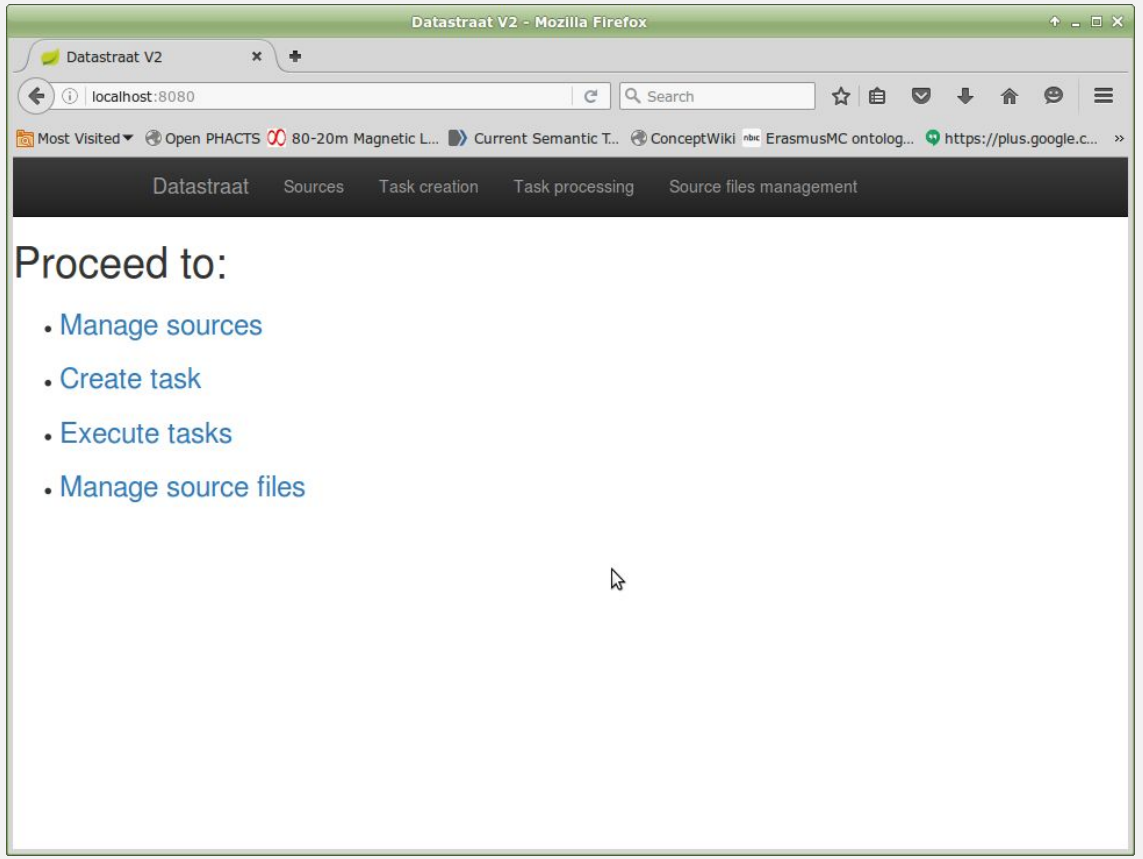
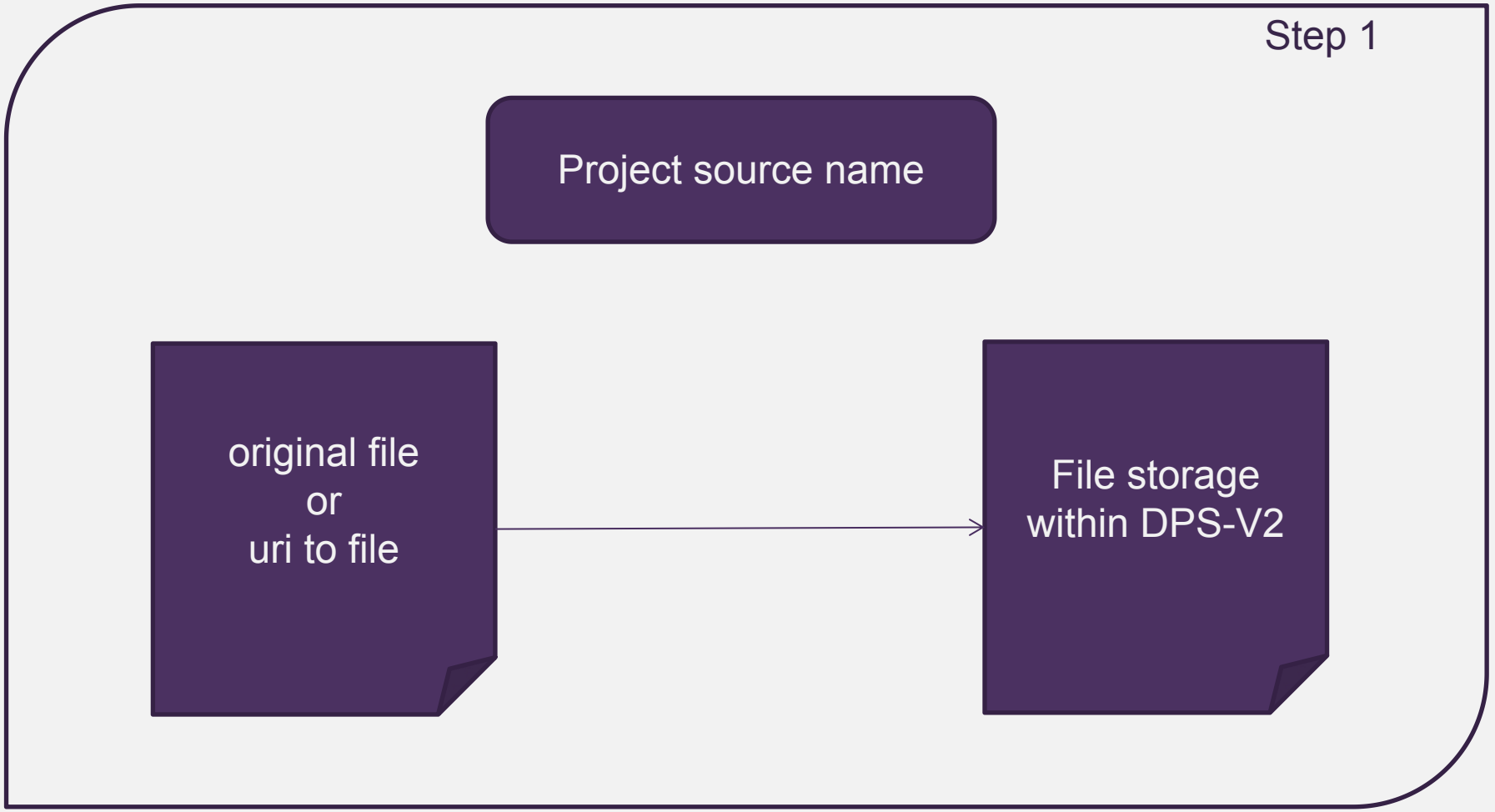
DPS V2

Our starting point for DPS

- ssh -f dps@178.63.49.197 -L 8082:127.0.0.1:8082 -N
- browser uri;
  - localhost:8082



Step 1; loading data into DPS

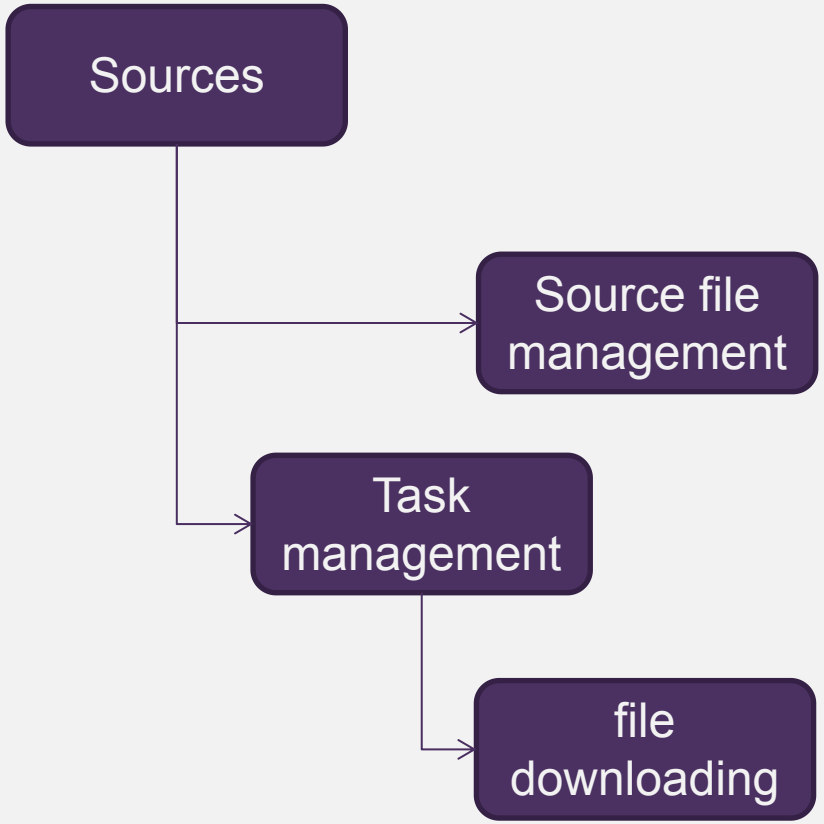


Uploading / downloading my data

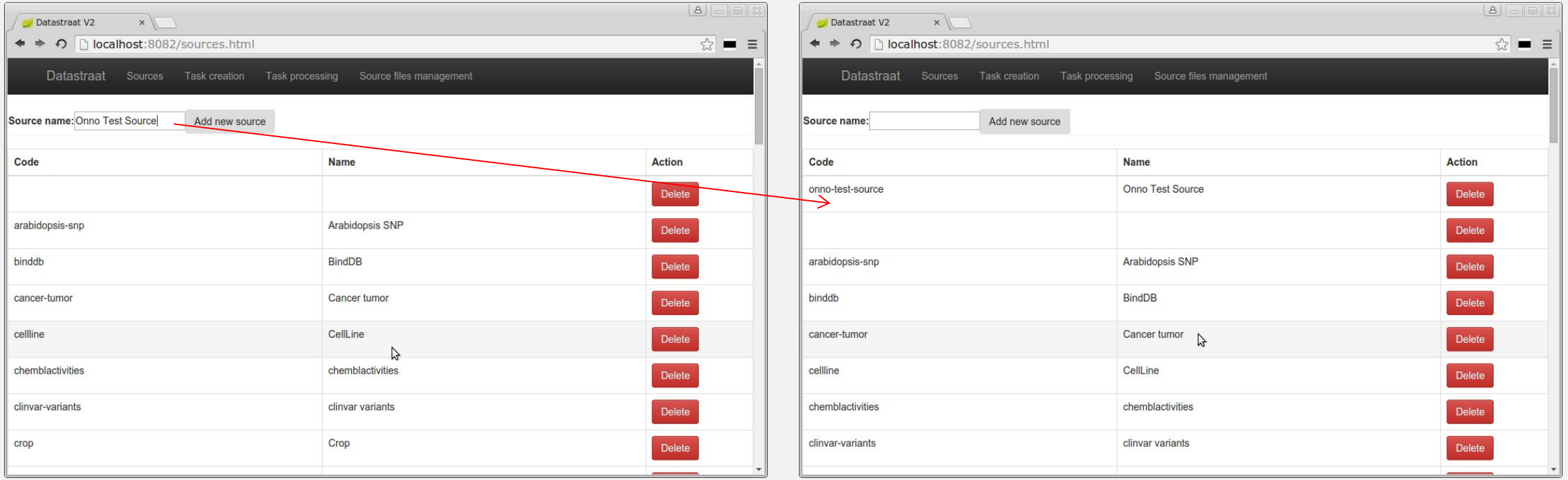
- Sources
  - give your project a reference name
    - Its a handle we use during other steps in subsequent processes
- Decide how you upload your data file
  - source file management, select input file, enter destination file name, upload

  OR

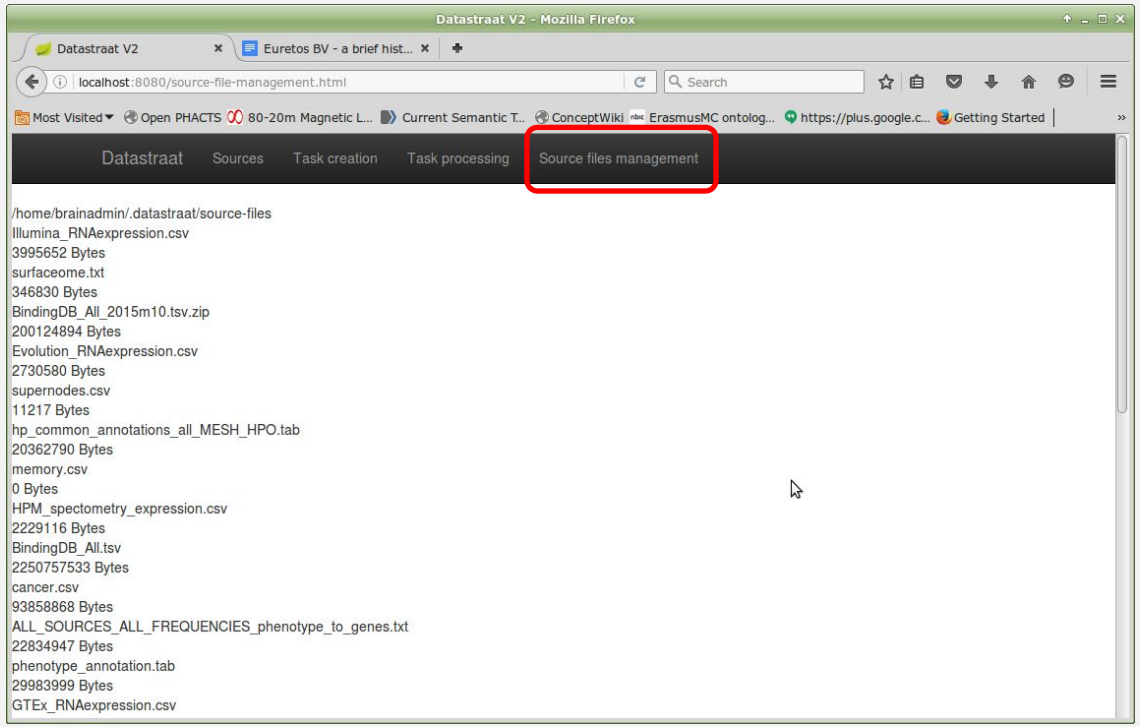
  - task management, file downloading through url definition

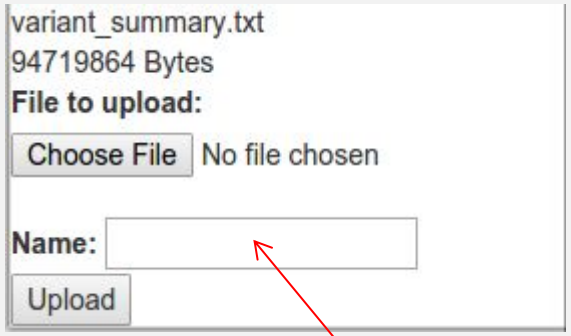


Add new source (project name)



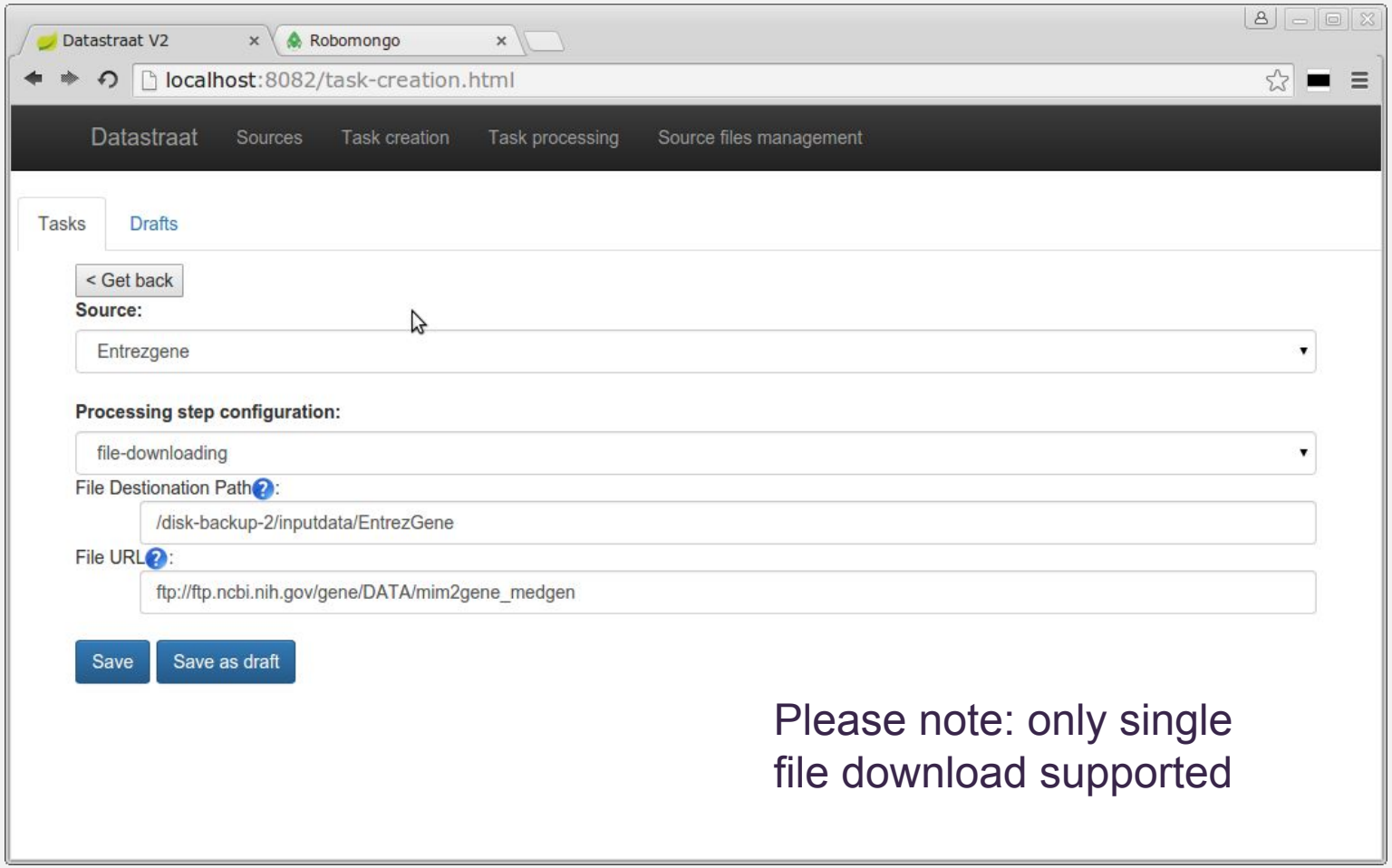
Source file management, *uploading* your file





Define the filename as used to store within DPS-V2

(or) Task management, *downloading* your file



Excercise step 1

Excercise step 1 (use your own name)

- Add a new source "Onno Test Source"
- Input file "onno.txt"

```
#subjectName, predicateName, objectName, pmid
"smoking","causes","cancer", "24950063"
smoking,relates,cancer,12907484
"excercise","prevents","cancer", 23665342
```

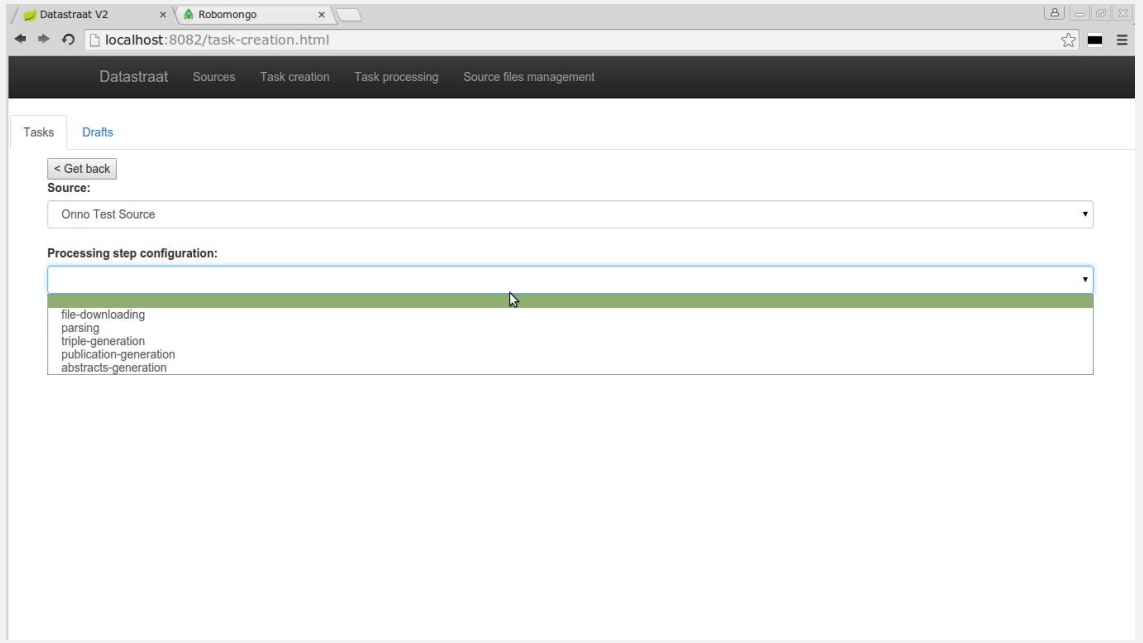
- Import it to the DPS server

Task management

Task management

- file downloading
  - step 1
- parsing
  - step 2
- triple-generation
  - step 3
- publication-generation
  - step 4
- abstracts-generation
  - step 5

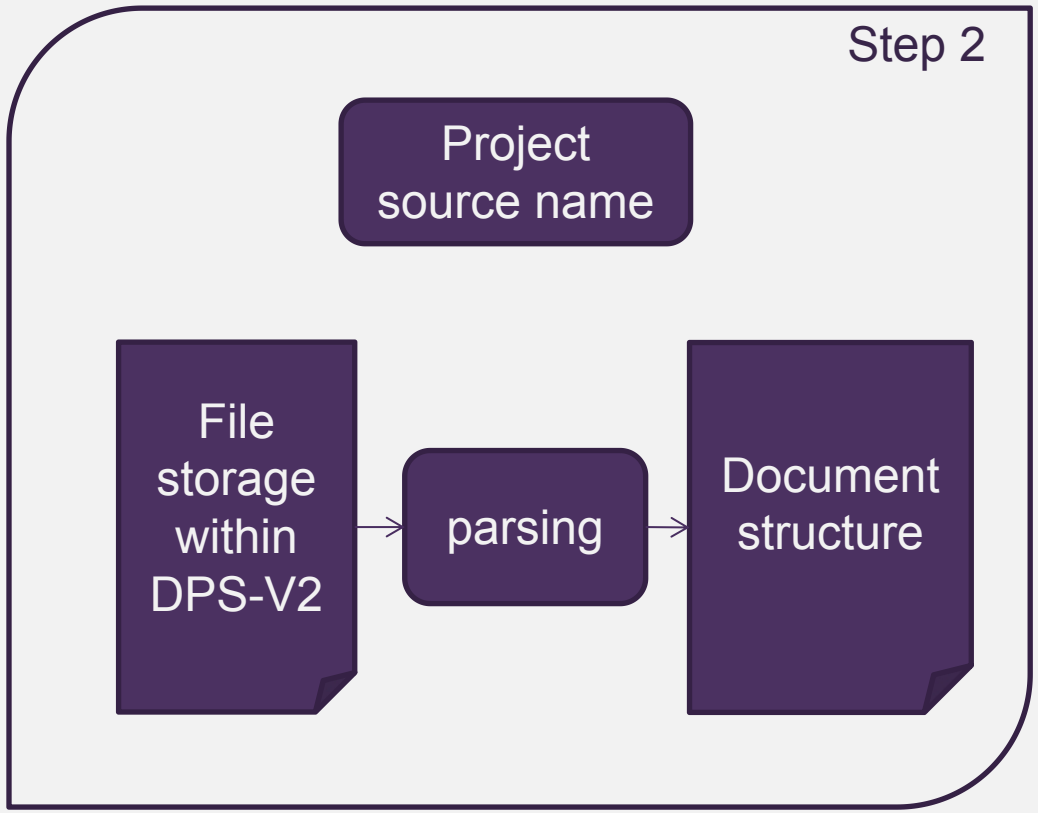
Training coverage



Parsing logic

Step 2; parsing data within DPS

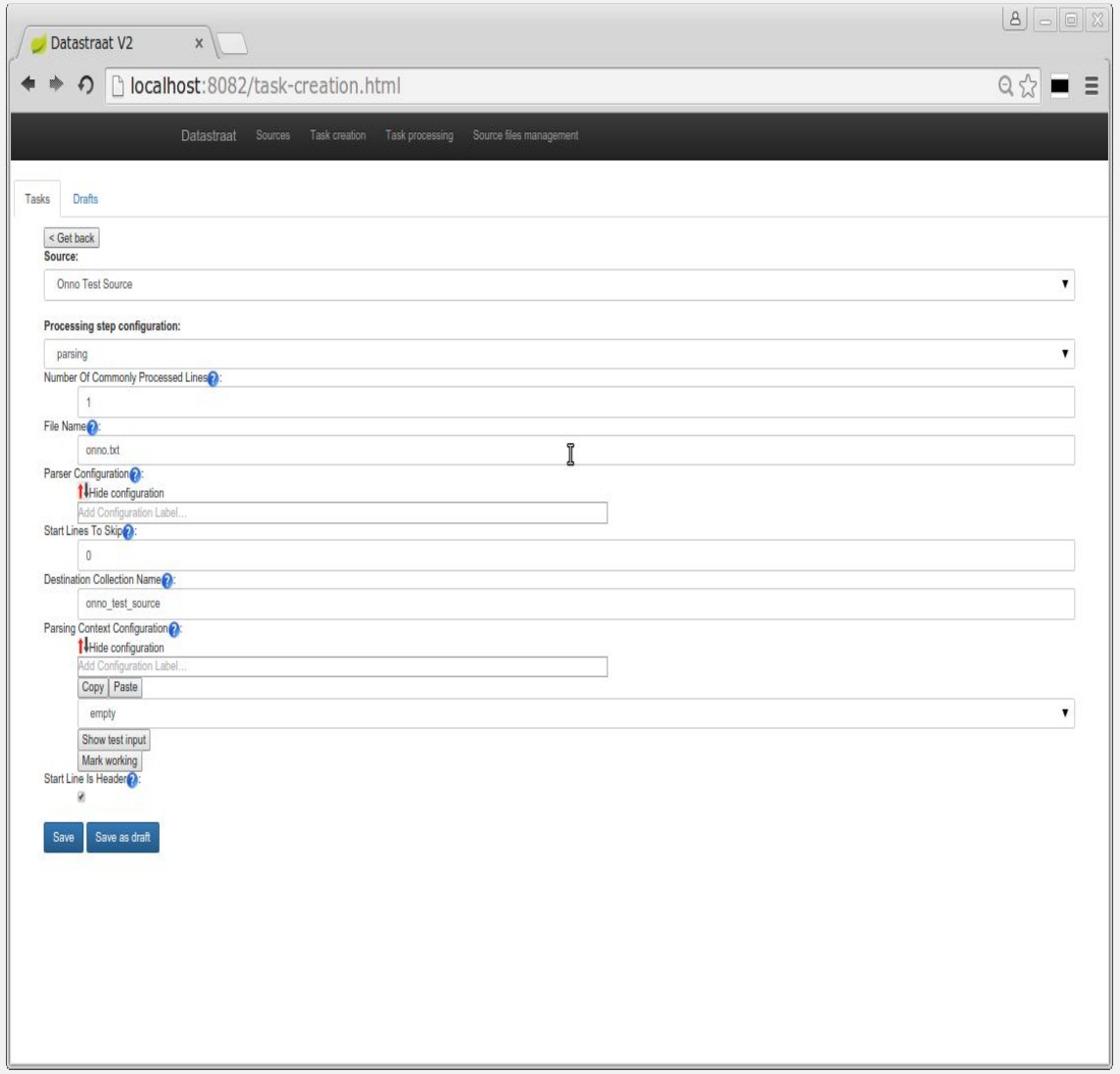
- The data file we stored on the server requires parsing to normalize into document format
- Its important to understand that all values are parsed as strings
- Parsing only transforms input to structured document format
- There is only one cleanup function, which is limited to removing header and / or tail of string line.



Creating the parser (step 2-1)

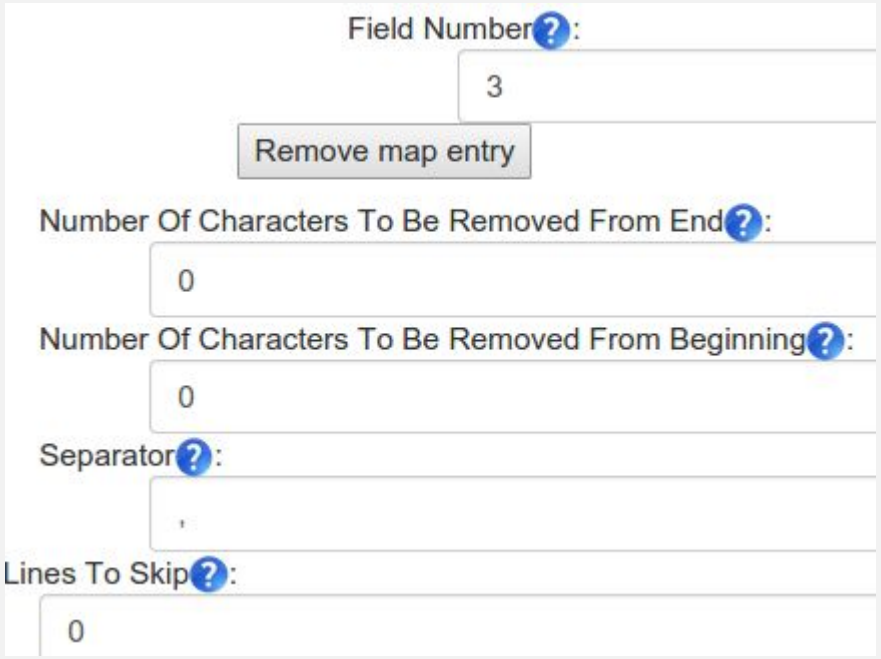
- select source (project)
- processing step; parsing
- select file name
- confirm first row is header true/false
- parsing context configuration; used for workflows, set to empty

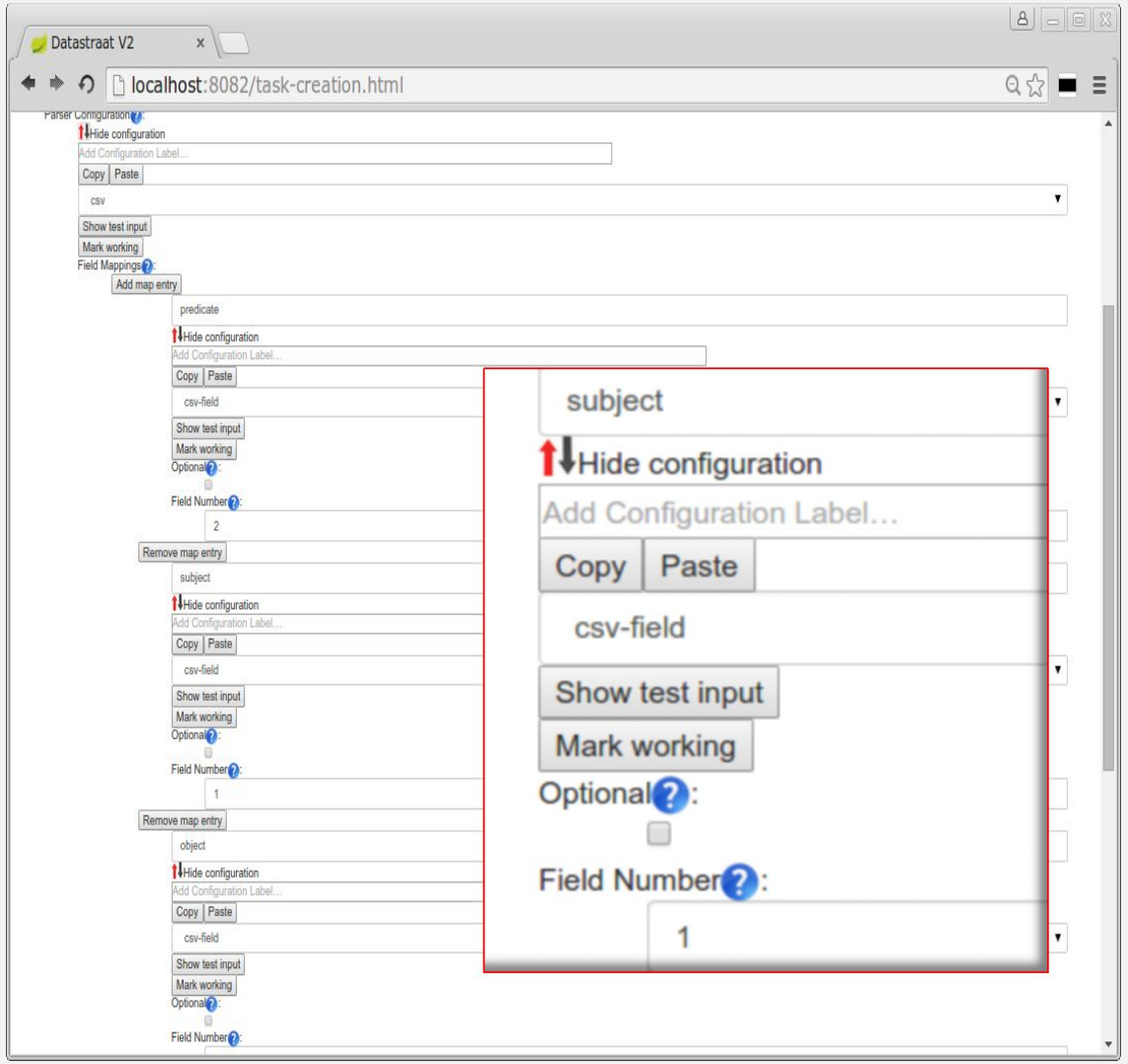
Validation function does not work for parsing



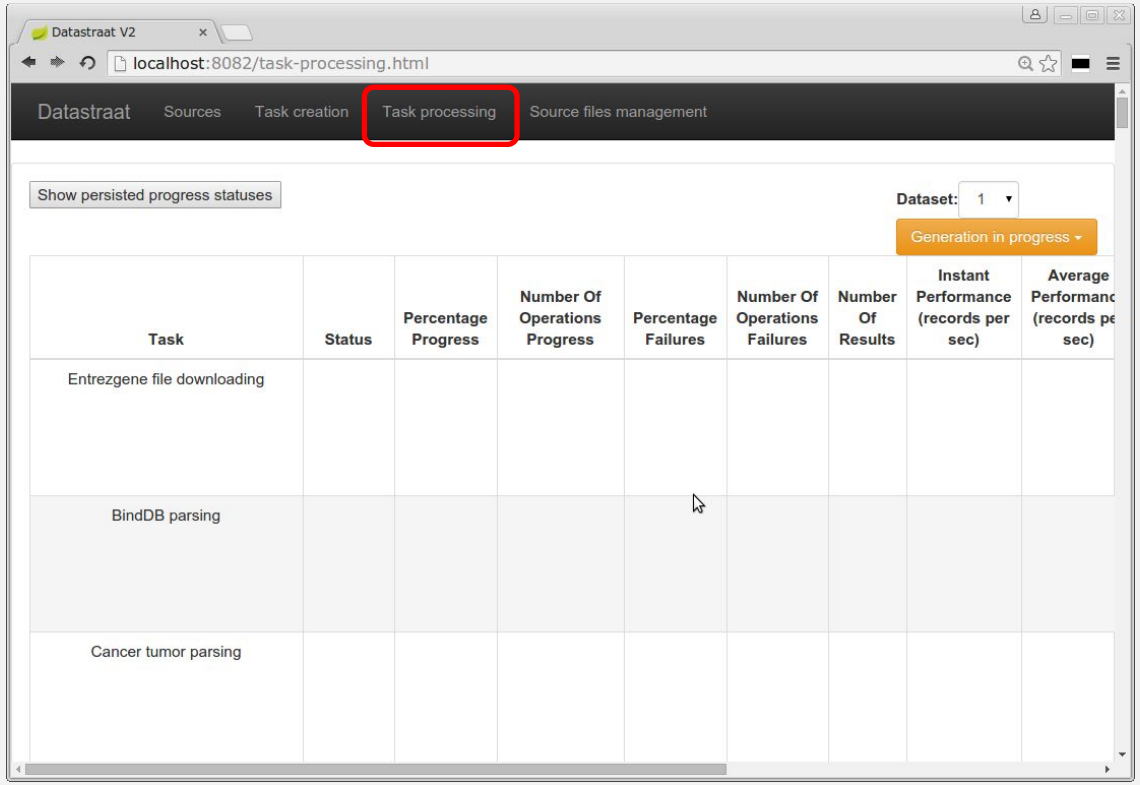
Creating the parser (step 2-2)

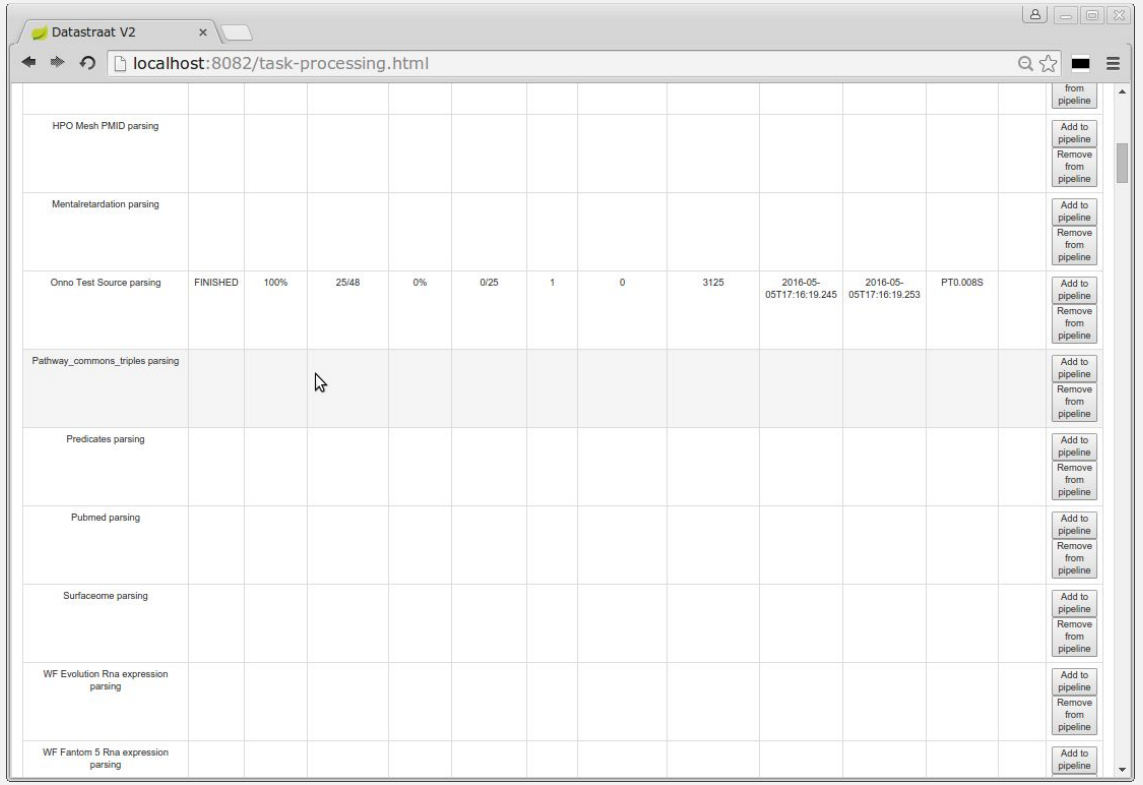
- Field Mapping definition
  - add map entry for each field



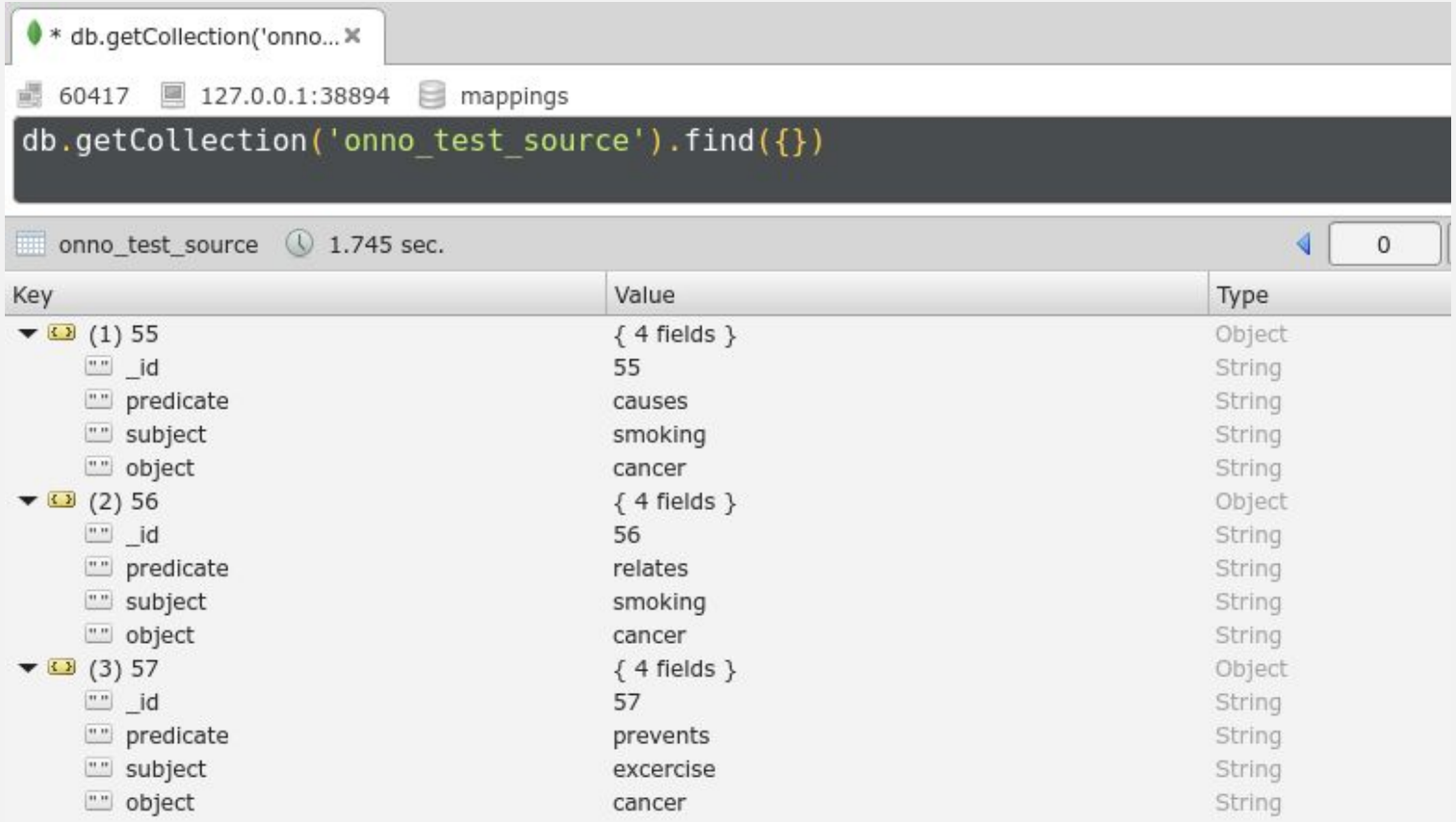


Execute the parser (step 2-3)





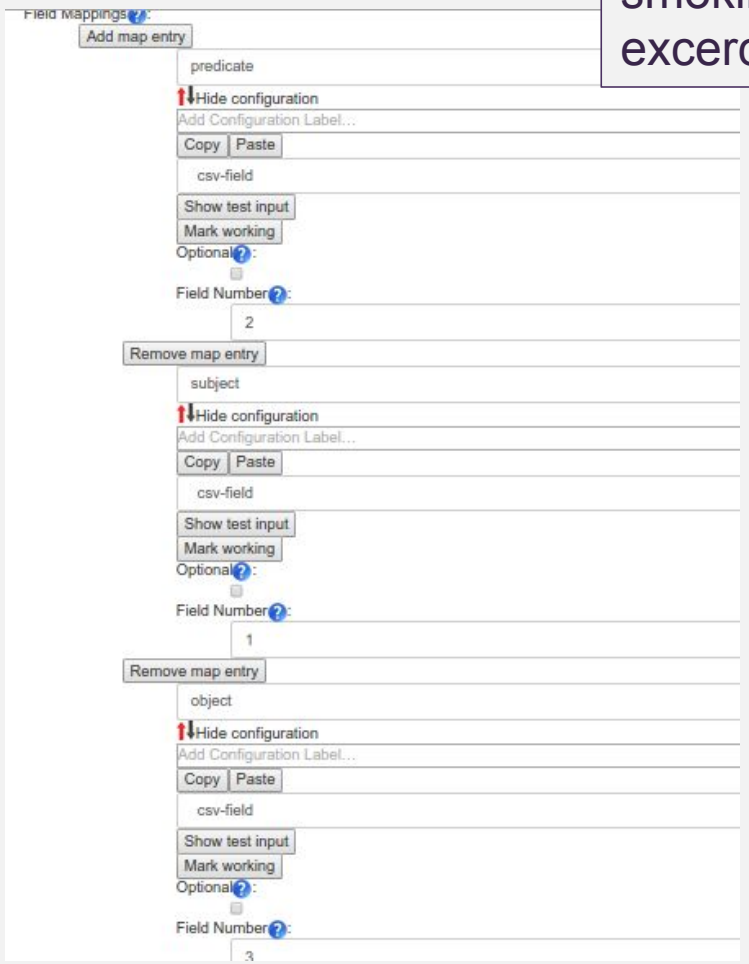
Resulting collection within documentstore



Summary parsing

CSV

```
#header stuff
smoking,causes,cancer
smoking,relates,cancer
excercise,prevents,cancer
```



This is document after we execute the task

```
db.getCollection('onno_test_source').find({})
```

onno_test_source 0.087 sec.

Key	Value
(1) 112	{ 4 fields }
_id	112
predicate	causes
subject	smoking
object	cancer
(2) 113	{ 4 fields }
_id	113
predicate	relates
subject	smoking
object	cancer
(3) 114	{ 4 fields }
_id	114
predicate	prevents
subject	excercise
object	cancer

csv-multi-line

```
"Gene","Gene name","Tumor","Level","Count patients","Total patients","Expression type"
"ENSG00000000003","TSPAN6","breast cancer","High","1","12","Staining"
"ENSG00000000003","TSPAN6","breast cancer","Medium","7","12","Staining"
"ENSG00000000003","TSPAN6","breast cancer","Low","2","12","Staining"
"ENSG00000000003","TSPAN6","breast cancer","Not detected","2","12","Staining"
"ENSG00000000003","TSPAN6","carcinoid","High","0","4","Staining"
"ENSG00000000003","TSPAN6","carcinoid","Medium","1","4","Staining"
"ENSG00000000003","TSPAN6","carcinoid","Low","1","4","Staining"
"ENSG00000000003","TSPAN6","carcinoid","Not detected","2","4","Staining"
```

example parser logic;
cancer tumor

```
db.getCollection('cancer_tumor').find({})
```

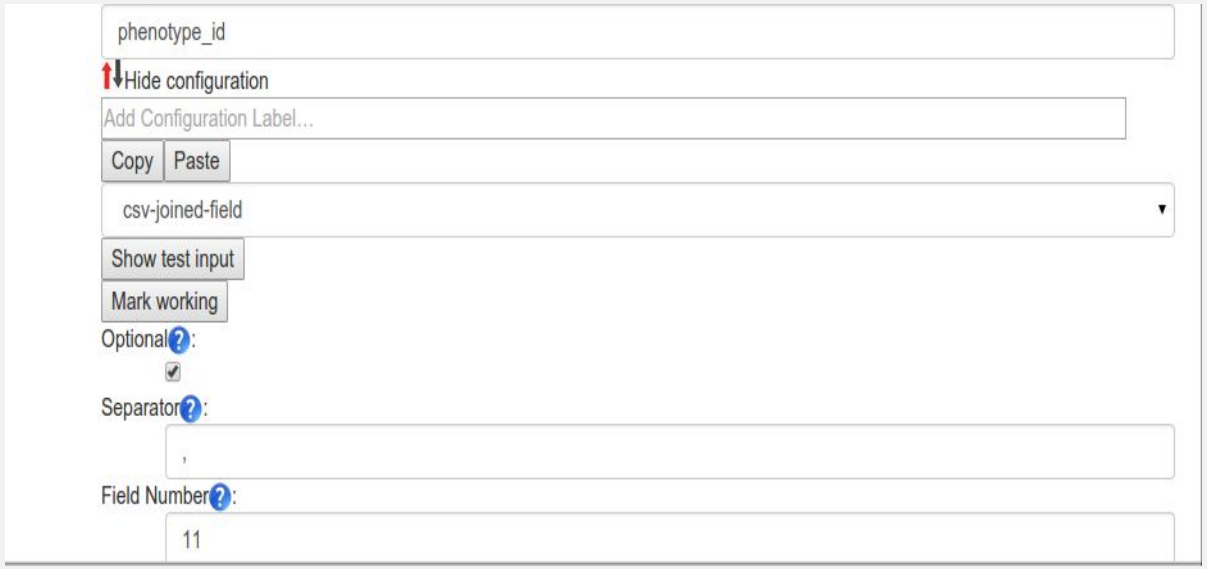
cancer_tumor 1.151 sec.

Key	Value
(1) 842465	{ 5 fields }
_id	842465
high	{ 6 fields }
expression_type	Staining
gene	ENSG00000000003
level	High
tumor	breast cancer
total_patients	12
count_patients	1
low	{ 6 fields }
expression_type	Staining
gene	ENSG00000000003
level	Low
tumor	breast cancer
total_patients	12
count_patients	2
not_detected	{ 6 fields }
expression_type	Staining
gene	ENSG00000000003
level	Not detected
tumor	breast cancer
total_patients	12
count_patients	2
medium	{ 6 fields }
expression_type	Staining
gene	ENSG00000000003
level	Medium

This is document after we execute the task.

csv-joined-field

```
15041 indel NM_014855.2(AP5Z1):c.80_83delGGATinsTGCTGTAAACTGTAACTGTAAA (p.Arg27_Ala362de
linsLeuLeuTer) 9907 AP5Z1 Pathogenic 397704705 - RCV000000012 N
MedGen:C3150901,OMIM:613647,ORPHA:306511 germline GRCh37 7 4820844 4820
847 7p22.1 no assertion criteria provided NM_014855.2:c.80_83delGGATinsTGCTGTAAACTGTAA
CTGTAAA NP_055670.1:p.Arg27_Ala362delinsLeuLeuTer 1 Jun 29, 2010 - OMIM
Allelic Variant:613653.0001 2 GGAT TGCTGTAAACTGTAACTGTAAA 1 NC_000007.13
```



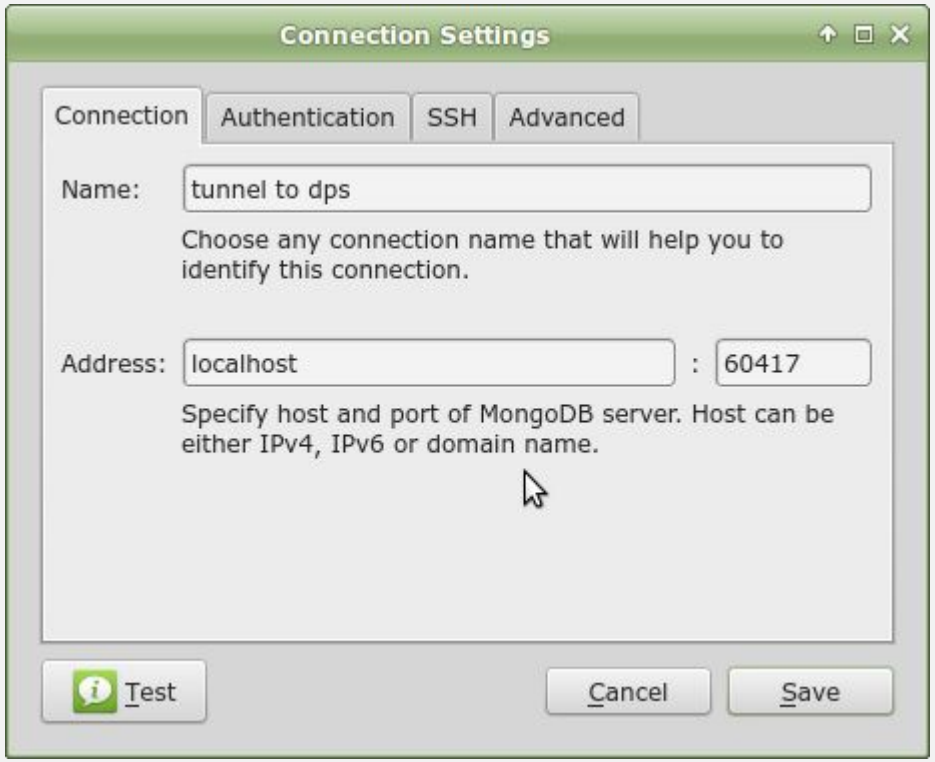
example parser logic;
clinvar_variants

Excercise step 2

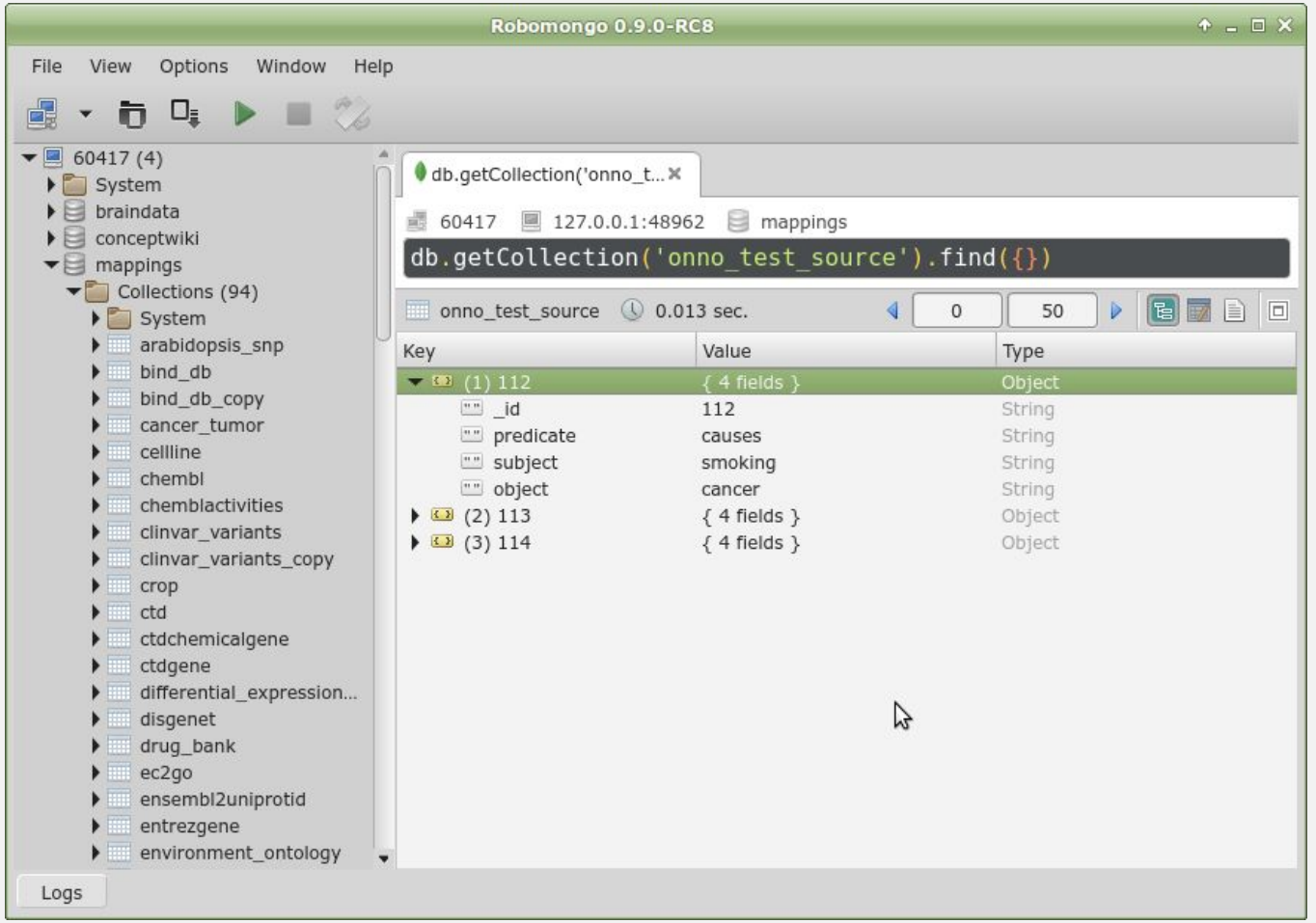
Excercise step 2 (use your own project)

- Create your own parsing
- Execute as task
- Check resulting document using Robomongo

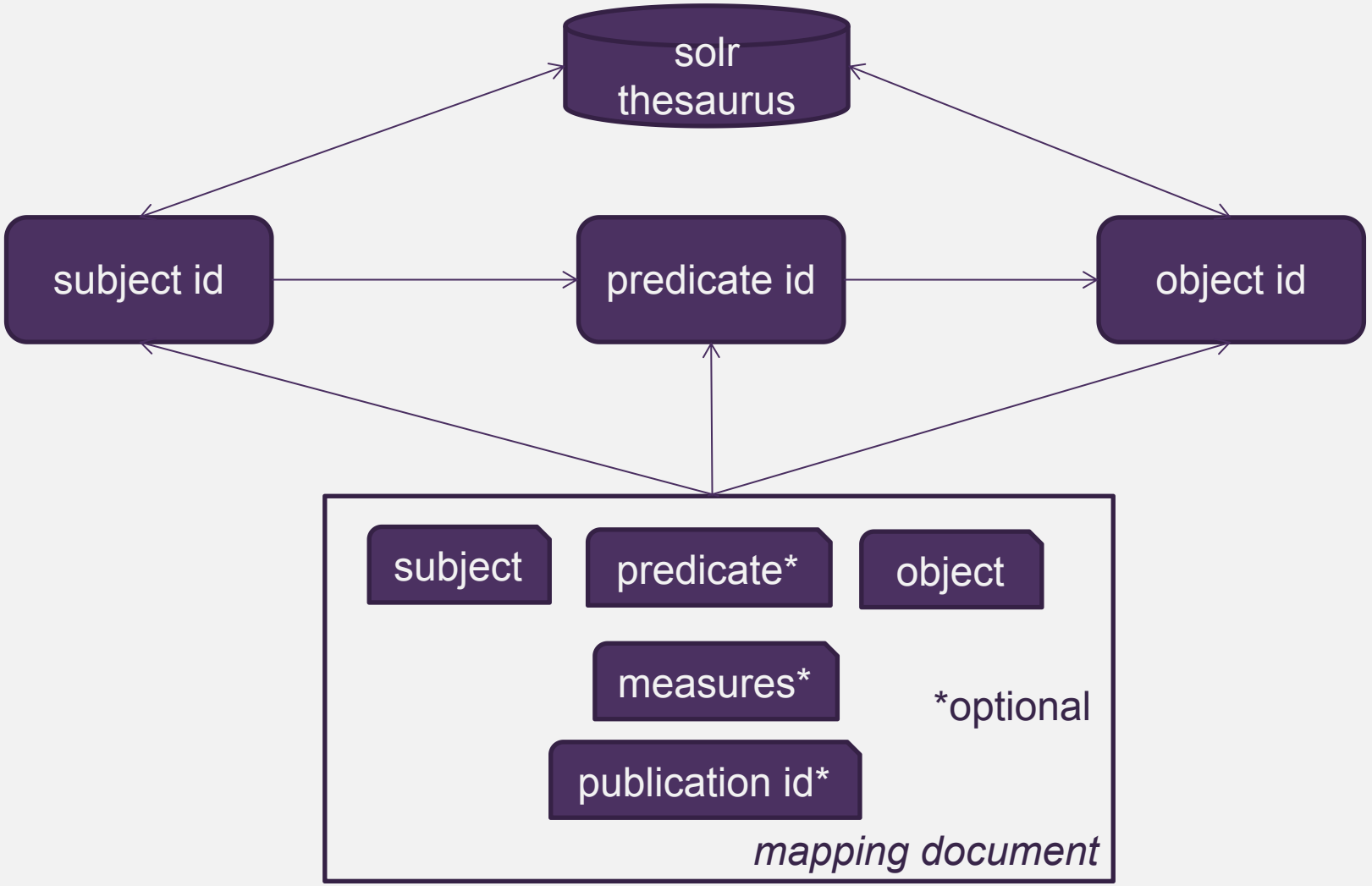
Configuration robomongo



Robomongo view

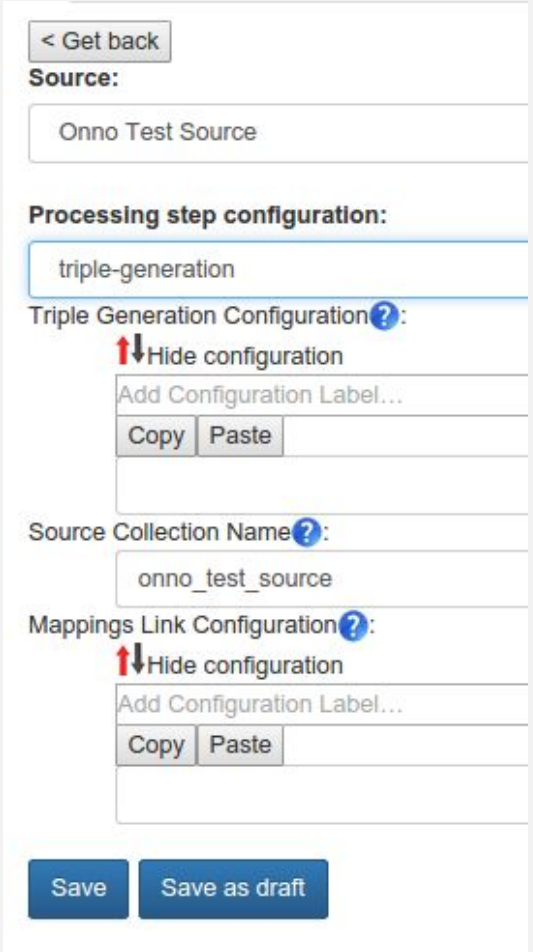


Creating triple generator logic

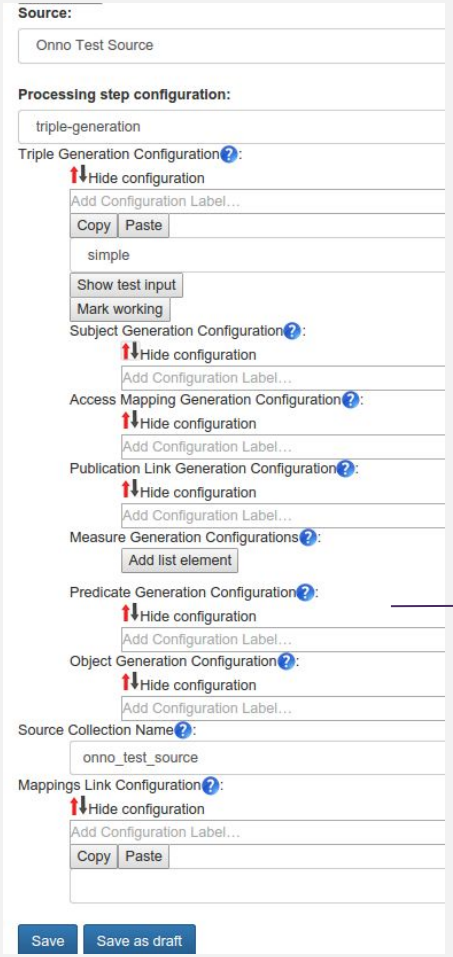


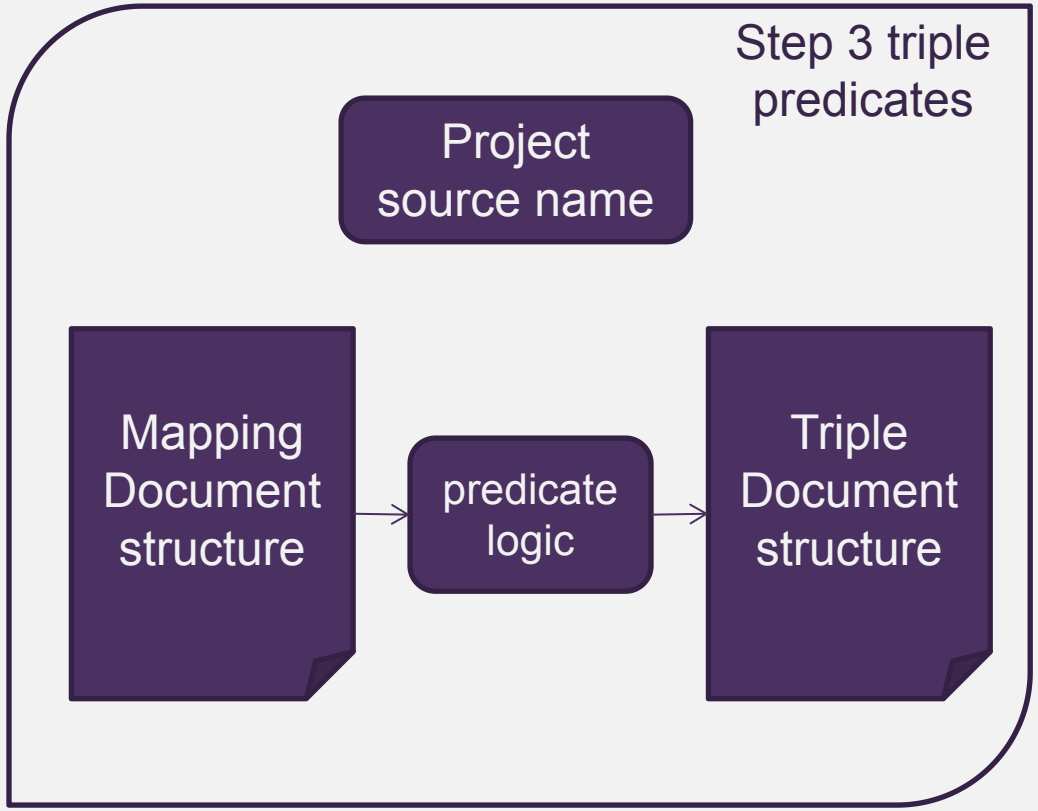
Main configuration elements

- Confirm the source collection name
- Mapping link configuration:
  - in case triple and publications are derived from same source collection, you need to confirm which field is used as shared identifier
- Triple Generation Configuration... see next slides



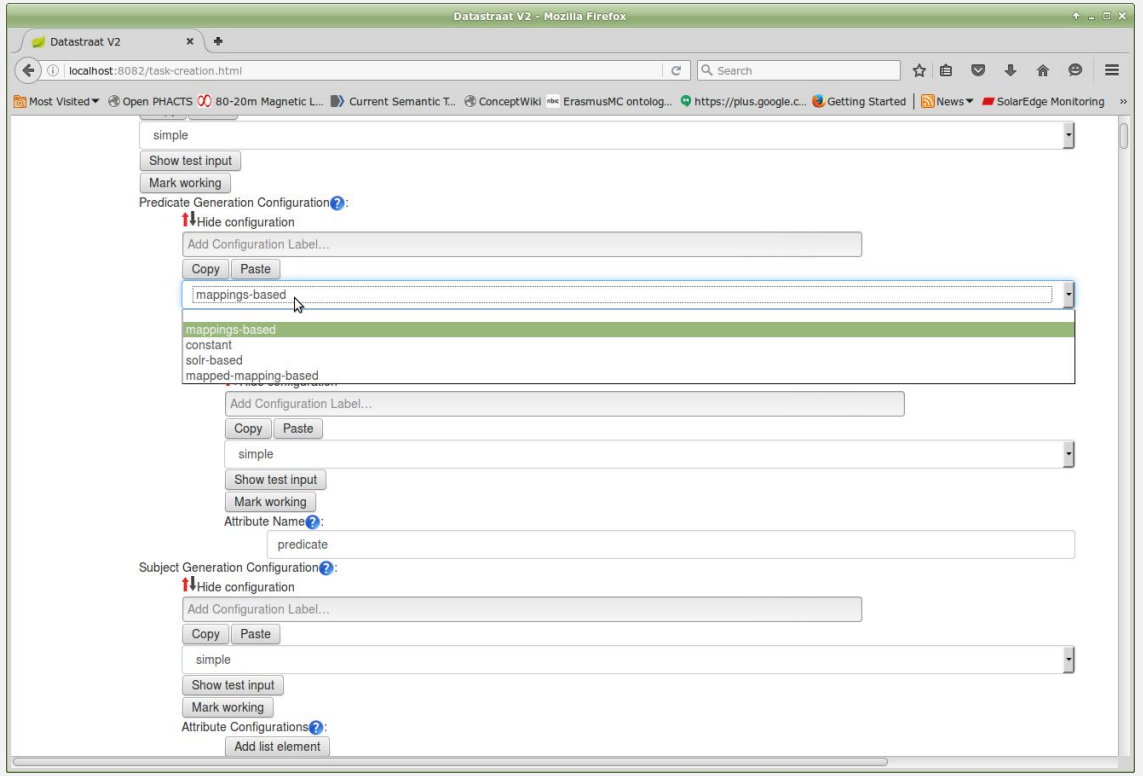
Predicate



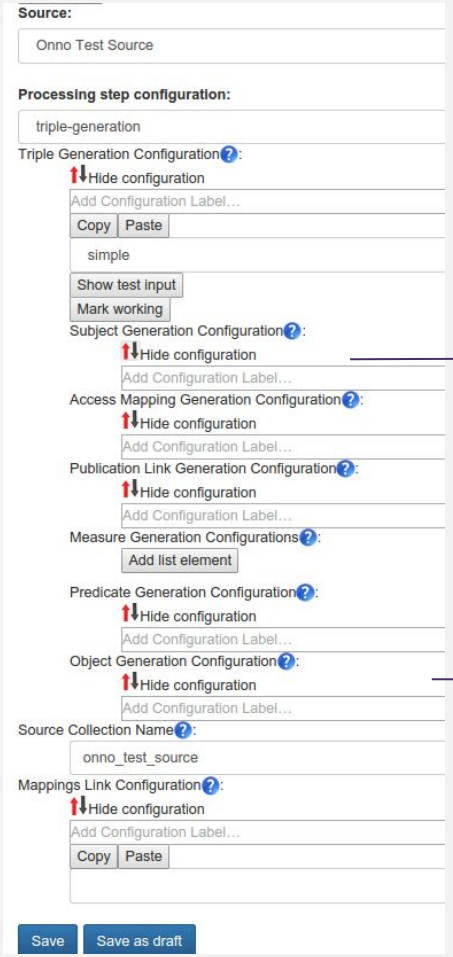


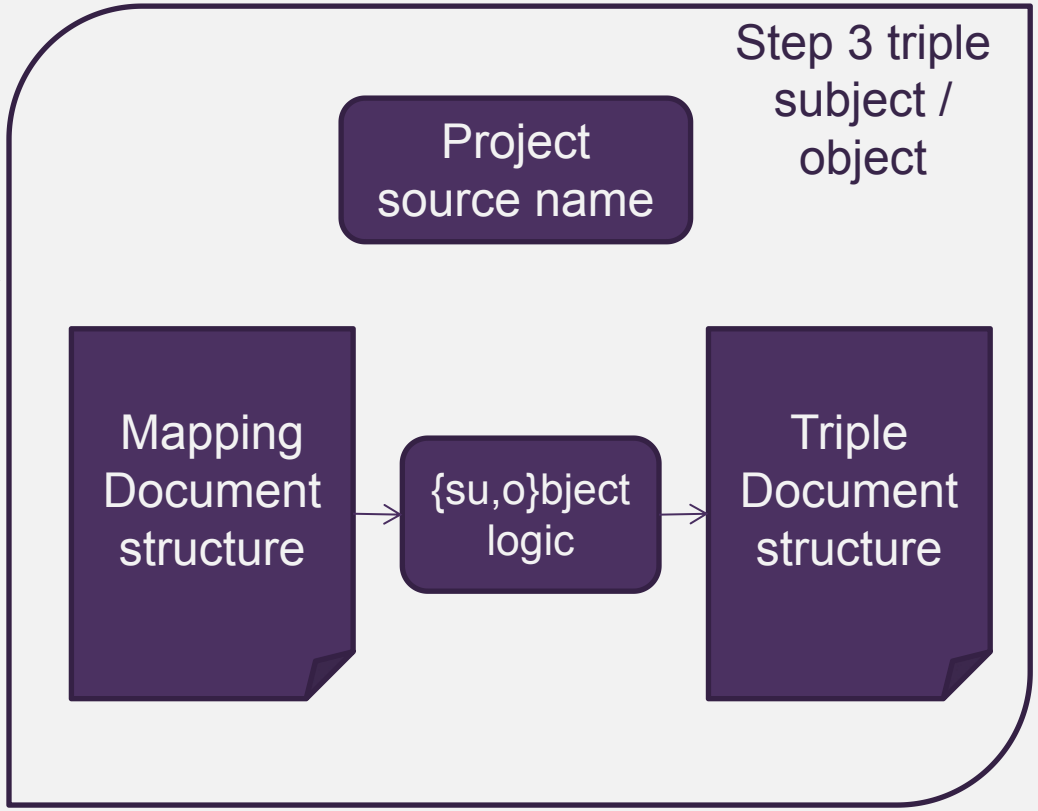
Predicate assignment

- Assignment / creation of predicate can be based on;
  - mapping-based
    - retrieve predicate name from mapping collection
  - constant
    - statically assigned by user into predicate name field
  - solr-based
    - solr request / response determines which predicate will be assigned
  - mapped-mapping-based
    - take a value from mapping collection and depending on value returns predicate name



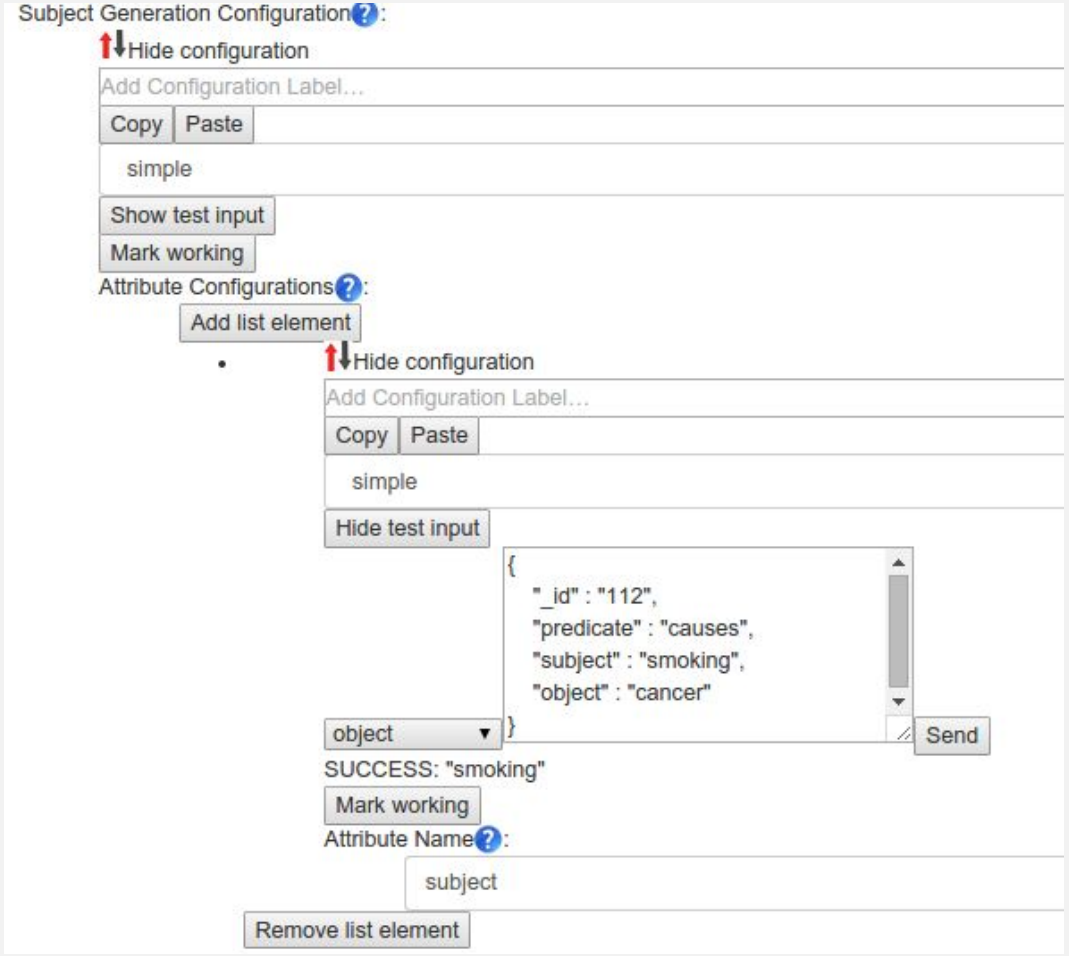
Subject / Object



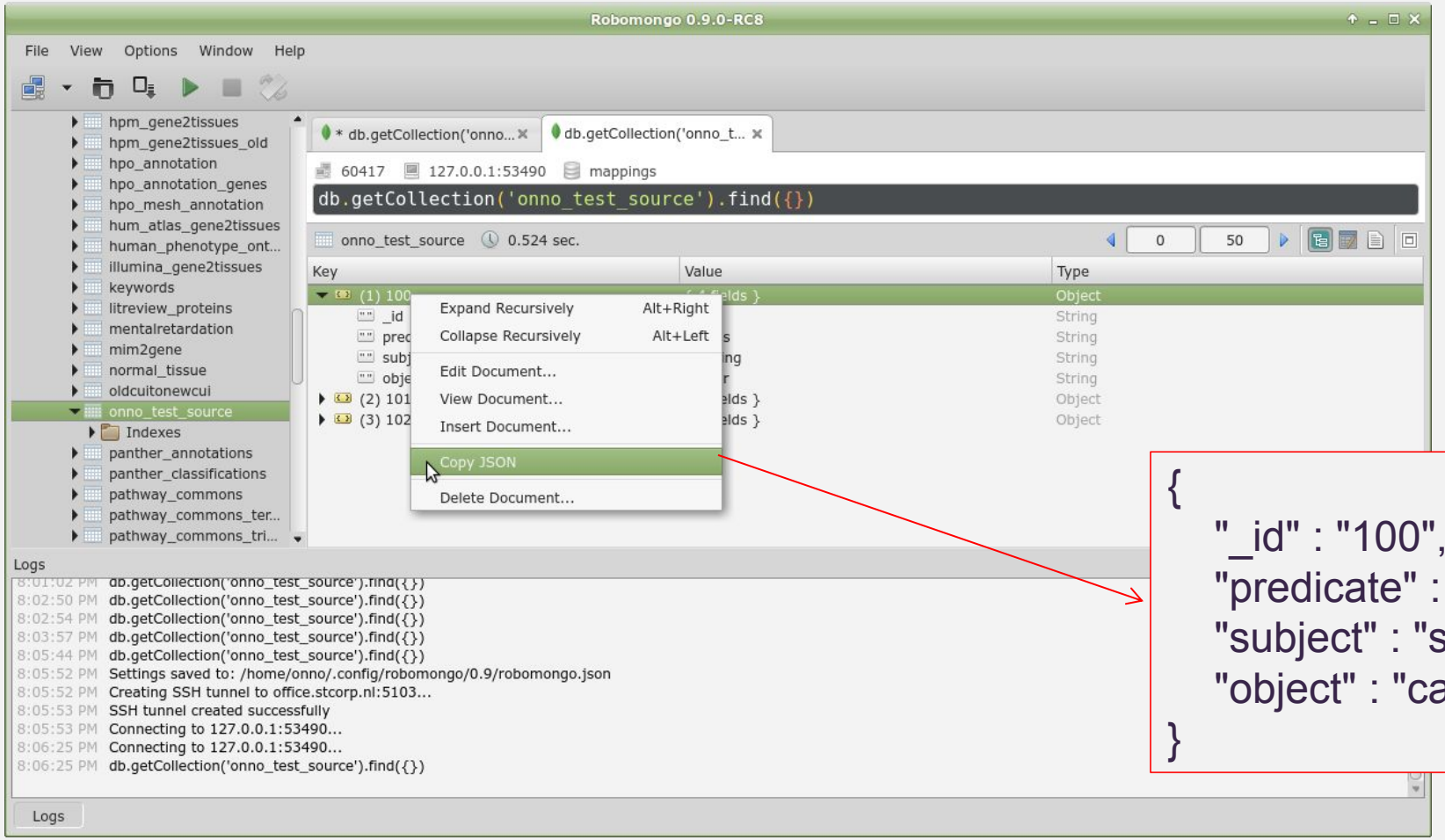


Subject / object -> retrieve values

- use simple to keep it simple
- attribute name is subject or object depending case
- you can test mapping field through copy / paste json object (see next slide)

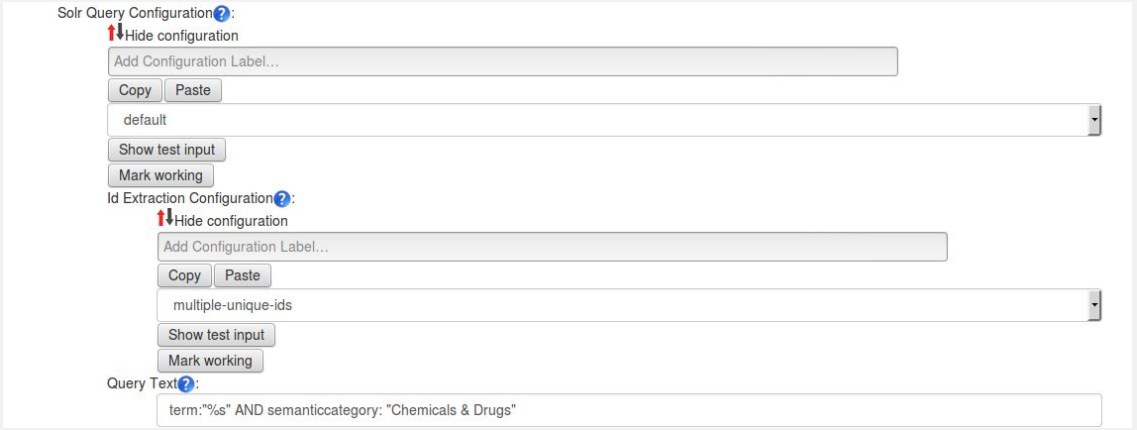


Robomongo (copy json)



Subject / Object -> Solr query

- Example solr queries:
  - term:"%s"
  - term:"%s" AND semantictype:"28" AND taxonomies:"arabidopsis thaliana"
  - (term:"%s" AND semantictype:"116")
  - (term:%s" AND (source:"umls" OR source:"uniprot"))
- Good idea to use your API access to collect key, value pairs?



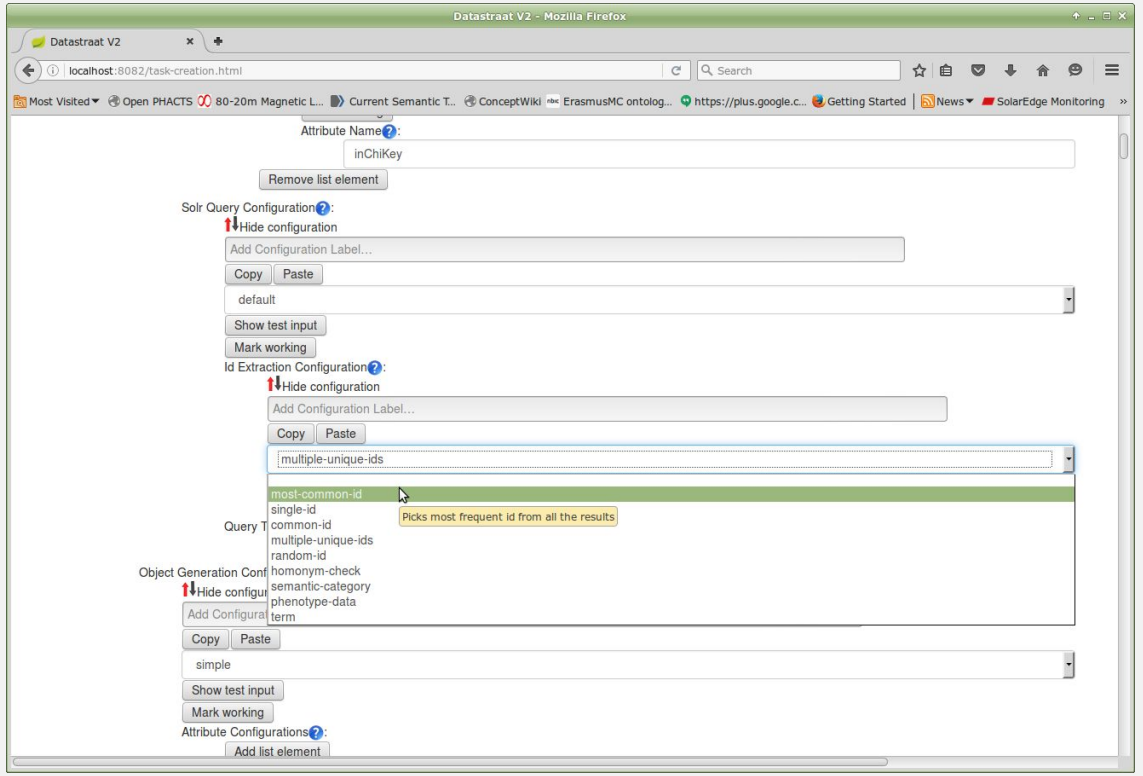
Note: keep in mind that semantic type value is numerical. This is different than api solr semantic type value

Subject / Object -> Solr response

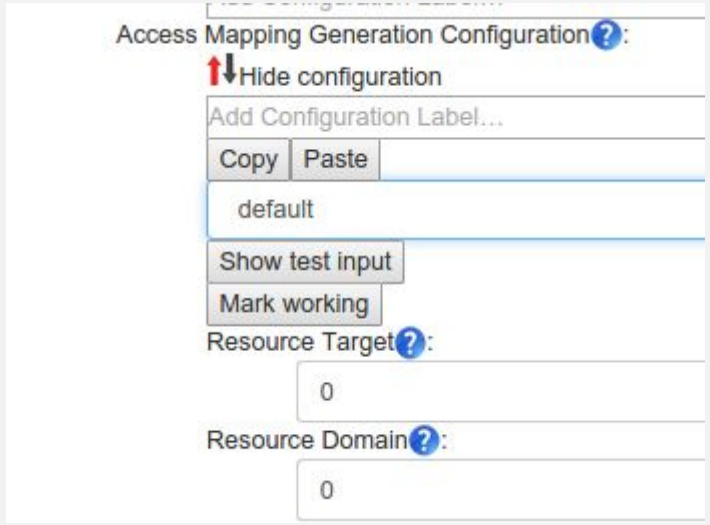
- what to do if (multiple) concept(s) are returned?

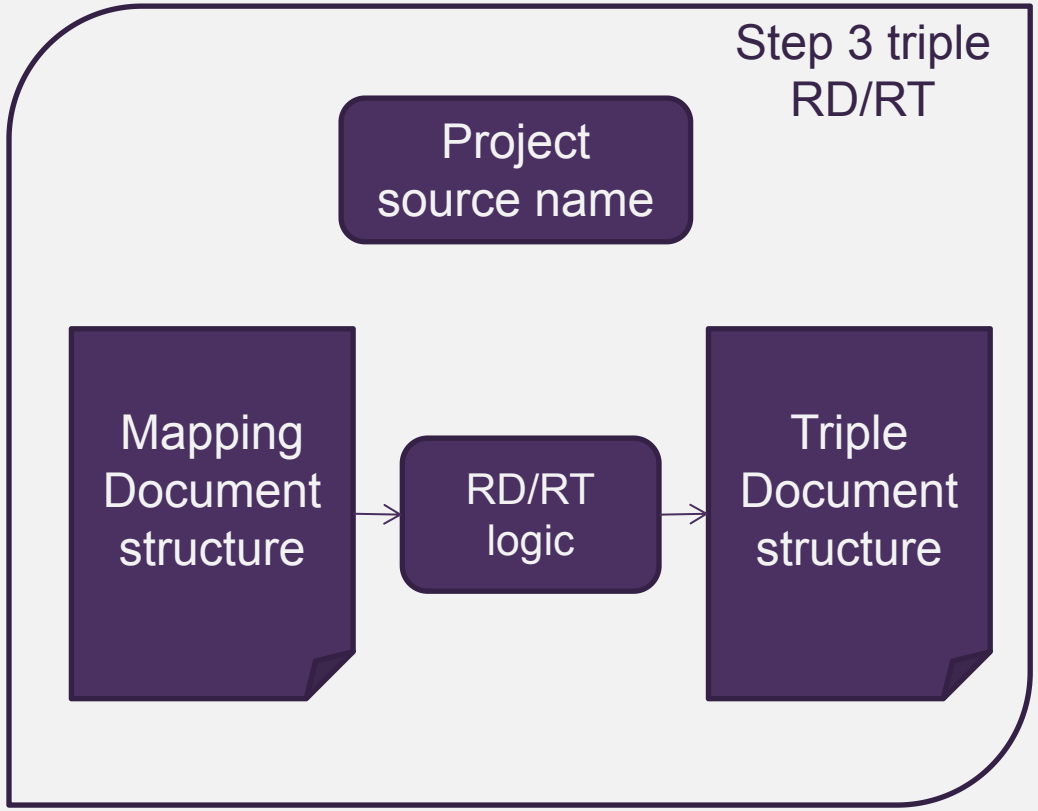
- most-common-id
  - in case multiple ids are found, use the id with most documents
  - when tie, we pick one
- single-id
  - we expect single id to use.
  - if multiple, skip resolution
- multiple-unique-ids
  - in case multiple ids are found, use them all
- random-id
  - in case multiple ids are found, pick one
- homonym-check
  - in case multiple ids are found, pick the most likely one
  - when tie, skip resolution

- Special cases (dont use);
- semantic-category
  - when match, return semantic category
- phenotype-data
  - when match, return phenotype data
- term
  - when match, return preferred term

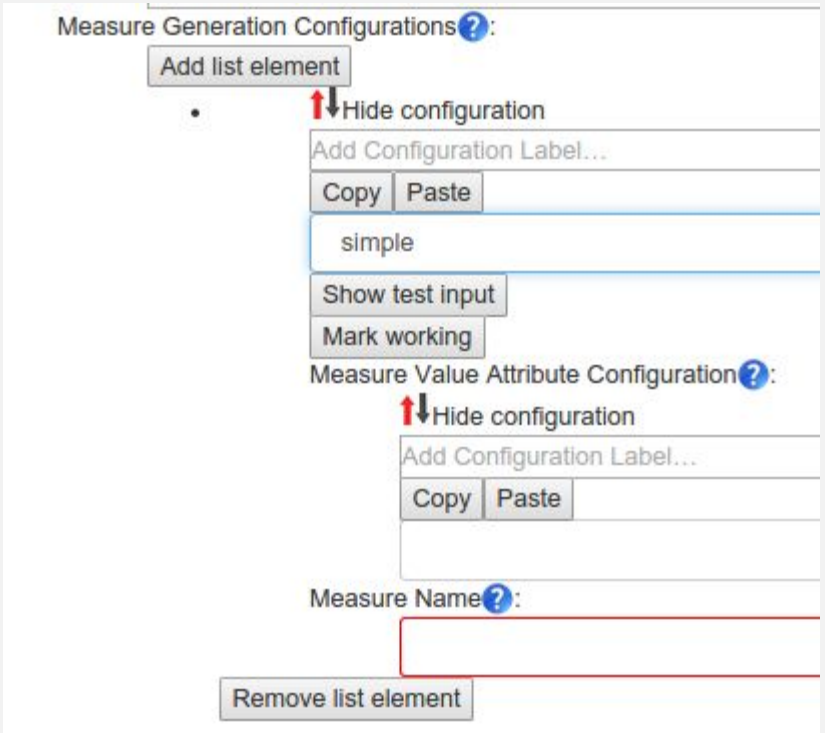


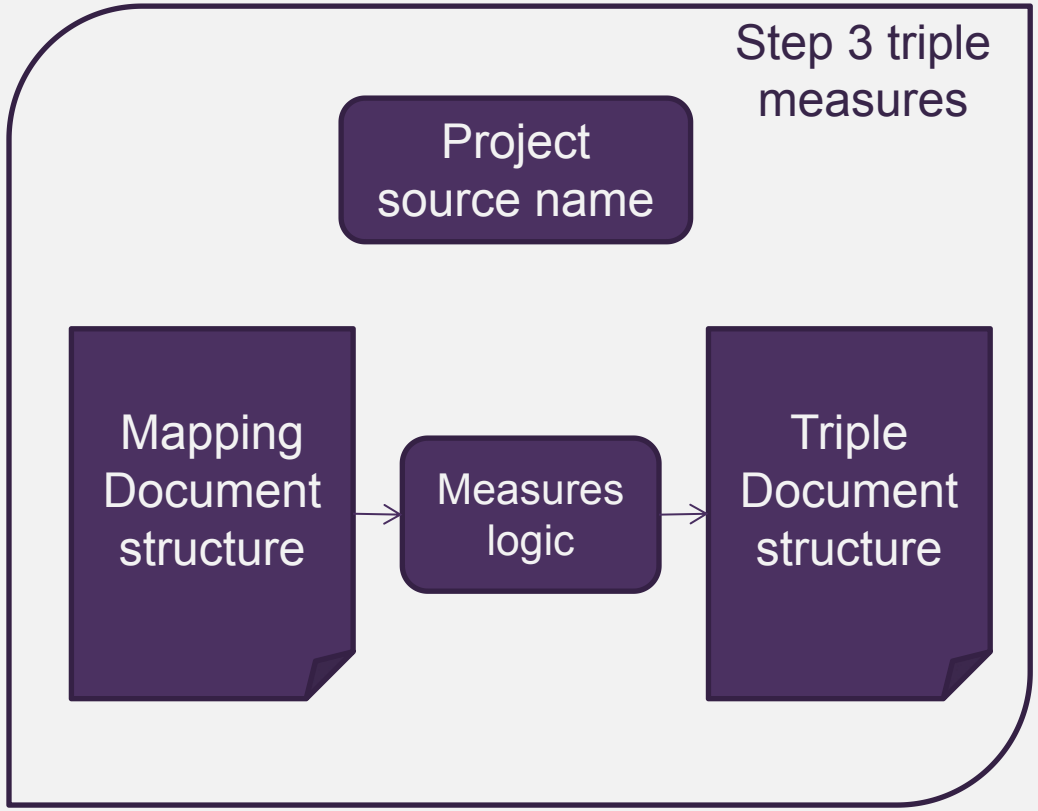
Access Mapping



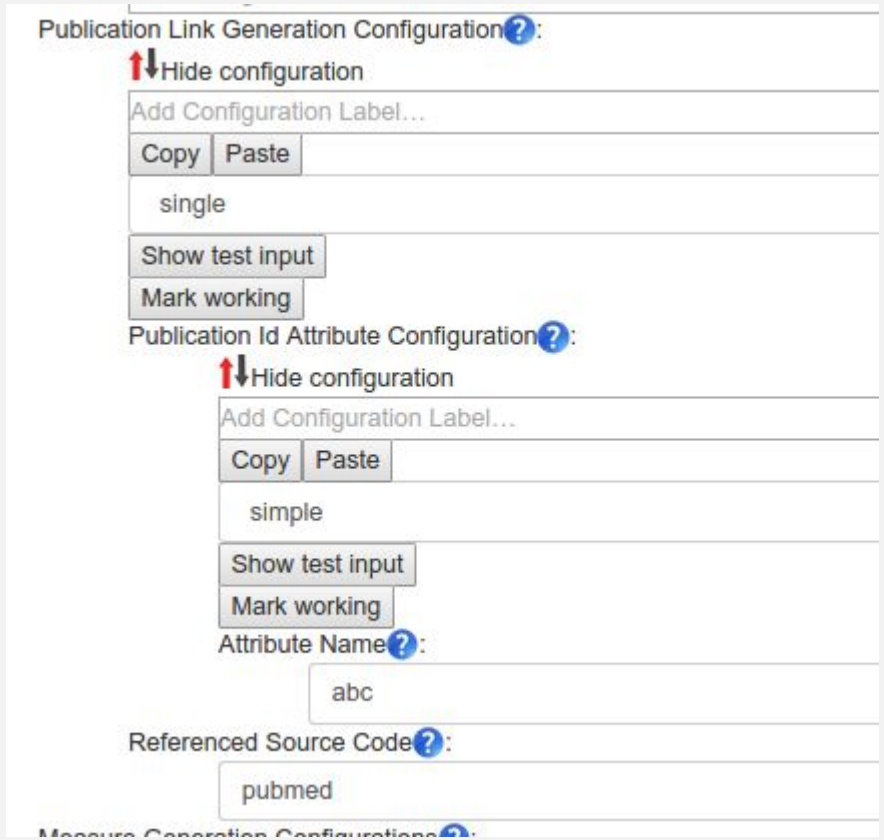


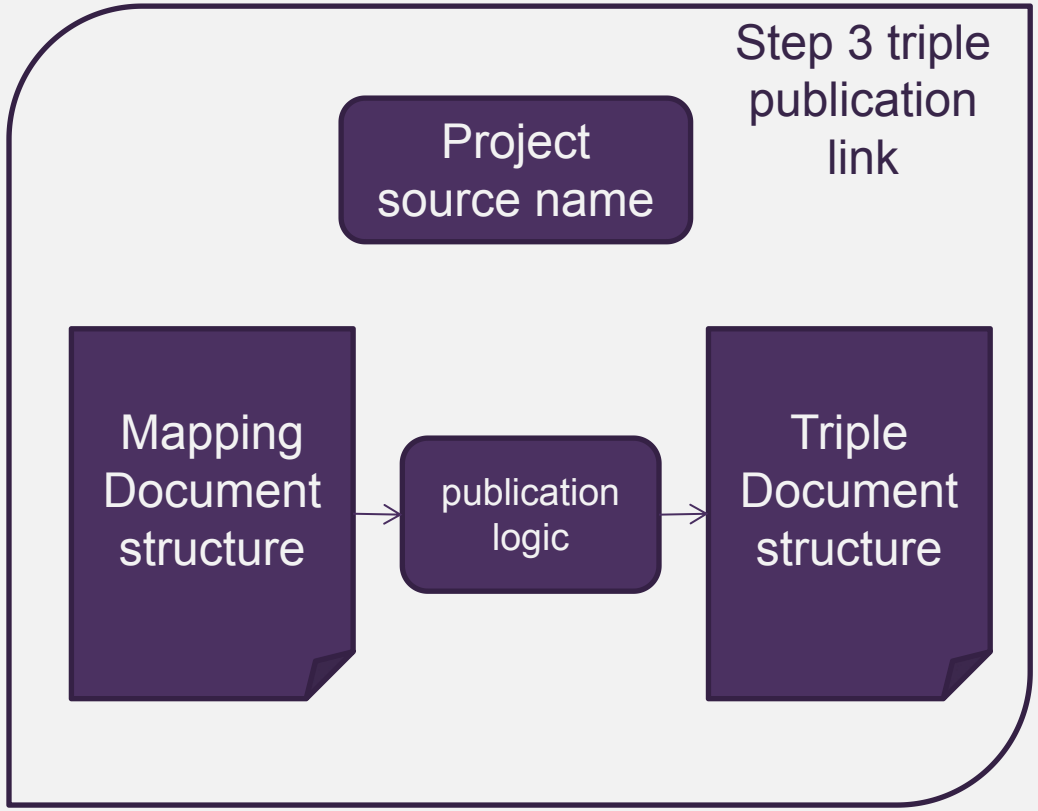
Measures





Publication links





Publication generation