Annotamentum Documentation

Release 5

Elmar Bucher

Oct 31, 2019

Contents

2014
09
An!
Annotamentum

Django admin based sample, reagent and experiment metadata tracking.

Summary: Annot is a web application, developed for biological wetlab experiment layout, sample and reagent logging, so that data is ready for sharing and analysis. On its core annot makes use of the acpipe_anjson library and acjson - assay coordinate json - file format. The use of controlled vocabulary from ontologies for sample and reagent annotation is enforced. Annot's modular implementation can be adapted to a variety of experimental paradigms.

Implementation: The annot web application is implemented in python3 utilizing roughly django, postgresql, nginx. Annot is deployed via docker container platform. Annot code it self is thereby packed into a debian docker container.

Annot was developed using for browser the firefox developer edition. However, all major browsers are supported.

Source code is available under GNU AGPLv3 license. This manual is written under the GNU FDLv1.3 license.

This are **links** to source code and a poster presentation at PyCon 2015 Montreal, Canada at the very beginning of project.

This user manual: is structured in tutorial, how-tos, reference and discussion.

CHAPTER 1

Tutorial

This tutorial will step-by-step guide you through the process of

1. populating annot with backed up controlled vocabulary and update it with the latest ontology version available online.
2. populating annot with backed up sample and reagent bricks.
3. populating annot with backed up study and investigation information.
4. populating annot with backed up experiment layouts and layout one acjson file yourself.
5. populating annot with backed up tracking information.
6. backup the work done.

1.1 Preparation

1. Before you follow this tutorial you have to install the development version of annot as described in *HowTo install annot*.
2. run `git clone https://gitlab.com/biotransistor/annotTutorial.git` to clone the tutorial material to you machine.
3. run `cp -r annotTutorial annot/web/` to copy the cloned annotTutorial folder into the annot/web/ folder from your annot installation.
4. run `rm -fr annot/web/annotTutorial/.git` to remove the .git folder in the copied annotTutorial folder.

1.2 Controlled Vocabulary

1. Enter annot
    1. `docker-machine ls` lists all the installed docker machines.
    2. `docker-machine start an0` starts the an0 docker machine, if needed.
    3. `eval $(docker-machine env an0)` loads the an0 environment variables.
2. run `docker exec -ti annot_webdev_1 /bin/bash` enter the annot_webdev_1 docker container
    1. `ls` should list among others the annotTutorial folder.
3. Load the backed up vocabulary.
    1. `cp annotTutorial/vocabulary_json/* ../media/vocabulary/backup/` copies the backed up vocabularies to the right place inside annot.
    2. `python manage.py vocabulary_loadbackup` will populate each vocabulary app first with the latest backup found at /usr/src/media/vocabulary/backup/‘, then, it will download the latest ontology version, if the online version is newer than the version already in the database, and update the database content with it.

If you get a `urllib.error.HTTPError: HTTP Error 401: Unauthorized` error, then your `APIKEY_BIOONTOLOGY` credential inside `annot/web/prjannot/crowbar.py` will most probably be wrong.

Now, let’s find out with which ontologies and version annot’s vocabularies were populated.

1. point your browser to `http://192.168.99.100/admin/` and login with your credentials.
2. click the red colored `Sys admin ctrl vocabularies` link. A table should pop up which lists all vocabularies and the information we were interested in.

1.3 Bricks

1. `python manage.py loaddata annotTutorial/brick_tsv/person_brick_20180331_235444_oo.json` loads the person brick into the database. The person brick is special kind of brick that doesn’t annotate data. The person brick is used to annotate the responsible person for each sample and reagent bricks. This is the reason that it is re-loaded a bit differently than the other bricks.
2. let’s have a look at the upload person bricks.
    1. point your browser to `http://192.168.99.100/admin/` and login with your credentials.
    2. click the yellow colored `Staff` link. A table should pop up, displaying the uploaded bricks.
3. `python manage.py antibody1_tsv2db annotTutorial/brick_tsv/antibody1_brick_20181024_003732_human.txt` load the primary antibody bricks.
    1. let’s have a look at the upload primary antibody bricks. Click the orange colored `Endpoint Primary Antibody` link to retrieve the database table.
4. `python manage.py antibody2_tsv2db annotTutorial/brick_tsv/antibody2_brick_20180510_020008_human.txt` loads the secondary antibody bricks.
5. `python manage.py cstain_tsv2db annotTutorial/brick_tsv/cstain_brick_20180503_020012_human.txt` loads the compound stain bricks.
6. `python manage.py compound_tsv2db annotTutorial/brick_tsv/compound_brick_20180511_020009_human.txt` loads the compound bricks.

7. `python manage.py proteinset_tsv2db annotTutorial/brick_tsv/proteinset_brick_20180502_020026_human.txt` loads the protein complex bricks.
8. `python manage.py protein_tsv2db annotTutorial/brick_tsv/protein_brick_20180502_020024_human.txt` loads the protein bricks.
9. The sample bricks are a bit special because of the `sample_parent` field. This means sample bricks might relate to other sample bricks. Because of that we first have to manually generate the `not_available-notavailable_notavailable_notavailable` and `not_yet_specified-notyetspecified_notyetspecified_notyetspecified` sample before we can upload the sample bricks with the usual command. For the same reason, we might have to call the human_tsv2db command several times till all sample bricks are loaded into the database.
   1. click the orange colored `Sample Homo Sapiens` `Add Human Brick` link.
   2. click `Save`. An error message: “`Please correct the errors below.`” will appear.
   3. Scroll down. For every “`This field is required.`” error choose `not_yet_specified`.
   4. when you reached the bottom click `Save`. This should bring you back to the database table. sample `not_yet_specified-notyetspecified_notyetspecified_notyetspecified` should now exist.
   5. click `not_yet_specified-notyetspecified_notyetspecified_notyetspecified`
   6. change `Sample`, `Provider`, `Provider_catalog_id`, `Provider_batch_id` to `not_available`.
   7. click `Save`. Sample `not_available-notavailable_notavailable_notavailable` should now exist too.
   8. now run command `python manage.py human_tsv2db annotTutorial/brick_tsv/human_brick_20180615_020026_human.txt`, if needed several time, until all sample bricks are loaded.

   Now, all backed up bricks should be loaded.
10. Let’s have a look how this brick information can be downloaded. All brick information can primary be downloaded in json and tsv format.
    1. click the orange colored `Perturbation_Protein` link.
    2. In the `Action` drop down list choose `Download protein page as as json file` and click `Go`.
    3. In the `Action` drop down list choose `Download protein page as as tsv file` and click `Go`.

This will download all protein bricks stored in the database once in json and once in tsv format. The download procedure is the same for any brick type (primary antibody, secondary antibody, compound stain, compound, protein, proteinset and homo sapiens sample).

Additionally, you can download related bricks form the investigation, study, assay run and acaxis layer. From there the bricks are downloadable in annot’s json and tsv standards and additionally in the lincs data standard. Please note: The lincs data standard contains not all stored brick information. Only the fields compatible with the lincs standard. Further, are the bricks re-grouped into an antibody, a small molecules, a protein and a cell line file, this as well because of the lincs standard.

1.3.1 Upload the Bricks to make them accessible in the Experiment Layout Layer!

Before any brick is accessible for experiment layout, it must be uploaded into the corresponding `Uploaded enpoint reagent bricks`, `Uploaded perturbation reagent bricks` or `Uploaded sample bricks` table. The first time after you install annot you have to do this via command line, because the database tables which the GUI relies on have to be initialized. After that you can populate the brick tables via command line or GUI.

On the command line:

1. `python manage.py brick_load` will upload all bricks.

On the GUI:

1. scroll down to the red colored `Appsabrick` block.
2. click the `Sys admin bricks` link. You will find a table with all the brick types and some information about their bricking. This is the place were you from now on can brick bricks by GUI.
    1. choose the brick types you like to brick by clicking the box in front of it.
    2. in the `Action` drop down list choose `Upload brick` and click `Go`.
3. go back to `Home › Appsabrick`
4. click the `Uploaded endpoint reagent bricks` or `Uploaded perturbation reagent bricks` or `Uploaded sample bricks`. This are the tables containing the uploaded bricks. Those are the bricks accessible for layout.

1.4 Investigations and Studies

Let's load from the backup what was stored in the Investigation and Study table under the black colored links.

1. run `docker exec -ti annot_webdev_1 /bin/bash` to enter annot by command line.
2. run `python manage.py loaddata annotTutorial/experiment_json/investigation_investigation_20181024_oo.json`
3. run `python manage.py loaddata annotTutorial/experiment_json/study_study_20181024_oo.json`
4. click on the black colored `Investigation` link to see the reloaded content.
5. click on the black colored `Study` link to see the reloaded content.

1.5 Experiment Layouts

Annot provides you with a super flexible way to layout any biological experiment. In a first step the three major axis from each biological experiment - sample, perturbation, endpoint - are laid out on the `Acaxis` layer. Then, these axes are pulled together on the `Assay run` layer.

For example, let's layout a lapatinib perturbation:

1. click the cyan `Set_of_Perturbation Add_perturbation_set` link.
2. enter the Setname: `ds-lapatinib_750nM_v1`
3. click inside the button next to `Brick:` this is a searchable drop down list. type `lapatinib` into the button. the list will immediately filter as you type. click on `compound-lapatinib_chebi49603-SelleckOwn_S1028_notavailable`. If you can not find lapatinib in the list because the list is still empty, remember that you first have to upload the bricks so

that they become accessible. Do so. Then the brick should appear in the drop down list. In this example we will only layout a lapatinib perturbation. If you would have to layout a 384 well plate with dozens of different perturbations keep on selecting all the reagent bricks you need.

4. click `Save`

Now let's layout the plate design. For this we will generate and edit the acjson template script code.

1. click the cyan `Set_of_Perturbation` `Add_perturbation_set` link.
2. click in the box in front of the `ds-lapatinib750nmv1` row. The box will turn blue and get a tick.
3. in the `Action` drop down list choose `Download selected set's as python3 acpipe template script` and click `Go`.
4. open the downloaded `acpipeTemplateCode_ds-lapatinib_750nM_v1.py` file in a plain text editor

Let's have a look a the generated template file in detail.

The *first part* is the so called header.

```
###
# title:  acpipeTemplateCode_ds-lapatinib_750nM_v1.py
#
# date: 2018-04-02
# license: GPL>=3
# author: bue
#
# description:
#   acpipe script to generate a perturbationset related acjson file.
#   template automatically generated by annot softawre.
#   check out: https://gitlab.com/biotransistor/annot
###
```

The header gives some basic information about the code in a comment section defined by hashtags (`#`).

The *second part* loads the libraries needed to interpret the program beside the python3 standard library. Those libraries are copy, json and acpipe_acjson.

```
# python library
import copy
import json

# acpipe library
# check out: https://gitlab.com/biotransistor/acpipe_acjson
import acpipe_acjson.acjson as ac
```

The *third part* builds an acjson object. Acjson stands for *assay coordinate java script object notation*. Acjson is the format we will use to layout the entire experiment. The acjson format is based on and totally compatible with the json syntax.

```
# build acjson
d_acjson = ac.acbuild(
    s_runid='ds-lapatinib_750nM_v1',
    s_runtype='annot_acaxcis',
    s_welllayout='?x?',
    ti_spotlayout=(1,),
    s_log='from the bricks'
)
```

Notably, in the template code, annot was able to specify `s_runid`, `s_runtype` and `s_log`. The question marks (`?`) in the code reflect that the wellplate layout has not yet been specified. You could for example specify a 384 wellplate (`16x24`) or a 96 wellplate (`8x12`) or petri dish (`1x1`) or a tube (`1x1`). We set `s_welllayout` to `1x1` because we will treat all our wells with the same lapatinib concentration. Annot can handle arrays with multiple spots per well. However, we will leave `ti_spotlayout` by the default which is `(1,)`, since lapatinib not is spotted to the array.

The *fourth part* describes the experimental layout.

```
# reagent: compound-lapatinib_chebi49603-SelleckOwn_S1028_notavailable
s_gent = 'lapatinib_chebi49603'
d_record = {s_gent: copy.deepcopy(ac.d_RECORDLONG)}
d_record[s_gent].update(copy.deepcopy(ac.d_SET))
d_record[s_gent].update(copy.deepcopy(ac.d_EXTERNALID))
d_record[s_gent].update({'manufacture': 'Selleck_Own'})
d_record[s_gent].update({'catalogNu': 'S1028'})
d_record[s_gent].update({'batch': 'not_available'})
d_record[s_gent].update({'conc': ?})
d_record[s_gent].update({'concUnit': 'nmolar_uo0000065'})
d_record[s_gent].update({'cultType': '?'})
d_record[s_gent].update({'timeBegin': ?})
d_record[s_gent].update({'timeEnd': ?})
d_record[s_gent].update({'timeUnit': 'hour_uo0000032'})
d_record[s_gent].update({'recordSet': ['ds-lapatinib_750nM_v1']})
d_record[s_gent].update({'externalId': 'LSM-1051'})
d_acjson = ac.acfuseaxisrecord(
    d_acjson,
    s_coor='?',
    s_axis='perturbation',
    d_record=d_record
)
```

As you con see, annot pulled as much information it could from the setname (recordSet) and the bricks (manufacture, catalogNu, batch, concUnit, timeUnit, externalId). However, we have not yet specified the actual concentration and time and how lapatinib will be provided to the cells (e.g. batch or fed or contiuose). Please set `conc` to `750` nmol, `timeBegin` to `0` hours, `timeEnd` to `72` hours and `cultType` to `batch`. Now, as we said this is a petri dish. Natural the `coor` coordinate will be `'1'`. Notice that the coordinate, even it is an integer, is given as string. This is because coor is a json dictionary key, and json dictionary keys, have to be strings to be compatible with the json syntax.

In the acjson format the coordinate numbering have to starting always by '1' and increases by 1, starting at the upper left corner, track every spot inside a well, well by well, from left to right, from top to bottom.

In the *last part* the acjson object is written into a json file.

```
# write to json file
print('write file: {}'.format(d_acjson['acid']))
with open(d_acjson['acid'], 'w') as f_acjson:
    json.dump(d_acjson, f_acjson, sort_keys=True)
```

Now lets generate and upload the acjson file.

This is python3 code. You will have to install python3 on your computer and an additional python3 library called acpipe_acjson to run this code.

1. How to install `python3` depends very much on the operating system you are running. Install per your system.
2. After you have installed python3, you can install acpipe_acjoson by running `pip3 install acpipe_acjson` from the command line.
3. run `python3 acpipeTemplateCode_ds-lapatinib_750nM_v1.py` to run

the modified template code from the command line. The resulting file's name `annot_acaxis-ds-lapatinib_750nM_v1_ac.json` is constructed out of runtype `annot_acaxis` and runid `ds-lapatinib_750nM_v1`. To study the json file open it in a webbrowser or text editor. You can change the last line of the code from `json.dump(d_acjson, f_acjson, sort_keys=True)` to `json.dump(d_acjson, f_acjson, indent=4, sort_keys=True)` to get better human readable output. However, the resulting file will as well take more disk space then a file without indent. The resulting acjson file will look like this:

```
{
    "1": {
        "endpoint": null,
        "iSpot": 1,
        "iWell": 1,
        "iixii": "1_1x1_1",
        "ixi": "1x1",
        "perturbation": {
            "lapatinib_chebi49603": {
                "batch": "not_available",
                "catalogNu": "S1028",
                "conc": 750,
                "concUnit": "nmolar_uo0000065",
                "cultType": "batch",
                "externalId": "LSM-1051",
                "manufacture": "Selleck_Own",
                "recordSet": ["ds-lapatinib_750nM_v1"],
                "timeBegin": 18,
                "timeEnd": 90,
                "timeUnit": "hour_uo0000032"
            }
        },
        "sample": null,
        "sxi": "Ax1"
    },
    "acid": "annot_acaxis-ds-lapatinib_750nM_v1_ac.json",
    "log": "from the bricks",
    "runid": "ds-lapatinib_750nM_v1",
    "runtype": "annot_acaxis",
    "spotlayout": [
        1
    ],
    "welllayout": "1x1"
}
```

4. to upload the generated acjson file
    1. click the cyan `Set_of_Perturbation` link.
    2. then click the `ds-lapatinib750nmv1` link.
    3. then click the `Browse...` button.
    4. search and choose the `annot_acaxis-ds-lapatinib_750nM_v1_ac.json` file and click `Open`.
    5. then click `Save`.
    6. in the `Acjson file` column should now appear a link `upload/acaxis/annot_acaxis-ds-lapatinib_750nM_v1_ac.json`. click this link.
    7. the uploaded json file should open in the browser. Use your `browser's back arrow` to go back to the perturbation set table.

   8. optional: install a json viewer addon in your browser, as described in *HowTo json files and youe web browser*. Then click again the `upload/acaxis/annot_acaxis-ds-lapatinib_750nM_v1_ac.json` link in the `Acjson file` column. The file should now appear nicely rendered.

5. now let's check the uploaded acjson against the brick content it was generated from.

   1. click in the box in front of the `ds-lapatinib750nmv1` row. The box will turn blue and get a tick.
   2. in the `Action` drop down list choose `Select selected set's acjson file against brick content`. You should receive a message "Ds-lapatinib750nmv1 # sucessfull checked.". Or a "Warning" or "Error" when something for annot not totally is as expected with the uploaded acjson file.

When everything worked as expected, then it is now time to store the modified python3 template code in your own source code repository. Just in case you later have to modify the layout and regenerate the acjson. Follow the instruction described in *HowTo backup acpipeTemplateCode_*.py* .

As a reference, all modified template scripts to generate all acjson used in this tutorial can be found inside the same folder as the acjson files. Have for example a look at `annotTutorial/experiment_acjson/acaxis/acpipeTemplateCode_es-1layout2v3.py`. This script layouts a 384 wellplate. Or have a look at `annotTutorial/experiment_acjson/runset/acpipeTemplateCode_mema-LI8C00201.py`. This is the script to generate the assay run acjson out of sample, perturbation, endpoint and superset acjsons, for mema assay LI8C00201.

The rest of the acjson used in this tutorial we will now restore from the backup.

1. run `docker exec -ti annot_webdev_1 /bin/bash` to enter annot by command line.
2. run `cp -r annotTutorial/experiment_acjson/* ../media/upload/` to copy the acjson file at the expected place.
3. run `python manage.py loaddata annotTutorial/experiment_json/acaxis_sampleset_20181024_oo.json`
4. run `python manage.py loaddata annotTutorial/experiment_json/acaxis_perturbationset_20181024_oo.json`
5. run `python manage.py loaddata annotTutorial/experiment_json/acaxis_endpointset_20181024_oo.json`
6. run `python manage.py loaddata annotTutorial/experiment_json/superset_acpipe_20181024_oo.json`
7. run `python manage.py loaddata annotTutorial/experiment_json/superset_supersetfile_20181024_oo.json`
8. run `python manage.py loaddata annotTutorial/experiment_json/superset_superset_20181024_oo.json`
9. run `python manage.py loaddata annotTutorial/experiment_json/runset_runset_20181024_oo.json` to load back the table content.

Now let's have a look what we can do with this acjson files.

1. The `Acjson_to_tsv_setting`. The acjson file stores information about each sample and reagent that was used in the screen. Now, in practice, when we download layout files or dataframes - which are generated straight out of the acjson files - we might not always be interested in all the details stored in the acjson files. We can tweak such downloads in the Acjson to tsv setting.

   1. click the cyan `Acjson_to_tsv_setting` `Add_acjson_to_tsv` link.
   2. click `Save` it should now have generated an entry for the user you are now logged in.
   3. click on your `username`.

   4. You will see a list of all informative fields stored in an acjson. By clicking the boxes you can choose which one of those you like to write out when you download a `tsv_long_layout` file, a `tsv_tidy_dataframes`, or a `tsv_unstacked_dataframe`.

2. Download `acjsons` and `layouts`. We described acjson files, how to generate and upload them, in the section above. Layout files are similar to Excel sheets that are commonly used for experimental layout description.

   1. click on the dark blue `Assay_Runs` link.
   2. click in the box in front of the `mema-LI8C00201` row.
   3. in the `Action` drop down list choose `Download select selected sets as acjson file`. This is the acjson file we before uploaded.
   4. in the `Action` drop down list choose `Download select selected sets as tsv_short layout file`. This will generate a bunch of layout files, one for each major setname, with no other content then the basic sample or reagent names.
   5. in the `Action` drop down list choose `Download select selected sets as tsv_long layout file`. This will generate a bunch of layout file with the reagent names and all content chosen in the `Acjson_to_tsv_setting` settings.

3. Downloading `dataframes`. Dataframes are tsv files in a format easy to upload into pandas or R

   1. click on the cyan `Set_of_Perturbation` link
   2. click in the box in front of the `es-1layout2v3` row. This is a simple 384 well plate layout.
   3. in the `Action` drop down list choose `Download select selected sets as tidy dataframes` and click `Go`.
   4. in the `Action` drop down list choose `Download select selected sets as unstacked dataframe` and click `Go`.

   For a simple 384 well screen, using the browser GUI can generate a dataframe in a reasonable amount of time. For larger screens the fastest and easiest way to generate dataframes is to download the acjson file and generate the dataframe locally.

   1. click on the dark blue `Asssay_Runs` link
   2. click in the box in front of the `mema-LI8C00201` row.
   3. in the `Action` drop down list choose `Download selected sets as acjson file`. This will download an acjson file named `annot_runset-mema-LI8C00201_ac.json`
   4. move the `annot_runset-mema-LI8C00201_ac.json` into your woking directory.
   5. in your work directory run `python3` to start a python shell.

   In the python3 shell type:

   1. `import json` loads the json library.
   2. `import acpipe_acjson.acjson as ac` loads the acjson library.
   3. `f_acjson = open("annot_runset-mema-LI8C00201_ac.json")` opens the file handle to the acjson file.
   4. `d_acjson = json.load(f_acjson)` loads the acjson file into a kind of a complicated python dictionary termed acjson object.
   5. `ac.acjson2dataframetsv(d_acjson, s_mode="tidy")` will generate three tidy stacked dataframe files named annot_runset-mema-LI8C00201dataframetsv_tidy_sample.tsv, annot_runset-mema-LI8C00201dataframetsv_tidy_perturbation.tsv, and annot_runset-mema-LI8C00201dataframetsv_tidy_endpoint.tsv

6. `ac.acjson2dataframetsv(d_acjson, s_mode="unstacked")` will generate an unstacked dataframe file in the working directory named annot_runset-mema-LI8C00201dataframetsv_unstacked_spe.tsv

1.6 Assay and Superset generation Tracking

Let's reload from the backup what was stored in the purple colored `Mema assay tracking` and `Superset tracking` tables.

1. run `docker exec -ti annot_webdev_1 /bin/bash` to enter annot by command line.
2. run `python manage.py loaddata annotTutorial/experiment_json/track_memasuperset_20181024_oo.json`
3. run `python manage.py loaddata annotTutorial/experiment_json/track_memaassay_20180331_oo.json`

Assay and superset tracking must be tailored to the particular experimental protocol. Alternatively, if you don't want to track superset and assay generation at all, you can easily erase these tables from your installation. Have a look at *About date tracking!* and *HowTo disable the date tracking app?* in the HowTo section of this manual.

1.7 Backup your Work

To back up your work done so far follow the instruction at *HowTo backup annot content?* in the HowTo section of this manual.

And with this we come to the end of this tutorial.

CHAPTER 2

HowTos

2.1 Annot

2.1.1 HowTo install annot?

This howto walks you step by step through the process of installing **development** and **production** version of annot.

1. On the host machine install docker, docker-compose and the docker-machine as described in *HowTo install the docker container platform?*
2. On the host machine install Git. Follow the instructions on the website specified for your operating system.
3. Get the Annot source code from the main fork. Run from the command line: `git clone https://gitlab.com/biotransistor/annot.git`

   (Alternatively, you can clone annot from your own fork. Howto forking the project is not described here.)
4. The cloned source code's `annot/pgsql.env` file contains a few PostgreSQL database configurations. Edit the DB_PASS entry:

```
[...]
DB_PASS=set_some_strong_random_postgresql_root_user_password.
[...]
```

5. Generate a BioPortal bioontology.org account. Go to your BioPortal account settings to figure out your API application interface key.
6. The `crowbar.py` file contains Django framework and annot related environment variables. Write a plain text crowbar.py file with the following content:

```
SECRET_KEY = "about_64_characters_long[->+<]"
PASSWD_DATABASE = "some_random_postgresql_annot_user_password"
APIKEY_BIOONTOLOGY = "your_BioPortal_bioontology.org_API_key"
```

(continues on next page)

(continued from previous page)

```
URL = "http://192.168.99.100/"
CONTACT = "you@emailaddress"
```

Adapt the SECRET_KEY, PASSWD_DATABASE, APIKEY_BIOONTOLOGY and CONTACT content inside the double quotes. For a local installation leave URL as it is.

Place this file under `annot/web/prjannot/crowbar.py`.

7. **development version only**: The `annot/dcdev.yml` file contains docker-compose related information. Edit the webdev and nginxdev volumes path according to your host machine environment:

```
webdev:
  [...]
  volumes:
    - /path/to/your/git/cloned/annot/web:/usr/src/app
  [...]

nginxdev:
  [...]
  volumes:
    - /path/to/your/git/cloned/annot/nginxdev/annotnginx.conf:/etc/nginx/nginx.
↪conf
  [...]
```

8. Build a docker machine in which the docker container will be installed, to run the development or production version of annot. Build the containers. Then fire up annot. You can name the machine however you like. In this example we named the machine *an0*.

   1. `docker-machine create --driver virtualbox --virtualbox-disk-size 20000 an0` this command creates the machine using VirtialBox as disk driver. The disk size is given in MB. Please adjust disk size to your needs.
   2. `docker-machine ls` lists all machines.
   3. `docker-machine start an0` fires up machine an0, if not yet running.
   4. `docker-machine env an0` get an0's environment variables.
   5. `eval "$(docker-machine env an0)"` sets an0's environment variables.
   6. `docker-machine ls` the an0 machine should now have an asterisk (`*`) in the ACTIVE column.
   7. cd into the cloned annot folder then execute the next steps.

   for the **development version**:

   1. `docker-compose -f dcdev.yml pull` pulls the basic containers.
   2. `docker-compose -f dcdev.yml build` builds all container.
   3. `docker-compose -f dcdev.yml up` fires up the docker containers and reports what goes on with the web framework.
   4. press `ctrl + c` to shut down the docker containers and give the prompt back.
   5. `docker-compose -f dcdev.yml up -d` fires up the docker containers and gives the prompt back.

   for the **production version**:

   1. `docker-compose -f dcusr.yml pull` pulls the basic containers.
   2. `docker-compose -f dcusr.yml build` builds all container.

   3. `docker-compose -f dcusr.yml up` fires up the docker containers and reports what goes on with the web framework.
   4. press `ctrl + c` to shut down the docker containers and gives the prompt back.
   5. `docker-compose -f dcusr.yml up -d` fires up the docker containers and gives the prompt back.

9. Setup PostgreSQL database and database user.
   1. `docker exec -ti annot_db_1 /bin/bash` to enter db docker container.
   2. `su postgres -s /bin/bash` to switch from unix root to unix postgres user.
   3. `createdb annot` creates a postgresql database named annot.
   4. `createuser -P annot` creates a database user named annot. When prompted enter the same database password as specified in `annot/web/prjannot/crowbar.py`
   5. `psql -U postgres -d annot -c"GRANT ALL PRIVILEGES ON DATABASE annot TO annot;"` does what it says.
   6. `exit` to exit as unix postgres user.
   7. `exit` to exit as unix root user and leaving as such the annot_db_1 docker container.

10. Generate database tables, a superuser and pull all static files together:

    for the **development version**:

    1. `docker exec -ti annot_webdev_1 /bin/bash` to enter the webdev docker container.

    for the **production version**:

    1. `docker exec -ti annot_web_1 /bin/bash` to enter the web docker container.

    then continuer:

    1. `python demigrations.py` will clean out the sql migration command folder from every app.
    2. `python manage.py makemigrations` generates the sql database migration commands.
    3. `python manage.py migrate` applies the generated sql migration commands.
    4. `python manage.py createsuperuser` creates a superuser for the annot web application.
    5. `python manage.py collectstatic` collects all static files needed by annot and put them into the right place to be served.
    6. `exit` to leave the container.

11. Fire up you favorite web browser and surf to the place where annot is running.
    1. `docker-machine ls` will give you the correct ip. Most probably 192.168.99.100.
    2. http://192.168.99.100/admin/ you can enter the annot GUI at the admin side. Use therefore the generate superuser credentials.

12. *production version only*:

    Annot can be set up so that it automatically checks for new versions of each ontology at midnight container time, and installs them and backups the whole annot content.

    1. run `docker exec -ti annot_web_1 /bin/bash` to enter the annot_web_1 docker container
    2. `/etc/init.d/cron status` to check the cron daemon status.
    3. `/etc/init.d/cron start` to start the cron daemon. Will enable check and backup at midnight container time. Backups are stored in at `/usr/src/media/`.

4. `date` to check for the docker containers local time.

Assuming you run a unix flavored host machine and cron is installed, your *host machine* can be setup to pull automatically the backups stored inside the docker container to the host machine every night. For this, you have to adjust and install the following cronjob.

At your *host machine*, inside the cloned annot project folder adjust `annot/web/nix/hostpull.sh`.

1. Change every `mymachine` to the docker machine name you gave. e.g `an0`.
2. Change every `/path/on/host/to/store/backup/` to the directory you would like to have your backups placed.

At the *host machine*, inside the cloned annot project folder adjust `annot/web/nix/hostcronjob.txt`

1. Make sure that `PATH` knows the location of the docker-machine binary. Run `which docker-machine` at the command line to find out the correct location.
2. Change the time `00 00` (which represents mm hh) to be 6 hours later than midnight inside the annot docker containers.
3. Change `/path/to/cloned/project/` to the directory where you have annot cloned to.
4. Change `/path/on/host/to/store/backup/` to the directory you would like to have your backups placed.

At the *host machine*, queue the cron job and start cron:

1. `crontab /path/to/cloned/project/annot/web/nix/hostcronjob.txt` to queue the job.
2. `/etc/init.d/cron status` to check cron daemon status.
3. `/etc/init.d/cron start` to start cron daemon, if needed.

If you run into troubles, the following cron documentation might come in handy. But keep in mind, this documentation was written for folks running the Ubuntu OS.

2.1.2 HowTo json files and youe web browser?

Howto make the acjson file uploaded to annot viewable in your browser?

- for *Ms Internet Explorer* this hack will make the json file viewable but it will not render them nicely.
- the Firefox developer Edition comes with an integrated json viewer.
- for *Chrome*, *Firefox*, *Opera* and *Safari* install this Json Lite browser Add-on which can render large json files quickly.
- for links json files are viewable but will not be rendered.

2.1.3 HowTo set up an additional annot user?

1. enter annot as superuser via GUI.
2. scroll down to the white colored `Authentication_and_Authorization` link on the bottom of the page.
3. click `Groups` `Add_Group`.
4. give `add`, `change` and `delete` `Permissions` for all `app*` django applications.
5. `Save` group.
6. go back to `Home › Authentication and Authorization`.

7. click `Users` `Add_User`.
8. set `Username` and `Password`.
9. give user `Staff_status` by clicking the box.
10. add user to the group generated before.
11. `Save` user.

2.1.4 HowTo fire up annot?

Once annot is installed as described in *HowTo install annot?* it can be fired up like this:

1. `docker-machine ls` lists all machines
2. `docker-machine start an0` fires up machine an0, if not yet running.
3. `docker-machine env an0` get an0's environment variables.
4. `eval "$(docker-machine env an0)"` sets an0's environment variables
5. `docker-machine ls` the an0 machine should now have an asterisk in the ACTIVE column.

for the **development version**:

- `docker-compose -f dcdev.yml up` fires up docker containers.

for the **production version**:

- `docker-compose -f dcusr.yml up` fires up docker containers.

2.1.5 HowTo enter annot?

First annot must be running as described in *HowTo fire up annot?* Then:

- To enter annot by GUI, point your browser at http://192.168.99.100/admin/ and use your annot user credentials.
- To enter the **development version** from the command line run: `docker exec -ti annot_webdev_1 /bin/bash`
- To enter the **production version** from the command line run: `docker exec -ti annot_web_1 /bin/bash`

2.1.6 HowTo get files from your host machine into annot?

for the **development version**:

1. move the files into the annot/web folder on your host machine.
2. run `docker exec -ti annot_webdev_1 /bin/bash` to enter the docker container.
3. the files will appear inside the /usr/src/app folder.

for the **production version**:

- rebuild the annot_web_1 container: this works because all relevant data is stored in the annot_fsdata_1 and annot_dbdata_1 containers.
    1. move the files into the `annot/web` folder on your host machine.
    2. `docker-compose -f dcusr.yml stop` to shut down the docker containers.

   3. `docker rm annot_web_1` to remove the annot_web_1 container.
   4. `docker-compose -f dcusr.yml build` to rebuild the annot_web_1 container from scratch.
   5. `docker-compose -f dcusr.yml up` to fire up annot again.

- cat data into the docker container
   1. tar or zip the data to one big file.
   2. `docker exec -i annot_web_1 bash -c "cat > bigfile.tar.gz" < /host/path/bigfile.tar.gz` to upload a big junk of data into the docker container.

2.1.7 HowTo get files from inside annot to your hostmachine?

for the **development version**:

1. run `docker exec -ti annot_webdev_1 /bin/bash` to enter the docker container
2. move the files into the `/usr/src/app` folder.
3. the files will appear inside the `annot/web` folder on your host machine.

for the **production version**:

- scp from inside the docker container:
   1. run `docker exec -ti annot_webdev_1 /bin/bash` to enter the docker container
   2. run something like `scp bigfile.tar.gz user@host.edu:`
- docker cp from the host machine:
   1. run something like `docker cp annot_web_1:/usr/src/path/to/the/bigfile.tar.gz`

2.1.8 HowTo list all available commands?

- In the GUI available commands can be found in each app in the **Action** drop down list.
- Enter annot from the command line. run `python manage.py`

2.1.9 HowTo backup annot content?

Annot can be backed up using the shell scripts we provide. Specifically:

1. enter annot by the command line.
2. `nix/cron_vocabulary.sh` this is a bash shell script, written to back up controlled vocabulary terms. Before annot backs up any vocabulary, it first updates the vocabulary to the latest ontology version. The backups are placed in folder `/usr/src/media/vocabulary/YYYYMMDD_json_backup_latestvocabulary/`
3. `nix/cron_brick.sh` back up brick in json and tsv format. The backups are placed in folder `/usr/src/media/vocabulary/YYYYMMDD_json_latestbrick` and `/usr/src/media/vocabulary/YYYYMMDD_tsv_latestbrick/`.
4. `nix/cron_experiment.sh` backs up the acaxis, superset, runset, track, study and investigation table and acjson and superset files.

In the *production version* a cron job can be enabled to automatically backup the annot content every night. How to do this is described in the last step of *HowTo install annot?*

2.1.10 HowTo backup acpipeTemplateCode_*.py code?

Warning: it is your responsibility to back up the modified python3 template code that generated the acsjon files, as these scripts not are stored in annot.

1. run `mkdir acaxis superset suerpsetfile runset` to gerenate the following folder structure:
   - acaxis
   - superset
   - suerpsetfile
   - runset
2. place all `acpipeTemplateCode_*.py` and *superset fiels* into the corresponding folders.

You only have to backup the *py* fiels and the *supersetfiles* as the *ac.json* files anytime can be regeneraetd.

2.1.11 HowTo fix anjson annotation that propagates from the acaxis to the superset to the runset layer?

The problem is the following: If you for example realize that you miss typed a concentration value in acjson on the acaxis layer, then you will have to fix this bug in the `acpipeTemplateCode_*.py`, generate the acjson file an regenerate all acjson files that depend on this acjson files. Doing such a fix via GUI is possible though really tedious. So we will make use of the command line to fix this bug.

1. set up a folderstructure as described in *HowTo backup acpipeTemplateCode_*.py code?*
2. fix the acpipeTemplateCode_*.py where necessary.
3. `cp annot/web/apptool/acjsonUpdater.py /to/the/root/of/the/folderstructure/.`
4. `cp annot/web/apptool/acjsonCheck.py /to/the/root/of/the/folderstructure/.`
5. run `python3 acjsonUpdater.py` from the root of your folder structure. this should re-gernerate all acjson files.
6. run `python3 acjsonCheck.py` from the root of your folder structure. this should check all superset and runset acjson against inconsistency with the acxis and superset acjson. The result will be written into a file named `YYYYmmdd_acjsoncheck.log` at the root folder.
7. copy the whole folder structure to `/usr/src/media/upload/` inside annot as described in *HowTo get files from your host machine into annot?*. Note: remove the `.git` folder form the root of *the copy*, if there is one, because this folder can cause troubles.

Now all your acjson files in annot should be updated to the latest version.

2.1.12 HowTo handle controlled vocabulary?

Please check out the *about controlled vocabulary* for a detailed discussion about the subject. This section just deals with the available annot commands.

- `python manage.py vocabulary_getupdate apponorganism_bioontology apponprotein_uniprot` searches and downloads the latest ontology version from ncbi taxon and uniprot. It will *not* update the database with it.
- `python manage.py vocabulary_getupdate` this command searches and downloads the latest ontology version for each vocabulary. It will *not* update the database.

- `python manage.py vocabulary_loadupdate apponorganism_bioontology apponprotein_uniprot` searches and downloads the latest ontology version, and updates the database, but only for ncbi taxon and uniprot.
- `python manage.py vocabulary_loadupdate` this command searches and downloads the latest ontology version for each vocabulary, and updates the database.
- `python manage.py vocabulary_loadbackup apponorganism_bioontology apponprotein_uniprot` First it will populate the ncbi taxonomy and uniprot vocabulary with the latest backup found at /usr/src/media/vocabulary/backup/‘. Then it will download the latest ontology version, and update the database content with it.
- `python manage.py vocabulary_loadbackup` will populate each vocabulary app First it with the latest backup found at /usr/src/media/vocabulary/backup/‘. Then it will download the latest ontology version, and update the database content with it. This command will break if a online ontology fails to be downloadable.
- `nix/cron_vocabulary.sh` is a shell script, written to back up each and every vocabulary one by one. Before annot backs up any vocabulary, it first update the vocabulary to the latest ontology version available. This script will not break, if a new online ontology version fails to be downloadable, which does happen.
- In the *production version* a cron job can be enabled to automatically check for all plugged in ontologies for new versions every night, installs them when available and backs up the local modifications. How to is described in the last step of *HowTo install annot?*

We have defined ontologies for categories where no established ontology exists. For example: Dye, Health status, Provider, Sample entity, Verification profile and Yield fraction. Terms added to these ontologies can be transformed to “original” ontology terms:

- `python manage.py vocabulary_backup2origin apponhealthstatus_own` will transform all added terms from the apponhealthstatus_own into original terms.
- `python manage.py vocabulary_backup2origin` will transform all added terms from every *_own ontology into original terms.

2.1.13 HowTo deal with huge ontologies?

Huge ontologies like Cellosaurus (apponsample_cellosaurus), ChEBI (apponcompound_ebi), Gene Ontology (apponcellularcomponent_go, appongeneontology_go, apponmolecularfunction_go), NCBI Taxonomy (apponorganism_bioontology), UniProt (apponprotein_uniprotemacs), can be filtered down to the relevant set of terms for your experimental paradigm using filter_idenitifier.txt or filter_term.txt files inside the particular django app. Check out the filter file in one of those ontologies apps for reference.

2.1.14 HowTo get detailed information about the ontologies in use?

A complete list of ontologies plugged into your current annot installation, their actual version, and the source it is pulled from can be found by clicking inside the GUI on the red colored `Sys_admin_ctrl_vocabularies` link.

2.1.15 HowTo handle bricks?

Bricks are the cell lines and reagents used in the wet lab. In annot those bricks can be specified and annotated by searchable drop down list boxes with controlled vocabulary.

There are three major types of bricks:

- sample bricks
- perturbation bricks

- endpoint bricks

There are currently seven minor types of bricks:

- antibody1: primary antibodies
- antibody2: secondary antibodies
- cstain: compound stains
- compound: chemical compounds
- protein: proteins
- proteinset: protein complexes
- human: human cell line samples

Bricks are highlighted orange in the GUI.

These are the four commands to deal with each minor brick type. The example for the protein brick type:

- `python manage.py protein_db2json` will download the content from the protein brick table into a json file. This format is easy processable by python and is handy for backups.
- `python manage.py protein_db2tsv` will download the content from the protein brick table into a tab separated value file. This is a handy format for folks who prefer Excel sheet over GUI for brick annotation and is a handy backup format.
- `python manage.py protein_json2db` will upload the protein brick json file to the database. The upload content will automatically be checked against valid controlled vocabulary.
- `python manage.py protein_tsv2db` will upload the protein brick tab separated value file into the database. Any additional columns will thereby be ignored. The content inside the expected columns will automatically be checked against valid controlled vocabulary.

2.1.16 HowTo annotate protein complexes?

In the GUI:

1. scroll to the orange colored `Appbrreagentprotein` section.
2. click `Perturbation_Proteinset`.
    1. under `Protein_set` choose the gene ontology cellular component identifier for the protein complex you wane annotate. E.g. COL1_go0005584.
    2. choose the `Provider`.
    3. enter `catalog_id`.
    4. enter `batch_id`.
    5. adjust `Availability`, `Final_concentration_unit` and `Time_unit` if necessary.
    6. click `Save`.

    Now Collagen 1 is a protein complex built out of two COL1A1_P02453 Collagen alpha-1 (I) chain proteins and one COL1A2_P02465 Collagen alpha-2 (I) chain protein. Both of these proteins have to be annotated.
3. click `Perturbation_Protein` and enter both proteins as usual.
    - Under `Protein_set` you must choose the proteinset generated before.
    - Enter the `Proteinset_ratio` 2:1.
    - Our lab convention is: Set `Availability` to False, because the single protein as such is not available.

   - Our lab convention is: Give the `Stock_solution_concentration` for the whole protein complex, do not divide by protein ratio, because there are protein complex reagents where the exact ratio is unknown.

4. now you should be able to upload this COL1_go0005584 protein set.

2.1.17 Howto make bricks accessible in the experiment layout layer?

Before any brick is accessible in experiment layout, it must be uploaded into the corresponding `Uploaded enpoint reagent bricks`, `Uploaded perturbation reagent bricks` or `Uploaded sample bricks` table. The very first time you install annot you have to do this by command line, because the database tables which the GUI relies on has to be initialized. After that you can populate the brick tables via command line or GUI.

from the command line:

1. `python manage.py brick_load` will upload all bricks.

from the GUI:

1. scroll to the bright orange colored `Sys_admin_brick` link and click.
2. select the brick types you like to upload.
3. In the `Action` drop down list choose `Upload brick` and click `Go`.

Where are the uploaded bricks stored?

1. enter the GUI
2. go to `Home › Appsabrick` (bright orange colored)
3. the `Uploaded endpoint reagent bricks`, `Uploaded endpoint reagent bricks` and `Uploaded endpoint reagent bricks` are the tables containing the uploaded bricks. Those are the bricks accessible for layout.

Note: If a brick (oragne colored) gets deleted, the uploaded brick inside the `Uploaded bricks` tables (bright orange colored) and any set that uses this uploaded brick that no longer exist will *not* be deleted! The entry in the `ok_brick` column of such uploaded bricks will change from a green tick to a red cross, the next time this brick type is uploaded.

2.1.18 HowTo layout experiments?

In a similar way the IPO input processing output paradigm can describes the structure of an information processing program, a biological experiment can be specified by sample, perturbation and endpoint description. The samples can thereby be regarded as input, perturbations as processing and endpoints as output. In annot assay coordinate model sample, perturbation and endpoint are represented as "axis". Below is in short described, who such axis are specified. Check out the Tutorial for an applied example.

About axis sets!

1. To define an axis set, one first has to gather the samples, the perturbation reagents, and the endpoint reagents used in the experiment.
   1. scroll to the cyan colored `Appacaxis` box.
   2. click the cyan `Set_of_Endpoints` and `Add` link to group together the endpoint brick used in an experiment.
   3. click the cyan `Set_of_Perturbation` and `Add` link to group together the perturbation bricks.

4. click the cyan `Set_of_Sample` and `Add` link to group together the sample bricks.

For `set_names` only alphanumeric keys, underscores and dashes are allowed [A-Za-z0-9-_]. The `dash` has a special function. The dash separates the major from the minor and possibly subminor setname. E.g. drug-plate1, drug-plate2 and drug-plate3-well2 are all member of the same major drug set. This becomes especially important later on when layout files and unstacked dataframes are retrieved form the acjson files, because the layout files will be grouped into folders according to their major sets name, and the unstacked dataframe will group the columns according to the major sets. If no dash is given, then the major and the minor set name are the same.

1. Second, the gathered samples and reagents have to be laid out. Python3 and the acpipe_acjson library must be installed on your computer. You can install the acpipe_acjson library with pip like this:

   1. `pip3 install acpipe_acjson` should do the trick.

   What follows is the description of the layout process on a perturbation set. But layout for sample and endpoint sets is done exactly the same way.

   1. click the cyan colored `Set_of_Perturbation` link.
   2. choose the set you would like to layout.
   3. in the `Action` drop down list choose `Download selected set's python3 acpipe template script` and click `Go` to download the template file.
   4. open the template file in a text editor. You will find python3 template code, generated based on set_name and the chosen bricks. Read the template code and replace all the question-marks, which are place holders for wellplate layout and each reagent's concentration and reaction time, with meaningful values.
   5. then run `python3 acpipeTemplateCode_*set-name*.py`. This will result in a `acpipe_acjson-*set-name*_ac.json` file.

2. Third, upload the generated acjson file and check for consistency.

   1. on the GUI click the name from the set you downloaded the template.
   2. scroll down to `Set Acjson file` and `Browse...` for the generate file to upload it.
   3. click `Save`
   4. in the `Set_of_Perturbation` table choose the set again. Then, in the `Action` drop down list choose `Check selected set's acjson file against brick content.` and click `Go`. After a little while, you should see a message `*set-name* # successfully checked` or a warning when the acjson content differs from the set_name or bricks settings.

About supersets!

Superset - stored in the blue colored `App4Superset` box - are optional.

Imagine for example you have pipette robot which helps you to produce randomized wellplates from reagents provided in eppendorf tubes.

You could store:

1. the eppendorf layout that you feed to the pipette robot as an ordinary `Set_of_Perturbation`.
2. store the pipette robots program code as `Superset_File`.
3. write a python3 library that can take the eppendorf layout acjson file and the robot program code as input to generates the `random plates layout acjson file`. store this library as `Python3_acpipe_module`.
4. Connect `eppendorf layout perturbations set`, `plate robot program code file`, `python3 acjson module` and resulting `random plate acjson file` as super set.

For any system in the lab you can imagine, you can write a python3 acpipe library and plug it into annot.

About run sets!

One runset represents one assay. An assay combines all 3 acjson axis: Sample, Perturbation, and Endpoint. The information can come from sampleset acjson files, perturbation set acjson files, endpoint acjson files, and superset acjson files.

1. scroll down to the dark blue colored `Assay_Runs` `Add` link.
2. give a `Runset_name`. Allowed are alphanumeric characters, dashes and underscore [A-Za-z0-9-_].
3. use the drop down list boxes to gather the related endpointsets, perturbationsets, samplesets, and supersets click `Save`.
4. in the `Action` drop down list choose `Download selected set's python3 acpipe template script` and click `Go` to download the template file.
5. modify the template code as appropriate and run it.
6. upload the resulting `Acjson file` to the set.
7. in the `Action` drop down list choose `Check runset against set of set acjson` and click `Go`. After a while You should see a message `*runset_name* # successfully checked` or a warning when the acjson content differs.

About date tracking!

The tracking layer enables assay and superset related date, protocol, and staff member metadata to be documented. The tracking site links are located in the purple colored `App2Track` box. The tracking app can be customized for different experimental protocols.

1. edit the `app2tacking/models.py` file to you needs
2. edit the `app2tacking/admin.py` file to you needs
3. enter annot by command line
4. run `python manage.py makemigrations`
5. run `python manage.py migrate`
6. edit the es_table constat and the os.system datadump call in `annot/web/appacaxis/management/commands/experiment_pkgjson.py` to have the backup packing properly updated

2.1.19 HowTo disable the date tracking app?

1. open `annot/web/prjannot/settings.py` with a text editer.
2. inside the INSTALLED_APPS tuple use a hashtag # to comment out `app2track`.
3. save the settings.py and leave the editor
4. run `docker-machine restart an0`, assuming your docker machines name is an0.
5. reload `http://192.168.99.100/admin/` page in your browser. `App2Track` should be gone.

2.1.20 HowTo handle study and investigation?

1. click the black colored `Studies` and `Add` link to gather `Assay_Runs` to a study.

2. click the black colored `Investigation` and `Add` link to gather `Studies` to an investigation. Those pages should be quite self explanatory.

2.2 Django

2.2.1 Howto enable the django-debug-toolbar?

1. open `annot/web/prjannot/settings.py` with a text editor.
2. delete the hashtags in front of DEBUG_TOOLBAR_CONFIG = { "SHOW_TOOLBAR_CALLBACK" : lambda request: True, }.
3. inside the INSTALLED_APPS tuple delete the hashtag in front of `debug_toolbar`.
4. inside the MIDDLEWARE_CLASSES tuple delete the hashtag in front of `debug_toolbar.middleware.DebugToolbarMiddleware`.
5. save the settings.py and leave the editor
6. enter annot from the command line
7. run `python manage.py collectstatic`
8. exit the container
9. run `docker-machine restart an0`, assuming your docker machines name is an0.
10. reload `http://192.168.99.100/admin/` page in your browser

2.3 Docker

2.3.1 HowTo install the docker platform?

Docker is able to run on Linux, Mac OSX, MS Windows, and many cloud platform flavors.

Install Docker Engine. Docker Machine and Docker Compose as described here: Install Docker. Additionally install VirtialBox, which will be used as docker-machine disk driver.

2.3.2 HowTo run the docker platform?

This howto will get you familiar with docker, as much as is needed to run docker as annot user or developer.

To successfully run docker you have to know a whole set of docker commands, from the docker engine, docker-compose, and docker-machine. The section below introduces a minimal set of commands needed to run annot. It is worthwhile to check out the list of all available docker engine, docker-compose, and docker-machine commands. There are many nice commands that may be very helpful for your specific application.

The docker platform can be booted either by starting the docker engine or by firing up a docker-machine. Annot as such, could run solely with the docker engine and docker-compose. However, we have chosen to make use of docker-machine to allow one physical computer to run more then one development version or a development and a deployed version simultaneously.

docker-machine commands

In this example the docker machines name is dev:

- `docker-machine` list all available docker-machine commands.
- `docker-machine --driver virtualbox dev` makes a docker machine labeled dev. Default disk size is 20GB.
- `docker-machine --virtualbox-disk-size "1000000" --driver virtualbox dev` makes a docker machine labeled dev with 1TB space. Then disk size is always given in MB.
- `docker-machine start dev` start docker machine dev.
- `docker-machine ls` lists all docker-machines (docker environments).
- `docker-machine ip` get the IP address of a machine to connect to it e.g. by a web browser.
- `docker-machine env dev` get dev's environment variables.
- `eval "$(docker-machine env dev)"` sets dev's environment variables.
- `docker-machine regenerate-certs dev` recreates ip certificates if needed. usually IPs are give by the order the machines are started. in case the IP of the dev changed the certificates have to be regenerated.
- `docker-machine upgrade dev` upgrades the dev machine to the latest version of docker.

docker engine commands

In the docker world, you have to be able to distinguish between a docker image and docker container. A docker image is a synonym for the container class (or type), A docker container is a actual instance of one of this container class.

Basic docker image related commands. In this example the image is labeled annot_web and has the id 0b8a78c6c379:

- `docker` list all the docker engine commands.
- `docker images` list all images.
- `docker rmi 0b8a78c6c379` delete one or more images.
- `docker rmi annot_web` delete one or more images.

Basic docker container related commands. In this example the container is labeled annot_web_1 and has the id 290ebef76c11:

- `docker` list all the docker engine commands.
- `docker ps` list running containers.
- `docker ps -a` list all containers.
- `docker run annnot_web_1 ls` run the ls command in a new container instance.
- `docker exec annnot_web_1 ls` execute the ls command in the annnot_web_1 container instance.
- `docker exec -ti annnot_web_1 /bin/bash` open an interactive terminal, which running the bash shell inside the annnot_web_1 container.
- `docker start annnot_web_1` start a stoped container.
- `docker restart annnot_web_1` restart a running container.
- `docker stop annnot_web_1` stop a running container.
- `docker rm annnot_web_1 annnot_nginx_1` delete one or more containers.

- `docker rm -v annnot_fsdata_1` delete the annnot_fsdata_1 container inclusive the volume inside the container

The slight difference between run and exec: docker run command will run on the annot_web_1 image and actually build a new container to run the command. Then new container will be automatically labeled. docker exec instead will run the command in an existing container. No new container will be build. In the case of annot you usually you do not want do create a new container.

docker-compose commands

Web applications like annot are usually built out of many containers. For example the development version of annot is out of five containers: annot_nginxdev_1, annot_webdev_1, annot_fsdata_1, annot_db_1, annot_dbdata_1. To orchestrate the whole container set you can run docker-compose commands. Nevertheless, it is important to know the low level docker engine commands, to be able to deal with single container out of the set. For run docker-compose commands, the container set and the connection between the containers have to be specified through a related yml file. In the following example this is the dcdev.yml file:

- `docker_compose` list all the docker compose commands.
- `docker-compose -f dcdev.yml build` build or rebuild the container set.
- `docker-compose -f dcdev.yml up` start the containers set. This don't gives the prompt back, but detailed output about the runnig containers. Press `ctrl + c` to stop the containers.
- `docker-compose -f dcdev.yml up -d` start the containers set in daemon mode. This gives the prompt back, but no detailed output about the runnig containers.
- `docker-compose -f dcdev.yml ps` list all running containers.
- `docker-compose -f dcdev.yml ps -a` list all containers.
- `docker-compose -f dcdev.yml start` start container set.
- `docker-compose -f dcdev.yml restart` restart container set.
- `docker-compose -f dcdev.yml stop` stop container set.
- `docker-compose -f dcdev.yml rm` remove stopped container set.

2.4 PostgreSQL

2.4.1 HowTo enter the postgresql database?

From the command line execute the following steps:

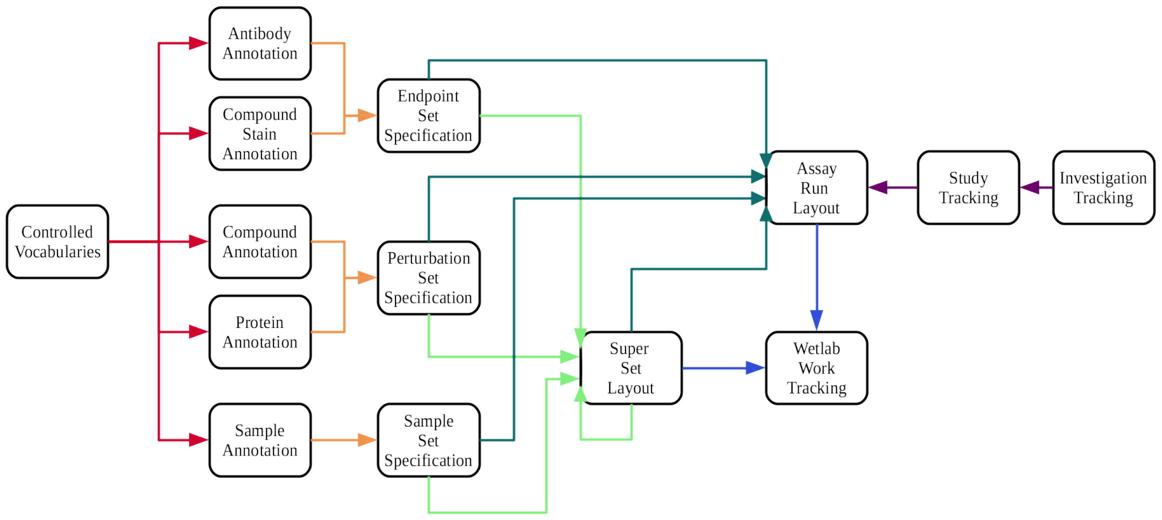
1. `docker exec -ti annot_db_1 /bin/bash` to enter annot_db_1 docker container as unix root user.
2. `su postgres -s /bin/bash` to switch to unix postgres user.
3. `psql -U annot -d annot` to enter the postgresql shell as database user named annot and connecting to the database named annot.
4. `\q` to quit the postgresql shell.
5. `exit` to exit as unix postgres user.
6. `exit` to exit as unix root user and leaving as such the annot_db_1 docker container.

CHAPTER 3

Reference

3.1 Annot workflow

This workflow representation gives an overview, how from a user point of view experiments with annot are annotated.



An! workflow

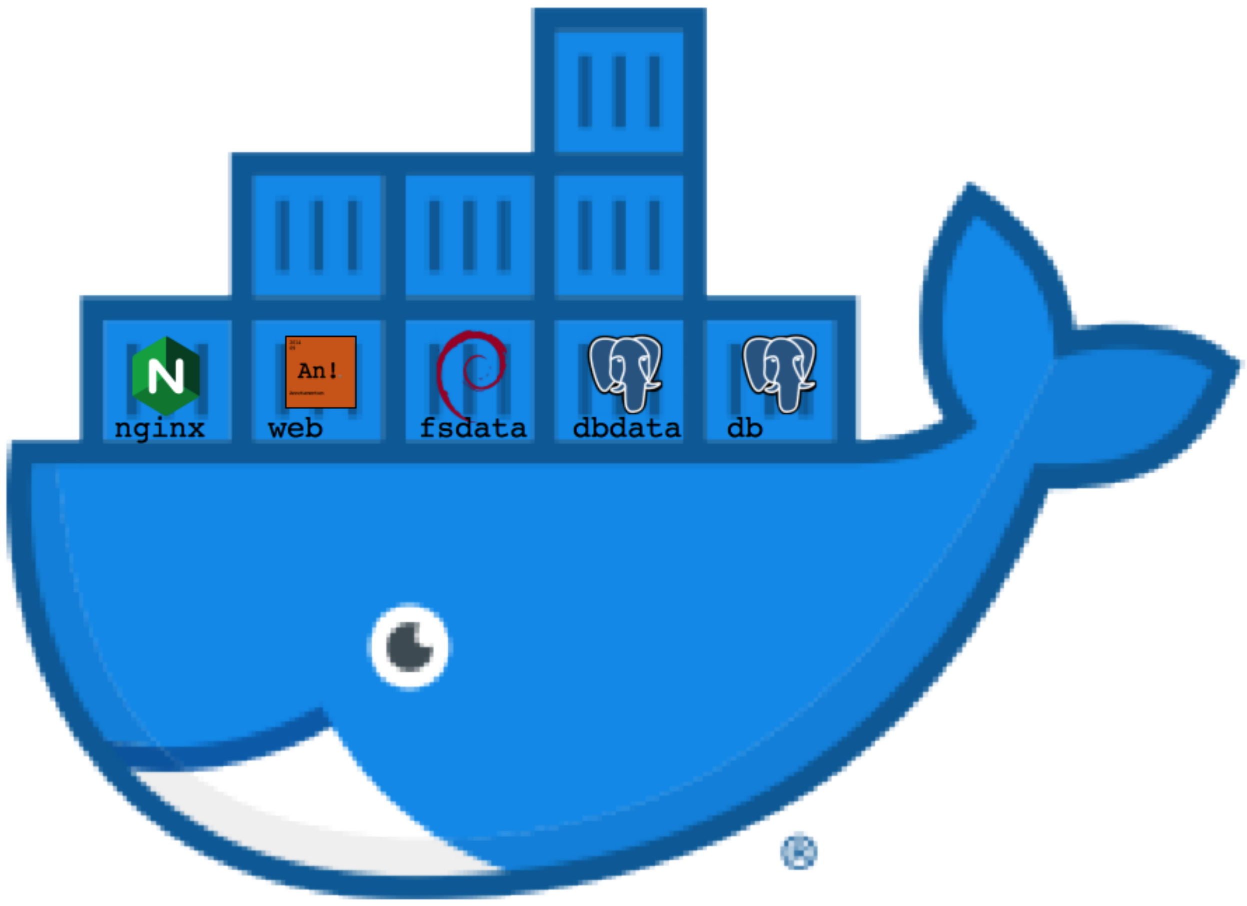
Assay reagents and samples are first annotated via the annot web interface or via Excel spreadsheet that can be uploaded into annot. This annotation step enforces the use of controlled vocabulary and official gene, protein, compound and cell line identifiers (red arrows). Annotated reagents and samples are next combined into endpoint, perturbation, and sample sets. In this step, additional experimental details can be specified, for example reagent concentrations, cell seeding density, or cell passage number (orange arrows).

For assays that involve robot pipetting, array spotting, or cyclic staining, super sets can be generated (light green arrows). Finally, for each assay run, endpoint, perturbation, sample and super sets are merged to a run specific assay layout (dark green arrows).

Assays and supersets that are regularly processes by the lab can be directly tracked in annot. There the exact data, protocol and the person who did the lab work can be specified (blue arrows). Lastly, assays can be grouped into studies and studies into investigations (purple arrows).

3.2 Docker and how the annot folder is structured

Annot is deployed within the docker distribution platform. In detail annot is packed into a docker-machine, where it is split into five docker containers:



An!

docker continer stack

1. annot_nginxdev_1 or anno_nginx_1 contains the web server.
2. annot_db_1 contains the database engine.
3. annot_dbdata_1 contains the data stored in the database.
4. annot_webdev_1 or annot_web_1 contains the actual annot code base.
5. annot_fsdata_1 contains all stored non-database data.

These five containers can be built and spun up together utilizing docker-compose command either with the dcdev.yml file for the development version or the dcusr.yml file for the production version.

In addition, these five containers define the main folder structure found in the annot folder.

1. nginx contains construction information for the annot_nginx_1 container.
2. nginxdev contains construction information for the annot_nginxdev_1 container.
3. dbdata contains construction information for the annot_dbdata_1 container.
4. web contains the actual annot code base and construction information for the annot_web_1 container.
5. webdev contains construction information for the annot_webdev_1 container.
6. fsdata contains construction information for the annot_fsdata_1 container.

For the annot_db_1 container is taken care in the dcdev.yml and dcusr.yml and the pgsql.env. all those file are found straight in the annot folder itself.

There is an additional folder - man - which contains the rst restructured text and md markdown files for this very user manual. The sphinx documentation generator tool can be used to generate the final documentation out of this files.

Further there is the LICENSE and a README.md file.

3.3 The nginx and nginxdev folder

Nginx and gunicorn serve as annot's web server backbone. Gunicorn figures thereby as unix WSGI (web server gate way interface) HTTP server. Nginx figures as HTTP proxy server.

The nginxdev and nginx folder contains a:

1. Dockerfile, which contains the container building instruction for annot_nginxdev_1 and annot_nginx_1. This containers are constructed out of the official nginx docker image.
2. annotnginxdev.conf or annotnginx.conf file which contains the particluar nginx configuration.

Gunicorn is called from the dcdev.yml or dcusr.yml file. The gunicorn library is listed in the requiremnet.txt in webdev and wev folder.

3.4 The dbdata folder

PostgreSQL and the psycopg2 library serve as annot's database backbone. PostgreSQL figures thereby as database engine, psycopg2 figures as python postgresql database adapter.

The dbdata folder contains a:

1. Dockerfile with the container building instruction for annot_dbdata_1. This container is constructed out of the official postgresql docker image.

The building instruction for annot_db_1 container (which contains the postgesql datbase engine) are part of the dcdev.yml and dcusr.yml file. The postgresql engine related configuration settings are stored in the pgsql.env file in the annot folder. This container is constructed out of the official postgresql docker image. The psycopg2 library is listed in the requiremnet.txt in webdev and web folder.

Splitting database engine and data into two containers (annot_db_1 and annot_dbdata_1 makes it really easy to update the database engine without loosing the data stored in the database.

3.5 The web, webdev and fsdata folder

The webdev and web folder contain a:

1. Dockerfile with the container building instruction for annot_webdev_1 and annot_web_1. These containers are constructed out of the official debian based python3 docker image.
2. requirement.txt file which lists the addition python libraries needed in the annot project.

The fsdata folder contains a

1. Dockerfile with the container building instruction for annot_fsdata_1. This container is constructed out of the official debian docker image.

Splitting the annot code base (annot_webdev_1 or annot_web_1) and filesystem part where files are stored (annot_fsdata_1) into separate containers makes it easy to update the annot code base without losing the data files stored on the filesystem.

3.5.1 Python3

Annot is written in the python3 language. For running Annot and especially for assay layouting you should at least be a bit familiar with this language.

3.5.2 acJson - the assay coordinate json file format

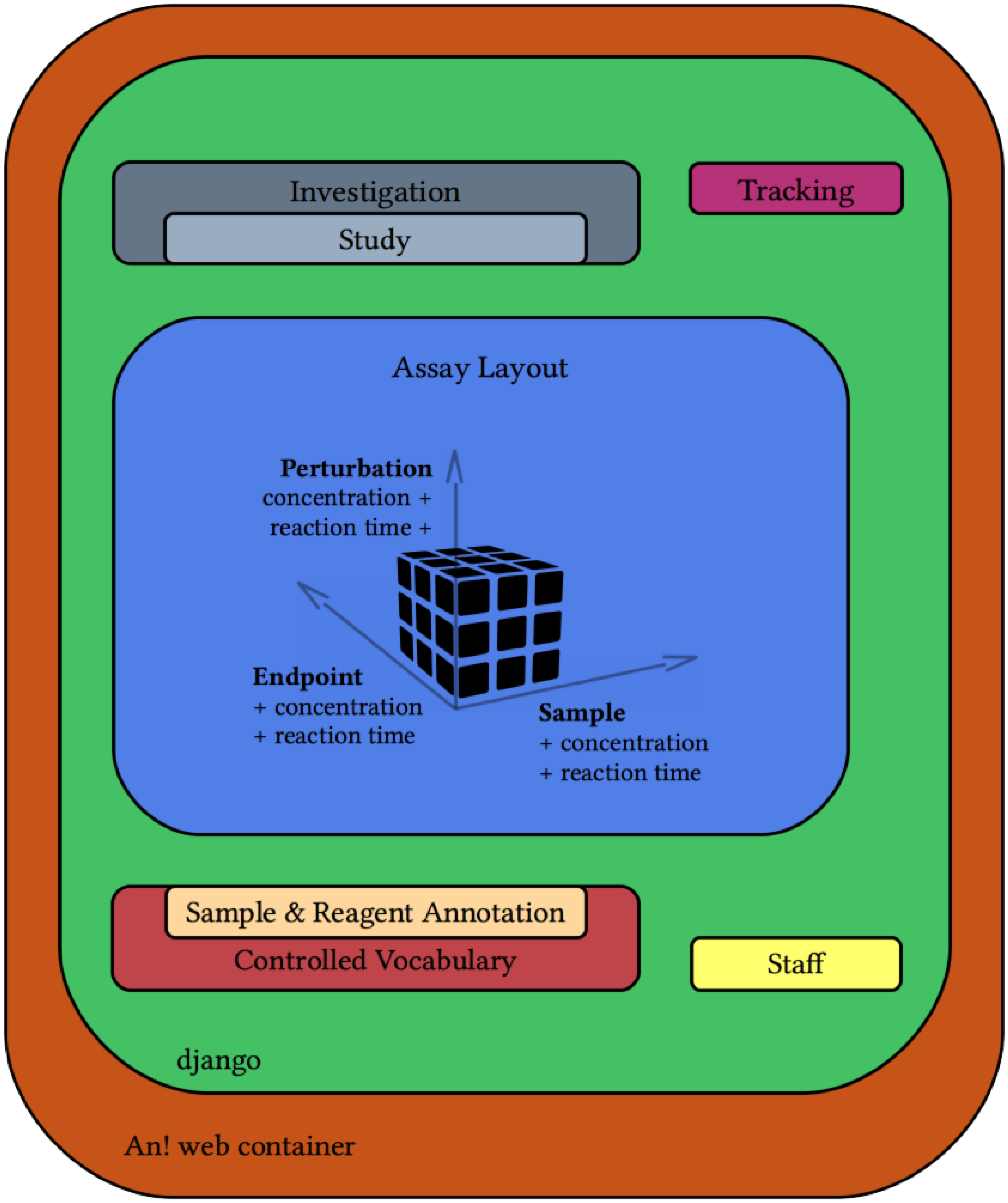
Annot's assay layout backbone is the acpipe_acjson library. Acpipe_acjson is a python3 library to handle the acjson file format, a file format developed to log complicated biological wet lab experiment layouts. Acjson file format complies fully withe the json standard.

3.5.3 Django

Annot is django based web application. Django as such is a python based web framework. Annot makes use of the django, django-admin - which is leveraged as annot's GUI (graphical user interface), and the external django-selectable library - which provides searchable dropdown list boxes to the django-admin based GUI. Particularly the lookups.py files found in several django apps are part of the django-selectable implementation.

If you not yet are familiar with djano and django-admin, then it's maybe a good idea to work through the tutorial from the official django documentation.

3.5.4 The folder structure inside the annot/web/ folder



An! django stack

The annot/web folder contains the actual annot code base. The django main project folder (prjannot) and all django app folders (app*) can be found here. The whole project is structured as following:

- prjannot: main project folder

- appon0investigation: investigation annotation app
- app1study: study annotation app
- app2tack: app for date tracking specific supersets or assay. this app have to be adjusted to the lab specific needs or can be commented out in the prjannot/settings.py file if not needed.
- app3runset: acjson based assay layout annotation app
- app4superset: acjson based superset layout annotation app
- appacaxis: acjson based sample, perturbation and endpoint layout annotation app
- appbrreagent*: annotation brick app for detailed reagent annotation.
- appbrsample*: annotation brick app detailed sample annotation.
- appbrxperson: app to store a list of scientist (staff) involved in the experiments logged by annot.
- appon*: controlled vocabulary app, one for each ontology.
- appsabrick: sample and reagent brick system administration app.
- appsavocabulary: controlled vocabulary system administration app.
- apptool: this app some command line and annot commands implemented for our lab specific need. this code can be adapted to the own lab specific needs, or the whole app can be commented out in the prjannot/settings.py file in not needed.
- nix: backup and maintenance related unix shell script and cron job code.

3.6 The man folder

The man folder contains this very documentation.

Documentation is mainly written in markdown, deployed via read the docs and generated using sphinx. Annot must be under your PYTHONPATH, to be able to be processable by sphinx.

If you would like to contribute on the manual, please read at least read the doc's getting started, get familiar with the basic of markdown, and check out Daniele Procida's "what nobody tells you about documentation" talk.

CHAPTER 4

Discussion

4.1 Why Annot?

Our overarching goal was to create a database to support the collection and access of controlled, structured experimental metadata to meet the needs of both computational and experimental scientists.

The common solution to this in biological research labs is to employ spreadsheets. While these benefit from being flexible and easily edited, they are subject to errors that result from manual entry, inadvertent auto-formatting, and version drift. Annot offers a robust solution to annotate - using controlled vocabulary - samples, reagents, and experimental details for established assays where multiple staff are involved. While Annot was written with an informatics agnostic end-user in mind, full system administration requires basic skills in Linux, Python3, and Django, as well as basic knowledge of relational databases. Because of the cost required to populate Annot with detailed sample and reagent annotation, it is most appropriate for large-scale, high-throughput experiments.

However, a major benefit to our approach is that data generated in different experimental settings can be integrated through a detailed description of each experimental condition along the dimensions of sample, perturbation, and endpoint. Moreover, the high cost of large-scale screening efforts warrants the time and effort required to adequately annotate it. Ultimately, approaches such as this will allow data to be better leveraged and utilized to make discoveries and biological insights.

4.2 About Controlled Vocabulary

Annot enforces controlled vocabulary for every sample and reagent. This means all terms used can be tracked back to controlled vocabulary from a specific ontology. Annot does this by creating an "annot id": an ontology term and id number linked together by an underscore (_). For example: "bovine insulin from UniProt" transforms into annot id INS_P01317. This approach helps limit variability in the nomenclature.

Annot id's are further restricted to use only alphanumeric character and the underscore. The official ontology identifier is found behind the last underscore.

If needed, the term part of the annot identifier can be adjusted to a term everyone in the lab is familiar with. The ontology identifier, on the other hand, should not be modified. For example: we changed the official term hyaluronic_acid_chebi16336 into HA_chebi16336. These types of changes should always happen before the term is used for sample or reagent annotation.

In the case needed terms are missing from a particular ontology, terms can be borrowed from an other ontology and added by clicking the `Add` button in the particular ontology (orange colored in the GUI).

We advocate the use of controlled vocabulary from existing, well established ontologies whenever possible. However, some terms do not exist in established ontologies. For example, all vocabulary from apponprovider_own. All of these terms will have "Own" as term id, so they are easily detectable. For example: Boots_Own for the boots pharmacy.

Should it ever happen that an id from an ontology becomes deprecated then the particular term will not be deleted. Instead in `Appsabbrick` (bright orange colored in the GUI) in the `Uploaded_endpoint_reagent_bricks` or `Uploaded_perturbation_reagent_bricks` or `Uploaded_sample_bricks` table the `ok_brick` field and the`ontology_term_status` field in the corresponding ontology (maroon colored in the GUI) will automatically be marked with a red cross.

In annot, the controlled vocabulary origin version contains the latest information pulled form the original source. Only the backup version will store adapted annot ids, our own added ontology terms, and deprecated ontology identifiers. Original version and backup files can be found inside annot at `/usr/src/media/vocabulary/`.

Further reading:

- HowTo handle controlled vocabulary?
- HowTo deal with huge ontologies?
- HowTo get detailed information about the ontologies in use?
- HowTo backup annot content?

4.3 About Proteins, Protein Isoforms, and Protein Complexes

Protein ids are a bit of a special case because some proteins have multiple known isoforms. For such cases we introduced an additional hierarchical separation character, the pipe symbol (|). For example, the canonical human insulin isoform: INS|1_P01308|1. Please note that in UniProt, the isoforms identifiers is officially separated by a dash from the protein identifier (e.g. P01308-1). Annot already uses the dash to separate the primary key parts in the annot brick identifiers which is why we adopted the pipe here.

If we could not identify an exact isoform, we always chose the canonical isoform as defined by UniProt. The boolean field isoform_explicit in the protein brick identifies which proteins have known isoforms and which simply use the canonical form.

Because UniProt doesn't cover protein complexes (i.e. COL1, ITGA2B1 or Laminin3B32), we used Gene Ontology cellular component identifiers, which resulted in annot ids like COL1_go0005584, ITGA2B1_go0034666, Laminin3B32_go0061801.

The unique annot id naming conventions make it very easy to spot key details about a protein. All details are not in the name, for example, the species the protein comes from (i.e. Human_9606 or Cow_9913). But this is annotated in the bricks in the `Protein_DNA_code_source` field.

Further reading:

- HowTo annotate protein complexes?

4.4 About not_yet_specified and not_available

Annot sets every empty field to not_yet_specified, regardless of whether information was not specified or the information was simply not available. This avoids the common problem of empty fields and confusion about how to handle missing data.

A sample or reagent brick, which has a not_yet_specified field in the primary key block, will in general not be uploadable. If however, primary key fields are marked as not_available, then we can upload the reagent brick. For example, if we do not have information about provider, catalog number, or lot number of the reagent DMSO, then we would have the following descriptor: DMSO_chebi28262-notavailable_notavailable_notavailable.

4.5 Programmer Contribution

- Elmar Bucher: main programmer.
- Cheryl Claunch: co-programmer on version 4.
- Derrick He: cron job backup routine implementation.
- Dave Kilburn and Laura Heiser: manual proofreading.

4.6 Contact Information

Contact Elmar Bucher at https://gitlab.com/biotransistor/annot or send an email to buchere at ohsu dot edu.

CHAPTER 5

GNU Free Documentation License

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.