Database Query Tool
Users Guide & Code Overview

5/15/2020

Mike Nute

Contents

User's Guide: Setup Command

- There is a command to be run on first download (ideally only once):

```
python3 query_tool.py --setup
```

- Does 3 things:
  1. Copies the config file from the 'doc' folder into the main folder. Sets the working folder in the config file to be the folder containing `query_tool.py`.
  2. Extracts the packaged containment file `containment_dict.json.gz`.
  3. Downloads and extracts the file containing the current NCBI taxonomy. (Creates a subfolder for it if it does not exist.)

Structure of the Containment JSON File

containment_dict.json:

```
{“metadata”:
    {“refseq1”: {<refseq1_properties>},
     “refseq2”: {…},
     “MTSV”: {<MTSV_properties>},
     (et cetera)…
    }
 “taxid_lists”:
    {“refseq1”: [taxon1, taxon3, …],
     “refseq2”: [taxon1, taxon3, taxon4…],
     “MTSV”: [<MTSV_taxa>],
     (et cetera)…
    }
}
```

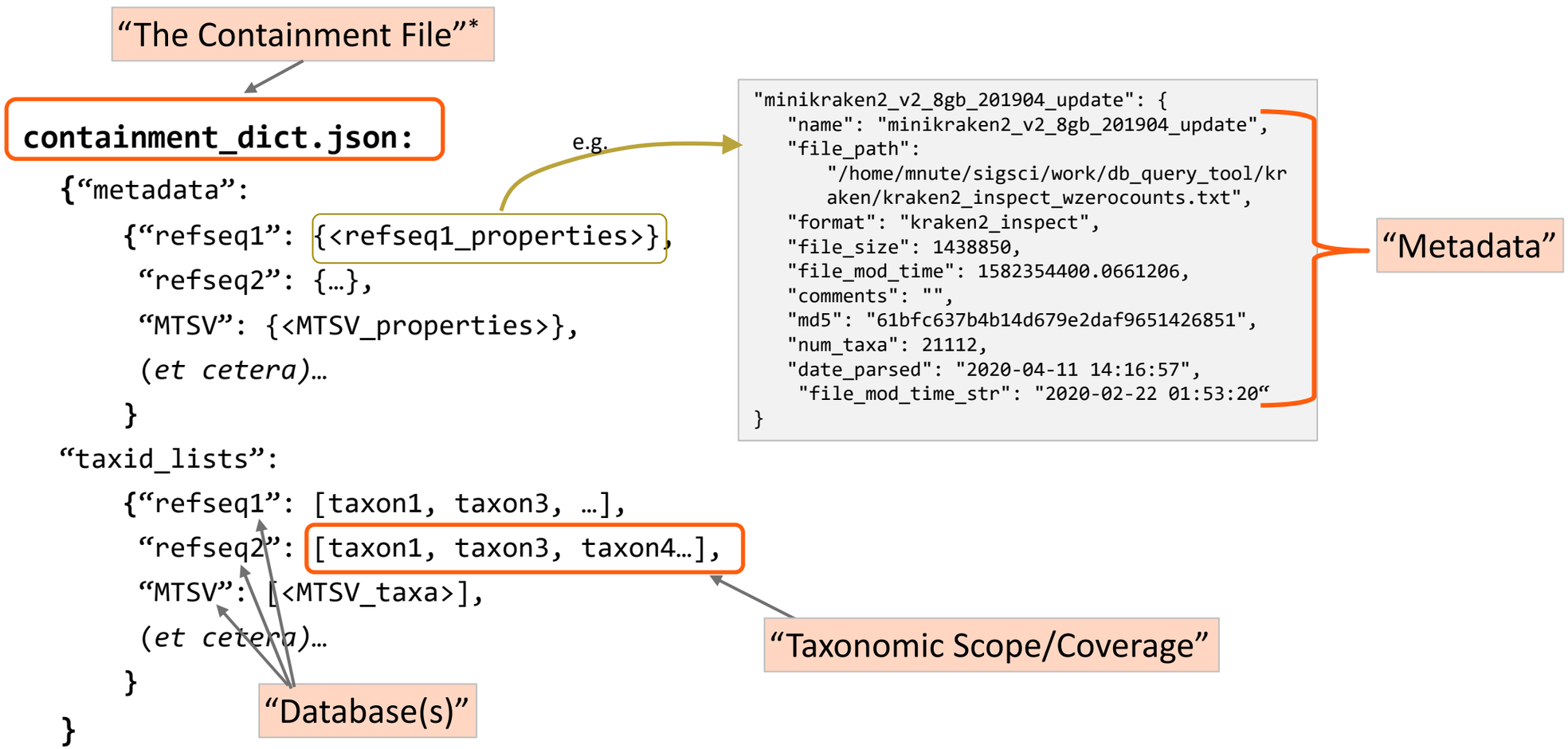
e.g.


```
"minikraken2_v2_8gb_201904_update": {
   "name": "minikraken2_v2_8gb_201904_update",
   "file_path":
      "/home/mnute/sigsci/work/db_query_tool/kr
      aken/kraken2_inspect_wzerocounts.txt",
   "format": "kraken2_inspect",
   "file_size": 1438850,
   "file_mod_time": 1582354400.0661206,
   "comments": "",
   "md5": "61bfc637b4b14d679e2daf9651426851",
   "num_taxa": 21112,
   "date_parsed": "2020-04-11 14:16:57",
    "file_mod_time_str": "2020-02-22 01:53:20“
}
```


- JSON file opens to python dictionary with two items.
- When file is opened:
  - Only “metadata” object is kept
  - Each taxon list moved into metadata dictionary
  - Each taxon list is converted to a python set
- Other notes:
  - This layout is for JSON readability
  - Current Version has 89 versions of RefSeq

Metadata for a Database of Database Metadata…?

Consistent Vocabulary is Needed:



*"Containment Dict" will refer to this object after is has been imported and in use in the code, where it as a slightly different form.

`dbqt_config`: Persistent Settings in the Config File

- `working_folder = ~/sigsci/metagenomics/scripts` [FOLDER]
  - Must be set first. No functionality unless this is a valid folder.
  - Sensible setup is to let other paths be relative to working folder (as below)

- `path_to_containment_file = ${working_folder}/containment_dict.json` [FILE]
  - Needs to be given for most operations.
  - Must be a valid file unless using command to build from scratch.

- `path_to_ncbi_taxonomy_nodes = ${working_folder}/ncbi_taxonomy/nodes.dmp` [FILE]
  - Path to 'nodes.dmp' file from NCBI reference taxonomy. Needed to query taxa against containment.
  - Command is available to automatically download this (using flag '`--download_ncbi_taxonomy`')
    - Parent folder of '`ncbi_taxonomy`' folder must be a valid directory. '`ncbi_taxonomy`' subfolder will be created.

- `path_to_db_import_manifest = ${working_folder}/db_import_manifest.txt` [FILE]
  - Used for building or updating the containment file.
  - A delimited text file with a manifest of files that should be imported.

- `refseq_folder = ${working_folder}/refseq/catalog_taxid` [FOLDER]
  - Partly deprecated now. A folder containing all the refseq databases.

Importing a New Database (MetaPhlan2 Example)

- Step 1: Find out where in the database the Taxon IDs are located
  - Example: MetaPhlan2 → sequence IDs contain either NCBI Accession ID or NCBI Gene ID:

```
>gi|345004010|ref|NC_015954.1|:c1336247-1335993
…
>gi|345004010|ref|NC_015954.1|:2775135-2775383
…
```

- [1]Step 2: Convert it to a delimited text file
  - Example: Use python/bash to a) make a list of NCBI Accession/Gene IDs and b) look up Taxon ID:

```
NZ_KI271599     1231336 accn
NZ_GG696043     621372  accn
NZ_KB850728     437663  accn
NZ_KB899102     1122202 accn
```

- Step 3: Define format specification and add this file to the manifest. Run "build_containment_dict" command.

[1]See the commands in the bash script `prepare_taxid_metadata.sh` for examples of how this is done.

Importing a New Database: Manifest Specifications

- Tab-delimited text file
- Single row header (see right)
- 4 Columns, in Order:
  - Name of the database
  - Path to the delimited text file
  - Format of the text file (see below)
  - Whether to proceed with import (1 = proceed, 0 = skip).
- Format of delimited text file:
  - Delimiter
  - Position of taxid column (0-index)
  - # Header rows to skip

Tables from db_import_manifest.xlsx

DB_Name	Path	Format	Import	Comments
RefSeq	$WORK/db_query_tool/refseq/catalog_taxid	refseq	1	RefSeq is par
minikraken_20171019_8GB	$WORK/db_query_tool/krakenuniq/seqid2taxi	seqid2taxid	1	(same forma
minikraken2_v2_8GB_201904_UPDATE	$WORK/db_query_tool/kraken/kraken2_inspect_wzerocounts.txt	kraken2_inspect	1	(made from c
kaiju_db_nr_euk	$WORK/db_query_tool/kaiju/nr_euk_2017_seqnames_fromPeter/kaiju_db_nr_euk_origUniqTa	first_col	1	(included wit
NCBI_nucl_gb	$WORK/db_query_tool/ncbi_taxonomy/accn2taxid_orig/nucl_gb.accession2taxid	accn2taxid	1	(from accessi
NCBI_nucl_wgs	$WORK/db_query_tool/ncbi_taxonomy/accn2taxid_orig/nucl_wgs.accession2taxid	accn2taxid	1	(from accessi
metaphlan_mpa_v20_m200	$WORK/humann2/metaphla 20_m200.TaxonIDlist.FINAL.			
MasonLab_Covid_Kraken_microDB_20200313	$WORK/db_query_tool/cov seqid2taxid.map	seqid2taxid	1	(kfrom seqid
MTSV_May-22-2019	$WORK/mtsv_db/20190522_taxid_list.txt	first_col	1	(based on ac
MTSV_Oct-28-2019	$WORK/mtsv_db/20191028_taxid_list.txt	first_col	1	(based on ac

This table can be saved as tab-delimited text to become a db_import_manifest file.

Format Specifications:

Name	delimiter	Taxid Col	Header	Config Param Line
kraken2_inspect	\t	4	0	kraken2_inspect = ('\t',4,0)
first_col	\t	0	0	first_col = (
refseq	\t	0	0	refseq = ('\
accn2taxid	\t	2	1	accn2taxid
seqid2taxid	\t	1	0	seqid2taxid = ('\t',1,0)

This tab is just a shortcut to making the strings for the config file.

Importing a New Database: Format Definitions in **dbqt_config**

- Recall: Import functions pull Taxid list from a *single column* of a *delimited text* file.
  - Taxon ID metadata usually in this form (or easily converted).
  - "Format" describes how to do this.
- A "format" needs:
  1. Delimiter
  2. Taxon ID Column Position (0-indexed)
  3. # Header Rows to skip
- Other Notes:
  - Format given in config file must be valid python code (see Fig. 1)
- **Example**: **`minikraken_20171019_8GB`**
  - Taxon list provided by developer in file called 'seqid2taxid' (Fig 2)
  - File is tab-delimited, taxon ID in second column (index = 1), no header
  - Thus format spec 'seqid2taxid' in dbqt_config (Fig. 1)

```
gi|6446580|ref|NC_000942.1|     129956
gi|6446580|ref|NC_000942.1|:10885-12369 129956
gi|6446580|ref|NC_000942.1|:1127-1654   129956
gi|6446580|ref|NC_000942.1|:12420-13889 129956
gi|6446580|ref|NC_000942.1|:13986-14585 129956
gi|6446580|ref|NC_000942.1|:14625-15311 129956
gi|6446580|ref|NC_000942.1|:1522-2815,2885-2892 129956
gi|6446580|ref|NC_000942.1|:15347-15571 129956
gi|6446580|ref|NC_000942.1|:15632-16345 129956
gi|6446580|ref|NC_000942.1|:16432-19161 129956
```

Fig 2: seqid2taxid file provided by kraken developers, corresponding to database file 'minikraken_20171019_8GB'.

```
[formats]
# Each of these defines a format for pulling taxon ids
from a delimited text file.
#   They must be written as a valid python string
containing a tuple. The fields
#   in each tuple are: 1) delimiter, 2) column (0-
indexed), 3) # of header rows to
#   skip.
accn2taxid = ('\t', 2, 1)
kraken2_inspect = ('\t', 4, 0)
first_col = ('\t', 0, 0)
refseq = ('\t', 0, 0)
seqid2taxid = ('\t', 1, 0)
```

Fig 1: Format Section from dbqt_config

Importing a New Database: Command-Line Arguments

- Command line flags available:
  - `-ICD, --cmd_inspect_contain`
    - Test opens the containment file and prints a summary of its contents. Mostly for debugging.
    - Was more useful before switching to readable JSON files.
  - `-IFL, --cmd_inspect_filelist`
    - Opens a db_import_manifest manifest and checks all of the files, paths and names in it for validity. Prints a summary of the manifest to the console.
  - `-CMO, --cmd_compare_sources`
    - Combination of the two above to generate a plan for building/updating the containment file.
    - Compares a file-import manifest to the contents of an existing containment file and determines which ones are overlapping or identical and which ones are new and must be added.
    - Any files with identical size and time_modified as in existing containment file are skipped.
  - `-BCD, --cmd_build_containment`
    - Runs same procedure as previous, but then executes the plan.
    - If the argument **`'--clober'`** is given, will overwrite previous containment_file.
- *Each flag has essentially its own subroutine to execute, though there is some sharing of functions.*

- Important Note:
  - When adding new a database(s) to the containment file, this program assumes that the taxa included will be found in a *single column of a delimited text file*.
  - Column can include duplicates, program will handle.
  - As long as this format is kept, importing can be done automatically.
  - Taxonomic information is often provided by developers in a delimited text format, though if it is not, any work to prepare that format is beyond the scope of the DBQT.
  - Any files containing the taxonomic coverage for a new database to be imported can be specified either in the config file or through a manifest file (in the code always called the '**`db_import_manifest`**'.

Querying

- **Currently only one command flag for this:**

  `-QRY, --cmd_query_taxids`

  - For each taxon ID and each database, 3 possible outcomes:
    - 1: Taxon ID is present in that database
    - 2: Taxon ID is not present but its species-level ancestor is
    - 0: Neither is present
  - Prints results to output file in the form at right.

- **Notes:**
  - For taxon IDs above species level, only outcomes 1/0 possible.
  - Taxon Id given using '-t' argument:

    `python3 query_tool.py –QRY –t <t-arg>`
  - <t-arg> can be:
    a) a single taxon id,
    b) a file containing a line-separated list of taxon ids,
    c) 'stdin' and then a line-separated list of taxon ids piped in.

Sample Results

```
DB Column Names:
   1: minikraken_20171019_8GB
   2: minikraken2_v2_8GB_201904_UPDATE
   3: kaiju_db_nr_euk
   4: NCBI_nucl_gb
   5: NCBI_nucl_wgs
   6: RefSeq_v98

    taxid      rank 0 1 2 3 4 5
  1913708   species - - 1 1 - -
   980453   species - - - 1 - -
   146582   species - - - 1 - -
  1950923   species - - - - 1 -
  1420363   species - - - 1 - -
  1367599   species - - - 1 - -
    48959   species - - - 1 - -
  1594871   species - - - 1 - -
    69507     genus - - - - - -
   241522 subspecies - - - 1 - -
  1068967   species - - - 1 - -
  1007150   species - - - 1 - -
   498356   no rank - 2 1 1 - -

(...truncated...)
```

Code Review: Adminstration

- global object **`options`** stores all settings
  - From config file and command line
- Two primary functions to populate it and reconcile validity of settings:
  - `def command_args_parse()`
    - Mainly just defines all of the available command line (CL) arguments and adds them to options using the **`argparse`** module.
  - `def command_args_postprocess():`
    - Imports config file parameters and reconciles with CL args
    - Sets up event logging using python **`logging`** module
    - Checks validity of config parameters and overrides them with CL args (if specially given) or defaults if invalid.
- Command arguments determine program execution
  - Only one allowed per execution
  - Variable names begin with **`cmd_`**
  - Funcion **`main()`** executes command parsing, calls one of the functions based on the command argument
- Several functions/commands for debugging hidden from help menu.

Code Review: NCBI Taxonomy Handling

- Downloading NCBI Reference Taxonomy
  - The latest NCBI Reference Taxonomy database is always located at:

    ftp://ftp.ncbi.nlm.nih.gov/pub/taxonomy/taxdmp.zip
  - User must download this and extract the file ‘nodes.dmp’, specify that path in the config.
  - A command flag has been provided to do this automatically using the command:

    ```
    python3 query_tool.py --download_ncbi_taxonomy
    ```
- Parsing **`nodes.dmp`**: `ncbi_taxonomy_parse_file()`
  - Returns a `dict` keyed by taxon_id (herein: the “NCBI dict”):

    ```
    {<taxon_id>: (<par>, <rank>, <assn_at_species>), … }
    ```
    - **Par**: taxon ID of parent
    - **Rank**: level of the taxonomy (e.g. phylum, species, etc…) (See Right). A python list of taxonomic ranks in descending order is in the code.
    - **Assn_at_species**: true if this taxon is at or below the species level and its ancestral lineage has an assignment at the species level.

Num	Rank
0	no rank
1	superkingdom
2	kingdom
3	subkingdom
4	superphylum
5	phylum
6	subphylum
7	superclass
8	class
9	subclass
10	infraclass
11	cohort
12	subcohort
13	superorder
14	order
15	suborder
16	infraorder
17	parvorder
18	superfamily
19	family
20	subfamily
21	tribe
22	subtribe
23	genus
24	subgenus
25	section
26	subsection
27	series
28	species group
29	species subgroup
30	species
31	subspecies
32	varietas
33	forma

Code Review: NCBI Taxonomy Handling (continued)

- Other Functions:
  - `def ncbi_taxonid_to_lineage_vector(taxid, ncbi_dict):`
    - Input: Taxon ID <taxid> (and the NCBI dict)
    - Output: a python list (length 34) representing the taxon IDs in the ancestral lineage of <taxid>
      - Unoccupied ranks are populated with -1
    - This is useful for calibrating a set of taxa at various ranks to their equivalent at a single rank.
  - `def ncbi_taxonomy_make_full_vector_lookup(ncbi_dict):`
    - Makes a lookup dictionary where the values are each full-length lineage vectors (as above).
    - I'm not positive whether this is currently used anywhere.

```
ncbi_tax_levels = ['no rank', 'superkingdom', 'kingdom', 'subkingdom', 'superphylum',
   'phylum', 'subphylum', 'superclass', 'class', 'subclass', 'infraclass', 'cohort',
   'subcohort', 'superorder', 'order', 'suborder', 'infraorder', 'parvorder', 'superfamily',
   'family', 'subfamily', 'tribe', 'subtribe', 'genus', 'subgenus', 'section', 'subsection',
   'series', 'species group', 'species subgroup', 'species', 'subspecies', 'varietas', 'forma']
```

Appendix: Changes Since the 2019 DQT

- Codebase is entirely functional now.
  - Executed at commandline with commandline arguments and subroutines to control program flow.
- Consolidated down to only a single pickled file for containment DB.
- Containment file as JSON instead of pickle
  - Human readability convenience
  - 75MB instead of 25MB for the containment_dict
- Config file for persistent parameters
  - Python ConfigParser module to interpret.
- Methods for constructing/inspecting/updating the containment file.
- Outputs print more consicely and readably.
- Lots of documentation in the code.