ColDoc Project Documentation

Release 0.1

Andrea C. G. Mennucci

Apr 06, 2021

Contents:

CHAPTER 1

ColDoc Project

There are two main frameworks currently used to present information (in particular, related to scientific fields).

- A document redacted with the standard *LaTeX* / *PDF* toolbox.
  - *Pros:* these documents are state-of-the-art quality.
  - *Cons:* the final user has no way of interacting with a PDF document (other than sending an e-mail to the original authors)
- The *Web 2.0* way (think of: Wikipedia or Stack Exchange).
  - *Pros:* in those frameworks, content is continuously developed and augmented by users.
  - *Cons:* the end result, though, is fragmented, and cannot (in general) be presented as an unified document.

A ColDoc tries to get the best of two publishing frameworks.

1.1 Code

The code is open source, it is available at https://github.com/mennucc/ColDoc_project

The code uses Django, that is implemented using the Python language; it also uses some JavaScript snippets for interactive features.

1.1.1 Authors

This software is Copyright 2019-21

Andrea C. G. Mennucci

1.1.2 License

See file *LICENSE.txt* in the code distribution

1.2 Features

1.2.1 Code, portal, document

This project is structured to keep separation between the *code*, the *portal*, and the *data*.

Foremost, we install the *code* following the instructions in the *install section*.

Then, we *deploy* the structure for a *portal* following the instructions in the *deploy section*.

Eventually, we add one or more *documents* in the *portal*. See *section* on how to prepare a document for its splitting and uploading into a *portal*.

(You may also deploy multiple *portals* using the same *code*).

1.2.2 Access management

Access to the whole document, or parts whereof, can be tuned.

See the *section on permissions* for details.

1.2.3 UUID and blobs

When a *LaTeX* document is inserted into a *portal*, it is *blobified*: it is divided in many small files, called *blobs*, each identified by an *UUID* (an unique identifier).

The purpose is twofold:

- each blob can be viewed conveniently by itself: the portal will compile an HTML representation of it, that is easily accessible (it also well adapts to mobile viewers), as well as a compact PDF representation (using the *standalone* class).
- The UUID is a permanent identifier for that content: even if other material is added before that part of LaTeX, the UUID associated to it will not change (contrary to ordinary references in LaTeX); UUIDs can also be used to reference from *outside* of the document, using appropriate web URLs.

Note that an UUID can reference to different versions of the same object:

- for images, there may be different formats available;
- for text, different languages may be available.

Moreover, the portal will compile the whole document, as PDF and as HTML. The whole document contains UUID markers, of the form *[XXX]*, that can be used to jump to the web page of that UUID; vice-versa in the web page of the UUID there are links to view that UUID in the context of the whole document.

1.3 Documentation

All documentation is in the "`docs`" directory.

The documentation is in RST format, so it is mostly standard text: you can read it in the files inside *docs/source*.

1.3.1 Compile

To compile the documentation, you will need the *sphinx* toolset. To install it:

```
pip3 install sphinx
```

or, if you prefer, in Debian-based systems (like Ubuntu):

```
sudo apt install python3-sphinx
```

Then

```
cd docs
make html
```

or any other format that you wish. Then start browsing by

```
firefox docs/build/html/index.html
```

1.4 Quick start

If just want to see the code in action: install the code and the prerequisite libraries as explained in the *install section*; then follow commands in the *test section* to create a test portal.

1.5 EDB portal

This software is used to run the portal https://coldoc.sns.it that serves a document containing math exercises (nicknamed *EDB*)

1.6 Getting help

To get more help:

coldoc.staff@sns.it

CHAPTER 2

Installing

The following instructions are for people running Debian/Ubuntu, and the *bash* shell. Other operating system may need some adjustments.

Download the latest code from GitHub

```
cd /home/.../.../somewhere
git clone https://github.com/mennucc/ColDoc_project
cd ColDoc_project
export COLDOC_SRC_ROOT=`pwd`
```

the last command sets the environmental variable *COLDOC_SRC_ROOT* to the directory where the code was downloaded. This is fundamental in the following. In this section, we will assume that the *CWD* (current working directory) is *COLDOC_SRC_ROOT*.

2.1 venv

Note that ColDoc needs *Python3* ; you may want to set up a virtualenv, so that Python3 is the default Python.

```
python3 -m venv venv
source venv/bin/activate
```

2.2 Prerequisites

ColDoc has some prerequisites: *Django* (version 2 or 3), *plasTex* (a patched version, see below), and others, as explained later.

To install them (but for plastex) you may use

```
pip3 install django BeautifulSoup4 pycountry django-guardian django-allauth django-
↪background-tasks django-select2
```

(only the first three are strictly needed, the others can be used to activate advanced features, as explained below)

2.3 Installing plasTex

Installing *plastex* is somewhat complex, since ColDoc needs a patched version.

The script *plastex/prepare.sh* can download and patch plastex for you: the patched version is then available in *plastex/plastex*. So you can install it, using *python3 setup.py install* inside the directory *plastex/plastex*.

2.4 Fix PdfLaTeX

Some TeX/LaTeX versions, by default, mangle the tags in the output PDF; then the cross-referencing machinery in ColDoc will not work.

To solve this problem, you should edit the file */usr/share/texlive/texmf-dist/dvipdfmx/dvipdfmx.cfg* and change *%C 0x0000* to *%C 0x0010*.

You may use the patch *patch/texmf.patch* for this.

Note that this file is not marked as a *configuration file* in Debian/Ubuntu, so it would be overwritten if the package *texlive-base* is upgraded; to avoid this problem, you may want to run (as *root* user)

```
dpkg-divert --add --rename /usr/share/texlive/texmf-dist/dvipdfmx/dvipdfmx.cfg
cp -a /usr/share/texlive/texmf-dist/dvipdfmx/dvipdfmx.cfg.distrib  /usr/share/texlive/
↪texmf-dist/dvipdfmx/dvipdfmx.cfg
patch  /usr/share/texlive/texmf-dist/dvipdfmx/dvipdfmx.cfg ${COLDOC_SRC_ROOT}/patch/
↪texmf.patch
```

Alternatively, you may add

```
\ifplastex\else
\special{dvipdfmx:config C 0x0010}
\special{xdvipdfmx:config C 0x0010}
\fi
```

to the preamble of all LaTeX documents.

CHAPTER 3

Testing

To test the code, you may use the tests in the *test* directory (go there and type *make* to see a list).

```
cd test
make
```

For example the commands

```
cd test
make clean
make django_deploy
make django_paper
make django_tasks &
make django_run &
```

will *blobify* the test document from *${COLDOC_SRC_ROOT}/test/paper/paper.tex* and create a coldoc called *paper*; then it will start all needed processes.

The data for the coldoc *paper* will be stored in *${COLDOC_SRC_ROOT}/test/tmp/test_site/coldocs/paper/blobs* so you can open the main file *${COLDOC_SRC_ROOT}/test/tmp/test_site/coldocs/paper/blobs/main.tex* with an editor, or compile it with *pdflatex* ; otherwise you can access the web portal at *http://localhost:8000*. and edit it thru the web interface. (Usernames and passwords for interacting with the test web server are printed when issuing *make django_deploy*) Note that if you edit the latex files on disk, then you will need to issue some commands to keep web interface in sync: see the section on *editing*

3.1 Blobify

Or you may want to *blobify* a document without using the Django web interface, just to see what it looks like. Create a temporary directory

```
tmpdir=$(mktemp  -d)
```

Then *blobify* the example document from the source into the temporary directory

```
python3 ${COLDOC_SRC_ROOT}/ColDoc/blob_inator.py --coldoc-nick=test --blobs-dir=$
↪{tmpdir} --ZS  --SAT  ${COLDOC_SRC_ROOT}/test/latex/latex_test.tex
```

Then open the main blob with an editor

```
editor ${tmpdir}/main.tex
```

CHAPTER 4

Deploying

Here we explain how to bootstrap a new ColDoc *web portal*, or *site*.

We assume that the ColDoc source code was downloaded in the directory whose name is saved in the environment variable COLDOC_SRC_ROOT. For details see the *install section*

ColDoc keeps a strict separation between *code* and *data*.

The same *code* directory can be used to run many *sites*.

In turn, each *site* can host many *documents*.

In the *install section* we installed the *code*. Here we will prepare the structure for the data and settings for a *portal*.

To start a new ColDoc site, you need to setup a directory containing some files. This process is called *deploying*. The directory name must be saved in the *COLDOC_SITE_ROOT* environmental variable.

You need to use a terminal where you can insert shell commands.

4.1 Serving with Apache

To serve the portal using Apache2 in Debian or Ubuntu, you may install the packages

```
sudo apt install apache2 libapache2-mod-wsgi-py3
```

It is advisable to put the portal under */var/www* (or otherwise, you should edit */etc/apache2/apache2.conf* otherwise *apache* will not serve your content). Here is an example shell code:

```
export COLDOC_SITE_ROOT=/var/www/test_site
sudo mkdir ${COLDOC_SITE_ROOT}
sudo chown owner.group ${COLDOC_SITE_ROOT}
```

where *owner.group* is who is performing the install.

4.2 Serving without Apache

If you want to run the portal by some other means (there are many ways to deploy Django, see here) then you may setup the test site anywhere, let's say */home/.../test_site* . Make sure that this directory is empty, and set its name in an environ variable as follows.

```
export COLDOC_SITE_ROOT=/home/.../test_site
mkdir ${COLDOC_SITE_ROOT}
```

4.3 Deploying the skeleton

In the following you may omit the part *${COLDOC_SRC_ROOT}/* if you are sure that the current working directory of the shell is the directory where the ColDoc source code is located.

This command will create the structure for the new ColDoc portal

```
python3 ${COLDOC_SRC_ROOT}/ColDocDjango/helper.py  deploy
```

In particular it will deploy the *config file* for the new document as ${COLDOC_SITE_ROOT}/config.ini. This contains some fundamental settings for the site, and it can also be used to activate/deactivate special features for the portal, such as: social authentication, background tasks, comments, *etc*. Edit it at taste.

4.4 Local variables

There are many settings for a Django portal (the *config.ini* file will setup only the most important ones).

At startup, Django reads a *settings.py* file. In this case, settings for a deployed site are read from three files:

- the general file *${COLDOC_SRC_ROOT}/ColDocDjango/settings.py* in the ColDoc code
- *${COLDOC_SRC_ROOT}/ColDocDjango/settings_local.py* if it exists
- ***${COLDOC_SITE_ROOT}/settings.py* from the *COLDOC_SITE_ROOT* directory where the** web site is deployed.

Each one overrides the previous.

To better test the code, you may want to create a file *${COLDOC_SRC_ROOT}/ColDocDjango/settings_local.py* to setup some variables to enable email sending, as in this example. Or you may want to enable them in *${COLDOC_SITE_ROOT}/settings.py* for your specific site.

```
MAIL_HOST = "smtp.server"
EMAIL_PORT = "587"
EMAIL_HOST_USER = "username"
EMAIL_HOST_PASSWORD = "password"
EMAIL_USE_TLS = True
DEFAULT_FROM_EMAIL = "Helpdesk <helpdesk@that_email>"
```

or to enhance the code, *e.g.* adding some mimetypes used in your *coldoc* s

```
import mimetypes
# https://bugs.freedesktop.org/show_bug.cgi?id=5455
for j in ('.gplt','.gnuplot'):
    mimetypes.add_type('application/x-gnuplot',j)
```

See in *${COLDOC_SRC_ROOT}/ColDocDjango/settings_suggested.py* for more examples.

4.5 Social auth

If you wish to use social authentication, you may set *use_allauth* to True in *${COLDOC_SITE_ROOT}/config.ini* and install *django-allauth*

Note that once you set 'use_allauth' to True, you cannot change it back to 'False'.

In particular, you may add stanzas for *django-allauth* in ${COLDOC_SITE_ROOT}/settings.py such as providers and configs, something like

```
INSTALLED_APPS += [
        'allauth.socialaccount.providers.google']
SOCIALACCOUNT_PROVIDERS = {
    'google': {
        'SCOPE': [
            'profile',
            'email',
        ],
        'AUTH_PARAMS': {
            'access_type': 'online',
        }
    }
}
```

and don't forget to connect to the *admin* interface and to create a *social application* in the database, that contains all credentials (in the above case, for Google OAuth2).

See django-allauth docs for more details

Moreover you may need to setup the Django smtp machinery, to send emails (emails are sent automatically to verify emails addresses or reset passwords).

4.6 Late adding of social auth

If you did not turn *social authentication* on at first, you may turn it on later, by following the above instructions; and then you have to run

```
python3 ${COLDOC_SRC_ROOT}/ColDocDjango/manage.py migrate
python3 ${COLDOC_SRC_ROOT}/ColDocDjango/manage.py collectstatic
```

to update the databases.

4.7 Initalize

Then initialize *django* for your deployed site

```
python3 ${COLDOC_SRC_ROOT}/ColDocDjango/manage.py migrate
python3 ${COLDOC_SRC_ROOT}/ColDocDjango/manage.py collectstatic
```

4.8 Add test material

To test the portal we may populate it with the test LaTeX document.

Before we create some fake users, to be able to interact with the portal.

```
python3 ${COLDOC_SRC_ROOT}/ColDocDjango/helper.py  create_fake_users
```

(The list of users and passwords will be printed on terminal)

We insert the test LaTeX document in the portal. Note that *jsmith* is the author of all blobs, and will have special access rights; similarly *ed_itor* is the editor, and will have access to some administrative information in the coldoc main page.

```
python3 ${COLDOC_SRC_ROOT}/ColDocDjango/blob_inator.py --coldoc-nick=test --ZS --
↪editor=ed_itor --author=jsmith  --SP --SAT    ${COLDOC_SRC_ROOT}/test/paper/paper.
↪tex
```

Then you should generate all PDF and HTML associated to the test paper

```
COLDOC_URL="http://localhost:8000/UUID/test/"
python3 ${COLDOC_SRC_ROOT}/ColDocDjango/latex.py --coldoc-nick=test --url-UUID=$
↪{COLDOC_URL}  all
```

(The command line option *–url-UUID* is needed so that the hyperlinks inside the PDF version will point to the correct URL)

4.9 Activate the Apache portal

If you are preparing the web site to be served by Apache2, you should

```
sudo chown -R www-data:www-data ${COLDOC_SITE_ROOT}
```

otherwise Apache will not be able to access it. Then set up Apache as follows:

```
sudo cp ${COLDOC_SITE_ROOT}/apache2.conf /etc/apache2/sites-available/test_site.conf
sudo a2ensite test_site
sudo systemctl reload apache2
```

4.10 Serve without Apache

Start the simplest Django server and access the portal

```
python3 ${COLDOC_SRC_ROOT}/ColDocDjango/manage.py  runserver 8000
firefox http://localhost:8000/
```

Note that in this case *django* will not serve the static files, unless you set *debug* to *True* in *${COLDOC_SITE_ROOT}/config.ini*.

4.11 Final remarks

ColDoc keeps a strict separation between *code* and *data*. You may even install the code using an account, let's say *coldoc_sw*, then deploy a portal, and assign all the files in the portal to a different user, let's say *coldoc_data*: in this case you need to tell Apache about this change, by adding the *user* and *group* directives in the line starting as *WSGIDaemonProcess*, as follows

```
WSGIDaemonProcess coldoc.group python-home=/...virtualenv.... python-path=${coldoc_
↪src_root}  locale=en_US.UTF-8  lang=en_US.UTF-8 user=coldoc_data group=coldoc_data
```

This may improve security.

CHAPTER 5

Blob and UUID

5.1 Tree of UUIDs

Any content in ColDoc is identified by an UUID, an unique identifier.

Each UUID has associated to it a list of metadata (see next section).

The ColDoc is a tree of UUIDs, connected by a parent-children relationship.

There is a special UUID called *root_uuid* usually 001. It is the root of the tree. (The *root_uuid* may be changed, it is stored as field *root_uuid* in the *DColDoc* model, but this is untested and may break the portal.)

5.2 Relationship between blobs and UUID

Any blob is identified by an UUID.

Vice versa, an UUID may refer to many blobs that have the same semantic content but are available in

- different langages (English, Italian..) and/or
- different content type (LaTeX, HTML, PDF, JPEG ...).

All these are blobs that are referred by the same UUID.

The list of languages is stored in the metadata *lang* , the list of content types is stored in *extension* (as filename extensions). (See next section).

The author can enter in the ColDoc system translations of a LaTeX blob in different languages; and can upload the same picture/graphic in different formats. (But this is still mostly TODO).

Currently the code is designed in this way:

- if the blob contains LaTeX then the only extension is *.tex* and there may be multiple languages;
- if the blob contains a LaTeX package then the only extension is *.sty* and the list of languages is empty;
- if the blob contains a LaTeX bibliography then the only extension is *.bib* and the list of languages is empty;

- all other cases are graphical blobs: the list of extensions explains all available content type; the list of languages is empty. (TODO it may be useful to have a graphical file available in different languages)

5.3 Blobs and views

The ColDoc portal also will convert the *blobs* into *views*: for each UUID (but not the *root_uuid*) that contains LaTeX, it will convert LaTeX to PDF and HTML; (TODO it may also convert images to different formats). This *view* contains only the material of that blob.

The ColDoc portal also will convert the entire document tree in a *main* view, available in PDF and HTML. The *main* view is internally associated to the *root_uuid*.

There are two versions of the *main* view.

- a version containing all the material, visible to *editors*; this *main* view is stored in the directory *blobs/UUID/0/0/1* of the root uuid;
- a reduced version, containing only the *public* and *open* parts; this is visible to anybody. (See the section on permissions). This reduced version view is stored in the directory *anon/UUID/0/0/1*.

For graphical content, there is no much difference between *blobs* and *views*, so an user that has *view_view* access will be able to view the blobs. (The precise definition of *graphical content* is encoded in *ColDoc.utils.is_image_blob*)

CHAPTER 6

Metadata

Here we explain all metadata that may be associated to blobs, and their meaning.

As explained in the previous section, the metadata is associated to the UUID (and not to the specific blob, as specified by language and file type)

6.1 Environment

Before proceeding, though, we clarify what we mean by *environment*.

LaTeX uses environments to delimit text elements, as in this example

```
\begin{Theorem}
  The hypothesis implies the thesis.
\end{Theorem}
```

Internally *ColDoc* identifies such environment as *E_Theorem* . The prefix *E_* helps identifying environments, and avoiding name collisions.

By passing the option *–split-environment environment* to *blob_inator*, you may specify which environments to split.

For example, *E_document* is the part between *\begin{document}* and *\end{document}*; note that this blob is always splitted since the option *–split-environment document* is already present as default into *blob_inator*.

ColDoc uses other *environments* :

- *main_file* is the main blob, the root of the tree
- *preamble* is the preamble, that is the part between *\documentclass* and *\begin{document}* ; this blob is always splitted, unless he argument *–dont-split-preamble* is passed to *blob_inator* (but this may break some parts of the portal).
- *input* or *include* are used for blobs that contain text from a LaTeX file that was include using *\input* or *\include*
- *input_preamble* is used for blobs that contain code from a LaTeX file that was include using *\input* while inside the preamble

- *usepackage* is used for blobs that contain packages; these are copied if found in the same directory of the main file
- *bibliography* is used for blobs that contain bibliography, as specified by the \bibliography command
- *section* is used for sections
- *paragraph* is used for long paragraphs of text (as specified by the *–split-paragraph* option)
- ***graphic_file* is used for blobs containing images (usually inserted using \*includegraphics*** or other commands specified with the option *–split-graphic* of *blob_inator*)

6.2 Metadata key list

This is the list of all keys in the metadata storage, and the meaning of their values. Note that a key may be repeated multiple times.

These keys are *static* : they are instantiated when the blob is first added to the tree (e.g. by using *blob_inator*), but are not changed when the blob content is subsequently edited.

- *coldoc* , the nickname of the ColDoc that this blob is part of
- *environ* , the value is the environ that contained this blob . See the previous section for details.
- *optarg* , the optional argument of the environment, as in this example.

```
\begin{Theorem}[Foobar's theorem]
  The hypothesis implies the thesis.
\end{Theorem}
```

  where the *optarg* would be equal to *Foobar's theorem*.

- *lang* , the languages available for this blob; more than one language may be available.
- *extension* , the extentions available for this blob; more than one extension may be available, for example a graphical file may be available a *.jpeg* and *.svg*. For blobs containined LaTeX, only *.tex* is allowed.
- *author* the list of people that contributed to this blob (this does not distinguish if somebody contributed only to a certain language version).
- ***original_filename* , the filename whose content was copied in this** blob (and children of this blob) by *blob_inator*; the extension of the filename (if any) is stripped; the path is not absolute, but is relative to the directory where the main LaTeX file was located.
- *uuid* , the UUID of this blob
- *parent_uuid* , the UUID of the parent of this blob; all blob have one, but for the blob with *environ=main_file*
- *child_uuid* , the UUID of the children of this blob; there may be none, one, or more than one
- *access* can be *open* , *public* or *private* . See the section on permissions.
- *creation_date*
- *modification_date* ; this is updated when the blob content is edited (this does not distinguish which language version was edited).
- ***latex_date* ; this is updated when the view (html and pdf) of this blob was last compiled** (this does not distinguish which language version was edited - the system automatically recompiles the language last edited).

These keys are derived from the content of the blob. Any direct change to this database would be lost as soon as the blob is changed. (In Django, they are stored in a SQL database for convenience; this database is called *ExtraMetadata*.)

- *M_* followed by a *name* that was provided as *–metadata-command name* . E.g. if *blob_inator* was invoked with the command

  ```
  blob_inator --metadata-command label --split-environment Theorem
  ```

  to parse this input

  ```
  \begin{Theorem}\label{tautol}
    The hypothesis implies the thesis.
  \end{Theorem}
  ```

  then the metadata for that blob would contain *environ=E_Theorem* and *M_label={tautol}*

- *S_* followed by an environment and then followed by *_M_name* ; this is used by metadata extracted from environments that are deeper in the tree than the current blob, but that are not splitted in a child blob. As in this example:

  ```
  blob_inator --metadata-command label --split-environment Theorem
  ```

  to parse this input

  ```
  \begin{Theorem}\label{tautol}
    The hypothesis implies the thesis.
    \begin{equation}\label{eq:forall}
      \forall x
    \end{equation}
  \end{Theorem}
  ```

  then a blob will contain this Theorem, and its metadata would contain *M_label={tautol}* and *S_E_equation_M_label={eq:forall}*

6.3 Metadata in source code

Metadata is represented and operated on by a Python Class.

The class interface is described as the base class *MetadataBase* in *ColDoc.classes*

This interface is implemented in the *FMetadata* class, that stores metadata in a file (this is independent of Django); and *DMetadata*, that stores metadata in the Django databases.

To write code that works with both implementations, it is important to use the *get* method, that always returns a list of values (even for properties that are known to be single valued).

The keys *coldoc*, *uuid*, *environ* are known to be single valued, and for convenience there is a Python *property* that returns the single value (or *None*).

Note that in *DMetadata* some objects are not strings:

- *author* is a *models.ManyToManyField* on the internal *User* class
- *coldoc* is a *models.ForeignKey* on the *DColDoc* model.

CHAPTER 7

UUID Permissions

(See in ColDocDjango/users.py for more details).

There is a list of permissions for each UUID. Currently it is: ’view_view’, ’view_log’, ’view_blob’, ’change_blob’, ’download’, ’commit’, ’view_dmetadata’, ’change_dmetadata’

Permissions are associated to the UUID of the blob, so they are the same for all languages and/or content types. (Internally, they are associated to the *DMetadata* class).

Inside Django, the complete name of such permissions is of the form *UUID.name*.

7.1 Permissions for a specific coldoc

For each permission above of the form *aaaa_bbbb* and any *coldoc* with nickname *cccc* there is also a permission *aaaa_bbbb_on_blob_inside_cccc*, that is specific for that coldoc.

- An user that has permission *aaaa_bbbb* automatically has permission *aaaa_bbbb_on_blob_inside_cccc* for any UUID in any coldoc.
- An user that has permission *aaaa_bbbb_on_blob_inside_cccc* automatically has permission *aaaa_bbbb* for any UUID in the coldoc with nickname *cccc*.

7.2 Special users

An author of a blob has all the above permissions for that blob.

An anonymous user (an user that accesses the portal and is not authenticated) has very limited permissions: s/he has the *view_view* permission only if the coldoc has the *anonymous_can_view* flag set to True, and the blob the UUID *access* state is *open* or *public*.

7.3 Meaning of permissions, and rules

This is the Permissions meaning and rule for each *UUID*.

(Recall that each UUID has an *access* metadata that can be *open* , *public* or *private*.)

- *view_view* : permission to view a a *view* (a representation of the blob, as a html or PDF). If the UUID *access* state is
  - ***open* or *public*, this is always granted to authenticated users; and** granted to anonymous users if the property *Anonymous can view* is set in the coldoc settings (an editor can change it from the main web page for the coldoc)
  - *private* , it is granted to the author or any user with *view_view* permission
- *view_blob* : permission to view real content of the blob. If the UUID *access* state is
  - *open* this is always granted to authenticated users.
  - *private* or *public* , it is granted to the author or any user with *view_blob* permission
- *download* : permission to download the content of this blob in nice formatted ways. If the UUID *access* state is
  - *open* this is always granted to authenticated users.
  - *private* or *public* , it is granted to the author or any user with *download*.

  Note that the *download* url also requires *view_view* permission.
- *view_log* : permission to view logs created by *LaTeX and plastex*

7.4 Local permissions

The *ColDoc* portal uses the *django-guardian* library, so that a specific permission can be given to an user for only one object.

Note that if the user has a certain permission for the whole coldoc, than it has that permission for any object in that coldoc. This only holds for permissions listed above (those associated to the *DMetadata* class, that start with *UUID.*).

7.5 Buying local permissions

There is a provision so that an user can buy certain permissions using *eulercoins*. For this, the library *django-wallet* must be installed (a special version, available on demand); then a function *PRICE_FOR_PERMISSION* must be defined in the *settings* file (an example is in the *settings_suggested.py* file): given a *user*, a *blob* (an instance of *DMetadata*) and a *permission*, the function will decide if the user can buy that permission for that object, and the price.

Note that an user must have *operate* permissions on *wallet* objects to buy something.

7.6 Access to protected content in the whole document

As aforementioned, the LaTeX data is stored on disk inside a *blobs* directory tree.

Two versions of the whole document are generated, one from the *blobs* tree, and in this case the generate document (both HTML and PDF) will contain all the material: this is the *private* version of the document.

Another version is from the *anon* tree. The *anon* tree is automatically generated as a copy of the *blobs* tree where all material with *access* set to *private* will be masked out. This is the *public* version of the whole document.

7.7 LaTeX macros

In the coldoc metadata there are three keys: *latex_macros_private*, *latex_macros_public* and *latex_macros_uuid*. These contain LaTeX macros.

When compiling the *private whole document* the *latex_macros_private* is automatically insert just after the *document-class* ; the *latex_macros_public* when compiling *public whole document*; the *latex_macros_uuid* when compiling one single blob in one UUID

The defaults are:

- *latex_macros_private* defaults to

  ```
  \newif\ifColDocPublic\ColDocPublicfalse
  \newif\ifColDocOneUUID\ColDocOneUUIDfalse
  ```

- *latex_macros_public* defaults to

  ```
  \newif\ifColDocPublic\ColDocPublictrue
  \newif\ifColDocOneUUID\ColDocOneUUIDfalse
  ```

- *latex_macros_uuid* defaults to

  ```
  \newif\ifColDocPublic\ColDocPublicfalse
  \newif\ifColDocOneUUID\ColDocOneUUIDtrue
  ```

Note that *ifColDocPublicfalse* is used when compiling each single blob by itself: this makes sense since in this case the web interface will make sure that only authorized users can access the content.

The value of these macros can be used to trigger different behaviours in the preamble and in the document.

7.8 Accessing the whole document

The whole document can be accessed using buttons *View whole document* and *View whole document, as PDF* in the main page of the coldoc.

These buttons will serve either the *private* or the *public* version.

If the user is an *editor*, or s/he has the *view_view* permission, then the content served from the buttons is the *private* version (compile from the material inside the *blobs* directory); note that in this case the HTML pages use a green theme, to distinguish; otherwise it is the the *public* version (compile from the material inside the *anon* directory); so that the generic user will not see the protected content; in this case the HTML pages use a blue theme, to distinguish.

Note that an user that is an *author* but not an *editor* will not see the protected content in the whole document: indeed it is not sensible to generate different whole document representations for each and any user.

CHAPTER 8

ColDoc Permissions

(See in ColDocDjango/users.py for more details).

There is a list of permissions for each ColDoc. Currently it is: ’add_blob’, ’delete_blob’, ’commit’, ’view_dcoldoc’, ’change_dcoldoc’

Inside Django, the complete name of such permissions is of the form *ColDocApp.name*.

8.1 Meaning of permissions, and rules

This is the Permissions meaning and rule for some of the above.

- ***add_blob*** [if an user has permission *add_blob* for the whole ColDoc,] and has permission *view_blob* for a specific UUID, then s/he can add a children UUID to that UUID. Moreover the author of a blob can always add children to that blob (unless *author_can_add_blob* flag is turned off in the ColDoc settings).

CHAPTER 9

Groups

For each coldoc *cccc* two groups are created, one named *coldoc_cccc_group_authors* and one *coldoc_cccc_group_editors*; *coldoc_cccc_group_authors* has all *UUID* permissions for that coldoc; *coldoc_cccc_group_editors* has all *ColDocApp* permissions for that coldoc, and also *UUID.view_view*, *UUID.view_log*.

CHAPTER 10

Blobify

This section explains how to import a LaTeX document into a ColDoc portal.

Since the portal will use *plastex* to convert LaTeX to HTML, and *pdflatex* with the *standalone* package for compact PDF representation of UUIDs, some changes have to be made.

Following are instructions, and you may also want to see in the directory *test/paper* for a complete working example.

In your LaTeX document, you should set the language as

```
\usepackage[english]{babel}
```

and not as

```
\documentclass[english]{article}
```

Then, right after the *documentclass* statement, add

```
\newif\ifplastex\plastexfalse
\ifplastex
\newif\ifstandalone\relax\standalonefalse\relax
\else
\usepackage{standalone}
\fi
```

This will load the package *standalone* only when compiling with standard LaTeX; it will also define the conditional *ifstandalone* to be true only when compiling a PDF in standalone mode.

10.1 Plastex tweaks

Then wrap all code that is not compatible with plastex (code that sets fonts etc etc) as follows

```
\ifplastex\else
 % set fonts ...
\fi
```

Also, you will have to replace some packages that do not work well with *plastex*, as in this example

```
\ifplastex
% plastex does not know of these
\def\eqref{\ref}
\fi
%
\ifplastex
% https://github.com/mathjax/MathJax/issues/1081
\def\sfrac{\frac}
\else
\usepackage{xfrac}
\fi
%
\ifplastex
% plastex does not know varioref
\def\vref{\ref}
\def\vpageref{\pageref}
\else
\usepackage{varioref}
\fi
```

See plastex docs for details

10.2 Standalone tweaks

You should also wrap all the code that modifies page geometry so that it is ignored in standalone mode, as in this example:

```
\ifplastex\else\ifstandalone\else
\usepackage[margin=18ex,headheight=16pt]{geometry}
\usepackage{fancyhdr}
\pagestyle{fancy}
\fi\fi
```

See standalone docs for details

10.3 Multiple LaTeX format

It is possible to prepare a LaTeX document that can be compiled using different engines: *pdflatex*, *xelatex* or *lualatex*

To this end, you should install the latest version of the *iftex* package from https://www.ctan.org/pkg/iftex

Then add a snippet in the document preamble as follows:

```
\usepackage{iftex}
%%%%%%%%% use conditionals to load some engine-specific packages
\ifplastex
 % code for plastex
 \newcommand\mathbbm[1]{{\mathbb{#1}}}
\else\iftutex
% code for xetex or luatex
  \input{preamble_xelatex}
```

```
\else
 % code for standard (pdf)latex
   \input{preamble_pdflatex}
\fi\fi
```

Then put in the file *preamble_xelatex.tex* all commands to setup fonts for *xelatex* or *lualatex*; and similarly in *preamble_pdflatex.tex* for *pdflatex*.

10.4 Downloading, and compiling single UUIDs

You should also move all your favorite customizations in a file *preamble_definitions.tex*

- loading of packages such as *amsthm*, *amsmath*
- definitions for theorems and such
- personal macros
- ...etc...

There is a provision in the *portal* so that users may download the LaTeX of a single UUID: the portal will add enough LaTeX code so that it will be possible to compile that UUID; so it will add to the bundle

- *preamble_pdflatex.tex* or *preamble_xelatex.tex*, for document-related definition;
- that file *preamble_definitions.tex* so that the user will have a copy of all the needed macros and definitions,

to be able to compile that blob.

10.5 Check it all

Check that the document compiles fine to HTML by invoking PlasTeX on your document using

```
plastex -d output_html document.tex
```

(it is recommended that you use the *plastex* version that was installed *in the installation phase*)

And check that it still compiles fine with standard *pdflatex*

Then try to import in a test portal. Setup the test portal as follows

```
export COLDOC_SITE_ROOT=${COLDOC_SRC_ROOT}/test/tmp/test_site
cd  ${COLDOC_SRC_ROOT}
make -C test clean
make -C test django_deploy
```

Then try to import your document in the portal

```
ColDocDjango/blob_inator.py --coldoc-site-root ${COLDOC_SITE_ROOT} --coldoc-
↪nick=testdoc --ZS --SAT  --split-sections --editor=ed_itor --author=jsmith  yourdir/
↪yourdocument.tex
```

note that:

- if your document best compiles with a specific engine, use the *–latex-engine* option of *blob_inator.py* to specify which;

- if you use non-standard commands to display images, add them to the command line options for *blob_inator.py* as *–split-graphic mypicturecommand*. (Warning: it is assumed that *mypicturecommand* uses the same syntax of *includegraphics*).

Then check if the document can be compiled

```
ColDocDjango/latex.py --coldoc-site-root ${COLDOC_SITE_ROOT} --coldoc-nick=testdoc --
↪url-UUID="http://localhost:8000/UUID/testdoc/" all
```

and eventually run the test portal

```
make -C test django_run &
```

and access the web portal at *http://localhost:8000*.

Try authenticating using the different users (see the output of *django_deploy* for usernames and passwords).

Check that everything looks fine.

Check in particular that images were imported correctly.

If you are not satisfied, or if something fails:

- tweak your document,
- try different command line options for *blob_inator.py*

If the result is satisfactory enough, that is, if only small changes are needed, you can also change the document *inside the portal* by editing the files inside *${COLDOC_SITE_ROOT}/coldocs/testdoc/blobs/*. Note that the data structure can be compiled from the command line, using

```
cd ${COLDOC_SITE_ROOT}/coldocs/testdoc/blobs/
pdflatex main.tex
plastex -d main_html main.tex
```

CHAPTER 11

Editing

There are many tools to operate on the coldoc; most have a *command line* and a *web* interface as well.

Command line tools have many options (not documented here), see respective *–help*.

One useful operation is to add new nodes to the tree of blobs. From command line,

```
python3 ${COLDOC_SRC_ROOT}/ColDocDjango/helper.py  add_blob
```

If you edit the blobs directly in the filesystem, and not using the web interface, then the django database will be desyncronized regarding metadata: run

```
python3 ${COLDOC_SRC_ROOT}/ColDocDjango/helper.py   --coldoc-nick NICK reparse_all
```

Moreover from time to time you will need to recreate the PDF and HTML representations.

Use

```
python3 ${COLDOC_SRC_ROOT}/ColDocDjango/latex.py   --coldoc-nick NICK main_private
```

to recreate the complete HTML PDF (visible only to editors); use

```
python3 ${COLDOC_SRC_ROOT}/ColDocDjango/latex.py   --coldoc-nick NICK main_public
```

to recreate the public HTML PDF (visible only to everybody); use

```
python3 ${COLDOC_SRC_ROOT}/ColDocDjango/latex.py   --coldoc-nick NICK tree
```

to recreate the HTML PDF for each blob (this is useful if you edited many blobs in the filesystem); use

```
python3 ${COLDOC_SRC_ROOT}/ColDocDjango/latex.py   --coldoc-nick NICK all
```

to run all of the above.

Note that when you edit a blob using the web interface, it is automatically reparsed and HTML and PDF are recomputed; but the *private* and *public* complete documents are not recompiled automatically, you have to either use the command above or the button in the web interface (visible only to editors).

CHAPTER 12

Indices and tables

- genindex
- modindex
- search