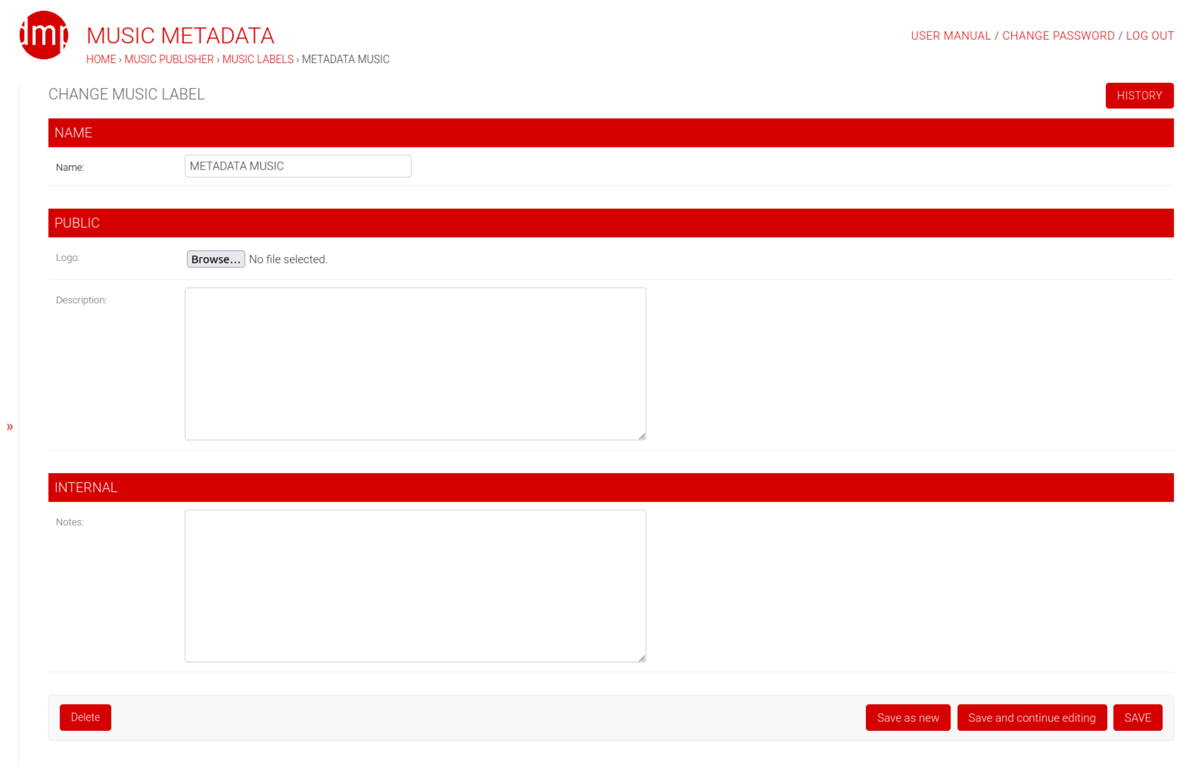
LAST OR BAND NAME	FIRST NAME	ISNI	RECORDINGS	PERF. WORKS
St. James	James	-	0	5
The Band		-	0	1

2 Performing Artists

Fig. 25: List view

There are no filters, only a search field. In the table, beside the three fields, there are two counters with links, to the *list of recordings* by this artist the *list of works* performed by this artist LIVE. It is also used if the recording data is not available.

9.4.3 Labels


MUSIC METADATA
HOME › MUSIC PUBLISHER › MUSIC LABELS › METADATA MUSIC
USER MANUAL / CHANGE PASSWORD / LOG OUT
HISTORY
Browse…
Delete
Save as new
Save and continue editing
SAVE

CHANGE MUSIC LABEL

NAME

Name: METADATA MUSIC

PUBLIC

Logo: No file selected.

Description:

INTERNAL

Notes:

Add View

Name

`Name` - for label name

Public

Note: This section is only visible if file uploads are configured.

This section has two fields:

- `Logo` - for uploading label logo
- `Description` - for public description

Internal

This section has only a single field `Notes`. You can use it in any way you like.

List View

The list views have counters with links:

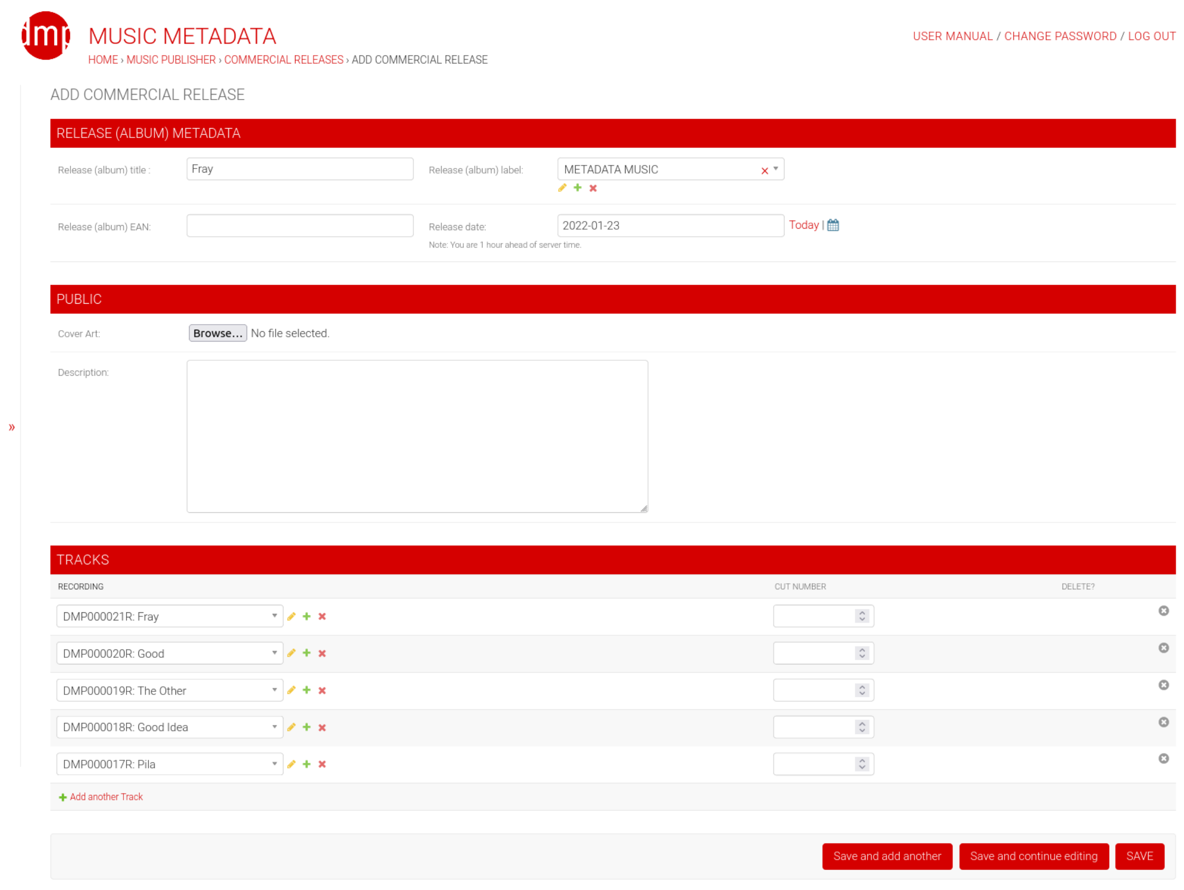
- for *recordings*, where this label was the `record label`,
- for *library releases*, where this label is the `release (album) label`,

- for *commercial releases*, where this label is the `release (album) label`.

9.5 Section: Releases

This section contains the models related to releases.

9.5.1 Commercial (General) Releases



MUSIC METADATA

USER MANUAL / CHANGE PASSWORD / LOG OUT

HOME › MUSIC PUBLISHER › COMMERCIAL RELEASES › ADD COMMERCIAL RELEASE

ADD COMMERCIAL RELEASE

RELEASE (ALBUM) METADATA

Release (album) title :	Fray	Release (album) label:	METADATA MUSIC
Release (album) EAN:		Release date:	2022-01-23 Today |

Note: You are 1 hour ahead of server time.

PUBLIC

Cover Art: Browse… No file selected.

Description:

TRACKS

RECORDING	CUT NUMBER	DELETE?
DMP000021R: Fray		
DMP000020R: Good		
DMP000019R: The Other		
DMP000018R: Good Idea		
DMP000017R: Pila		

+ Add another Track

The most typical example of a release used to be a vinyl record album, then a CD. It is often referred to as *product*.

Add view

Commercial (general) and *library* releases are actually one model with two different sets of views.

They share basic 4 fields, as well as inline `tracks`:

- `Release title`
- `Release EAN`
- `Release label`
- `Release date`
- `Tracks:`
  - `Recording`

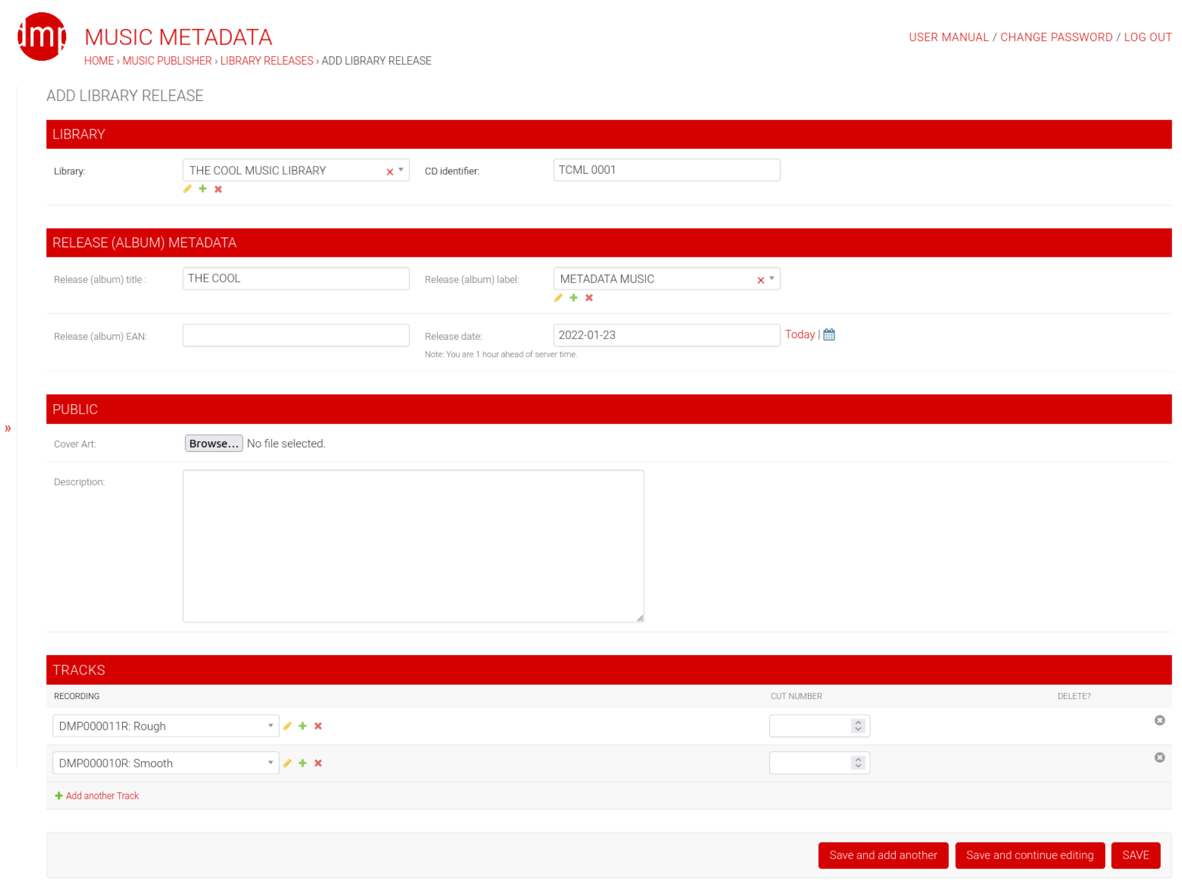
   – `Cut number`

Note: **Track** in this software means *recording in a release*.

List view

`List view` is quite simple, only three columns, `Release (album) title`, `Release (album) label` and count of tracks with link to `Recordings`.

9.5.2 Library Releases


MUSIC METADATA
USER MANUAL / CHANGE PASSWORD / LOG OUT
HOME › MUSIC PUBLISHER › LIBRARY RELEASES › ADD LIBRARY RELEASE

ADD LIBRARY RELEASE

LIBRARY

Library:	THE COOL MUSIC LIBRARY	CD identifier:	TCML 0001

RELEASE (ALBUM) METADATA

Release (album) title:	THE COOL	Release (album) label:	METADATA MUSIC
Release (album) EAN:		Release date:	2022-01-23 Today | Note: You are 1 hour ahead of server time.

PUBLIC

Cover Art:	Browse... No file selected.
Description:	

TRACKS

RECORDING	CUT NUMBER	DELETE?
DMP000011R: Rough		
DMP000010R: Smooth		

Add view

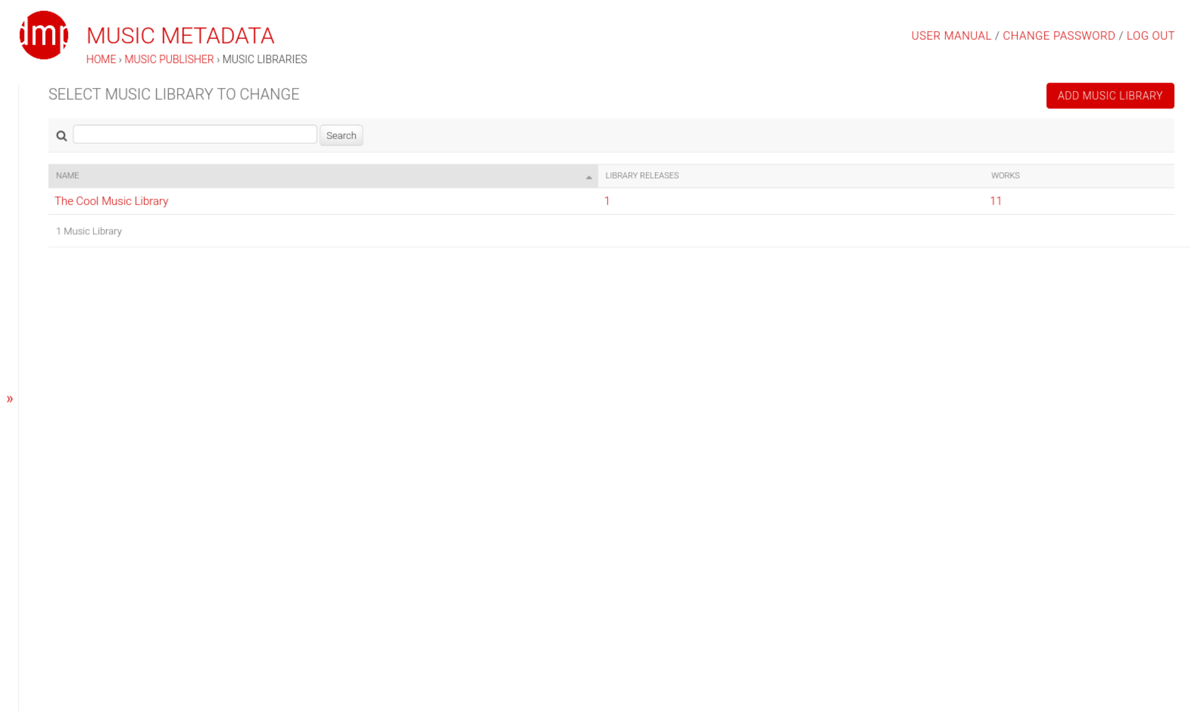
Commercial (general) and *library* releases are actually one model with two different sets of views. The only difference is that *library* releases have two additional fields, both required:

- `Library`
- `CD identifier` - a CWR field name for *release code*

List view

`List view` has 6 columns, 3 more than *commercial* releases. Two of them are for the two aforementioned field. The last one is a counter and a link to *works*. This field will list works that have `library release` field set to this library release.

9.5.3 Libraries


MUSIC METADATA
HOME › MUSIC PUBLISHER › MUSIC LIBRARIES
USER MANUAL / CHANGE PASSWORD / LOG OUT
ADD MUSIC LIBRARY
Search

SELECT MUSIC LIBRARY TO CHANGE

NAME	LIBRARY RELEASES	WORKS
The Cool Music Library	1	11

1 Music Library

Label model only has a single field: `name`.

However, the list views have counters with links:

- for *works* in this library,
- for *library releases* in this library.

CHAPTER 10

Integration (Rest API)

DMP is very good at data management and validation, but not made for public presentation of this data. Still, it makes no sense to enter the same data over and over again. Now you don't have to.

DMP has provides several browsable read-only API endpoints for integration with other software, most notably user's website.

The address of the API root, relative to the home page, is: `api/v1/`.

10.1 Featured Releases and Artists

Releases, artists, writers and labels now feature fields `image` and `description`, to be used for public content presentation. Recordings feature `audio file` field for the same reason.

There are endpoints for getting lists of all artists and releases (both commercial and library), with data in either `image` or `description` field, as well as details about an artist or a release. Details contain data about recordings (including audio files if they exist), record labels, underlying musical works and writers.

- `/api/v1/artists/`
- `/api/v1/releases/`

These endpoints are not publicly available, they are protected by `Basic HTTP Authentication`. It is recommended to create a dedicated user, has to be active, and has to have permission `Can view Performing Artist` and/or `Can view Release`.

One use example is to provide list of artists and/or releases on your website through a plugin. You do it once, and then your website will always be up-to-date, as long as you enter the data in DMP.

Warning: THIS FEATURE IS BEING DEVELOPED, IT IS NOT READY FOR PRODUCTION!

10.2 Shareable Playlists

A sharable playlist can be accessed through a normal HTML interface, or through a REST API endpoint. Both URLs can be found in the `change view`.

There is currently no way to get a list of all secret playlist.

10.3 Backup Metadata

- `/api/v1/backup_metadata/`

This endpoint can be used to get all the metadata about all works and releases. However, public data (descriptions, images and audio files) are not included.

It is available only to a `superuser`, because it's purpose is to provide one-time backup if you choose to move to a different system.

Note: If you are moving from DMP to That Green Thing, the migration is fully automated.

CHAPTER 11

For Developers

This technical section is targeting software developers.

- Code: https://github.com/matijakolaric-com/django-music-publisher/
- PYPI: https://pypi.org/project/django-music-publisher/

11.1 music_publisher

Django-Music-Publisher (DMP) is open source software for managing music metadata, registration/licencing of musical works and royalty processing.

`music_publisher` app is the only Django app in this project.

11.1.1 music_publisher.apps

Django app definition for *`music_publisher`*.

class `music_publisher.apps.`**`MusicPublisherConfig`**(*app_name*, *app_module*)

Bases: `django.apps.config.AppConfig`

Configuration for Music Publisher app.

`label`

app label

Type str

`name`

app name

Type str

`verbose_name`

app verbose name

Type str

`ready`()

Validate settings when ready to prevent deployments with invalid settings.

11.1.2 music_publisher.societies

Create society tuple and dict.

`music_publisher.societies.`**`SOCIETIES`**
(tis-n, Name (Country))

Type tuple

`music_publisher.societies.`**`SOCIETY_DICT`**
{tis-n, Name (Country)}

Type dict

11.1.3 music_publisher.validators

CWR-compatibility field-level validation.

For formats that allow dashes and dots (ISWC, IPI Base), the actual format is from CWR 2.x specification: ISWC without and IPI Base with dashes.

`music_publisher.validators.`**`check_ean_digit`**(*ean*)
EAN checksum validation.

Parameters **`ean`** (*`str`*) – EAN

Raises `ValidationError`

`music_publisher.validators.`**`check_iswc_digit`**(*iswc*, *weight*)
ISWC / IPI Base checksum validation.

Parameters

- **`iswc`** (*`str`*) – ISWC or IPI Base #
- **`weight`** (*`int`*) – 1 for ISWC, 2 for IPI Base #

Raises `ValidationError`

`music_publisher.validators.`**`check_ipi_digit`**(*all_digits*)
IPI Name checksum validation.

Parameters **`all_digits`** (*`str`*) – IPI Name #

Raises `ValidationError`

`music_publisher.validators.`**`check_isni_digit`**(*all_digits*)
ISNI checksum validation.

Parameters **`all_digits`** (*`str`*) – ISNI

Raises `ValidationError`

`music_publisher.validators.`**`check_dpid`**(*dpid*)
Calculate the checksum. A valid number should have a checksum of 1.

`class` `music_publisher.validators.`**`CWRFieldValidator`**(*field: str*)
Bases: `object`

Validate fields for CWR compliance.

`field`
Validation service name of the field being validated

Type str

`deconstruct`()
Return a 3-tuple of class import path, positional arguments, and keyword arguments.

`music_publisher.validators.`**`validate_publisher_settings`**`()`

CWR-compliance validation for publisher settings.

`music_publisher.validators.`**`validate_settings`**`()`

CWR-compliance validation for settings.

This is used to prevent deployment with invalid settings.

11.1.4 music_publisher.base

Contains base (abstract) classes used in `models`

`class` `music_publisher.base.`**`NotesManager`**

Bases: `django.db.models.manager.Manager`

Manager for objects inheriting from `NotesBase`.

Defers `NotesBase.notes` field.

`get_queryset``()`

Defer `NotesBase.notes` field.

`class` `music_publisher.base.`**`NotesBase`**`(*args, **kwargs)`

Bases: `django.db.models.base.Model`

Abstract class for all classes that have notes.

`notes`

Notes, free internal text field

Type django.db.models.TextField

`class` `music_publisher.base.`**`DescriptionBase`**`(*args, **kwargs)`

Bases: `django.db.models.base.Model`

Abstract class for all classes that have publicly visible descriptions.

`description`

Public description

Type django.db.models.TextField

`class` `music_publisher.base.`**`TitleBase`**`(*args, **kwargs)`

Bases: `django.db.models.base.Model`

Abstract class for all classes that have a title.

`title`

Title, used in work title, alternate title, etc.

Type django.db.models.CharField

`class` `music_publisher.base.`**`PersonBase`**`(*args, **kwargs)`

Bases: `django.db.models.base.Model`

Base class for all classes that contain people with first and last name.

This includes writers and artists. For bands, only the last name field is used.

`first_name`

First Name

Type django.db.models.CharField

`last_name`

Last Name

Type django.db.models.CharField

class `music_publisher.base.`**`SocietyAffiliationBase`**(**args*, ***kwargs*)

Bases: `django.db.models.base.Model`

Abstract base for all objects with CMO affiliations

`pr_society`

Performing Rights Society Code

Type django.db.models.CharField

`mr_society`

Mechanical Rights Society Code

Type django.db.models.CharField

`sr_society`

Sync. Rights Society Code

Type django.db.models.CharField

class `music_publisher.base.`**`IPIBase`**(**args*, ***kwargs*)

Bases: `django.db.models.base.Model`

Abstract base for all objects containing IPI numbers.

`ipi_base`

IPI Base Number

Type django.db.models.CharField

`ipi_name`

IPI Name Number

Type django.db.models.CharField

`_can_be_controlled`

used to determine if there is enough data for a writer to be controlled.

Type django.db.models.BooleanField

`clean_fields`(**args*, ***kwargs*)

Data cleanup, allowing various import formats to be converted into consistently formatted data.

class `music_publisher.base.`**`IPIWithGeneralAgreementBase`**(**args*, ***kwargs*)

Bases: *`music_publisher.base.IPIBase`*, *`music_publisher.base.SocietyAffiliationBase`*

Abstract base for all objects with general agreements.

`saan`

Society-assigned agreement number, in this context it is used for general agreements, for specific agreements use *`models.WriterInWork.saan`*.

Type django.db.models.CharField

`generally_controlled`

flags if a writer is generally controlled (in all works)

Type django.db.models.BooleanField

`publisher_fee`

this field is used in calculating publishing fees

Type django.db.models.DecimalField

`clean`()

Clean the data and validate.

`clean_fields`(**args*, ***kwargs*)

Data cleanup, allowing various import formats to be converted into consistently formatted data.

class `music_publisher.base.`**`AccountNumberBase`**`(`**args*, ***kwargs*`)`
Bases: `django.db.models.base.Model`

Abstract base for all objects with an account number.

`account_number`
account number, used for royalty processing

Type django.db.models.CharField

`clean_fields``(`**args*, ***kwargs*`)`
Account Number cleanup

class `music_publisher.base.`**`ArtistBase`**`(`**args*, ***kwargs*`)`
Bases: *`music_publisher.base.PersonBase`*, *`music_publisher.base.NotesBase`*, *`music_publisher.base.DescriptionBase`*

Performing artist base class.

`isni`
International Standard Name Id

Type django.db.models.CharField

`clean_fields``(`**args*, ***kwargs*`)`
ISNI cleanup

class `music_publisher.base.`**`WriterBase`**`(`**args*, ***kwargs*`)`
Bases: *`music_publisher.base.PersonBase`*, *`music_publisher.base.IPIWithGeneralAgreementBase`*, *`music_publisher.base.NotesBase`*, *`music_publisher.base.DescriptionBase`*, *`music_publisher.base.AccountNumberBase`*

Base class for writers.

class `music_publisher.base.`**`LabelBase`**`(`**args*, ***kwargs*`)`
Bases: *`music_publisher.base.NotesBase`*, *`music_publisher.base.DescriptionBase`*

Music Label base class.

`name`
Label Name

Type django.db.models.CharField

class `music_publisher.base.`**`LibraryBase`**`(`**args*, ***kwargs*`)`
Bases: `django.db.models.base.Model`

Music Library base class.

`name`
Library Name

Type django.db.models.CharField

class `music_publisher.base.`**`ReleaseBase`**`(`**args*, ***kwargs*`)`
Bases: *`music_publisher.base.DescriptionBase`*

Music Release base class

`cd_identifier`
CD Identifier, used when origin is library

Type django.db.models.CharField

`library`
Library Name

Type django.db.models.CharField

release_date
Date of the release

Type django.db.models.DateField

ean
EAN code

Type django.db.models.CharField

release_label
Label Name

Type django.db.models.CharField

release_title
Title of the release

Type django.db.models.CharField

11.1.5 music_publisher.models

Concrete models.

They mostly inherit from classes in *base*.

class music_publisher.models.**Artist**(**args*, ***kwargs*)
Bases: *music_publisher.base.ArtistBase*

Performing artist.

get_dict()
Get the object in an internal dictionary format

Returns internal dict format

Return type dict

artist_id
Artist identifier

Returns Artist ID

Return type str

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

class music_publisher.models.**Label**(**args*, ***kwargs*)
Bases: *music_publisher.base.LabelBase*

Music Label.

label_id
Label identifier

Returns Label ID

Return type str

get_dict()
Get the object in an internal dictionary format

Returns internal dict format

Return type dict

exception DoesNotExist
Bases: `django.core.exceptions.ObjectDoesNotExist`

exception MultipleObjectsReturned
Bases: `django.core.exceptions.MultipleObjectsReturned`

class `music_publisher.models.`**Library**(**args*, ***kwargs*)
Bases: *`music_publisher.base.LibraryBase`*

Music Library.

library_id
Library identifier

Returns Library ID

Return type str

get_dict()
Get the object in an internal dictionary format

Returns internal dict format

Return type dict

exception DoesNotExist
Bases: `django.core.exceptions.ObjectDoesNotExist`

exception MultipleObjectsReturned
Bases: `django.core.exceptions.MultipleObjectsReturned`

class `music_publisher.models.`**Release**(**args*, ***kwargs*)
Bases: *`music_publisher.base.ReleaseBase`*

Music Release (album / other product)

library
Foreign key to *`models.Library`*

Type django.db.models.ForeignKey

release_label
Foreign key to *`models.Label`*

Type django.db.models.ForeignKey

recordings
M2M to *`models.Recording`* through *`models.Track`*

Type django.db.models.ManyToManyField

release_id
Release identifier.

Returns Release ID

Return type str

get_dict(*with_tracks=False*)
Get the object in an internal dictionary format

Parameters **`with_tracks`** (*`bool`*) – add track data to the output

Returns internal dict format

Return type dict

exception DoesNotExist
Bases: `django.core.exceptions.ObjectDoesNotExist`

exception MultipleObjectsReturned
Bases: `django.core.exceptions.MultipleObjectsReturned`

class music_publisher.models.**LibraryReleaseManager**
Bases: `django.db.models.manager.Manager`

Manager for a proxy class *`models.LibraryRelease`*

get_queryset()
Return only library releases

Returns Queryset with instances of *`models.LibraryRelease`*

Return type django.db.models.query.QuerySet

get_dict(*qs*)
Get the object in an internal dictionary format

Parameters **qs** (*`django.db.models.query.QuerySet`*) –

Returns internal dict format

Return type dict

class music_publisher.models.**LibraryRelease**(**args*, ***kwargs*)
Bases: *`music_publisher.models.Release`*

Proxy class for Library Releases (AKA Library CDs)

objects
Database Manager

Type *LibraryReleaseManager*

clean()
Make sure that release title is required if one of the other "non-library" fields is present.

Raises `ValidationError` – If not compliant.

get_origin_dict()
Get the object in an internal dictionary format.

This is used for work origin, not release data.

Returns internal dict format

Return type dict

exception DoesNotExist
Bases: `music_publisher.models.DoesNotExist`

exception MultipleObjectsReturned
Bases: `music_publisher.models.MultipleObjectsReturned`

class music_publisher.models.**CommercialReleaseManager**
Bases: `django.db.models.manager.Manager`

Manager for a proxy class *`models.CommercialRelease`*

get_queryset()
Return only commercial releases

Returns Queryset with instances of *`models.CommercialRelease`*

Return type django.db.models.query.QuerySet

get_dict(*qs*)
Get the object in an internal dictionary format

Parameters **qs** (*`django.db.models.query.QuerySet`*) –

Returns internal dict format

Return type dict

class `music_publisher.models.`**`CommercialRelease`**(**args*, ***kwargs*)

Bases: *`music_publisher.models.Release`*

Proxy class for Commercial Releases

`objects`

Database Manager

Type *CommercialReleaseManager*

`exception DoesNotExist`

Bases: `music_publisher.models.DoesNotExist`

`exception MultipleObjectsReturned`

Bases: `music_publisher.models.MultipleObjectsReturned`

class `music_publisher.models.`**`PlaylistManager`**

Bases: `django.db.models.manager.Manager`

Manager for a proxy class *`models.Playlist`*

`get_queryset`()

Return only commercial releases

Returns Queryset with instances of *`models.CommercialRelease`*

Return type django.db.models.query.QuerySet

`get_dict`(*qs*)

Get the object in an internal dictionary format

Parameters **qs** (*`django.db.models.query.QuerySet`*) –

Returns internal dict format

Return type dict

class `music_publisher.models.`**`Playlist`**(**args*, ***kwargs*)

Bases: *`music_publisher.models.Release`*

Proxy class for Playlists

`objects`

Database Manager

Type *CommercialReleaseManager*

`clean`(**args*, ***kwargs*)

Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

`exception DoesNotExist`

Bases: `music_publisher.models.DoesNotExist`

`exception MultipleObjectsReturned`

Bases: `music_publisher.models.MultipleObjectsReturned`

class `music_publisher.models.`**`Writer`**(**args*, ***kwargs*)

Bases: *`music_publisher.base.WriterBase`*

Writers.

`original_publishing_agreement`

Foreign key to `models.OriginalPublishingAgreement`

Type django.db.models.ForeignKey

`clean`(**args*, ***kwargs*)

Check if writer who is controlled still has enough data.

writer_id
Writer ID for CWR

Returns formatted writer ID

Return type str

get_dict()
Create a data structure that can be serialized as JSON.

Returns JSON-serializable data structure

Return type dict

exception DoesNotExist
Bases: `django.core.exceptions.ObjectDoesNotExist`

exception MultipleObjectsReturned
Bases: `django.core.exceptions.MultipleObjectsReturned`

class `music_publisher.models.`**WorkManager**
Bases: `django.db.models.manager.Manager`

Manager for class *models.Work*

get_queryset()
Get an optimized queryset.

Returns Queryset with instances of *models.Work*

Return type django.db.models.query.QuerySet

get_dict(*qs*)
Return a dictionary with works from the queryset

Parameters **qs** (*django.db.models.query import QuerySet*) –

Returns dictionary with works

Return type dict

class `music_publisher.models.`**Work**(**args*, ***kwargs*)
Bases: *music_publisher.base.TitleBase*

Concrete class, with references to foreign objects.

_work_id
permanent work id, either imported or fixed when exports are created

Type django.db.models.CharField

iswc
ISWC

Type django.db.models.CharField

original_title
title of the original work, implies modified work

Type django.db.models.CharField

release_label
Foreign key to *models.LibraryRelease*

Type django.db.models.ForeignKey

last_change
when the last change was made to this object or any of the child objects, basically used in filtering

Type django.db.models.DateTimeField

artists
Artists performing the work

Type django.db.models.ManyToManyField

writers
Writers who created the work

Type django.db.models.ManyToManyField

objects
Database Manager

Type *WorkManager*

work_id
Create Work ID used in registrations.

Returns Internal Work ID

Return type str

is_modification()
Check if the work is a modification.

Returns True if modification, False if original

Return type bool

clean_fields(**args*, ***kwargs*)
Deal with various ways ISWC is written.

static get_publisher_dict()
Create data structure for the publisher.

Returns JSON-serializable data structure

Return type dict

get_dict(*with_recordings=True*)
Create a data structure that can be serialized as JSON.

Normalize the structure if required.

Returns JSON-serializable data structure

Return type dict

exception DoesNotExist
Bases: `django.core.exceptions.ObjectDoesNotExist`

exception MultipleObjectsReturned
Bases: `django.core.exceptions.MultipleObjectsReturned`

class music_publisher.models.**AlternateTitle**(**args*, ***kwargs*)
Bases: `music_publisher.base.TitleBase`

Concrete class for alternate titles.

work
Foreign key to Work model

Type django.db.models.ForeignKey

suffix
implies that the title should be appended to the work title

Type django.db.models.BooleanField

get_dict()
Create a data structure that can be serialized as JSON.

Returns JSON-serializable data structure

Return type dict

exception DoesNotExist
Bases: `django.core.exceptions.ObjectDoesNotExist`

exception MultipleObjectsReturned
Bases: `django.core.exceptions.MultipleObjectsReturned`

class `music_publisher.models.`**`ArtistInWork`**`(`**args*, ***kwargs*`)`
Bases: `django.db.models.base.Model`

Artist performing the work (live in CWR 3).

artist
FK to Artist

Type django.db.models.ForeignKey

work
FK to Work

Type django.db.models.ForeignKey

get_dict()

Returns taken from `models.Artist.get_dict()`

Return type dict

exception DoesNotExist
Bases: `django.core.exceptions.ObjectDoesNotExist`

exception MultipleObjectsReturned
Bases: `django.core.exceptions.MultipleObjectsReturned`

class `music_publisher.models.`**`WriterInWork`**`(`**args*, ***kwargs*`)`
Bases: `django.db.models.base.Model`

Writers who created this work.

At least one writer in work must be controlled. Sum of relative shares must be (roughly) 100%. Capacity is limited to roles for original writers.

work
FK to Work

Type django.db.models.ForeignKey

writer
FK to Writer

Type django.db.models.ForeignKey

saan
Society-assigned agreement number between the writer and the original publisher, please note that this field is for SPECIFIC agreements, for a general agreement, use `base.IPIBase.saan`

Type django.db.models.CharField

controlled
A complete mistery field

Type django.db.models.BooleanField

relative_share
Initial split among writers, prior to publishing

Type django.db.models.DecimalField

capacity
Role of the writer in this work

Type django.db.models.CharField

publisher_fee
Percentage of royalties kept by publisher

Type django.db.models.DecimalField

clean_fields(**args*, ***kwargs*)
Turn SAAN into uppercase.

Parameters

- ***args** – passing through
- ****kwargs** – passing through

Returns SAAN in uppercase

Return type str

clean()
Make sure that controlled writers have all the required data.

Also check that writers that are not controlled do not have data that can not apply to them.

get_agreement_dict()
Get agreement dictionary for this writer in work.

get_dict()
Create a data structure that can be serialized as JSON.

Returns JSON-serializable data structure

Return type dict

exception DoesNotExist
Bases: `django.core.exceptions.ObjectDoesNotExist`

exception MultipleObjectsReturned
Bases: `django.core.exceptions.MultipleObjectsReturned`

class `music_publisher.models.`**Recording**(**args*, ***kwargs*)
Bases: `django.db.models.base.Model`

Recording.

release_date
Recording Release Date

Type django.db.models.DateField

duration
Recording Duration

Type django.db.models.TimeField

isrc
International Standard Recording Code

Type django.db.models.CharField

record_label
Record Label

Type django.db.models.CharField

clean_fields(**args*, ***kwargs*)
ISRC cleaning, just removing dots and dashes.

Parameters

- ***args** – may be used in upstream
- ****kwargs** – may be used in upstream

Returns return from `django.db.models.Model.clean_fields()`

`complete_recording_title`
Return complete recording title.

Returns str

`complete_version_title`
Return complete version title.

Returns str

`title`
Generate title from various fields.

`recording_id`
Create Recording ID used in registrations

Returns Internal Recording ID

Return type str

`get_dict`(*with_releases=False*, *with_work=True*)
Create a data structure that can be serialized as JSON.

Parameters

- **`with_releases`** (*bool*) – add releases data (through tracks)
- **`with_work`** (*bool*) – add work data

Returns JSON-serializable data structure

Return type dict

`exception DoesNotExist`
Bases: `django.core.exceptions.ObjectDoesNotExist`

`exception MultipleObjectsReturned`
Bases: `django.core.exceptions.MultipleObjectsReturned`

`class` `music_publisher.models.`**`Track`**(**args*, ***kwargs*)
Bases: `django.db.models.base.Model`

Track, a recording on a release.

`recording`
Recording

Type django.db.models.ForeignKey

`release`
Release

Type django.db.models.ForeignKey

`cut_number`
Cut Number

Type django.db.models.PositiveSmallIntegerField

`get_dict`()
Create a data structure that can be serialized as JSON.

Returns JSON-serializable data structure

Return type dict

`exception DoesNotExist`
Bases: `django.core.exceptions.ObjectDoesNotExist`

`exception MultipleObjectsReturned`
Bases: `django.core.exceptions.MultipleObjectsReturned`

class music_publisher.models.**DeferCwrManager**

Bases: `django.db.models.manager.Manager`

Manager for CWR Exports and ACK Imports.

Defers *CWRExport.cwr* and `AckImport.cwr` fields.

get_queryset()

Return a new QuerySet object. Subclasses can override this method to customize the behavior of the Manager.

class music_publisher.models.**CWRExport**(**args*, ***kwargs*)

Bases: `django.db.models.base.Model`

Export in CWR format.

Common Works Registration format is a standard format for registration of musical works world-wide. Exports are available in CWR 2.1 revision 8 and CWR 3.0 (experimental).

nwr_rev

choice field where user can select which version and type of CWR it is

Type django.db.models.CharField

cwr

contents of CWR file

Type django.db.models.TextField

year

2-digit year format

Type django.db.models.CharField

num_in_year

CWR sequential number in a year

Type django.db.models.PositiveSmallIntegerField

works

included works

Type django.db.models.ManyToManyField

description

internal note

Type django.db.models.CharField

version

Return CWR version.

filename

Return CWR file name.

Returns CWR file name

Return type str

filename3

Return proper CWR 3.x filename.

Format is: CWYYnnnnSUB_REP_VM - m - r.EXT

Returns CWR file name

Return type str

filename2

Return proper CWR 2.x filename.

Returns CWR file name

Return type str

get_record(*key*, *record*)

Create CWR record (row) from the key and dict.

Parameters

- **key** (*str*) – type of record
- **record** (*dict*) – field values

Returns CWR record (row)

Return type str

get_transaction_record(*key*, *record*)

Create CWR transaction record (row) from the key and dict.

This methods adds transaction and record sequences.

Parameters

- **key** (*str*) – type of record
- **record** (*dict*) – field values

Returns CWR record (row)

Return type str

yield_iswc_request_lines(*works*)

Yield lines for an ISR (ISWC request) in CWR 3.x

yield_publisher_lines(*publisher*, *controlled_relative_share*)

Yield SPU/SPT lines.

Parameters

- **publisher** (*dict*) – dictionary with publisher data
- **controlled_relative_share** (*Decimal*) – sum of manuscript shares for controlled writers

Yields *str* – CWR record (row/line)

yield_registration_lines(*works*)

Yield lines for CWR registrations (WRK in 3.x, NWR and REV in 2.x)

Parameters **works** (*list*) – list of work dicts

Yields *str* – CWR record (row/line)

get_party_lines(*work*)

Yield SPU, SPT, OPU, SWR, SWT, OPT and PWR lines

Parameters **work** – musical work

Yields *str* – CWR record (row/line)

get_other_lines(*work*)

Yield ALT and subsequent lines

Parameters **work** – musical work

Yields *str* – CWR record (row/line)

get_header()

Construct CWR HDR record.

yield_lines(*works*)

Yield CWR transaction records (rows/lines) for works

Parameters **works** (*query*) – *models.Work* query

Yields *str* – CWR record (row/line)

create_cwr(*publisher_code=None*)
Create CWR and save.

exception DoesNotExist
Bases: `django.core.exceptions.ObjectDoesNotExist`

exception MultipleObjectsReturned
Bases: `django.core.exceptions.MultipleObjectsReturned`

class `music_publisher.models.`**WorkAcknowledgement**(**args*, ***kwargs*)
Bases: `django.db.models.base.Model`

Acknowledgement of work registration.

date
Acknowledgement date

Type django.db.models.DateField

remote_work_id
Remote work ID

Type django.db.models.CharField

society_code
3-digit society code

Type django.db.models.CharField

status
2-letter status code

Type django.db.models.CharField

TRANSACTION_STATUS_CHOICES
choices for status

Type tuple

work
FK to Work

Type django.db.models.ForeignKey

get_dict()
Return dictionary with external work IDs.

Returns JSON-serializable data structure

Return type dict

exception DoesNotExist
Bases: `django.core.exceptions.ObjectDoesNotExist`

exception MultipleObjectsReturned
Bases: `django.core.exceptions.MultipleObjectsReturned`

class `music_publisher.models.`**ACKImport**(**args*, ***kwargs*)
Bases: `django.db.models.base.Model`

CWR acknowledgement file import.

filename
Description

Type django.db.models.CharField

society_code
3-digit society code, please note that `choices` is not set.

Type models.CharField

society_name
Society name, used if society code is missing.

Type models.CharField

date
Acknowledgement date

Type django.db.models.DateField

report
Basically a log

Type django.db.models.CharField

cwr
contents of CWR file

Type django.db.models.TextField

exception DoesNotExist
Bases: `django.core.exceptions.ObjectDoesNotExist`

exception MultipleObjectsReturned
Bases: `django.core.exceptions.MultipleObjectsReturned`

class `music_publisher.models.`**DataImport**(**args*, ***kwargs*)
Bases: `django.db.models.base.Model`

Importing basic work data from a CSV file.

This class just acts as log, the actual logic is in `data_import`.

exception DoesNotExist
Bases: `django.core.exceptions.ObjectDoesNotExist`

exception MultipleObjectsReturned
Bases: `django.core.exceptions.MultipleObjectsReturned`

`music_publisher.models.`**smart_str_conversion**(*value*)
Convert to Title Case only if UPPER CASE.

`music_publisher.models.`**change_case**(*sender*, *instance*, ***kwargs*)
Change case of CharFields from `music_publisher`.

11.1.6 music_publisher.cwr_templates

Django templates for CWR generation.

`music_publisher.cwr_templates.`**TEMPLATES_21**
Record templates for CWR 2.1

Type dict

`music_publisher.cwr_templates.`**TEMPLATES_22**
Record templates for CWR 2.2, based on 2.1

Type dict

`music_publisher.cwr_templates.`**TEMPLATES_30**
Record templates for CWR 3.0

Type dict

`music_publisher.cwr_templates.`**TEMPLATES_31**
Record templates for CWR 3.1, based on 3.0

Type dict

11.1.7 music_publisher.templatetags

Template tags for *music_publisher*

11.1.8 music_publisher.templatetags.cwr_filters

Filters used in parsing of CWR files.

`music_publisher.templatetags.cwr_filters.`**`perc`**`(`*value*`)`
Display shares as human-readable string.

`music_publisher.templatetags.cwr_filters.`**`soc_name`**`(`*value*`)`
Display society name

`music_publisher.templatetags.cwr_filters.`**`capacity`**`(`*value*`)`
Display writer capacity/role

`music_publisher.templatetags.cwr_filters.`**`agreement_type`**`(`*value*`)`
Display publishing agreement type

`music_publisher.templatetags.cwr_filters.`**`status`**`(`*value*`)`
Display acknowledgement status

`music_publisher.templatetags.cwr_filters.`**`flag`**`(`*value*`)`
Display flag value

`music_publisher.templatetags.cwr_filters.`**`orimod`**`(`*value*`)`
Display original or modification

`music_publisher.templatetags.cwr_filters.`**`terr`**`(`*value*`)`
Display territory

`music_publisher.templatetags.cwr_filters.`**`ie`**`(`*value*`)`
Display Included / Excluded

`music_publisher.templatetags.cwr_filters.`**`role`**`(`*value*`)`
Display publisher role/capacity

11.1.9 music_publisher.templatetags.cwr_generators

Filters used in generation of CWR files.

`music_publisher.templatetags.cwr_generators.`**`rjust`**`(`*value*, *length*`)`
Format general numeric fields.

`music_publisher.templatetags.cwr_generators.`**`ljust`**`(`*value*, *length*`)`
Format general alphanumeric fields.

`music_publisher.templatetags.cwr_generators.`**`soc`**`(`*value*`)`
Format society fields.

`music_publisher.templatetags.cwr_generators.`**`cwrshare`**`(`*value*`)`
Get CWR-compatible output for share fields

11.1.10 music_publisher.templatetags.dmp_dashboard

Filter used in DMP dashboard.

`music_publisher.templatetags.dmp_dashboard.`**`yield_sections`**`(`*model_dict*, *sections*`)`
Convert model dictionary according to section structure

`music_publisher.templatetags.dmp_dashboard.`**`dmp_model_groups`**`(`*model_list*`)`
Return groups of models.

11.1.11 music_publisher.forms

Forms and formsets.

class `music_publisher.forms.`**`LibraryReleaseForm`**(**args, **kwargs*)

Bases: `django.forms.models.ModelForm`

Custom form for `models.LibraryRelease`.

class `music_publisher.forms.`**`PlaylistForm`**(**args, **kwargs*)

Bases: `django.forms.models.ModelForm`

Custom form for `models.LibraryRelease`.

class `music_publisher.forms.`**`AlternateTitleFormSet`**(*data=None, files=None, instance=None, save_as_new=False, prefix=None, queryset=None, **kwargs*)

Bases: `django.forms.models.BaseInlineFormSet`

Formset for `AlternateTitleInline`.

`clean`()

Performs these checks: if suffix is used, then validates the total length

Returns None

Raises `ValidationError`

class `music_publisher.forms.`**`WorkForm`**(**args, **kwargs*)

Bases: `django.forms.models.ModelForm`

Custom form for `models.Work`.

Calculate values for readonly field version_type.

class `music_publisher.forms.`**`ACKImportForm`**(*data=None, files=None, auto_id='id_%s', prefix=None, initial=None, error_class=<class 'django.forms.utils.ErrorList'>, label_suffix=None, empty_permitted=False, instance=None, use_required_attribute=None, renderer=None*)

Bases: `django.forms.models.ModelForm`

Form used for CWR acknowledgement imports.

`acknowledgement_file`

Field for file upload

Type FileField

`clean`()

Perform usual clean, then process the file, returning the content field as if it was the TextField.

class `music_publisher.forms.`**`WriterInWorkFormSet`**(*data=None, files=None, instance=None, save_as_new=False, prefix=None, queryset=None, **kwargs*)

Bases: `django.forms.models.BaseInlineFormSet`

Formset for `WriterInWorkInline`.

`clean`()

Performs these checks: at least one writer must be controlled, at least one writer music be Composer or Composer&Lyricist sum of relative shares must be ~100%

Returns None

Raises `ValidationError`

class `music_publisher.forms.`**`DataImportForm`**(*data=None, files=None, auto_id='id_%s', prefix=None, initial=None, error_class=<class 'django.forms.utils.ErrorList'>, label_suffix=None, empty_permitted=False, instance=None, use_required_attribute=None, renderer=None*)

Bases: `django.forms.models.ModelForm`

Form used for data imports.

`data_file`
Field for file upload

Type FileField

`clean`()
This is the actual import process, if all goes well, the report is saved.

Raises `ValidationError`

11.1.12 music_publisher.admin

Main interface for *`music_publisher`*.

All views are here, except for *`royalty_calculation`*.

class `music_publisher.admin.`**`ImageWidget`**(*attrs=None*)
Bases: `django.forms.widgets.ClearableFileInput`

class `music_publisher.admin.`**`AudioPlayerWidget`**(*attrs=None*)
Bases: `django.forms.widgets.ClearableFileInput`

class `music_publisher.admin.`**`MusicPublisherAdmin`**(*model, admin_site*)
Bases: `django.contrib.admin.options.ModelAdmin`

Parent class to all admin classes.

class `music_publisher.admin.`**`ArtistInWorkInline`**(*parent_model, admin_site*)
Bases: `django.contrib.admin.options.TabularInline`

Inline interface for *`models.ArtistInWork`*.

`model`
alias of *`music_publisher.models.ArtistInWork`*

class `music_publisher.admin.`**`RecordingInline`**(*parent_model, admin_site*)
Bases: `django.contrib.admin.options.StackedInline`

Inline interface for *`models.Recording`*, used in *`WorkAdmin`*.

`get_fieldsets`(*request, obj=None*)
Hook for specifying fieldsets.

`model`
alias of *`music_publisher.models.Recording`*

class music_publisher.admin.**ArtistAdmin**(*model*, *admin_site*)
Bases: *music_publisher.admin.MusicPublisherAdmin*

Admin interface for *models.Artist*.

get_fieldsets(*request*, *obj=None*)
Hook for specifying fieldsets.

last_or_band(*obj*)
Placeholder for models.Artist.last_name.

save_model(*request*, *obj*, *form*, **args*, ***kwargs*)
Save, then update last_change of the works whose CWR registration changes due to this change.

get_queryset(*request*)
Optimized queryset for changelist view.

work_count(*obj*)
Return the work count from the database field, or count them. (dealing with legacy)

recording_count(*obj*)
Return the work count from the database field, or count them. (dealing with legacy)

class music_publisher.admin.**LabelAdmin**(*model*, *admin_site*)
Bases: *music_publisher.admin.MusicPublisherAdmin*

Admin interface for *models.Label*.

get_fieldsets(*request*, *obj=None*)
Hook for specifying fieldsets.

get_queryset(*request*)
Optimized queryset for changelist view.

commercialrelease_count(*obj*)
Return the work count from the database field, or count them. (dealing with legacy)

libraryrelease_count(*obj*)
Return the work count from the database field, or count them. (dealing with legacy)

recording_count(*obj*)
Return the work count from the database field, or count them. (dealing with legacy)

save_model(*request*, *obj*, *form*, **args*, ***kwargs*)
Save, then update last_change of the corresponding works.

class music_publisher.admin.**LibraryAdmin**(*model*, *admin_site*)
Bases: *music_publisher.admin.MusicPublisherAdmin*

Admin interface for *models.Library*.

get_queryset(*request*)
Optimized queryset for changelist view.

libraryrelease_count(*obj*)
Return the work count from the database field, or count them. (dealing with legacy)

work_count(*obj*)
Return the work count from the database field, or count them. (dealing with legacy)

save_model(*request*, *obj*, *form*, **args*, ***kwargs*)
Save, then update last_change of the corresponding works.

class music_publisher.admin.**TrackInline**(*parent_model*, *admin_site*)
Bases: django.contrib.admin.options.TabularInline

Inline interface for *models.Track*, used in *LibraryReleaseAdmin* and *CommercialReleaseAdmin*.

model
alias of *music_publisher.models.Track*

class music_publisher.admin.**PlaylistTrackInline**(*parent_model*, *admin_site*)
Bases: *music_publisher.admin.TrackInline*

class music_publisher.admin.**ReleaseAdmin**(*model*, *admin_site*)
Bases: *music_publisher.admin.MusicPublisherAdmin*

Admin interface for *models.Release*.

has_module_permission(*request*)
Return False

has_add_permission(*request*)
Return False

has_change_permission(*request*, *obj=None*)
Return False

has_delete_permission(*request*, *obj=None*)
Return False

class music_publisher.admin.**LibraryReleaseAdmin**(*model*, *admin_site*)
Bases: *music_publisher.admin.MusicPublisherAdmin*

Admin interface for *models.LibraryRelease*.

form
alias of *music_publisher.forms.LibraryReleaseForm*

get_fieldsets(*request*, *obj=None*)
Hook for specifying fieldsets.

get_inline_instances(*request*, *obj=None*)
Limit inlines in popups.

save_model(*request*, *obj*, *form*, **args*, ***kwargs*)
Save, then update `last_change` of the corresponding works.

get_queryset(*request*)
Optimized queryset for changelist view.

work_count(*obj*)
Return the work count from the database field, or count them. (dealing with legacy)

track_count(*obj*)
Return the work count from the database field, or count them. (dealing with legacy)

create_json(*request*, *qs*)
Batch action that downloads a JSON file containing library releases.

Returns JSON file with selected works

Return type JsonResponse

get_actions(*request*)
Custom action disabling the default `delete_selected`.

class music_publisher.admin.**PlaylistAdmin**(*model*, *admin_site*)
Bases: *music_publisher.admin.MusicPublisherAdmin*

Admin interface for *models.Playlist*.

form
alias of *music_publisher.forms.PlaylistForm*

get_inline_instances(*request*, *obj=None*)
Limit inlines in popups.

get_queryset (*request*)
Optimized queryset for changelist view.

track_count (*obj*)
Return the work count from the database field, or count them. (dealing with legacy)

class music_publisher.admin.**CommercialReleaseAdmin** (*model*, *admin_site*)
Bases: *music_publisher.admin.MusicPublisherAdmin*

Admin interface for *models.CommercialRelease*.

get_fieldsets (*request*, *obj=None*)
Hook for specifying fieldsets.

get_inline_instances (*request*, *obj=None*)
Limit inlines in popups.

get_queryset (*request*)
Optimized queryset for changelist view.

track_count (*obj*)
Return the work count from the database field, or count them. (dealing with legacy)

create_json (*request*, *qs*)
Batch action that downloads a JSON file containing commercial releases.

Returns JSON file with selected commercial releases

Return type JsonResponse

get_actions (*request*)
Custom action disabling the default `delete_selected`.

class music_publisher.admin.**WriterAdmin** (*model*, *admin_site*)
Bases: *music_publisher.admin.MusicPublisherAdmin*

Interface for *models.Writer*.

get_fieldsets (*request*, *obj=None*)
Return the fieldsets.

Depending on settings, MR and PR affiliations may not be needed. See *WriterAdmin.get_society_list()*

static get_society_list ()
List which society fields are required.

Mechanical and Sync affiliation is not required if writers don't collect any of it, which is the most usual case.

save_model (*request*, *obj*, *form*, **args*, ***kwargs*)
Perform normal save_model, then update last_change of all connected works.

get_queryset (*request*)
Optimized queryset for changelist view.

work_count (*obj*)
Return the work count from the database field, or count them. (dealing with legacy)

class music_publisher.admin.**AlternateTitleInline** (*parent_model*, *admin_site*)
Bases: `django.contrib.admin.options.TabularInline`

Inline interface for *models.AlternateTitle*.

model
alias of *music_publisher.models.AlternateTitle*

formset
alias of *music_publisher.forms.AlternateTitleFormSet*

complete_alt_title(*obj*)
Return the complete title, see `models.AlternateTitle.__str__()`

class `music_publisher.admin.`**WriterInWorkInline**(*parent_model*, *admin_site*)
Bases: `django.contrib.admin.options.TabularInline`

Inline interface for *`models.WriterInWork`*.

model
alias of *`music_publisher.models.WriterInWork`*

formset
alias of *`music_publisher.forms.WriterInWorkFormSet`*

class `music_publisher.admin.`**WorkAcknowledgementInline**(*parent_model*, *admin_site*)
Bases: `django.contrib.admin.options.TabularInline`

Inline interface for *`models.WorkAcknowledgement`*, used in *`WorkAdmin`*.

Note that normal users should only have a 'view' permission.

model
alias of *`music_publisher.models.WorkAcknowledgement`*

class `music_publisher.admin.`**WorkAdmin**(*model*, *admin_site*)
Bases: *`music_publisher.admin.MusicPublisherAdmin`*

Admin interface for *`models.Work`*.

This is by far the most important part of the interface.

actions
batch actions used: *`create_cwr()`*, *`create_json()`*

Type tuple

inlines
inlines used in change view: *`AlternateTitleInline`*, *`WriterInWorkInline`*, *`RecordingInline`*, *`ArtistInWorkInline`*, *`WorkAcknowledgementInline`*,

Type tuple

form
alias of *`music_publisher.forms.WorkForm`*

writer_last_names(*obj*)
This is a standard way how writers are shown in other apps.

percentage_controlled(*obj*)
Controlled percentage (sum of relative shares for controlled writers)

Please note that writers in work are already included in the queryset for other reasons, so no overhead except summing.

work_id(*obj*)
Return *`models.Work.work_id`*, make it sortable.

cwr_export_count(*obj*)
Return the count of CWR exports with the link to the filtered changelist view for *`CWRExportAdmin`*.

recording_count(*obj*)
Return the count of CWR exports with the link to the filtered changelist view for *`CWRExportAdmin`*.

get_queryset(*request*)
Optimized queryset for changelist view.

class InCWRListFilter(*request*, *params*, *model*, *model_admin*)
Bases: `django.contrib.admin.filters.SimpleListFilter`

Custom list filter if work is included in any of CWR files.

lookups (*request*, *model_admin*)
Simple Yes/No filter

queryset (*request*, *queryset*)
Filter if in any of CWR files.

class ACKSocietyListFilter (*request*, *params*, *model*, *model_admin*)
Bases: `django.contrib.admin.filters.SimpleListFilter`

Custom list filter of societies from ACK files.

lookups (*request*, *model_admin*)
Simple Yes/No filter

queryset (*request*, *queryset*)
Filter on society sending ACKs.

class ACKStatusListFilter (*request*, *params*, *model*, *model_admin*)
Bases: `django.contrib.admin.filters.SimpleListFilter`

Custom list filter on ACK status.

lookups (*request*, *model_admin*)
Simple Yes/No filter

queryset (*request*, *qs*)
Filter on ACK status.

class HasISWCListFilter (*request*, *params*, *model*, *model_admin*)
Bases: `django.contrib.admin.filters.SimpleListFilter`

Custom list filter on the presence of ISWC.

lookups (*request*, *model_admin*)
Simple Yes/No filter

queryset (*request*, *queryset*)
Filter on presence of `iswc`.

class HasRecordingListFilter (*request*, *params*, *model*, *model_admin*)
Bases: `django.contrib.admin.filters.SimpleListFilter`

Custom list filter on the presence of recordings.

lookups (*request*, *model_admin*)
Simple Yes/No filter

queryset (*request*, *queryset*)
Filter on presence of `models.Recording`.

get_search_results (*request*, *queryset*, *search_term*)
Deal with the situation term is work ID.

save_model (*request*, *obj*, *form*, **args*, ***kwargs*)
Set last_change if the work form has changed.

save_formset (*request*, *form*, *formset*, *change*)
Set last_change for the work if any of the inline forms has changed.

create_cwr (*request*, *qs*)
Batch action that redirects to the add view for `CWRExportAdmin` with selected works.

create_json (*request*, *qs*)
Batch action that downloads a JSON file containing selected works.

Returns JSON file with selected works

Return type JsonResponse

get_labels_for_csv (*works*, *repeating_column_nr=0*, *simple=False*)
Return the list of labels for the CSV file.

get_rows_for_csv(*works*)
Return rows for the CSV file, including the header.

create_csv(*request, qs*)
Batch action that downloads a CSV file containing selected works.

Returns JSON file with selected works

Return type JsonResponse

get_actions(*request*)
Custom action disabling the default `delete_selected`.

get_inline_instances(*request, obj=None*)
Limit inlines in popups.

class music_publisher.admin.**RecordingAdmin**(*model, admin_site*)
Bases: *music_publisher.admin.MusicPublisherAdmin*

Admin interface for *models.Recording*.

class HasISRCListFilter(*request, params, model, model_admin*)
Bases: `django.contrib.admin.filters.SimpleListFilter`

Custom list filter on the presence of ISRC.

lookups(*request, model_admin*)
Simple Yes/No filter

queryset(*request, queryset*)
Filter on presence of *iswc*.

class HasAudioFilter(*request, params, model, model_admin*)
Bases: `django.contrib.admin.filters.SimpleListFilter`

Custom list filter on the presence of audio file.

lookups(*request, model_admin*)
Simple Yes/No filter

queryset(*request, queryset*)
Filter on presence of *iswc*.

get_fieldsets(*request, obj=None*)
Hook for specifying fieldsets.

get_queryset(*request*)
Optimized query regarding work name

recording_id(*obj*)
Return *models.Recording.recording_id*, make it sortable.

title(*obj*)
Return the recording title, which is not the necessarily the title field.

work_link(*obj*)
Link to the work the recording is based on.

artist_link(*obj*)
Link to the recording artist.

label_link(*obj*)
Link to the recording label.

class music_publisher.admin.**CWRExportAdmin**(*model, admin_site*)
Bases: `django.contrib.admin.options.ModelAdmin`

Admin interface for *models.CWRExport*.

work_count(*obj*)
Return the work count from the database field, or count them. (dealing with legacy)

get_preview(*obj*)
Get CWR preview.

If you are using highlighing, then override this method.

view_link(*obj*)
Link to the CWR preview.

download_link(*obj*)
Link for downloading CWR file.

get_queryset(*request*)
Optimized query with count of works in the export.

get_readonly_fields(*request*, *obj=None*)
Read-only fields differ if CWR has been completed.

get_fields(*request*, *obj=None*)
Shown fields differ if CWR has been completed.

has_add_permission(*request*)
Return false if CWR delivery code is not present.

has_delete_permission(*request*, *obj=None*)
If CWR has been created, it can no longer be deleted, as it may have been sent. This may change once the delivery is automated.

has_change_permission(*request*, *obj=None*)
If object exists, it can only be edited in changelist.

get_form(*request*, *obj=None*, ***kwargs*)
Set initial values for work IDs.

add_view(*request*, *form_url=''*, *extra_context=None*, *work_ids=None*)
Added work_ids as default for wizard from `WorkAdmin.create_cwr()`.

change_view(*request*, *object_id*, *form_url=''*, *extra_context=None*)
Normal change view with two sub-views defined by GET parameters:

Parameters

- **preview** – that returns the preview of CWR file,
- **download** – that downloads the CWR file.

save_related(*request*, *form*, *formsets*, *change*)
`save_model()` passes the main object, which is needed to fetch CWR from the external service, but only after related objects are saved.

class music_publisher.admin.**AdminWithReport**(*model*, *admin_site*)
Bases: `django.contrib.admin.options.ModelAdmin`

The parent class for all admin classes with a report field.

print_report(*obj*)
Mark report as HTML-safe.

class music_publisher.admin.**ACKImportAdmin**(*model*, *admin_site*)
Bases: `music_publisher.admin.AdminWithReport`

Admin interface for `models.ACKImport`.

get_form(*request*, *obj=None*, ***kwargs*)
Returns a custom form for new objects, default one for changes.

get_fields(*request*, *obj=None*)
Return different fields for add vs change.

process (*request*, *ack_import*, *file_content*, *import_iswcs=False*)
Create appropriate WorkAcknowledgement objects, without duplicates.

Big part of this code should be moved to the model, left here because messaging is simpler.

save_model (*request*, *obj*, *form*, *change*)
Custom save_model, it ignores changes, validates the form for new instances, if valid, it processes the file and, upon success, calls `super().save_model`.

has_add_permission (*request*)
Return false if CWR delivery code is not present.

has_delete_permission (*request*, *obj=None*, **args*, ***kwargs*)
Deleting ACK imports is a really bad idea.

has_change_permission (*request*, *obj=None*)
Deleting this would make no sense, since the data is processed.

get_preview (*obj*)
Get CWR preview.

If you are using highlighing, then override this method.

view_link (*obj*)
Link to CWR ACK preview.

change_view (*request*, *object_id*, *form_url=''*, *extra_context=None*)
Normal change view with a sub-view defined by GET parameters:

Parameters **preview** – that returns the preview of CWR file.

class music_publisher.admin.**DataImportAdmin** (*model*, *admin_site*)
Bases: *music_publisher.admin.AdminWithReport*

Data import from CSV files.

Only the interface is here, the whole logic is in *data_import*.

form
alias of *music_publisher.forms.DataImportForm*

get_fields (*request*, *obj=None*)
Return different fields for add vs change.

has_delete_permission (*request*, *obj=None*, **args*, ***kwargs*)
Deleting data imports is a really bad idea.

has_change_permission (*request*, *obj=None*)
Deleting this would make no sense, since the data is processed.

get_form (*request*, *obj=None*, *change=False*, ***kwargs*)
Return a Form class for use in the admin add view. This is used by add_view and change_view.

save_model (*request*, *obj*, *form*, *change*)
Custom save_model, it ignores changes, validates the form for new instances, if valid, it processes the file and, upon success, calls `super().save_model`.

11.1.13 music_publisher.data_import

All the code related to importing data from external files.

Currently, only works (with writers, artists, library data and ISRCs) are imported. (ISRCs will be used for importing recording data the in future.)

class music_publisher.data_import.**DataImporter** (*filelike*, *user=None*)
Bases: `object`

log (*obj*, *message*, *change=False*)
Helper function for logging history.

static get_clean_key (*value*, *tup*, *name*)
Try to match either key or value from a user input mess.

process_writer_value (*key*, *key_elements*, *value*)
Clean a value for a writer and return it.

If it is a 'controlled', then also calculate general agreement. Always return a tuple.

unflatten (*in_dict*)
Create a well-structured dictionary with cleaner values.

get_writers (*writer_dict*)
Yield Writer objects, create if needed.

get_artists (*artist_dict*)
Yield Artist objects, create if needed.

get_library_release (*library_name*, *cd_identifier*)
Yield LibraryRelease objects, create if needed.

process_row (*row*)
Process one row from the incoming data.

run ()
Run the import.

11.1.14 music_publisher.royalty_calculation

This module is about processing royalty statements.

It processes files in the request-response cycle, not in background workers. Therefore, focus is on speed. Nothing is written to the database, and SELECTs are optimised and performed in one batch.

music_publisher.royalty_calculation.**get_id_sources** ()
Yield choices, fixed and societies.

music_publisher.royalty_calculation.**get_right_types** ()
Yield fixed options.

They will be extended with columns in JS and prior to validation.

class music_publisher.royalty_calculation.**RoyaltyCalculationForm** (**args*, ***kwargs*)
Bases: `django.forms.forms.Form`

The form for royalty calculations.

is_valid ()
Append additional choices to various fields, prior to the actual validation.

class music_publisher.royalty_calculation.**RoyaltyCalculation** (*form*)
Bases: `object`

The process of royalty calculation.

filename
Return the filename of the output file.

fieldnames
Return the list of field names in the output file.

get_work_ids ()
Find work unambiguous work identifiers.

Returns set of work identifier from the file

get_work_queryset(*work_ids*)
Return the appropriate queryset based on work ID source and ids.

Returns queryset with `models.WriterInWork` objects. `query_id` has the matched field value.

generate_works_dict(*qs*)
Generate the works cache.

Returns dict (works) of lists (writerinwork) of dicts

generate_writer_dict()
Generate the writers cache. :returns: dict (writer) of dicts

get_works_and_writers()
Get work and writer data.

Extract all work IDs, then perform the queries and put them in dictionaries. When the actual file processing happens, no further queries are required.

process_row(*row*)
Process one incoming row, yield multiple output rows.

out_file_path
This method creates the output file and outputs the temporary path.

Note that the process happens is several passes.

class `music_publisher.royalty_calculation.`**RoyaltyCalculationView**(***kwargs*)
Bases: `django.contrib.auth.mixins.PermissionRequiredMixin`, `django.views.generic.edit.FormView`

The view for royalty calculations.

form_class
alias of `RoyaltyCalculationForm`

render_to_response(*context*, ***response_kwargs*)
Prepare the context, required since we use admin template.

dispatch(*request*, **args*, ***kwargs*)
Royalty processing works only with TemporaryFileUploadHandler.

form_valid(*form*)
This is where the magic happens.

11.1.15 music_publisher.tests

Tests for `music_publisher`.

The folder includes these files:

- CW200001DMP_000.V21 - CWR 2.1 registration file
- CW200002DMP_0000_V3-0-0.SUB - CWR 3.0 registration file
- CW200003DMP_0000_V3-0-0.ISR - CWR3.0 ISWC request file
- CW200001052_DMP.V21 - CWR 2.1 acknowledgement file
- dataimport.csv - used for data imports
- royaltystatement.csv - CSV royalty statement
- royaltystatement_200k_rows.csv - CSV royalty statement with 200.000 rows, used for load testing.

Actual tests are in `music_publisher.tests.tests`.

11.1.16 music_publisher.tests.tests

Tests for `music_publisher`.

This software has almost full test coverage. The only exceptions are instances of `Exception` being caught during data imports. (User idiocy is boundless.)

Most of the tests are functional end-to-end tests. While they test that code works, they don't always test that it works as expected.

Still, it must be noted that exports are tested against provided templates (made in a different software, not using the same code beyond Python standard library).

More precise tests would be better.

`music_publisher.tests.tests.`**`get_data_from_response`**`(`*response*`)`
Helper for extracting data from HTTP response in a way that can be fed back into POST that works with Django Admin.

`class` `music_publisher.tests.tests.`**`DataImportTest`**`(`*methodName='runTest'*`)`
Bases: `django.test.testcases.TestCase`

Functional test for data import from CSV files.

`classmethod setUpClass``()`
Hook method for setting up class fixture before running tests in the class.

`test_log``()`
Test logging during import.

`test_unknown_key_exceptions``()`
Test exceptions not tested in functional tests.

`class` `music_publisher.tests.tests.`**`AdminTest`**`(`*methodName='runTest'*`)`
Bases: `django.test.testcases.TestCase`

Functional tests on the interface, and several related unit tests.

Note that tests build one atop another, simulating typical work flow.

`classmethod create_original_work``()`
Create original work, three writers, one controlled, with recording, alternate titles, included in a commercial release.

`classmethod create_modified_work``()`
Create modified work, original writer plus arranger, with recording, alternate titles.

`classmethod create_copublished_work``()`
Create work, two writers, one co-published

`classmethod create_duplicate_work``()`
Create work, two writers, one co-published, duplicate.

`classmethod create_writers``()`
Create four writers with different properties.

`classmethod create_cwr2_export``()`
Create a NWR and a REV CWR2 Export.

`classmethod create_cwr3_export``()`
Create a WRK and an ISR CWR3 Export.

`classmethod create_work_acknowledgements``()`
Create work acknowledgements.

`classmethod setUpClass``()`
Class setup.

Creating users. Creating instances of classes of less importance:

- label,
- library,
- artist,
- releases,

then calling the methods above.

test_strings()
Test __str__ methods for created objects.

test_unknown_user()
Several fast test to make sure that an unregistered user is blind.

test_super_user()
Testing index for superuser covers all the cases.

test_super_user_with_files()
Testing index for superuser covers all the cases.

test_staff_user()
Test that a staff user can access some urls.

Please note that most of the work is in other tests.

test_staff_user_with_files()
Testing index for superuser covers all the cases.

test_cwr_previews()
Test that CWR preview works.

test_cwr_downloads()
Test that the CWR file can be downloaded.

test_json()
Test that JSON export works.

test_cwr_nwr()
Test that CWR export works.

test_csv()
Test that CSV export works.

test_label_change()
Test that `models.Label` objects can be edited.

test_library_change()
Test that `models.Library` objects can be edited.

test_library_change_2()
Test that `models.Library` objects can be edited.

test_artist_change()
Test that `models.Artist` objects can be edited.

test_commercialrelease_change()
Test that `models.CommercialRelease` can be edited.

test_libraryrelease_change()
Test that `models.LibraryRelease` can be edited.

test_audit_user()
Test that audit user can see, but not change things.

test_generally_controlled_not_controlled()
Test that a *controlled* flag must be set for a writer who is generally controlled.

test_generally_controlled_missing_capacity()
Test that if *controlled* flag is set, the *capacity* must be set as well.

test_controlled_but_no_writer()
Test that a line without a writer can not have *controlled* set.

test_controlled_but_missing_data()
The requirements for a controlled writer are higher, make sure they are obeyed when setting a writer as controlled.

test_writer_switch()
Just replace one writer with another, just to test last change

test_not_controlled_extra_saan()
SAAN can not be set if a writer is not controlled.

test_not_controlled_extra_fee()
Publisher fee can not be set if a writer is not controlled.

test_bad_alt_title()
Test that alternate title can not have disallowed characters.

test_unallowed_capacity()
Some capacieties are allowed only in modifications.

test_missing_capacity()
At least one of the additional capacieties must be set for modifications.

test_none_controlled()
At least one Writer in Work line must be set as controlled.

test_wrong_sum_of_shares()
Sum of shares must be (roughly) 100%

test_wrong_capacity_in_copublishing_modification()
Test the situation where one writer appears in two rows, once as controlled, once as not with different capacities.

test_altitle_sufix_too_long()
A suffix plus the base title plus one space in between must be 60 characters or less.

test_ack_import_and_work_filters()
Test acknowledgement import and then filters on the change view, as well as other related views.

These tests must be together, ack import is used in filters.

test_data_import_and_royalty_calculations()
Test data import, ack import and royalty calculations.

This is the normal process, work data is entered, then the registration follows and then it can be processed in royalty statements.

This test also includes load testing, 200.000 rows must be imported in under 10-15 seconds, performed 4 times with different algos and ID types.

test_bad_data_import()
Test bad data import.

test_recording_filters()
Test Work changelist filters.

test_search()
Test Work search.

test_simple_save()
Test saving changed Work form.

test_create_cwr_wizard()
Test if CWR creation action works as it should.

test_create_cwr_wizard_no_publisher_code()
Publisher code is required for CWR generation, it must fail if attempted otherwise.

class `music_publisher.tests.tests.`**`CWRTemplatesTest`**`(`*methodName='runTest'*`)`
Bases: `django.test.testcases.SimpleTestCase`

A test related to CWR Templates.

`test_templates``()`
Test CWR 2.1, 2.2 and 3.0 generation with empty values.

class `music_publisher.tests.tests.`**`ValidatorsTest`**`(`*methodName='runTest'*`)`
Bases: `django.test.testcases.TestCase`

Test all validators.

Note that validators are also validating settings.

class `music_publisher.tests.tests.`**`ModelsSimpleTest`**`(`*methodName='runTest'*`)`
Bases: `django.test.testcases.TransactionTestCase`

These tests are modifying objects directly.

`test_work``()`
A complex test where a complete Work objects with all related objects is created.

class `music_publisher.tests.tests.`**`OtherFunctionalTest`**`(`*methodName='runTest'*`)`
Bases: `django.test.testcases.SimpleTestCase`

These tests are testing things not tested otherwise.

Python Module Index

Index