CanDIG Documentation

Release 1.1.0

CanDIG Team

Dec 10, 2019

CONTENTS

This the documentation for version 1.1.0 of the candig-server.

The candig-server is a CanDIG project, and can be installed via `pip install candig-server`.

If you need help or want to report a bug, please refer to our *Contribution Guideline*

CHAPTER

ONE

SETUP

1.1 Introduction

Warning: This documentation is a work-in-progress.

The candig-server is a CanDIG project, aiming to provide distributed, secure data access to participating institutions across Canada. You can learn more about our work from https://www.distributedgenomics.ca

It has been built on the ga4gh-server with extended functionalities including native support for peer-to-peer communications, clinical and pipeline metadata support, as well as the upgraded codebase, written in Python 3.

1.2 Prepare Data For Ingestion

Warning: This part is a work in progress.

The candig-server has a general guideline on which data are accepted.

The following sections will talk about how to prepare for the data that is accepted by the candig-server, and provide you with some mock data for testing.

1.2.1 Clinical and Pipeline Metadata

File format required for ingestion

At this time, `ingest` command is the only way to ingest any clinical and pipeline metadata, detailed instructions for ingestion can be found under *Command Line Interface*.

However, before you run the `ingest` command, you need to prepare for a json file that conforms to the ingest standard format.

For clinical data, it is a json object, with *metadata* as the key. However, for pipeline data, its key is *pipeline_metadata*. Make sure you have the correct key specified.

The value of the key is a list of objects. Each object should have the table name as the key, and the object as its value. Therefore, it is possible to specify multiple tables in one single object. However, each table can only specified once, due to the uniqueness of the key in the object.

If you need to specify, for example, two samples for one patient. You should specify the second sample as an independent object in the list, as shown below. For all clinical data objects, you always need to specify *patientId*.

> **Warning:** Please do not include Tier information yourself. Use the *load_tier* that comes with *candig-ingest* to load tiers. More details follow.

```
{
    "metadata": [
        {
            "Patient": {
                "patientId": "Patient_12345",
                "patientIdTier": 0
            },
             "Sample": {
                "sampleId": "Sample_1",
                "sampleIdTier": 0,
                "patientId": "Patient_12345",
                "patientIdTier": 4
            }
        }
        {

            "Sample": {
                "sampleId": "Sample_2",
                "sampleIdTier": 0,
                "patientId": "Patient_12345",
                "patientIdTier": 4
            },
        }
    ]
}
```

Similar structure is used for pipeline metadata, however, for all pipeline metadata objects, you should always include `sampleId`.

How to load tiers

If you have a valid json file, but missing tier information, you should use the `load_tier` utility provided by the *candig-ingest* to load the tier information.

The `load_tiers` command is the preferred way to load tier information. It does not come with candig-server by default, to use it, you need to install *candig-ingest* by running:

pip install candig-ingest

To tier the data, you need to run

```
usage: load_tiers <project_name> <json_filepath> <tier_filepath> <output_filepath>
```

Examples:

```
$ load_tiers pog mock.json tier.tsv mock_tier.json
```

Mock data for testing

We provide some mock data files, should you want to use them to quickly test your server.

They are available from https://github.com/CanDIG/candig-ingest/tree/master/candig/ingest/mock_data Use the `clinical_metadata_tier[1,2,3].json` files.

Note: If you are interested in testing the 1k genome data, you can find a `ingest-compliant` clinical mock dataset here https://github.com/CanDIG/candig-ingest/releases/tag/v1.3.1

This contains all individuals information.

Migrate data from RedCap Cloud

If your clinical meta data is on RedCapCloud, we provide a script that would transform the related data into ready-to-ingest format.

It is available from here: https://github.com/CanDIG/redcap-cloud

1.2.2 Data Use Ontology

To enable future automated discovery, we have adopted the use of *Data Use Ontology (DUO)* Terms to describe our datasets. For the current version of candig-server, you have the option to use a json file to define your dataset.

You can find a list of DUO Terms through this csv file. You can also use an ontology parsing tool of your choice to visualize/parse a more complete list of DUO's raw OWL definition.

Warning: We only support a limited subset of the DUO terms.

Terms whose ID has an underscore, such as *DUO_0000031*, as well as *DUO:0000022* and *DUO:0000025* are not supported, as we expect these terms to be updated in the near future.

If you think an ID should be supported, but is not. You can let us know by opening an issue on our Github repo.

The supported IDs are listed below.

```
[
    "DUO:0000001", "DUO:0000002", "DUO:0000003", "DUO:0000004", "DUO:0000005",
    "DUO:0000006", "DUO:0000007", "DUO:0000011", "DUO:0000012", "DUO:0000014",
    "DUO:0000015", "DUO:0000016", "DUO:0000017", "DUO:0000018", "DUO:0000019",
    "DUO:0000020", "DUO:0000021", "DUO:0000024", "DUO:0000026", "DUO:0000027"
    "DUO:0000028", "DUO:0000029", "DUO:0000042"
]
```

Note: We do not currently provide API to look up the definitions via their ID.

If one of the supported ids listed above is not in the csv file provided above, you may be able to look up their definitions via EBI's DUO page.

To ingest the DUO Terms, you need to prepare a json file listed like below. You should only specify *id* in a single DUO object, unless the *modifier* is also required, then you specify the *id* along with the *modifier*.

```
{
    "duo": [
        {
```

(continues on next page)

(continued from previous page)

```
            "id": "DUO:0000021"
        },
        {
            "id": "DUO:0000024",
            "modifier": "2020-01-01"
        }
    ]
}
```

> **Warning:** For now, *DUO:0000024* is the only DUO Term that requires *modifier*. The modifier has to be formatted exactly like *YYYY-MM-DD*, invalid dates will be rejected.

When your file is ready, run the *add-dataset-duo* command to populate the DUO information of the dataset. Please note that this command will always overwrite the existing DUO information stored.

1.2.3 Reads, Variants and References Data

If you are interested in testing the candig-server with some variants data, we provide a mock dataset here: https://github.com/CanDIG/test_data/releases

Currently, there are three groups of test data provided, containing clinical, pipeline metadata, as well as the variants data. We have provided a loading script, note that you might need to modify the DB path, or the dataset name.

We provide three *group* datasets since we often use it to test federation of three test servers.

1.3 Development Setup

Thanks for your interest in the candig-server. If you have any questions, please let us know

> **Warning:** This guide is a work in progress, and is incomplete.

1.3.1 Standalone candig-server Setup

You will need Python 3.6.x to successfully run the candig-server.

First, create a virtualenv with Python 3.6.x

and run

```
pip clone https://github.com/CanDIG/candig-server.git
pip install dev-requirements.txt
```

The default DB location for development server is *candig-example-data/registry.db*.

Assume your local *python3* instance is of Python 3.6, with sqlite3 and other required modules built, you can run the following script to set up a dev server in a few minutes. Even better, it will be populated with two datasets with mock data.

The script has been tested on CentOS machines, and we expect the install to run OK on most unix distributions. You can run it on *MacOS*, but *wget* likely won't work and you should either download the files yourself, or use a replacement for *wget*.

See *Command Line Interface* for more information regarding ingesting data and starting up server.

> **Warning:** Please note that by default, the script installs the server on the directory *test_server*. You can either change it, or make sure that you do not have a *test_server* directory.

```
# Wipe previous virtualenv
if [ -d test_server ];
     then rm -r test_server;
fi


# Server 1
echo "setting up test_server"
python3 -m venv test_server
cd test_server
source bin/activate
pip install -U pip
pip install -U setuptools


pip install candig-server
pip install candig-ingest==1.3.1

mkdir candig-example-data

wget https://raw.githubusercontent.com/CanDIG/candig-ingest/master/candig/ingest/mock_
↪data/clinical_metadata_tier1.json
wget https://raw.githubusercontent.com/CanDIG/candig-ingest/master/candig/ingest/mock_
↪data/clinical_metadata_tier2.json
ingest candig-example-data/registry.db mock1 clinical_metadata_tier1.json
ingest candig-example-data/registry.db mock2 clinical_metadata_tier2.json

echo "test server set up has completed."

candig_server --host 0.0.0.0 --port 3000
```

Optionally, you can run two servers with federation set up. Assume you have server A running at `0.0.0.0:3000`, you need to run the script again to set up a server B. Use this file as the clinical data for your server B, and run your server B at `0.0.0.0:3001`.

Now that you have both servers installed, you need to add them to be the peer of each other

For server A, you need to run

```
candig_repo add-peer candig-example-data/registry.db http://0.0.0.0:3001
```

For server B, you need to run

```
candig_repo add-peer candig-example-data/registry.db http://0.0.0.0:3001
```

You do not need to have anything running on the peer when you execute the *add-peer* command. It simply registeres that URL as a peer.

Now, you will get federated response from both servers A and B. You can certainly choose to run them on different ports, or different servers, the script makes these assumptions only for your convenience.

1.3.2 Tyk and Keycloak Setup

It is possible to run set up a local test server, with Tyk and Keycloak set up. These two components provide authentication capabilities.

For more information, refer to https://github.com/CanDIG/candig_tyk.

1.4 Production Deployment

The candig-server has a *Configuration file*. allows Flask and application specific configuration values to be set.

1.4.1 Configuration file

The candig-server is a Flask application and uses the standard Flask configuration file mechanisms. Many configuration files will be very simple, and will consist of just one directive instructing the server where to find the data repository; example, we might have

```
DATA_SOURCE = "/path/to/registry.db"
```

For production deployments, we shouldn't need to add any more configuration than this, as the other keys have sensible defaults. However, all of Flask's builtin configuration values are supported, as well as the extra custom configuration values documented here. For information on ingesting data, see *Command Line Interface*.

When debugging deployment issues, it may be useful to turn on extra debugging information as follows:

```
DEBUG = True
```

> **Warning:** Debugging should only be used temporarily and not left on by default. Running the server with Flask debugging enable is insecure and should never be used in a production environment.

Configuration Values

DEFAULT_PAGE_SIZE The default maximum number of values to fill into a page when responding to search queries. If a client does not specify a page size in a query, this value is used.

Depending on your audiences or your dataset, you could choose to set a number between 1000 to 4000. If you are part of a federated network, it is best to choose a single number and use that within the federated network.

MAX_RESPONSE_LENGTH The approximate maximum size of the server buffer used when creating responses. This is somewhat smaller than the size of the JSON response returned to the client. When a client makes a search request with a given page size, the server will process this query and incrementally build a response until (a) the number of values in the page list is equal to the page size; (b) the size of the internal buffer in bytes is >= MAX_RESPONSE_LENGTH; or (c) there are no more results left in the query.

The default is 1024 * 1024, which is equivalent to roughly 1 MB. You can make this bigger, though we don't recommend setting anything too big, e.g., bigger than 10 MB.

REQUEST_VALIDATION Set this to True to strictly validate all incoming requests to ensure that they conform to the protocol. This may result in clients with poor standards compliance receiving errors rather than the expected results.

This defaults to *True*. Don't change this unless you have a particularly good reason to.

1.4.2 Docker Deployment

Refer to this page for Docker installation instructions

https://github.com/CanDIG/candig_compose

1.4.3 Access List Setup

An example access list file looks like below. It is a tab-separated file. The server default name for the file is `access_list.txt` (not .tsv) due to backward compatibility reasons.

You can place your own access_list file anywhere you like, but you will need to specify the location at *ACCESS_LIST*. Under production environment, you should define the location of the file to be somewhere secure.

```
issuer      username        project1        project2        project3        projectN

https://candigauth.bcgsc.ca/auth/realms/candig      userA   4       4       4       4
https://candigauth.bcgsc.ca/auth/realms/candig      userB   4               0       1

https://candigauth.uhnresearch.ca/auth/realms/CanDIG        userC   4       3       2␣
↪       1
https://candigauth.uhnresearch.ca/auth/realms/CanDIG        userD                   4␣
↪       4
```

1.5 Command Line Interface

With the exception of using `ingest` to ingest the clinical and pipeline metadata in bulk, `candig_repo` command line interface is used for all other operation.

The registry contains links to files, as well as some metadata.

> **Warning:** Because the data model objects returned via APIs are created at server start-up, at this time, you have to restart the server for the data you ingest to be reflected.

For instructions on adding metadata in bulk, see *ingest*.

When you are done ingesting data, you may start up your server instance by running the `candig_server` command, see *Start up candig-server* for more information.

1.5.1 Initialize/Remove Dataset

This section contains commands that initialize the dataset, give you the overview of the data repository, as well as deleting the dataset.

You do not need to use `init` to initialize the dataset if you already prepared a json file of clinical information. You can run the `ingest` command directly and it will take care of everything for you.

init

> **Warning:** If you already prepared a json file that conforms to our standard clinical or pipeline metadata, you can run `ingest` command directly without running `init`.
>
> For detailed instructions, see *ingest*.

The `init` command initialises a new registry DB at a given file path. Unless you have a clinical json file ready that can be ingested with `ingest`, you need to run this to initialize your DB.

Initialize a data repository

```
usage: candig_repo init [-h] [-f] registryPath
```

Positional Arguments

registryPath the location of the registry database

Named Arguments

-f, --force do not prompt for confirmation

Examples:

```
$ candig_repo init registry.db
```

list

The `list` command is used to print the contents of a repository to the screen. It is an essential tool for administrators to understand the structure of the repository that they are managing.

Note: The `list` command is under development and will be much more sophisticated in the future. In particular, the output of this command should improve considerably in the near future.

List the contents of the repo

```
usage: candig_repo list [-h] registryPath
```

Positional Arguments

registryPath the location of the registry database

Examples:

```
$ candig_repo list registry.db
```

verify

The `verify` command is used to check that the integrity of the data in a repository. The command checks each container object in turn and ensures that it can read data from it. Read errors can occur for any number of reasons (for example, a VCF file may have been moved to another location since it was added to the registry), and the `verify` command allows an administrator to check that all is well in their repository.

Note: The `verify` command is currently under review.

Verifies the repository by examing all data files

```
usage: candig_repo verify [-h] registryPath
```

Positional Arguments

registryPath	the location of the registry database

Examples:

```
$ candig_repo verify registry.db
```

add-dataset

Creates a new dataset in a repository. A dataset is an arbitrary collection of ReadGroupSets, VariantSets, VariantAnnotationSets and FeatureSets. Each dataset has a name, which is used to identify it in the repository manager.

Warning: If you already prepared a json file that conforms to our standard clinical or pipeline metadata, you can run `ingest` command directly without running `add-dataset`.

For detailed instructions, see *ingest*.

Add a dataset to the data repo

```
usage: candig_repo add-dataset [-h] [-A ATTRIBUTES] [-d DESCRIPTION]
                               registryPath datasetName
```

Positional Arguments

registryPath	the location of the registry database
datasetName	the name of the dataset

Named Arguments

-A, --attributes	additional attributes for the message expressed as JSON
-d, --description	The human-readable description of the dataset.

Examples:

```
$ candig_repo add-dataset registry.db 1kg -d 'Example dataset using 1000 genomes data'
```

Adds the dataset with the name `1kg` and description `'Example dataset using 1000 genomes data'` to the registry database `registry.db`.

add-dataset-duo

Create/update new Data Use Ontology Information for an existing dataset. Note that you have to have an existing dataset to be able to use this command. When you need to update the DUO info, simply run the command with updated DUO Json file.

Add DUO info to a dataset

```
usage: candig_repo add-dataset-duo [-h]
                                   registryPath datasetName
                                   dataUseOntologyFile
```

Positional Arguments

registryPath the location of the registry database

datasetName the name of the dataset

dataUseOntologyFile Path to your duo config json file.

Examples:

```
$ candig_repo add-dataset-duo registry.db mock1 duo.json
```

Adds the Data Use Ontology info to the dataset with the name `mock1`.

To learn about how to prepare a json file that contains DUO info for a dataset, and a list of DUO IDs that are allowed, see the `Data Use Ontology` section under *Prepare Data For Ingestion*.

remove-dataset

Removes a dataset from the repository and recursively removes all objects (ReadGroupSets, VariantSets, etc) within this dataset.

Remove a dataset from the data repo

```
usage: candig_repo remove-dataset [-h] [-f] registryPath datasetName
```

Positional Arguments

registryPath the location of the registry database

datasetName the name of the dataset

Named Arguments

-f, --force do not prompt for confirmation

Examples:

```
$ candig_repo remove-dataset registry.db dataset1
```

Deletes the dataset with name `dataset1` from the repository represented by `registry.db`

remove-dataset-duo

Remove new Data Use Ontology Information for an existing dataset.

Remove DUO info from a dataset

```
usage: candig_repo remove-dataset-duo [-h] [-f] registryPath datasetName
```

Positional Arguments

registryPath the location of the registry database

datasetName the name of the dataset

Named Arguments

-f, --force do not prompt for confirmation

Examples:

```
$ candig_repo remove-dataset-duo registry.db mock1
```

Removes the Data Use Ontology info to the dataset with the name `mock1`.

1.5.2 Add/Remove Clinical & Pipeline Metadata

This section contains commands that let you ingest data into the clinical and pipeline metadata tables, as well as the commands that delete them.

The `ingest` command is the only way to ingest clinical or pipeline data in bulk. It encapsulates all the write operations into a single transaction. To learn about preparing the json files for the `ingest` command, see *Prepare Data For Ingestion*

All of the `remove` commands for removing clinical tables require you to specify their `name`, note that the `name` here is actually their unique identifier, typically is composed of their patientId, sometimes along with some other ID or timestamp information. This is the same `name` you see in the records of these clinical or pipeline data records.

ingest

The `ingest` command is the preferred way to import metadata in bulk. It does not come with candig-server by default, to use it, you need to install *candig-ingest* by running:

pip install candig-ingest

To import metadata in bulk, you need to have a specially formatted json file. A mock json file is available from https://github.com/CanDIG/candig-ingest/blob/master/candig/ingest/mock_data/clinical_metadata_tier1.json

To ingest the data, you need to run

```
usage: ingest registryPath datasetName metadataPath
```

If the dataset does not exist, it will create a new dataset of this name. There is no need to run `init` command before running `ingest`.

Examples:

```
$ ingest registry.db mock1 mock_data.json
```

remove-patient

remove a patient.

Remove an Patient from the repo

```
usage: candig_repo remove-patient [-h] [-f]
                                  registryPath datasetName patientName
```

Positional Arguments

registryPath the location of the registry database

datasetName the name of the dataset

patientName the name of the patient

Named Arguments

-f, --force do not prompt for confirmation

Examples:

```
$ candig_repo remove-patient registry.db mock1 PATIENT_81202
```

remove-enrollment

remove a enrollment.

Remove an Enrollment from the repo

```
usage: candig_repo remove-enrollment [-h] [-f]
                                     registryPath datasetName enrollmentName
```

Positional Arguments

registryPath the location of the registry database

datasetName the name of the dataset

enrollmentName	the name of the enrollment

Named Arguments

-f, --force	do not prompt for confirmation

Examples:

```
$ candig_repo remove-enrollment registry.db mock1 PATIENT_81202_2005-08-23
```

remove-treatment

remove a treatment.

Remove an Treatment from the repo

```
usage: candig_repo remove-treatment [-h] [-f]
                                    registryPath datasetName treatmentName
```

Positional Arguments

registryPath	the location of the registry database
datasetName	the name of the dataset
treatmentName	the name of the treatment

Named Arguments

-f, --force	do not prompt for confirmation

Examples:

```
$ candig_repo remove-treatment registry.db mock1 PATIENT_81202_2005-08-23
```

remove-sample

remove a sample.

Remove an Sample from the repo

```
usage: candig_repo remove-sample [-h] [-f] registryPath datasetName sampleName
```

Positional Arguments

registryPath	the location of the registry database
datasetName	the name of the dataset
sampleName	the name of the sample

Named Arguments

-f, --force	do not prompt for confirmation

Examples:

```
$ candig_repo remove-sample registry.db mock1 PATIENT_81202_SAMPLE_33409
```

remove-diagnosis

remove a diagnosis.

Remove an Diagnosis from the repo

```
usage: candig_repo remove-diagnosis [-h] [-f]
                                    registryPath datasetName diagnosisName
```

Positional Arguments

registryPath	the location of the registry database
datasetName	the name of the dataset
diagnosisName	the name of the diagnosis

Named Arguments

-f, --force	do not prompt for confirmation

Examples:

```
$ candig_repo remove-diagnosis registry.db mock1 PATIENT_81202_SAMPLE_33409
```

remove-tumourboard

remove a tumourboard.

Remove an Tumourboard from the repo

```
usage: candig_repo remove-tumourboard [-h] [-f]
                                      registryPath datasetName tumourboardName
```

Positional Arguments

registryPath	the location of the registry database
datasetName	the name of the dataset
tumourboardName	the name of the tumourboard

Named Arguments

-f, --force	do not prompt for confirmation

Examples:

```
$ candig_repo remove-tumourboard registry.db mock1 PATIENT_81202_SAMPLE_33409
```

remove-outcome

remove a outcome.

Remove an Outcome from the repo

```
usage: candig_repo remove-outcome [-h] [-f]
                                  registryPath datasetName outcomeName
```

Positional Arguments

registryPath	the location of the registry database
datasetName	the name of the dataset
outcomeName	the name of the outcome

Named Arguments

-f, --force	do not prompt for confirmation

Examples:

```
$ candig_repo remove-outcome registry.db mock1 PATIENT_81202_2016-10-11
```

remove-complication

remove a complication.

Remove an Complication from the repo

```
usage: candig_repo remove-complication [-h] [-f]
                                       registryPath datasetName
                                       complicationName
```

Positional Arguments

registryPath	the location of the registry database
datasetName	the name of the dataset
complicationName	the name of the complication

Named Arguments

-f, --force do not prompt for confirmation

Examples:

```
$ candig_repo remove-complication registry.db mock1 PATIENT_81202_2016-10-11
```

remove-consent

remove a consent.

Remove an Consent from the repo

```
usage: candig_repo remove-consent [-h] [-f]
                                  registryPath datasetName consentName
```

Positional Arguments

registryPath the location of the registry database

datasetName the name of the dataset

consentName the name of the consent

Named Arguments

-f, --force do not prompt for confirmation

Examples:

```
$ candig_repo remove-consent registry.db mock1 PATIENT_81202_2016-10-11
```

remove-chemotherapy

remove a chemotherapy.

Remove an Chemotherapy from the repo

```
usage: candig_repo remove-chemotherapy [-h] [-f]
                                       registryPath datasetName
                                       chemotherapyName
```

Positional Arguments

registryPath the location of the registry database

datasetName the name of the dataset

chemotherapyName the name of the chemotherapy

Named Arguments

-f, --force	do not prompt for confirmation

Examples:

```
$ candig_repo remove-chemotherapy registry.db mock1 PATIENT_81202_2016-10-11
```

remove-immunotherapy

remove a immunotherapy.

Remove an Immunotherapy from the repo

```
usage: candig_repo remove-immunotherapy [-h] [-f]
                                        registryPath datasetName
                                        immunotherapyName
```

Positional Arguments

registryPath	the location of the registry database
datasetName	the name of the dataset
immunotherapyName	the name of the immunotherapy

Named Arguments

-f, --force	do not prompt for confirmation

Examples:

```
$ candig_repo remove-immunotherapy registry.db mock1 PATIENT_81202_2016-10-11
```

remove-radiotherapy

remove a radiotherapy.

Remove an Radiotherapy from the repo

```
usage: candig_repo remove-radiotherapy [-h] [-f]
                                       registryPath datasetName
                                       radiotherapyName
```

Positional Arguments

registryPath	the location of the registry database
datasetName	the name of the dataset
radiotherapyName	the name of the radiotherapy

Named Arguments

-f, --force	do not prompt for confirmation

Examples:

```
$ candig_repo remove-radiotherapy registry.db mock1 PATIENT_81202_2016-10-11
```

remove-celltransplant

remove a celltransplant.

Remove an Celltransplant from the repo

```
usage: candig_repo remove-celltransplant [-h] [-f]
                                         registryPath datasetName
                                         celltransplantName
```

Positional Arguments

registryPath	the location of the registry database
datasetName	the name of the dataset
celltransplantName	the name of the celltransplant

Named Arguments

-f, --force	do not prompt for confirmation

Examples:

```
$ candig_repo remove-celltransplant registry.db mock1 PATIENT_81202_2016-10-11
```

remove-surgery

remove a surgery.

Remove an Surgery from the repo

```
usage: candig_repo remove-surgery [-h] [-f]
                                  registryPath datasetName surgeryName
```

Positional Arguments

registryPath	the location of the registry database
datasetName	the name of the dataset
surgeryName	the name of the surgery

Named Arguments

-f, --force do not prompt for confirmation

Examples:

```
$ candig_repo remove-surgery registry.db mock1 PATIENT_81202_2016-10-11
```

remove-study

remove a study.

Remove an Study from the repo

```
usage: candig_repo remove-study [-h] [-f] registryPath datasetName studyName
```

Positional Arguments

registryPath the location of the registry database

datasetName the name of the dataset

studyName the name of the study

Named Arguments

-f, --force do not prompt for confirmation

Examples:

```
$ candig_repo remove-study registry.db mock1 PATIENT_81202_2016-10-11
```

remove-slide

remove a slide.

Remove an Slide from the repo

```
usage: candig_repo remove-slide [-h] [-f] registryPath datasetName slideName
```

Positional Arguments

registryPath the location of the registry database

datasetName the name of the dataset

slideName the name of the slide

Named Arguments

-f, --force do not prompt for confirmation

Examples:

```
$ candig_repo remove-slide registry.db mock1 PATIENT_81202_2016-10-11
```

remove-labtest

remove a labtest.

Remove an Labtest from the repo

```
usage: candig_repo remove-labtest [-h] [-f]
                                  registryPath datasetName labtestName
```

Positional Arguments

registryPath the location of the registry database

datasetName the name of the dataset

labtestName the name of the labtest

Named Arguments

-f, --force do not prompt for confirmation

Examples:

```
$ candig_repo remove-labtest registry.db mock1 PATIENT_81202_2016-10-11
```

1.5.3 Add/Remove Genomics Data

add-referenceset

Adds a reference set derived from a FASTA file to a repository. Each record in the FASTA file will correspond to a Reference in the new ReferenceSet. The input FASTA file must be compressed with `bgzip` and indexed using `samtools faidx`. Each ReferenceSet contains a number of metadata values (.e.g. `species`) which can be set using command line options.

Add a reference set to the data repo

```
usage: candig_repo add-referenceset [-h] [-A ATTRIBUTES] [-r] [-n NAME]
                                    [-d DESCRIPTION] [--species SPECIES]
                                    [--isDerived ISDERIVED]
                                    [--assemblyId ASSEMBLYID]
                                    [--sourceAccessions SOURCEACCESSIONS]
                                    [--sourceUri SOURCEURI]
                                    registryPath filePath
```

Positional Arguments

registryPath the location of the registry database

filePath The path of the FASTA file to use as a reference set. This file must be bgzipped and indexed.

Named Arguments

-A, --attributes additional attributes for the message expressed as JSON

-r, --relativePath store relative path in database

-n, --name The name of the reference set

-d, --description The human-readable description of the reference set.

--species The species ontology term as a JSON string

--isDerived Indicates if this reference set is derived from another

--assemblyId The assembly id

--sourceAccessions The source accessions (pass as comma-separated list)

--sourceUri The source URI

Examples:

```
$ candig_repo add-referenceset registry.db hs37d5.fa.gz \
    --description "NCBI37 assembly of the human genome" \
    --species '{"termId": "NCBI:9606", "term": "Homo sapiens"}' \
    --name NCBI37 \
    --sourceUri ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/reference/phase2_
↪reference_assembly_sequence/hs37d5.fa.gz
```

Adds a reference set used in the 1000 Genomes project using the name `NCBI37`, also setting the `species` to 9606 (human).

add-ontology

> **Warning:** This command, as well as all ontology-related operations are under review. They might undergo changes in the near future.

Adds a new ontology to the repository. The ontology supplied must be a text file in OBO format. If you wish to serve sequence or variant annotations from a repository, a sequence ontology (SO) instance is required to translate ontology term names held in annotations to ontology IDs. Sequence ontology definitions can be downloaded from the Sequence Ontology site.

Adds an ontology in OBO format to the repo. Currently, a sequence ontology (SO) instance is required to translate ontology term names held in annotations to ontology IDs. Sequence ontology files can be found at https://github.com/The-Sequence-Ontology/SO-Ontologies

```
usage: candig_repo add-ontology [-h] [-r] [-n NAME] registryPath filePath
```

Positional Arguments

registryPath the location of the registry database

filePath The path of the OBO file defining this ontology.

Named Arguments

-r, --relativePath store relative path in database

-n, --name The name of the ontology

Examples:

```
$ candig_repo add-ontology registry.db path/to/so-xp.obo
```

Adds the sequence ontology `so-xp.obo` to the repository using the default naming rules.

add-variantset

Adds a variant set to a named dataset in a repository. Variant sets are currently derived from one or more non-overlapping VCF/BCF files which may be either stored locally or come from a remote URL. Multiple VCF files can be specified either directly on the command line or by providing a single directory argument that contains indexed VCF files. If remote URLs are used then index files in the local file system must be provided using the `-I` option.

Note: Starting from 0.9.3, you now need to specify a `patientId` and a `sampleId`. The server does not validate either, so please double check to make sure the IDs are correct.

Add a variant set to the data repo based on one or more VCF files.

```
usage: candig_repo add-variantset [-h] [-r] [-I indexFiles [indexFiles ...]]
                                  [-n NAME] [-R REFERENCESETNAME]
                                  [-O ONTOLOGYNAME] [-A ATTRIBUTES] [-a]
                                  registryPath datasetName patientId sampleId
                                  dataFiles [dataFiles ...]
```

Positional Arguments

registryPath the location of the registry database

datasetName the name of the dataset

patientId the ID of the patient

sampleId the ID of the sample

dataFiles The VCF/BCF files representing the new VariantSet. These may be specified either one or more paths to local files or remote URLS, or as a path to a local directory containing VCF files. Either a single directory argument may be passed or a list of file paths/URLS, but not a mixture of directories and paths.

Named Arguments

-r, --relativePath store relative path in database

-I, --indexFiles The index files for the VCF/BCF files provided in the dataFiles argument. These must be provided in the same order as the data files.

-n, --name The name of the VariantSet

-R, --referenceSetName the name of the reference set to associate with this VariantSet

-O, --ontologyName the name of the sequence ontology instance used to translate ontology term names to IDs in this VariantSet

-A, --attributes additional attributes for the message expressed as JSON

-a, --addAnnotationSets If the supplied VCF file contains annotations, create the corresponding VariantAnnotationSet.

Examples:

```
$ candig_repo add-variantset registry.db 1kg PATIENT_123 SAMPLE_123 1kgPhase1/ -R␣
↪NCBI37
```

Adds a new variant set to the dataset named `1kg` in the repository defined by the registry database `registry.db` using the VCF files contained in the directory `1kgPhase1` that belong to PATIENT_123 and SAMPLE_123. Note that this directory must also contain the corresponding indexes for these files. We associate the reference set named `NCBI37` with this new variant set. Because we do not provide a `--name` argument, a name is automatically generated using the default name generation rules.

```
$ candig_repo add-variantset registry.db 1kg PATIENT_123 SAMPLE_123 \
    1kgPhase1/chr1.vcf.gz -n phase1-subset -R NCBI37
```

Like the last example, we add a new variant set to the dataset `1kg`, with one VCF and the corresponding patientId and sampleId. We also specify the name for this new variant set to be `phase1-subset`.

```
$ candig_repo add-variantset registry.db 1kg PATIENT_123 SAMPLE_123 \
    --name phase1-subset-remote -R NCBI37 \
    --indexFiles ALL.chr1.phase1_release_v3.20101123.snps_indels_svs.genotypes.vcf.gz.
↪tbi ALL.chr2.phase1_release_v3.20101123.snps_indels_svs.genotypes.vcf.gz.tbi \
    ftp://ftp.ncbi.nlm.nih.gov/1000genomes/ftp/release/20110521/ALL.chr1.phase1_
↪release_v3.20101123.snps_indels_svs.genotypes.vcf.gz \
```

This example performs the same task of creating a subset of the phase1 VCFs, but this time we use the remote URL directly and do not keep a local copy of the VCF file. Because we are using remote URLs to define the variant set, we have to download a local copy of the corresponding index files and provide them on the command line using the `--indexFiles` option.

add-readgroupset

Adds a readgroup set to a named dataset in a repository. Readgroup sets are currently derived from a single indexed BAM file, which can be either stored locally or based on a remote URL. If the readgroup set is based on a remote URL, then the index file must be stored locally and specified using the `--indexFile` option.

Each readgroup set must be associated with the reference set that it is aligned to. The `add-readgroupset` command first examines the headers of the BAM file to see if it contains information about references, and then looks for a reference set with name equal to the genome assembly identifer defined in the header. (Specifically, we read the `@SQ` header line and use the value of the `AS` tag as the default reference set name.) If this reference set exists, then the readgroup set will be associated with it automatically. If it does not (or we cannot find the appropriate information in the header), then the `add-readgroupset` command will fail. In this case, the user must provide the name of the reference set using the `--referenceSetName` option.

Add a read group set to the data repo

```
usage: candig_repo add-readgroupset [-h] [-n NAME] [-R REFERENCESETNAME]
                                    [-A ATTRIBUTES] [-r] [-I INDEXFILE]
                                    registryPath datasetName patientId
                                    sampleId dataFile
```

Positional Arguments

registryPath the location of the registry database

datasetName the name of the dataset

patientId the ID of the patient

sampleId the ID of the sample

dataFile The file path or URL of the BAM file for this ReadGroupSet

Named Arguments

-n, --name The name of the ReadGroupSet

-R, --referenceSetName the name of the reference set to associate with this ReadGroupSet

-A, --attributes additional attributes for the message expressed as JSON

-r, --relativePath store relative path in database

-I, --indexFile The file path of the BAM index for this ReadGroupSet. If the dataFile argument is a local file, this will be automatically inferred by appending '.bai' to the file name. If the dataFile is a remote URL the path to a local file containing the BAM index must be provided

Examples:

```
$ candig_repo add-readgroupset registry.db 1kg \
    path/to/HG00114.chrom11.ILLUMINA.bwa.GBR.low_coverage.20120522.bam
```

Adds a new readgroup set for an indexed 1000 Genomes BAM file stored on the local file system. The index file follows the usual convention and is stored in the same directory as the BAM file and has an extra `.bai` extension. The name of the readgroup set is automatically derived from the file name, and the reference set automatically set from the BAM header.

```
$ candig_repo add-readgroupset registry.db 1kg PATIENT_123 SAMPLE_123 candig-example-
↪data/HG00096.bam \
    -R GRCh37-subset -n HG0096-subset
```

Adds a new readgroup set based on a subset of the 1000 genomes reads for the HG00096 sample from the example data used in the reference server. In this case we specify that the reference set name `GRCh37-subset` be associated with the readgroup set. We also override the default name generation rules and specify the name `HG00096-subset` for the new readgroup set.

```
$ candig_repo add-readgroupset registry.db 1kg PATIENT_123 SAMPLE_123 \
    -n HG00114-remote
    -I /path/to/HG00114.chrom11.ILLUMINA.bwa.GBR.low_coverage.20120522.bam.bai
    ftp://ftp.ncbi.nlm.nih.gov/1000genomes/ftp/phase3/data/HG00114/alignment/HG00114.
↪chrom11.ILLUMINA.bwa.GBR.low_coverage.20120522.bam
```

Adds a new readgroups set based on a 1000 genomes BAM directly from the NCBI FTP server. Because this readgroup set uses a remote FTP URL, we must specify the location of the `.bai` index file on the local file system.

add-featureset

> **Warning:** Before you add the feature set, you should make sure to index some of the columns in your generated DB. Specifically, you should make sure that you both `gene_name` and `type` should be indexed. If you don't, queries to this endpoint, and endpoints that depend on this, e.g., `variants/gene/search` will be very very slow.
>
> To create a composite index on aforementioned fields, open the featureset DB you generated via the sqlite browser, then run `CREATE INDEX name_type_index ON FEATURE (gene_name, type);`. You should carefully review your use-case and index other fields accordingly.

Adds a feature set to a named dataset in a repository. Feature sets must be in a '.db' file. An appropriate '.db' file can be generate from a GFF3 file using scripts/generate_gff3_db.py.

Add a feature set to the data repo

```
usage: candig_repo add-featureset [-h] [-A ATTRIBUTES] [-r]
                                  [-R REFERENCESETNAME] [-O ONTOLOGYNAME]
                                  [-C CLASSNAME]
                                  registryPath datasetName filePath
```

Positional Arguments

registryPath the location of the registry database

datasetName the name of the dataset

filePath The path to the converted SQLite database containing Feature data

Named Arguments

-A, --attributes additional attributes for the message expressed as JSON

-r, --relativePath store relative path in database

-R, --referenceSetName the name of the reference set to associate with this feature set

-O, --ontologyName the name of the sequence ontology instance used to translate ontology term names to IDs in this feature set

-C, --className the name of the class used to fetch features in this feature set

Examples:

```
$ candig_repo add-featureset registry.db 1KG gencode.db \
    -R hg37 -O so-xp-simple
```

Adds the feature set *gencode* to the registry under the *1KG* dataset. The flags set the reference genome to be hg37 and the ontology to use to *so-xp-simple*.

add-continuousset

Adds a continuous set to a named dataset in a repository. Continuous sets must be in a bigWig file. The bigWig format is described here: http://genome.ucsc.edu/goldenPath/help/bigWig.html. There are directions for converting wiggle files to bigWig files on the page also. Files in the bedGraph format can be converted using bedGraphToBigWig (https://www.encodeproject.org/software/bedgraphtobigwig/).

Add a continuous set to the data repo

```
usage: candig_repo add-continuousset [-h] [-r] [-R REFERENCESETNAME]
                                     [-C CLASSNAME]
                                     registryPath datasetName filePath
```

Positional Arguments

registryPath	the location of the registry database
datasetName	the name of the dataset
filePath	The path to the file contianing the continuous data

Named Arguments

-r, --relativePath	store relative path in database
-R, --referenceSetName	the name of the reference set to associate with this continuous set
-C, --className	the name of the class used to fetch features in this continuous set

Examples:

```
$ candig_repo add-continuousset registry.db 1KG continuous.bw \
    -R hg37
```

Adds the continuous set *continuous* to the registry under the *1KG* dataset. The flags set the reference genome to be hg37.

init-rnaquantificationset

Initializes a rnaquantification set.

Initializes an RNA quantification set

```
usage: candig_repo init-rnaquantificationset [-h] registryPath filePath
```

Positional Arguments

registryPath	the location of the registry database
filePath	The path to the resulting Quantification Set

Examples:

```
$ candig_repo init-rnaquantificationset repo.db rnaseq.db
```

Initializes the RNA Quantification Set with the filename rnaseq.db.

add-rnaquantification

Adds a rnaquantification to a RNA quantification set.

RNA quantification formats supported are currently kallisto and RSEM.

Add an RNA quantification to the data repo

```
usage: candig_repo add-rnaquantification [-h] [--biosampleName BIOSAMPLENAME]
                                         [--sampleId SAMPLEID]
                                         [--patientId PATIENTID]
                                         [--readGroupSetName READGROUPSETNAME]
                                         [--featureSetNames FEATURESETNAMES]
                                         [-n NAME] [-d DESCRIPTION] [-t]
                                         [-A ATTRIBUTES]
                                         filePath quantificationFilePath
                                         format registryPath datasetName
```

Positional Arguments

filePath The path to the RNA SQLite database to create or modify

quantificationFilePath The path to the expression file.

format format of the quantification input data

registryPath the location of the registry database

datasetName the name of the dataset

Named Arguments

--biosampleName Biosample Name

--sampleId SampleId

--patientId PatientId

--readGroupSetName Read Group Set Name

--featureSetNames Comma separated list

-n, --name The name of the rna quantification

-d, --description The human-readable description of the RnaQuantification.

-t, --transcript sets the quantification type to transcript

-A, --attributes additional attributes for the message expressed as JSON

Examples:

```
$ candig_repo add-rnaquantification rnaseq.db data.tsv \
         kallisto candig-example-data/registry.db brca1 \
        --biosampleName HG00096 --featureSetNames gencodev19
        --readGroupSetName HG00096rna --transcript
```

Adds the data.tsv in kallisto format to the *rnaseq.db* quantification set with optional fields for associating a quantification with a Feature Set, Read Group Set, and Biosample.

add-rnaquantificationset

When the desired RNA quantification have been added to the set, use this command to add them to the registry.

Add an RNA quantification set to the data repo

```
usage: candig_repo add-rnaquantificationset [-h] [-R REFERENCESETNAME]
                                            [-n NAME] [-A ATTRIBUTES]
                                            registryPath datasetName filePath
```

Positional Arguments

registryPath the location of the registry database

datasetName the name of the dataset

filePath The path to the converted SQLite database containing RNA data

Named Arguments

-R, --referenceSetName the name of the reference set to associate with this RnaQuantificationSet

-n, --name The name of the RnaQuantificationSet

-A, --attributes additional attributes for the message expressed as JSON

Examples:

```
$ candig_repo add-rnaquantificationset registry.db brca1 rnaseq.db \
    -R hg37 -n rnaseq
```

Adds the RNA quantification set *rnaseq.db* to the registry under the *brca1* dataset. The flags set the reference genome to be hg37 and the name of the set to *rnaseq*.

add-phenotypeassociationset

Adds an rdf object store. The cancer genome database Clinical Genomics Knowledge Base http://nif-crawler.neuinfo.org/monarch/ttl/cgd.ttl, published by the Monarch project, is the supported format for Evidence.

Adds phenotypes in ttl format to the repo.

```
usage: candig_repo add-phenotypeassociationset [-h] [-n NAME] [-A ATTRIBUTES]
                                               registryPath datasetName
                                               dirPath
```

Positional Arguments

registryPath the location of the registry database

datasetName the name of the dataset

dirPath The path of the directory containing ttl files.

Named Arguments

-n, --name	The name of the PhenotypeAssociationSet
-A, --attributes	additional attributes for the message expressed as JSON

Examples:

```
$ candig_repo add-phenotypeassociationset registry.db dataset1 /monarch/ttl/cgd.ttl -
↪n cgd
```

remove-referenceset

Removes a reference set from the repository. Attempting to remove a reference set that is referenced by other objects in the repository will result in an error.

Remove a reference set from the repo

```
usage: candig_repo remove-referenceset [-h] [-f] registryPath referenceSetName
```

Positional Arguments

registryPath	the location of the registry database
referenceSetName	the name of the reference set

Named Arguments

-f, --force	do not prompt for confirmation

Examples:

```
$ candig_repo remove-referenceset registry.db NCBI37
```

Deletes the reference set with name `NCBI37` from the repository represented by `registry.db`

remove-ontology

Removes an ontology from the repository. Attempting to remove an ontology that is referenced by other objects in the repository will result in an error.

Remove an ontology from the repo

```
usage: candig_repo remove-ontology [-h] [-f] registryPath ontologyName
```

Positional Arguments

registryPath	the location of the registry database
ontologyName	the name of the ontology

Named Arguments

-f, --force	do not prompt for confirmation

Examples:

```
$ candig_repo remove-ontology registry.db so-xp
```

Deletes the ontology with name `so-xp` from the repository represented by `registry.db`

remove-variantset

Removes a variant set from the repository. This also deletes all associated call sets and variant annotation sets from the repository.

Remove a variant set from the repo

```
usage: candig_repo remove-variantset [-h] [-f]
                                     registryPath datasetName variantSetName
```

Positional Arguments

registryPath	the location of the registry database
datasetName	the name of the dataset
variantSetName	the name of the variant set

Named Arguments

-f, --force	do not prompt for confirmation

Examples:

```
$ candig_repo remove-variantset registry.db dataset1 phase3-release
```

Deletes the variant set named `phase3-release` from the dataset named `dataset1` from the repository represented by `registry.db`.

remove-readgroupset

Removes a read group set from the repository.

Remove a read group set from the repo

```
usage: candig_repo remove-readgroupset [-h] [-f]
                                       registryPath datasetName
                                       readGroupSetName
```

Positional Arguments

registryPath the location of the registry database

datasetName the name of the dataset

readGroupSetName the name of the read group set

Named Arguments

-f, --force do not prompt for confirmation

Examples:

```
$ candig_repo remove-readgroupset registry.db dataset1 HG00114
```

Deletes the readgroup set named `HG00114` from the dataset named `dataset1` from the repository represented by `registry.db`.

remove-featureset

Removes a feature set from the repository.

Remove a feature set from the repo

```
usage: candig_repo remove-featureset [-h] [-f]
                                     registryPath datasetName featureSetName
```

Positional Arguments

registryPath the location of the registry database

datasetName the name of the dataset

featureSetName the name of the feature set

Named Arguments

-f, --force do not prompt for confirmation

Examples:

```
$ candig_repo remove-featureset registry.db 1KG gencode-genes
```

Deletes the feature set named `gencode-genes` from the dataset named `1KG` from the repository represented by `registry.db`.

remove-continuousset

Removes a continuous set from the repository.

Remove a continuous set from the repo

```
usage: candig_repo remove-continuousset [-h] [-f]
                                        registryPath datasetName
                                        continuousSetName
```

Positional Arguments

registryPath the location of the registry database

datasetName the name of the dataset

continuousSetName the name of the continuous set

Named Arguments

-f, --force do not prompt for confirmation

Examples:

```
$ candig_repo remove-continuousset registry.db 1KG continuous
```

Deletes the feature set named `continuous` from the dataset named `1KG` from the repository represented by `registry.db`.

remove-rnaquantificationset

Removes a rna quantification set from the repository.

Remove an RNA quantification set from the repo

```
usage: candig_repo remove-rnaquantificationset [-h] [-f]
                                               registryPath datasetName
                                               rnaQuantificationSetName
```

Positional Arguments

registryPath the location of the registry database

datasetName the name of the dataset

rnaQuantificationSetName the name of the RNA Quantification Set

Named Arguments

-f, --force do not prompt for confirmation

Examples:

```
$ candig_repo remove-rnaquantificationset registry.db dataset1 ENCFF305LZB
```

Deletes the rnaquantification set named `ENCFF305LZB` from the dataset named `dataset1` from the repository represented by `registry.db`.

remove-phenotypeassociationset

Removes an rdf object store.

Remove an phenotypes from the repo

```
usage: candig_repo remove-phenotypeassociationset [-h] [-f]
                                                  registryPath datasetName
                                                  name
```

Positional Arguments

registryPath	the location of the registry database
datasetName	the name of the dataset
name	The name of the phenotype association set

Named Arguments

-f, --force	do not prompt for confirmation

Examples:

```
$ candig_repo remove-phenotypeassociationset registry.db dataset1  cgd
```

add-biosample

Warning: This command is deprecated, and may be removed soon in future. Use ingest command to add Sample-related information.

Adds a new biosample to the repository. The biosample argument is a JSON document according to the GA4GH JSON schema.

Add a Biosample to the dataset

```
usage: candig_repo add-biosample [-h]
                                 registryPath datasetName biosampleName
                                 biosample
```

Positional Arguments

registryPath	the location of the registry database
datasetName	the name of the dataset
biosampleName	the name of the biosample
biosample	the JSON of the biosample

Examples:

```
$ candig_repo add-biosample registry.db dataset1 HG00096 '{"individualId": "abc"}'
```

Adds the biosample named HG00096 to the repository with the individual ID “abc”.

add-individual

Warning: This command is deprecated, and may be removed soon in future. Use ingest command to add Patient-related information.

Adds a new individual to the repository. The individual argument is a JSON document following the GA4GH JSON schema.

Add an Individual to the dataset

```
usage: candig_repo add-individual [-h]
                                  registryPath datasetName individualName
                                  individual
```

Positional Arguments

registryPath	the location of the registry database
datasetName	the name of the dataset
individualName	the name of the individual
individual	the JSON of the individual

Examples:

```
$ candig_repo add-individual registry.db dataset1 HG00096 '{"description": "A␣
↪description"}'
```

remove-biosample

Removes a biosample from the repository.

Remove a Biosample from the repo

```
usage: candig_repo remove-biosample [-h] [-f]
                                    registryPath datasetName biosampleName
```

Positional Arguments

registryPath	the location of the registry database
datasetName	the name of the dataset
biosampleName	the name of the biosample

Named Arguments

-f, --force	do not prompt for confirmation

Examples:

```
$ candig_repo remove-biosample registry.db dataset1 HG00096
```

Deletes the biosample with name `HG00096` in the dataset `dataset1` from the repository represented by `registry.db`

remove-individual

Removes an individual from the repository.

Remove an Individual from the repo

```
usage: candig_repo remove-individual [-h] [-f]
                                     registryPath datasetName individualName
```

Positional Arguments

registryPath	the location of the registry database
datasetName	the name of the dataset
individualName	the name of the individual

Named Arguments

-f, --force	do not prompt for confirmation

Examples:

```
$ candig_repo remove-individual registry.db dataset1 HG00096
```

Deletes the individual with name `HG00096` in the dataset `dataset1` from the repository represented by `registry.db`

1.5.4 Start up candig-server

candig_server

There are a number of optional parameters to start up the server.

When no paramters are set, running `candig-server` would start up the server at `http://127.0.0.1:8000`.

You may supply your own config file (.py), as indicated below. This `config.py` specifies the `DATA_SOURCE` to be at a custom location, and the `DEFAULT_PAGE_SIZE` to be 1500, overridding the default values for both.

```
DATA_SOURCE = '/home/user/dev/data.db'
DEFAULT_PAGE_SIZE = 1500
```

```
usage: candig_server [-h] [--port PORT] [--host HOST] [--config CONFIG]
                     [--config-file CONFIG_FILE] [--tls] [--gunicorn]
                     [--certfile CERTFILE] [--keyfile KEYFILE]
                     [--dont-use-reloader] [--workers WORKERS]
                     [--timeout TIMEOUT] [--worker_class WORKER_CLASS]
                     [--epsilon EPSILON] [--version]
                     [--disable-urllib-warnings]
```

Examples:

```
$ candig_server --host 0.0.0.0 --port 3000 --config-file config.py
```

CHAPTER

TWO

USAGE

2.1 API Usage & Sample Queries

Warning: This guide is a work in progress, and is incomplete.

This section will provide instructions on API usages. Upon logging in, you will see a dashboard that gives high level overview of your authorized datasets.

When you click on the top right corner, you will see an *API Info* section, where a Swagger UI is provided. That UI does not yet contain any information on */search* and */count* endpoints. Please refer to the resources here for instructions on using these two endpoints.

Warning: Please note that while reasonable efforts have been put in to validate the request and response schemas for the swagger UI definitions, at this time, we cannot guarantee it to be 100% accurate. Please open a ticket if you encounter any problems.

2.1.1 Sample queries for all metadata services

This includes all of the endpoints under Clinical and Pipeline Metadata Services.

Even though some of the sample queries below are made to specific endpoint, the same format applies to any pipeline or clinical metadata endpoint. Typically you will need to change the field in your filters, depending on which table you are querying on.

How to write filter objects

As you can see from Sample Queries II to IV, it is possible to submit optional *filters* list along with your request.

filters is a list of *filter* objects. There are two types of filter objects.

Type I: Compare the *field*'s value to a string

```
{
    "field": "gender",
    "operator": "=",
    "value": "female"
}
```

In the filter object above, you specify *gender* as the field, and value to be the *female*, = as the operator. This filter object asks server to find records whose *gender* is female.

Note: For Type I filter object, the supported operators are >, <, >=, <=, =, ==, !=, contains. Note that = and == are equivalent.

Type II: Check if a record's field belongs to a list

```
{
    "field": "provinceOfResidence",
    "operator": "in",
    "values": ["British Columbia", "Ontario"]
}
```

In the filter object above, you specify *provinceOfResidence* as the field, and *values* to be the *["British Columbia", "Ontario"]*, *in* as the operator.

This filter object asks server to find records whose *provinceOfResidence* is one of *British Columbia* or *Ontario*.

Note: For Type II filter object, *in* is the only supported operator. Also, you specify *values*, instead of *value*.

Fetch Datasets

Description: To fetch all of the datasets, make a query to datasets/search endpoint with an empty body. You need datasetId for subsequent queries to other endpoints.

Query:

```
{}
```

Sample Query I

Description: Fetch all of the items from the specified dataset from any clinical or pipeline metadata service endpoint.

Query:

```
{
    "datasetId": "WyIxa2dlbm9tZSJd"
}
```

Sample Query II

Description: Fetch all of the items from the patients/search endpoint, whose gender is female.

Note: We support a number of operators in the filters, which include: >, <, >=, <=, =, !=, and contains.

Query:

```
{
    "datasetId": "WyIxa2dlbm9tZSJd",
    "filters": [
        {
            "field": "gender",
            "operator": "=",
            "value": "female"
        }
    ]
}
```

Sample Query III

Description: Fetch all of the items from the patients/search endpoint, whose gender is female and whose ethnicity is NOT 'GBR'.

Note: You can specify more than one filter under filters.

```
{
    "datasetId": "WyIxa2dlbm9tZSJd",
    "filters": [
        {
            "field": "gender",
            "operator": "=",
            "value": "female"
        },
        {
            "field": "ethnicity",
            "operator": "!=",
            "value": "GBR"
        }
    ]
}
```

Sample Query IV

Description: Fetch all of the items from the patients/search endpoint, whose ethnicity is one of GBR, FIN or ESN.

Note: Note that this query is very different from the previous ones. To specify a list of values you are interested in, you need to: Specify `values` in your filters, instead of `value`. Specify a list of values that you are interested in. Specify `in` as the operator.

```
{
    "datasetId": "WyIxa2dlbm9tZSJd",
    "filters": [
        {
            "field": "ethnicity",
            "operator": "in",
            "values": [
                "GBR",
```

(continues on next page)

(continued from previous page)

```
                "FIN",
                "ESN"
            ]
        }
    ]
}
```

2.1.2 Sample queries for all variants services

This mainly includes the /variantsets/search, /variants/search and /variantsbygene endpoints.

Sample Query I

Description: Fetch all of the variantsets associated with a particular dataset.

Endpoint: *variantsets/search*

Note: This query is the same as Sample Query I under Metadata services, but it is the same across metadata, variantSets, referenceSets, etc.

```
{
    "datasetId": "WyIxa2dlbm9tZSJd"
}
```

Sample Query II

Description: Search for variants within the range between the start and end that are on chromesome 22, from the designated variantSets.

Endpoint: *variants/search*

```
{
    "start": "50158561",
    "end": "50158565",
    "referenceName": "22",
    "variantSetIds": [
        "yourVariantSetId1",
        "yourVariantSetId2"
    ]
}
```

Sample Query III

Description: Search for variants within the range between the start and end that are on chromesome 22, from all variantsets that are associated with one particular dataset.

Endpoint: *variants/search*

Warning: You should never attempt to specify both datasetId and variantSetIds.

```
{
    "datasetId": "WyIxa2dlbm9tZSJd",
    "start": "50158561",
    "end": "50158565",
    "referenceName": "22"
}
```

Sample Query IV

Description: Search for variants that are associated with a particular gene.

Warning: Do not use `/variantsbygenesearch` endpoint, it has been deprecated.

Endpoint: */variantsbygenesearch* or */variants/gene/search*

```
{
    "datasetId": "WyIxa2dlbm9tZSJd",
    "gene": "ABCD",
}
```

2.1.3 Instructions for /search and /count endpoints

You need to write complex queries to be able to use the `/search` and `/count` endpoints. It has 4 mandatory fields, `datasetId`, `logic`, `components`, and `results`. Queries for both endpoints are largely the same, and the differences will be explained below.

You always need to specify datasetId in your query.

You may want to look at the sample queries first before you can look at the *how to* instructions below.

Warning: You may specify `pageToken` only when you see a `nextPageToken` returned in the previous response. You cannot set `pageSize` for requests made to `/search` and `/count` endpoints.

How to write logic

Logic is where you specify the relationship between various components. The only operators you will need to specify are either AND or OR. When writing the logic, the operation becomes the key.

Write Logic for multiple components

This is possibly the most common use-case, where you want to find records that satisfy multiple filters you set.

For example, conditionA and conditionB and condition C would be written as below.

In this example, the records will have to satisfy conditions of all three components.

```
{
    "logic": {
        "and": [
            {
                "id": "conditionA"
            },
            {
                "id": "conditionB"
            },
            {
                "id": "conditionC"
            }
        ]
    }
}
```

In the above example, you can replace the *and* key with *or*. In this case, the records will only need to fulfill the condition of any single component.

Write Logic for 1 component

When you only have 1 component in your query, however, you may only specify *id*.

```
{
    "logic": {
        "id": "condition1"
    }
}
```

For this case only, that is, when you only have one single *id* in your logic. Optionally, you can specify the *negate* flag, which would basically negate the logic of the componenet.

```
{
    "logic": {
        "id": "condition1",
        "negate": true
    }
}
```

Write Logic for nested components

It is possible to write more complex logic, with multiple nested operations involved. However, the examples explained above should suffice most basic needs.

As an example, the following request is the equivalent to `AB(CD)`, which is equivalent to `(AB)   (CD)`.

```
{
    "logic": {
        "and": [
            {
                "id": "A"
            },
            {
                "id": "B"
```

(continues on next page)

(continued from previous page)

```
            },
            {
                "or": [
                    {
                        "id": "C"
                    },
                    {
                        "id": "D"
                    }
                ]
            }
        ]
    }
}
```

How to write components

The `components` part is a list, each corresponding to a filter of a specified table. Be careful that the id has to match with the one you specified in the logic part of your query. It can be almost any string, but they have to match.

In a `component`, you specify the tables you want to search on to be the key.

There are 3 different types of `components` objects.

Components for Clinical tables

```
{
    "components": [
        {
            "id": "condition1",
            "patients": {
                "filters": [
                    {
                        "field": "provinceOfResidence",
                        "operator": "!=",
                        "value": "Ontario"
                    }
                ]
            }
        }
    ]
}
```

You write the filter objects the same way you would write for individual endpoints. If you need a reminder on that, check *How to write filter objects*.

Components for /variants/search endpoint

```
{
    "components": [
        {
            "id": "condition1",
```

(continues on next page)

(continued from previous page)

```
            "variants": {
                "start": "100000",
                "end": "500000",
                "referenceName": "1"
            }
        }
    ]
}
```

Note that you can also specify *variantSetIds* in here, which will limit the scope to your list of *variantSetIds*. If you don't specify any, by default, it will try to search through all variantSets associated with this dataset.

Components for /variantsbygenesearch endpoint

Warning: Note that this component is deprecated, and may be removed in subsequent releases. Use *Components for /variants/gene/search endpoint*.

```
{
    "components": [
        {
            "id": "condition1",
            "variantsByGene": {
                "gene": "MUC1"
            }
        }
    ]
}
```

Components for /variants/gene/search endpoint

```
{
    "components": [
        {
            "id": "condition1",
            "variants": {
                "gene": "MUC1"
            }
        }
    ]
}
```

How to write results

In the `results` part of your query, you will need to specify the table you want the server to return. For a query made to the /search endpoint, you can simply specify the table name.

Warning: The only endpoints that are accepted here are all clinical metadata endpoints, as well as `variants`.

Results section for Clinical tables

```
{
    "results": {
        "table": "patients",
        "fields": ["gender", "ethnicity"]
    }
}
```

Warning: *fields* is a list of fields that you want the server to return. It is optional for /search endpoint, but mandatory for */count* endpoint. If you do not specify this in */search* endpoint, the server will just return all the fields.

Results section for /variants endpoint

Warning: Please be considerate when you are submitting any */variants* request, we recommend you to not search more than 1 million bps at a time. If you have a lot of variantSets, you should limit your search size to 100,000.

```
{
    "results": {
        "table": "variants",
        "start": "1232555",
        "end": "1553222",
        "referenceName": "1"
    }
}
```

Results section for /variantsbygenesearch endpoint

Warning: This endpoint is deprecated. Use *Results section for /variants/gene/search endpoint*.

Warning: Please note that while you need to specify the table name to be *variantByGene*, it still returns a list of variants in its response.

```
{
    "results": {
        "table": "variantsByGene",
        "gene": "MUC1"
    }
}
```

Results section for /variants/gene/search endpoint

```
{
    "results": {
        "table": "variants",
        "gene": "MUC1"
    }
}
```

Sample Query I

Description: Return a list of patients, whose diseaseResponseOrStatus is “Complete Response”, AND have a courseNumber that is not 100.

> **Warning:** The example query below only works for the /search endpoint, as it did not specify *fields*.

Query:

```
{
    "datasetId": "yourDatasetId",
    "logic": {
        "and": [
            {
                "id": "A"
            },
            {
                "id": "B"
            }
        ]
    },
    "components": [
        {
            "id": "A",
            "outcomes": {
                "filters": [
                    {
                        "field": "diseaseResponseOrStatus",
                        "operator": "==",
                        "value": "Complete Response"
                    }
                ]
            }
        },
        {
            "id": "B",
            "treatments": {
                "filters": [
                    {
                        "field": "courseNumber",
                        "operator": "!=",
                        "value": "100"
                    }
                ]
            }
```

(continues on next page)

(continued from previous page)

```
        }
    ],
    "results": [
        {
            "table": "patients"
        }
    ]
}
```

Sample Query II

Description: Return the aggregated stats on patients' gender and ethnicity who have mutations present between "50158561" and "50158565" on chromosome 22, from the list of variantsetIds.

```
{
    "datasetId": "yourDatasetId",
    "logic": {
        "id": "A"
    },
    "components": [
        {
            "id": "A",
            "variants": {
                "start": "50158561",
                "end": "50158565",
                "referenceName": "22",
                "variantSetIds": [
                    "yourVariantSetId_1",
                    "yourVariantSetId_2",
                    "yourVariantSetId_3",
                    "yourVariantSetId_4",
                    "yourVariantSetId_5",
                    "yourVariantSetId_6",
                    "yourVariantSetId_7",
                    "yourVariantSetId_8"
                ]
            }
        }
    ],
    "results": [
        {
            "table": "patients",
            "fields": [
                "gender",
                "ethnicity"
            ]
        }
    ]
}
```

Sample Query III

Description: Return the aggregated stats on patients' gender and ethnicity, who have mutations present between "50158561" and "50158565" on chromosome 22.

Note: Note: Since a list of `variantSetIds` was not specified, the server will attempt to locate all variantSets associated with the dataset. If you have a lot of variantSets associated with this particular dataset, the query might take some time.

```
{
    "datasetId": "yourDatasetId",
    "logic": {
        "id": "A"
    },
    "components": [
        {
            "id": "A",
            "variants": {
                "start": "50158561",
                "end": "50158565",
                "referenceName": "22"
            }
        }
    ],
    "results": [
        {
            "table": "patients",
            "fields": [
                "gender",
                "ethnicity"
            ]
        }
    ]
}
```

Sample Query IV

Description: Retrieve all the variants between 50100000 and 50158565 on chromosome 22 associated with an individual `[HG00105]`.

```
{
    "datasetId": "yourDatasetId",
    "logic": {
        "id": "A"
    },
    "components": [
        {
            "id": "A",
            "patients": {
                "filters": [
                    {
                        "field": "patientId",
                        "operator": "==",
                        "value": "HG00105"
                    }
                ]
            }
        }
    ],
```

(continues on next page)

(continued from previous page)

```
    "results": [
        {
            "table": "variants",
            "start": "50100000",
            "end": "50158565",
            "referenceName": "22"
        }
    ]
}
```

CHAPTER

THREE

APPENDIX

3.1 Contribution Guideline

> **Warning:** This guide is a work in progress, and is incomplete.

If you encounter a bug, or have a problem of using the package, please contact us by opening an issue at https://github.com/candig/candig-server.

3.1.1 GitHub workflow

We mainly employ three different types of branches: feature branches, develop branch, and master branch.

Feature branches are used to resolve a limited set of issues, and typically follows the naming convention of `username/fix_one_particular_issue`. When initiating a PR, you should request it to be merged back into the `develop` branch. The commits in individual feature branches are usually squashed, and code review usually happens at this step.

Develop branch is used to host code that has passed all the tests, but may not yet be production-ready, As a developer, you are welcome to play with this branch to test some of the new functionalities.

Commits in the `develop` branch is merged into the `master` branch once in a while. Release is tagged directly from the `master` branch, and follows the convention of `vX.Y.Z` (X, Y, Z are all integers.

A `hotfix` branch may be employed to fix production-critical issue that was later found on the `master` branch. They are parched directly to the `master` branch, and will be synced back to `develop` branch later.

If you would like to contribute code, please fork the package to your own git repository, then initiate a PR to be merged into develop.

3.2 Status

Please refer to https://github.com/CanDIG/candig-server/releases for the latest release notes.

3.2.1 Release Notes

1.1.0

https://github.com/CanDIG/candig-server/releases/tag/v1.1.0

1.0.3

https://github.com/CanDIG/candig-server/releases/tag/v1.0.3

1.0.2

https://github.com/CanDIG/candig-server/releases/tag/v1.0.2

1.0.1

https://github.com/CanDIG/candig-server/releases/tag/v1.0.1

1.0.0

https://github.com/CanDIG/candig-server/releases/tag/v1.0.0