
BioThings SDK

BioThings team

Feb 28, 2024

PRODUCTS

1	What's BioThings?	1
2	BioThings SDK	3
3	BioThings API	5
4	BioThings Studio	7
5	Installation	9
6	Quick Start	11
6.1	BioThings Studio	11
6.2	BioThings CLI	66
6.3	BioThings Standalone	74
6.4	BioThings Hub	85
6.5	BioThings Web	110
6.6	DataTransform Module	117
6.7	biothings.web	122
6.8	biothings.tests	150
6.9	biothings.utils	153
6.10	biothings.hub	217
6.11	biothings.cli	308
	Python Module Index	313
	Index	315

CHAPTER
ONE

WHAT'S BIOTHINGS?

We use “**BioThings**” to refer to objects of any biomedical entity-type represented in the biological knowledge space, such as genes, genetic variants, drugs, chemicals, diseases, etc.

**CHAPTER
TWO**

BIOTHINGS SDK

SDK represents “Software Development Kit”. BioThings SDK provides a [Python-based](#) toolkit to build high-performance data APIs (or web services) from a single data source or multiple data sources. It has the particular focus on building data APIs for biomedical-related entities, a.k.a “*BioThings*”, though it’s not necessarily limited to the biomedical scope. For any given “*BioThings*” type, BioThings SDK helps developers to aggregate annotations from multiple data sources, and expose them as a clean and high-performance web API.

The BioThings SDK can be roughly divided into two main components: data hub (or just “hub”) component and web component. The hub component allows developers to automate the process of monitoring, parsing and uploading your data source to an [Elasticsearch](#) backend. From here, the web component, built on the high-concurrency [Tornado Web Server](#), allows you to easily setup a live high-performance API. The API endpoints expose simple-to-use yet powerful query features using [Elasticsearch’s full-text query capabilities](#) and [query language](#).

CHAPTER
THREE

BIOTHINGS API

We also use “*BioThings API*” (or *BioThings APIs*) to refer to an API (or a collection of APIs) built with BioThings SDK. For example, both our popular [MyGene.Info](#) and [MyVariant.Info](#) APIs are built and maintained using this BioThings SDK.

**CHAPTER
FOUR**

BIO THINGS STUDIO

BioThings Studio is a buildin, pre-configured environment used to build and administer BioThings API. At its core is the *Hub*, a backend service responsible for maintaining data up-to-date, producing data releases and update API frontends.

CHAPTER

FIVE

INSTALLATION

You can install the latest stable BioThings SDK release with pip from PyPI, like:

```
pip install biothings
```

You can install the latest development version of BioThings SDK directly from our github repository like:

```
pip install git+https://github.com/biothings/biothings.api.git#egg=biothings
```

Alternatively, you can download the source code, or clone the [BioThings SDK](#) repository and run:

```
python setup.py install
```


QUICK START

We recommend to follow [this tutorial](#) to develop your first BioThings API in our pre-configured BioThings Studio development environment.

6.1 BioThings Studio

BioThings Studio is a pre-configured environment used to build and administer BioThings API. At its core is the **Hub**, a backend service responsible for maintaining data up-to-date, producing data releases, and updating API frontends.

6.1.1 A. Tutorial

This tutorial will guide you through **BioThings Studio** by showing, in a first part, how to convert a simple flat file to a fully operational BioThings API. In a second part, this API will enrich for more data.

Note: You may also want to read the [developer's guide](#) for more detailed informations.

Note: The following tutorial uses a docker-compose file to run the **BioThings Studio** and **Hub**. This file is available [here](#)

1. What you'll learn

Through this guide, you'll learn:

- how to run a docker-compose to run your favorite API
- how to run that image inside a Docker container and how to access the **BioThings Studio** application
- how to integrate a new data source by defining a data plugin
- how to define a build configuration and create data releases
- how to create a simple, fully operational BioThings API serving the integrated data
- how to use multiple datasources and understand how data merge is done

2. Prerequisites

Using **BioThings Studio** requires a Docker server up and running, some basic knowledge about commands to run and use containers. Images have been tested on Docker >=17.

You can install your own Docker server (on recent Ubuntu systems, `sudo apt-get install docker.io` is usually enough). You may need to point Docker images directory to a specific hard drive to get enough space, using `-g` option:

```
# /mnt/docker points to a hard drive with enough disk space
sudo echo 'DOCKER_OPTS="-g /mnt/docker"' >> /etc/default/docker
# restart to make this change active
sudo service docker restart
```

Alternatively, if you have a Mac or Windows, you can install [Docker Desktop](#). It will install the docker server for you. Once you have Docker Desktop installed, go to settings->resources->advanced. You should give at least 80% of your resources to Docker for each category. This will prevent your Docker from crashing if you are running a large datasource or build.

3. Installation

BioThings Studio is available as a docker-compose file at our [github repository](#). Clone the repository to your local.

A **BioThings Studio** instance exposes several services on different ports:

- **8080:** **BioThings Studio** web application port
- **7022:** **BioThings Hub** SSH port
- **7080:** **BioThings Hub** REST API port
- **7081:** **BioThings Hub** REST API port, read-only access
- **9200:** ElasticSearch port
- **27017:** MongoDB port
- **8000:** BioThings API, once created, it can be any non-privileged (>1024) port
- **9000:** [Cerebro](#), a webapp used to easily interact with ElasticSearch clusters

Note: Ports 8080, 7022, 7080, 9200, 27017, 8000, 9000 are exposed by default in the docker-compose.yml file.

```
$ docker compose up -d --build
```

We can follow the starting sequence using `docker logs` command:

```
$ docker logs -f biothings
ARG
SSH keys not yet created, creating
Generating SSH Keys for BioThings Hub...
SSH Key has been generated, Public Key:
```

Please refer to [Filesystem overview](#) and [Services check](#) for more details about Studio's internals.

We can now access **BioThings Studio** using the dedicated web application (see [webapp overview](#)).

4. Getting start with data plugin

In this section we'll dive in more details on using the **BioThings Studio** and **Hub**. We will be integrating a simple flat file as a new datasource within the **Hub**, declare a build configuration using that datasource, create a build from that configuration, then a data release and finally instantiate a new API service and use it to query our data.

The whole source code is available at <https://github.com/biothings/tutorials/tree/master>, each branch pointing to a specific step in this tutorial.

4.1. Input data

For this tutorial, we will use several input files provided by **PharmGKB**, freely available in their [download](#) section, under “Annotation data”:

- `annotations.zip`: contains a file `var_drug_ann.tsv` about variant-gene-drug annotations. We'll use this file for the first part of this tutorial.
- `drugLabels.zip`: contains a file `drugLabels.byGene.tsv` describing, per gene, which drugs have an impact of them
- `occurrences.zip`: contains a file `occurrences.tsv` listing the literature per entity type (we'll focus on gene type only)

The last two files will be used in the second part of this tutorial when we'll add more datasources to our API.

These files will be downloaded by the **Hub** when we trigger the dumper. These files will go into a folder named `data_folder` by default. This will be explained in more detail in the [Data Plugin](#) section.

4.2. Parser

In order to ingest this data and make it available as an API, we first need to write a parser. Data is pretty simple, tab-separated files, and we'll make it even simpler by using `pandas` python library. The first version of this parser is available in branch `pharmgkb_v1` at https://github.com/biothings/tutorials/blob/pharmgkb_v1/parser.py. After some boilerplate code at the beginning for dependencies and initialization, the main logic is the following:

```
def load_annotations(data_folder):
    results = []
    for rec in dat:

        if not rec["Gene"] or pandas.isna(rec["Gene"]):
            logging.warning("No gene information for annotation ID '%s'", rec["Annotation_ID"])
            continue
        _id = re.match(".* \((.*?)\)", rec["Gene"]).groups()[0]
        # We'll remove space in keys to make queries easier. Also, lowercase is preferred
        # for a BioThings API. We'll use an helper function `dict_convert()` from BioThings
        # SDK
        process_key = lambda k: k.replace(" ", "_").lower()
        rec = dict_convert(rec, keyfn=process_key)
        results.setdefault(_id, []).append(rec)

    for _id, docs in results.items():
        doc = {"_id": _id, "annotations" : docs}
        yield doc
```

Our parsing function is named `load_annotations`, it could be named anything else, but it has to take a folder path `data_folder` containing the downloaded data. This path is automatically set by the Hub and points to the latest version available. More on this later.

```
infile = os.path.join(data_folder, "var_drug_ann.tsv")
assert os.path.exists(infile)
```

It is the responsibility of the parser to select, within that folder, the file(s) of interest. Here we need data from a file named `var_drug_ann.tsv`. Following the moto “don’t assume it, prove it”, we make that file exists.

Note: In this case, an assertion isn’t necessary as code will fail anyway if the file doesn’t exist. But it’s a good practice to make sure the file exists before trying to open it. Also, it’s a good practice to use `os.path.join()` to build the path to the file, as it will automatically use the right path separator depending on the operating system.

```
dat = pandas.read_csv(infile, sep="\t", squeeze=True, quoting=csv.QUOTE_NONE).to_
      dict(orient='records')
results = []
for rec in dat:
    ...
```

We then open and read the TSV file using `pandas.read_csv()` function. At this point, a record `rec` looks like the following:

```
{'Alleles': 'A',
 'Annotation ID': 608431768,
 'Chemical': 'warfarin (PA451906)',
 'Chromosome': 'chr1',
 'Gene': 'EPHX1 (PA27829)',
 'Notes': nan,
 'PMID': 19794411,
 'Phenotype Category': 'dosage',
 'Sentence': 'Allele A is associated with decreased dose of warfarin.',
 'Significance': 'yes',
 'StudyParameters': '608431770',
 'Variant': 'rs1131873'}
```

Keys are uppercase, for a BioThings API, we like to have them as lowercase. More importantly, we want to remove spaces in those keys as querying the API in the end will be hard with spaces. We’ll use a special helper function from BioThings SDK to process these.

```
process_key = lambda k: k.replace(" ", "_").lower()
rec = dict_convert(rec, keyfn=process_key)
```

Finally, because there could be more than one record by gene (ie. more than one annotation per gene), we need to store those records as a list, in a dictionary indexed by gene ID. The final documents are assembled in the last loop.

```
...
results.setdefault(_id, []).append(rec)

for _id,docs in results.items():
    doc = {"_id": _id, "annotations" : docs}
    yield doc
```

Note: The `_id` key is mandatory and represents a unique identifier for this document. The type must be a string. The `_id` key is used when data from multiple datasources are merged together, that process is done according to its value (all documents sharing the same `_id` from different datasources will be merged together). Due to the indexing limitation, the length of the `_id` key should be kept no more than 512.

Note: In this specific example, we read the whole content of this input file in memory, then store annotations per gene. The data itself is small enough to do this, but memory usage always needs to be cautiously considered when we write a parser.

Note: In this case, the final documents are assembled within a generator function, which is a good practice to save memory. You may see within our [Biothings github organization](#) that we have plugins where we return a dictionary or a list of documents. This is also fine, but it is recommended to use a generator function when possible.

4.3. Data plugin

Parser is ready, it's now time to glue everything together and build our API. We can easily create a new datasource and integrate data using **BioThings Studio**, by declaring a *data plugin*. Such plugin is defined by:

- a folder containing a `manifest.json` file, where the parser and the input file location are declared
- all necessary files supporting the declarations in the manifest, such as a python file containing the parsing function for instance.

This folder must be located in the `plugins` directory (by default `/data/biothings_studio/plugins`, where the **Hub** monitors changes and reloads itself accordingly to register data plugins. Another way to declare such plugin is to register a `github` repository that contains everything useful for the datasource. This is what we'll do in the following section.

Note: Whether the plugin comes from a `github` repository or directly found in the `plugins` directory doesn't really matter. In the end, the code will be found in that same `plugins` directory, whether it comes from a `git clone` command while registering the `github` URL or from folder(s) and file(s) manually created in that location. However, when developing a plugin, it's easier to directly work on local files first so we don't have to regularly update the plugin code (`git pull`) from the webapp, to fetch the latest code. That said, since the plugin is already defined in `github` in our case, we'll use the `github` repo registration method.

The corresponding data plugin repository can be found at https://github.com/biothings/tutorials/tree/pharmgkb_v1. The manifest file looks like this:

```
{
  "version": "0.2",
  "requires" : ["pandas"],
  "dumper" : {
    "data_url" : ["https://s3.pgkb.org/data/annotations.zip",
                  "https://s3.pgkb.org/data/drugLabels.zip",
                  "https://s3.pgkb.org/data/occurrences.zip"],
    "uncompress" : true
  },
  "uploader" : {
    "url" : "https://s3.pgkb.org/data/upload"
  }
}
```

(continues on next page)

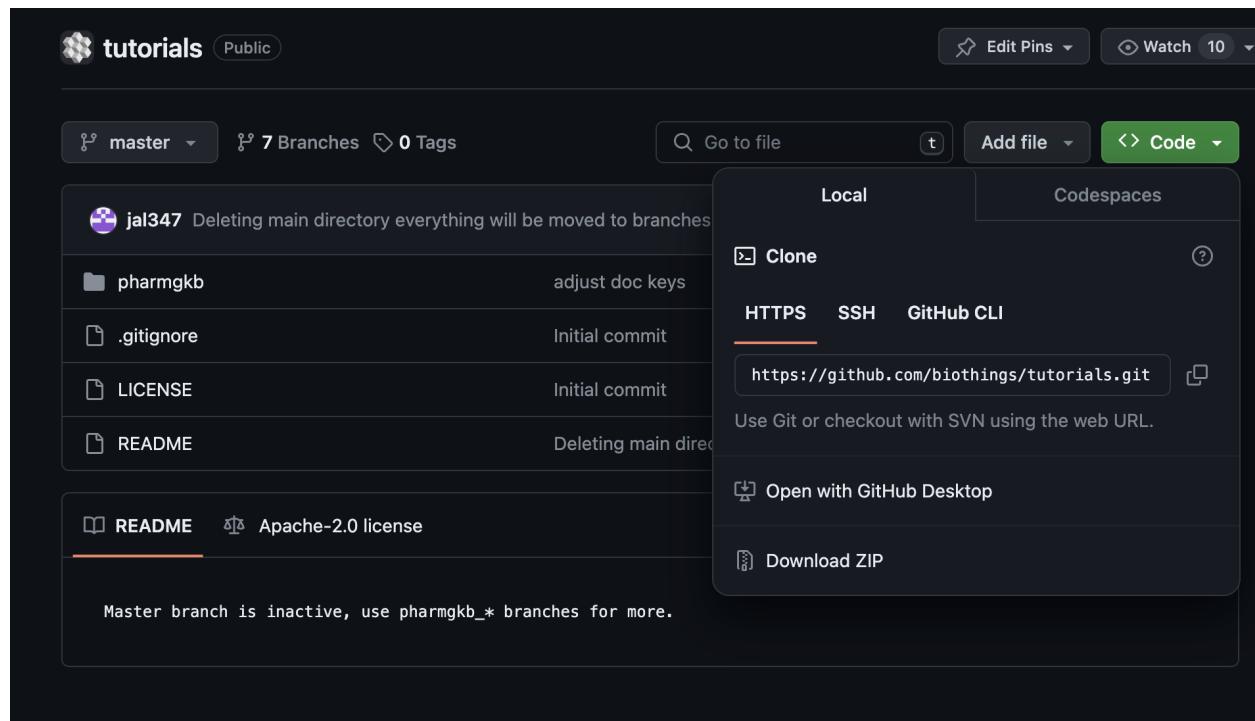
(continued from previous page)

```
"parser" : "parser:load_annotations",
"on_duplicates" : "error"
}
}
```

- *version* specifies the manifest version (it's not the version of the datasource itself) and tells the Hub what to expect from the manifest.
- parser uses pandas library, we declare that dependency in *requires* section.
- the *dumper* section declares where the input files are, using *data_url* key. In the end, we'll use 3 different files so a list of URLs is specified there. A single string is also allowed if only one file (ie. one URL) is required. Since the input file is a ZIP file, we first need to uncompress the archive, using *uncompress* : *true*.
- the *uploader* section tells the **Hub** how to upload JSON documents to MongoDB. *parser* has a special format, *module_name:function_name*. Here, the parsing function is named *load_annotations* and can be found in *parser.py* module. ‘*on_duplicates*’ : ‘*error*’ tells the **Hub** to raise an error if we have documents with the same *_id* (it would mean we have a bug in our parser).

For more information about the other fields, please refer to the [plugin specification](#).

Let's register that data plugin using the Studio. First, copy the repository URL:



Now go to the Studio web application at <http://localhost:8080>, click on the tab, then icon, this will open a side bar on the left. Click on *New data plugin*, you will be asked to enter the github URL. Click “OK” to register the data plugin.

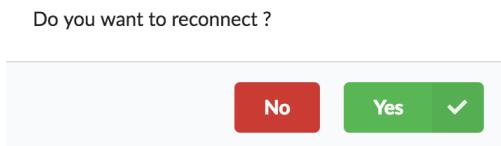


Interpreting the manifest coming with the plugin, **BioThings Hub** has automatically created for us:

- a *dumper* using HTTP protocol, pointing to the remote file on the CGI website. When downloading (or dumping) the data source, the dumper will automatically check whether the remote file is more recent than the one we may have locally, and decide whether a new version should be downloaded.
- and an *uploader* to which it “attached” the parsing function. This uploader will fetch JSON documents from the parser and store those in MongoDB.

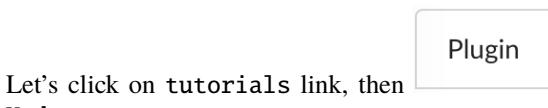
At this point, the **Hub** has detected a change in the datasource code, as the new data plugin source code has been pulled from github locally inside the container. In order to take this new plugin into account, the **Hub** needs to restart to load the code. The webapp should detect that reload and should ask whether we want to reconnect, which we’ll do!

Hub is restarting



Once you reconnect, you will have to do a hard refresh on your webpage, for example, `cmd + shift + r` on a Mac or `ctrl + shift + r` on a Windows/Linux.

Since we fetch source code from branch `master`, which doesn’t contain any manifest file. We need to switch to another branch (this tutorial is organized using branches, and also it’s a perfect opportunity to learn how to use a specific branch/commit using **BioThings Studio**...)



Let's click on `tutorials` link, then . In the textbox on the right, enter `pharmgkb_v1` then click on `Update`.

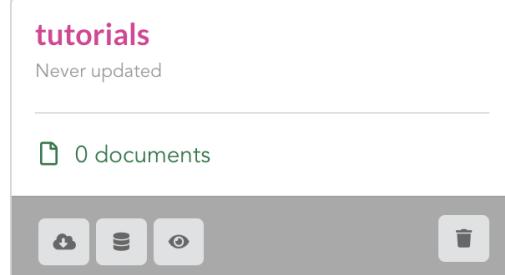


BioThings Studio will fetch the corresponding branch (we could also have specified a commit hash for instance), source code changes will be detected and the Hub will restart. The new code version is now visible in the plugin tab

Note: Remember to do a hard refresh again before continuing as the hub will attempt to restart.

Plugin	Mapping
URL	https://github.com/sirloon/pharmgkb.git
Release	pharmgkb_v1
Source folder	/data/biothings_studio/plugins/pharmgkb
Last download	2020-01-15T23:50:29.027Z (a few seconds ago)
Duration	0.58s

If we click back on  PharmGKB appears fully functional, with different actions available:



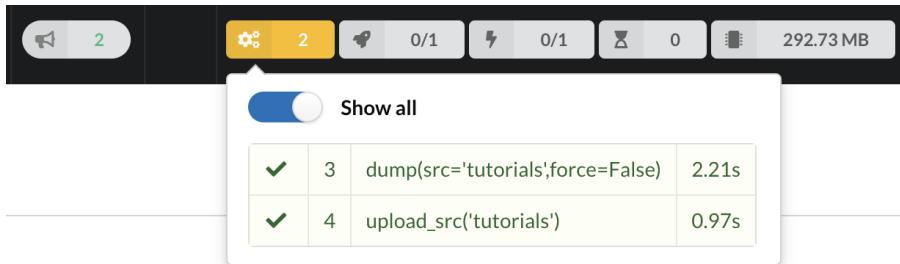
The screenshot shows the PharmGKB datasource page. At the top, there's a section for 'tutorials' with a note 'Never updated'. Below it is a section for '0 documents'. A grey toolbar at the bottom contains four icons: a cloud (Dumper), a cylinder (Uploader), an eye (Inspect), and a trash can (Delete). To the right of the toolbar is a large grey button labeled 'Sources'.

-  is used to trigger the dumper and (if necessary) download remote data
-  will trigger the uploader (note it's automatically triggered if a new version of the data is available)
-  can be used to “inspect” the data, more of that later

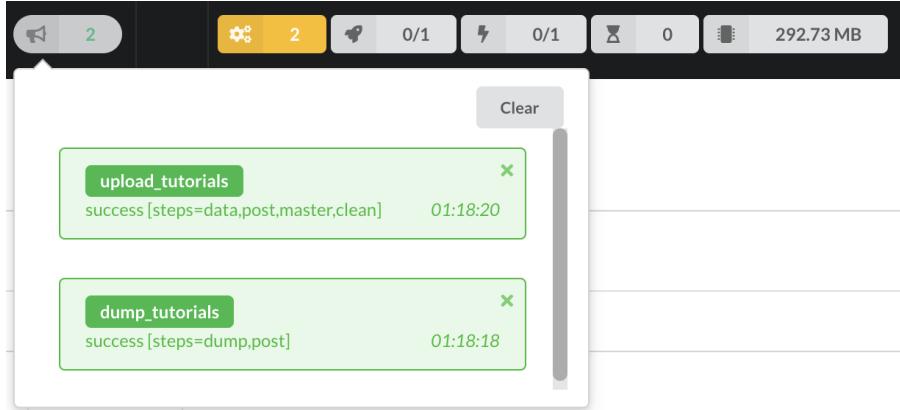
Let's open the datasource by clicking on its title to have more information. *Dumper* and *Uploader* tabs are rather empty since none of these steps have been launched yet. Without further waiting, let's trigger a dump to integrate this new

datasource. Either go to *Dump* tab and click on  or click on  to go back to the sources list and click on  at the bottom of the datasource.

The dumper is triggered, and after few seconds, the uploader is automatically triggered. Commands can be listed by clicking at the top of the page. So far we've run 3 commands to register the plugin, dump the data and upload the JSON documents to MongoDB. All succeeded.



We also have new notifications as shown by the speakerphone icon number on the left. Let's have a quick look:



Going back to the source's details, we can see the *Dumper* has been populated. We now know the release number, the data folder, when the last download was, how long it tooks to download the file, etc...

Dumper	Uploader	Plugin	Mapping
Release	2020-01-05		
Status	success		
Data folder	/data/biothings_studio/datasources/pharmgkb/2020-01-05		
Last download	2020-01-15T23:58:03.366Z (a minute ago)		
Duration	1.43s		
Dumper	biothings.hub.dataplugin.assistant.AssistedDumper_pharmgkb		

Same for the *Uploader* tab, we now have 979 documents uploaded to MongoDB. Exact number may change depending on when the source file that is downloaded.

Dumper	Uploader	Plugin	Mapping
<hr/>			
Release			2020-01-05
Data folder			/data/biotothings_studio/datasources/pharmgkb/2020-01-05
Status			success
Last upload			2020-01-15T23:58:10.015Z (2 minutes ago)
Duration			0.61s
Documents uploaded			979
Uploader			biotothings.hub.dataplugin.assistant.AssistedUploader_pharmgkb

4.4. Inspection and mapping

Now that we have integrated a new datasource, we can move forward. Ultimately, data will be sent to Elasticsearch, an indexing engine. In order to do so, we need to tell Elasticsearch how the data is structured and which fields should be indexed (and which should not). This step consists of creating a “mapping”, describing the data in Elasticsearch terminology. This can be a tedious process as we would need to dig into some tough technical details and manually write this mapping. Fortunately, we can ask **BioThings Studio** to inspect the data and suggest a mapping for it.

In order to do so, click on *Mapping* tab, then click on  **Inspect data**.

We can inspect the data for different purposes:

- **Mode**
 - **type**: inspection will report any types found in the collection, giving detailed information about the structure of documents coming from the parser. Note results aren't available from the webapp, only in MongoDB.
 - **stats**: same as type but gives numbers (count) for each structures and types found. Same as previous, results aren't available in the webapp yet.
 - **mapping**: inspect the date types and suggest an Elasticsearch mapping. Will report any error or types incompatible with ES.

Here we'll stick to mode **mapping** to generate that mapping. There are other options used to explore the data to inspect:

- **Limit**: limit the inspected documents.
- **Sample**: randomize the documents to inspect (1.0 = consider all documents, 0.0 = skip all documents, 0.5 = consider every other documents)

The last two options can be used to reduce the inspection time of huge data collection, or you're absolutely sure the same structure is returned for any documents output from the parser.

Inspect data: mvcgi

Plugin Mapping

Selecting more than one mode won't affect much the performance, running time will roughly be the same.

mapping

- ▶ **mapping**
Analyzes data so the inspection results can be converted into an ElasticSearch mapping (used during indexing step)
- ▶ **type**
Builds a map of all types involved in the data, providing a summary of its structure
- ▶ **stats**
Performs in-depth analysis about the data, including type map and basic statistics, showing how volumetry fits over data structure

Optional parameters

Limit...
Restrict inspection to this number of documents. If empty, all documents are inspected.

Sampling (eg. 0.5)...
Randomly pick documents to inspect. Value is a float between 0 and 1.0. If sampling is 1.0, all documents are picked, if 0.0, none of them. Combined with parameter "limit", it allows to randomly inspect a subset of the data.

Cancel OK

Since the collection is very small, inspection is fast. But... it seems like we have a problem

Mapping from inspection

Validate on localhub ▾ >> Commit

⚠ Found errors while generating the mapping:

- More than one type (key:'notes',types:[<class 'biothings.utils.common.splitstr'>, <class 'biothings.utils.common.nan'>])

Mapping can't be generated until those errors are fixed. Please fix the parser or the data and try again.

For debugging purposes, below is a pre-mapping structure, where errors can be spot.

```
{
  "_id": {
    "__type__": "str"
  },
  "annotations": {
    "__type__": "list",
    "annotation_id": {
      "__type__": "int"
    }
  }
}
```

More than one type was found for a field named **notes**. Indeed, if we scroll down on the *pre-mapping* structure, we can see the culprit:

```
},
"notes": {
    "__type__:splitstr": {},
    "__type__:nan": {}
},
"sentence": {
```

This result means documents sometimes have `notes` key equal to `NaN`, and sometimes equal to a string (a splittable string, meaning there are spaces in it). This is a problem for ElasticSearch because it wouldn't index the data properly. And furthermore, ElasticSearch doesn't allow `NaN` values anyway. So we need to fix the parser. The fixed version is available in branch `pharmgkb_v2` (go back to Plugin tab, enter that branch name and update the code). The fix consists in [removing `key/value`](#) from the records, whenever a value is equal to `NaN`.

```
rec = dict_sweep(rec,vals=[np.nan])
```

Once fixed, we need to re-upload the data, and inspect it again. This time, no error, our mapping is valid:

```
{
    "annotations": {
        "properties": {
            "annotation_id": {
                "type": "integer"
            },
            "pmid": {
                "type": "integer"
            },
            "phenotype_category": {
                "normalizer": "keyword_lowercase_normalizer",
                "type": "keyword"
            },
            "studyparameters": {
                "normalizer": "keyword_lowercase_normalizer",
                "type": "keyword"
            },
            "chromosome": {
                "normalizer": "keyword_lowercase_normalizer",
                "type": "keyword"
            },
            "variant": {
                "type": "text"
            }
        }
}
```

For each highlighted field, we can decide whether we want the field to be searchable or not, and whether the field should be searched by default when querying the API. We can also change the type for that field, or even switch to “advanced mode” and specify your own set of indexing rules. Let's click on “gene” field and make it searched by default. Let's also do the same for field “variant”.

Modify indexing rules

Field: gene
Path: annotations.gene

- Index this field
- Search this field by default

Change type text

```
{
  "type": "text",
  "copy_to": [
    "all"
  ]
}
```

- ▶ **Enable index** allows a field to be searchable. If indexing is disabled, values are still stored and returned in results, but they can't be directly queried. Indexing takes disk space and can also impact performances, only index fields which make sense to query.
- ▶ When **Search by default** is enabled, field can be searched without specifying the full path. Note: `_id` field is an exception, path is not required.
Ex:
 - When searching by default is disabled, searching `gene` field requires to specify the full path:
`/query?q=annotations.gene:value_to_search`
 - When searching by default is enabled, path can be omitted, `value_to_search` will be searched in all fields declared as searchable by default.
`/query?q=value_to_search`.
- ▶ ElasticSearch field data type can be changed if needed. See [Field datatypes](#) for more information. Note: only core datatypes are available in this list
- ▶ Field definition can also be manually specified in the text box, using JSON notation, for more advanced usage.

Cancel

OK

Indeed, by checking the “Search by default” checkbox, we will be able to search for instance gene symbol “ABL1” with `/query?q=ABL1` instead of `/query?q=annotations.gene:ABL1`. Same for “variant” field where we can specify a rsid.

After this modification, you should see at the top of the mapping, let's save our changes clicking on Save. Also, before moving forward, we want to make sure the mapping is valid, let's click on Validate on localhub. You should see this success message:

Mapping from inspection

Save Validate on localhub Commit

Mapping has successfully been validated.

Note: “Validate on localhub” means **Hub** will send the mapping to ElasticSearch by creating a temporary, empty index to make sure the mapping syntax and content are valid. It's immediately deleted after validation (whether successful or not). Also, “localhub” is the default name of an environment. Without further manual configuration, this is the only development environment available in the Studio, pointing to embedded ElasticSearch server.

Everything looks fine, the last step is to “commit” the mapping, meaning we're ok to use this mapping as the official, registered mapping that will actually be used by ElasticSearch. Indeed the left side of the page is about inspected mapping, we can re-launch the inspection as many times as we want, without impacting active/registered mapping (this is useful when the data structure changes). Click on Commit then “YES”, and you now should see the final, registered mapping on the right:

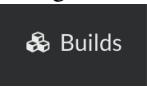
Registered mapping

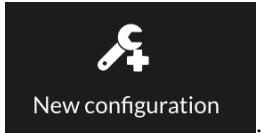
 Save  Validate on localhub ▾

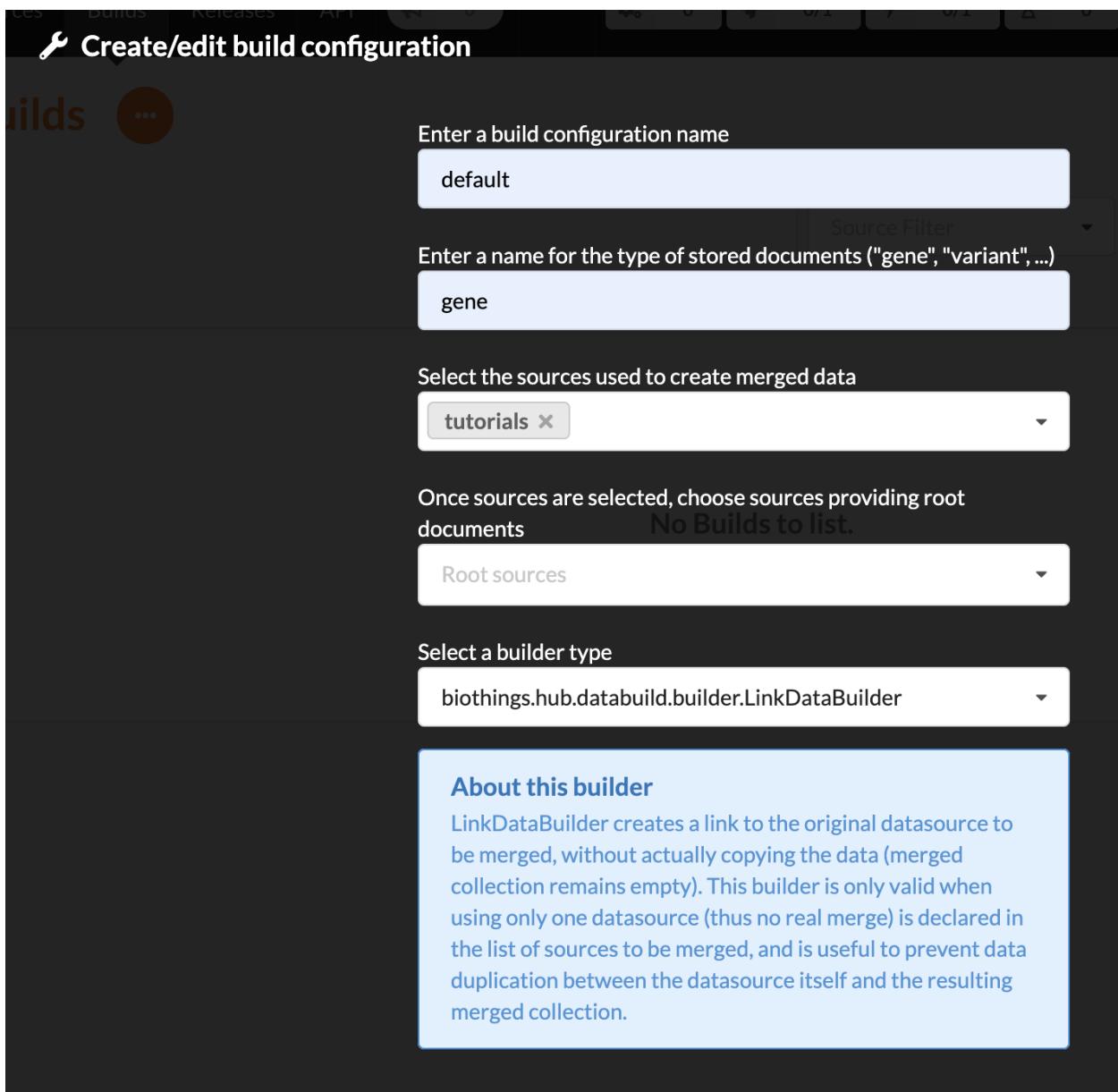
```
{  
    "annotations": {  
        "properties": {  
            "annotation_id": {  
                "type": "integer"  
            },  
            "pmid": {  
                "type": "integer"  
            },  
            "phenotype_category": {  
                "normalizer": "keyword_lowercase_normalizer",  
                "type": "keyword"  
            },  
            "studyparameters": {  
                "normalizer": "keyword_lowercase_normalizer",  
                "type": "keyword"  
            },  
            "chromosome": {  
                "normalizer": "keyword_lowercase_normalizer",  
                "type": "keyword"  
            },  
            "variant": {  
                "type": "text",  
                "copy_to": [  
                    "all"  
                ]  
            },  
            "gene": {  
                "type": "text",  
                "copy_to": [  
                    "all"  
                ]  
            }  
        }  
    }  
}
```

4.5. Build

Once we have integrated data and a valid ElasticSearch mapping, we can move forward by creating a build configuration.

A *build configuration* tells the **Hub** which datasources should be merged together, and how. Click on 

then  and finally, click on 

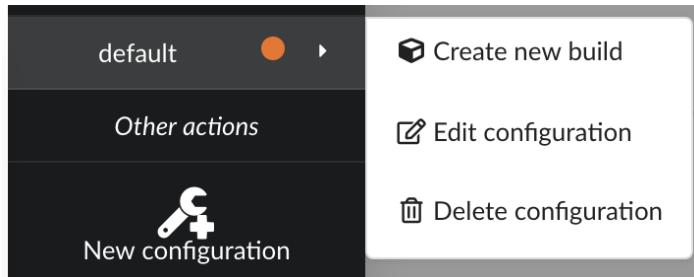


- enter a *name* for this configuration. We're going to have only one configuration created through this tutorial so it doesn't matter, let's make it “default”
- the *document type* represents the kind of documents stored in the merged collection. It gives its name to the annotate API endpoint (eg. /gene). This source is about gene annotations, so “gene” it is...
- open the dropdown list and select the *sources* you want to be part of the merge. We only have one, “pharmgkb”
- in *root sources*, we can declare which sources are allowed to create new documents in the merged collection. If a root source is declared, data from other sources will only be merged if documents previously exist with same IDs (documents coming from root sources). If not, data is discarded. Finally, if no root source is declared, any data sources can generate a new document in the merged data. In our case, we can leave it empty (no root sources specified, all sources can create documents in the merged collection).
- selecting a builder is optional, but for the sake of this tutorial, we'll choose `LinkDataBuilder`. This special builder will fetch documents directly from our datasources *pharmgkb* when indexing documents, instead of duplicating documents into another connection (called *target* or *merged* collection). We can do this (and save

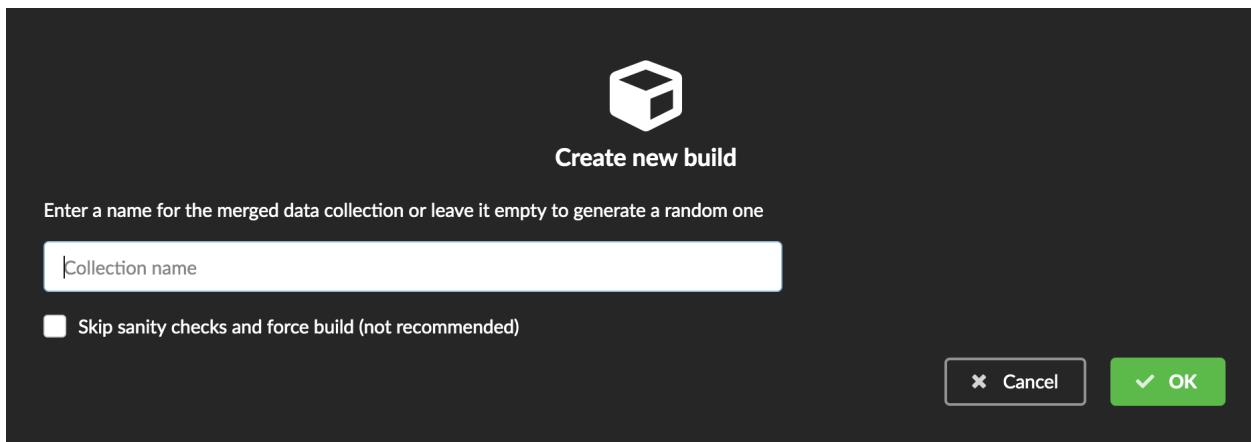
time and disk space) because we only have one datasource here.

- the other fields are for advanced usage and are out-of-topic for this tutorial

Click “OK” and open the menu again, you should see the new configuration available in the list.



Click on it and create a new build.



You can give a specific name for that build, or let the **Hub** generate one for you. Click “OK”, after a few seconds, you should see the new build displayed on the page.

default

default_20240109_9fbrkmja

1,018 documents 20240109

Build time: Built a few seconds ago

Datasource	Version
tutorials	2024-01-05

Open it by clicking on its name. You can explore the tabs for more information about it (sources involved, build times, etc...). The “Release” tab is the one we’re going to use next.

4.6. Data release

If not there yet, open the new created build and go the “Release” tab. This is the place where we can create new data releases. Click on **New release**.

Create new release

Select the type of release
full

Enter a name for the index (or leave it empty to have the same name as the build)
Index name

Select an indexer environment to create the index on
localhub (elasticsearch:9200)

Note: sources providing root documents, or *root sources*, are sources allowed to create a new document in a build (merged data). If a root source is declared, data from other sources will only be merged if documents previously exist with same IDs (documents coming from root sources). If not, data is discarded. Finally, if no root source is declared, any data sources can generate a new document in the merged data.

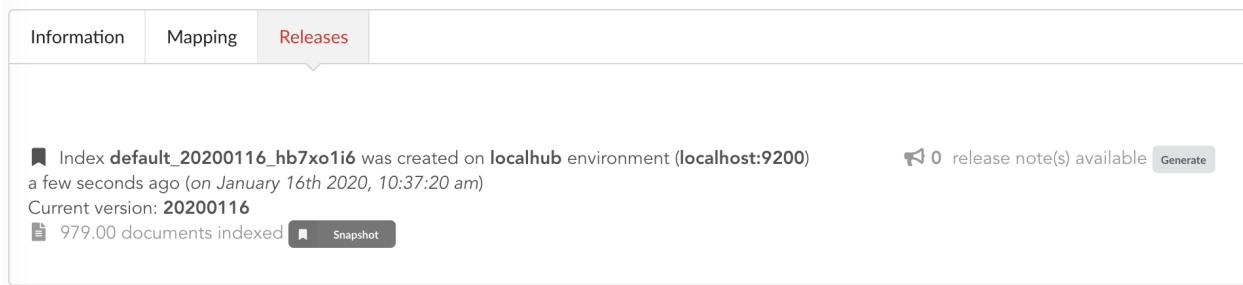
Cancel **OK**

Since we only have one build available, we can't generate an *incremental* release, so we'll have to select *full* this time.

Click “OK” to launch the process.

Note: Should there be a new build available (coming from the same configuration), and should there be data differences, we could generate an incremental release. In this case, **Hub** would compute a diff between previous and new builds and generate diff files (using [JSON diff](#) format). Incremental releases are usually smaller than full releases, usually take less time to deploy (applying diff data) unless diff content is too big (there's a threshold between using an incremental and a full release, depending on the hardware and the data, because applying a diff requires you to first fetch the document from ElasticSearch, patch it, and then save it back).

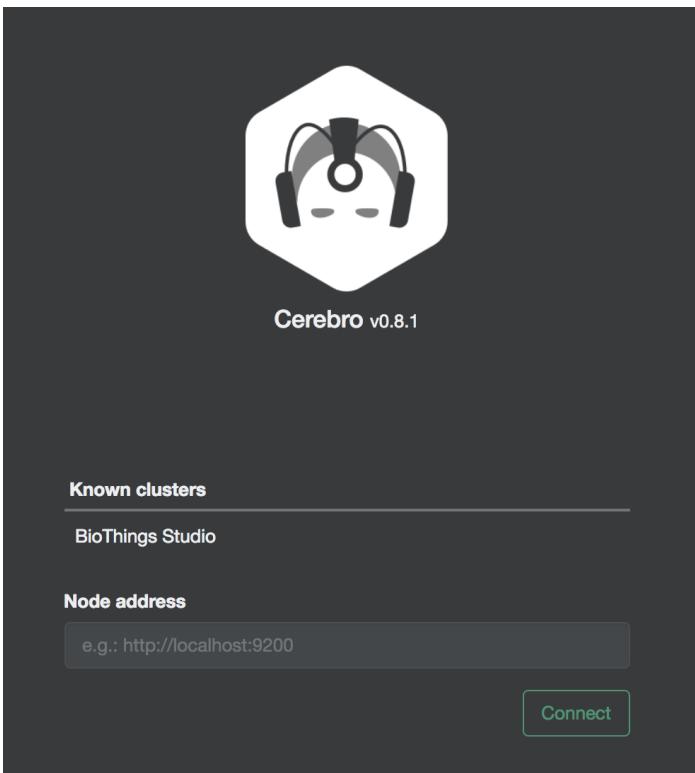
Hub will directly index the data on its locally installed ElasticSearch server (`localhub` environment). After few seconds, a new *full* release is created.



Index `default_20200116_hb7xo1i6` was created on `localhub` environment (`localhost:9200`)
a few seconds ago (on January 16th 2020, 10:37:20 am)
Current version: `20200116`
979.00 documents indexed [Snapshot](#)

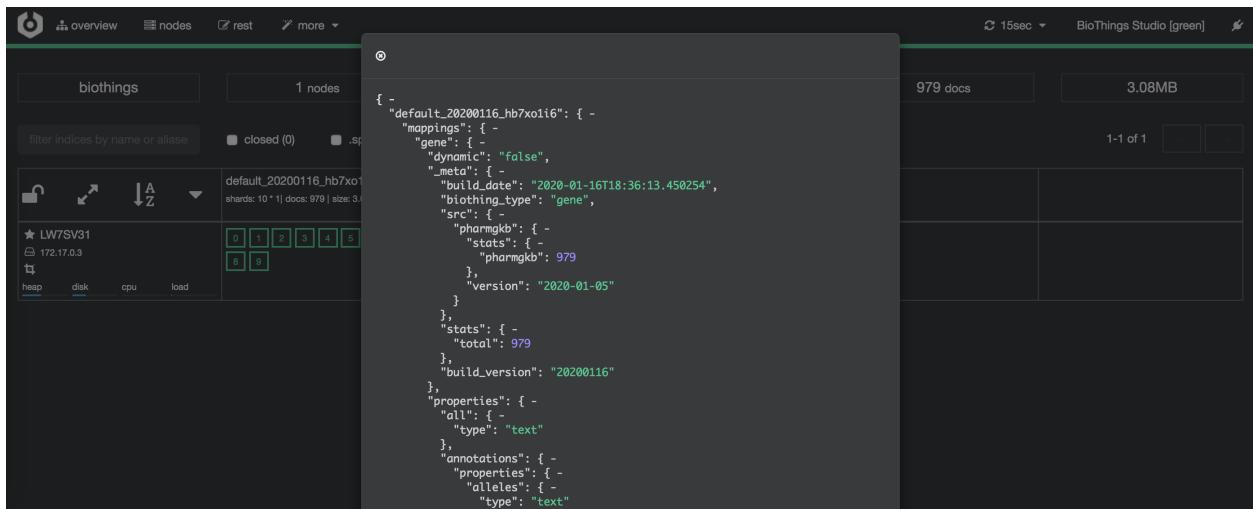
We can easily access ElasticSearch server using the application **Cerebro**, which comes pre-configured with the studio. Let's access it through <http://localhost:9000/#/connect> (assuming ports 9200 and 9000 have properly been mapped, as mentioned earlier). **Cerebro** provides an easy-to-manage ElasticSearch and check/query indices.

Click on the pre-configured server named **BioThings Studio**.



Clicking on an index gives access to different information, such as the mapping, which also contains metadata (sources

involved in the build, releases, counts, etc...)



4.7. API creation

At this stage, a new index containing our data has been created on ElasticSearch, it is now time for final step. Click on



We'll name it *pharmgkb* and have it running on port 8000.

Note: Spaces are not allowed in API names

>Create new API mgkb v1

Enter a name for the API

Enter a description for the API (optional)

Select a backend the API will connect to to serve the data

Specify a port

Once form is validated a new API is listed.

pharmgkb	
ElasticSearch host	localhost:9200
Index	default_20200116_hb7xo1i6
Document type	gene
API port	8000

To turn on this API instance, just click on  , you should then see a  label on the top right corner, meaning the API can be accessed:

pharmgkb

running

- Metadata: <http://6d5a7a0aa54a:8000/metadata>
- Query: http://6d5a7a0aa54a:8000/query?q=*

Note: When running, queries such `/metadata` and `/query?q=*` are provided as examples. They contain a hostname set by Docker though (it's the Docker instance's hostname), which probably means nothing outside of Docker's context. In order to use the API you may need to replace this hostname by the one actually used to access the Docker instance.

4.8. Tests

Assuming API is accessible through <http://localhost:8000>, we can easily query it with `curl` for instance. The endpoint `/metadata` gives information about the datasources and build date:

```
$ curl localhost:8000/metadata
{
  "biothing_type": "gene",
  "build_date": "2020-01-16T18:36:13.450254",
  "build_version": "20200116",
  "src": {
    "pharmgkb": {
      "stats": {
        "pharmgkb": 979
      },
      "version": "2020-01-05"
    }
  },
  "stats": {
    "total": 979
  }
}
```

Let's query the data using a gene name (results truncated):

```
$ curl localhost:8000/query?q=ABL1
{
  "max_score": 7.544187,
  "took": 70,
  "total": 1,
  "hits": [
    {
      "_id": "PA24413",
      "_score": 7.544187,
      "annotations": [
        {
          "alleles": "T",
          "annotation_id": 1447814556,
          "chemical": "homoharringtonine (PA166114929)"
        }
      ]
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```

    "chromosome": "chr9",
    "gene": "ABL1 (PA24413)",
    "notes": "Patient received received omacetaxine, treatment had been stopped after two cycles because of clinical intolerance, but a major molecular response and total disappearance of the T315I clone was obtained. Treatment with dasatinib was then started and after 34-month follow-up the patient is still in major molecular response.",
    "phenotype_category": "efficacy",
    "pmid": 25950190,
    "sentence": "Allele T is associated with response to homoharringtonine in people with Leukemia, Myelogenous, Chronic, BCR-ABL Positive as compared to allele C.",
    "significance": "no",
    "studyparameters": "1447814558",
    "variant": "rs121913459"
  },
  {
    "alleles": "T",
    "annotation_id": 1447814549,
    "chemical": "nilotinib (PA165958345)",
    "chromosome": "chr9",
    "gene": "ABL1 (PA24413)",
    "phenotype_category": "efficacy",
    "pmid": 25950190,
    "sentence": "Allele T is associated with resistance to nilotinib in people with Leukemia, Myelogenous, Chronic, BCR-ABL Positive as compared to allele C.",
    "significance": "no",
    "studyparameters": "1447814555",
    "variant": "rs121913459"
  }
]
}
]
}
}

```

Note: We don't have to specify annotations.gene, the field in which the value "ABL1" should be searched, because we explicitly asked ElasticSearch to search that field by default (see *fieldbydefault*)

Finally, we can fetch a variant by its PharmGKB ID:

```
$ curl "localhost:8000/gene/PA134964409"
{
  "_id": "PA134964409",
  "_version": 1,
  "annotations": [
    {
      "alleles": "AG + GG",
      "annotation_id": 1448631680,
      "chemical": "etanercept (PA449515)",
      "chromosome": "chr1",
      "gene": "GBP6 (PA134964409)",
      "phenotype_category": "efficacy",
    }
  ]
}
```

(continues on next page)

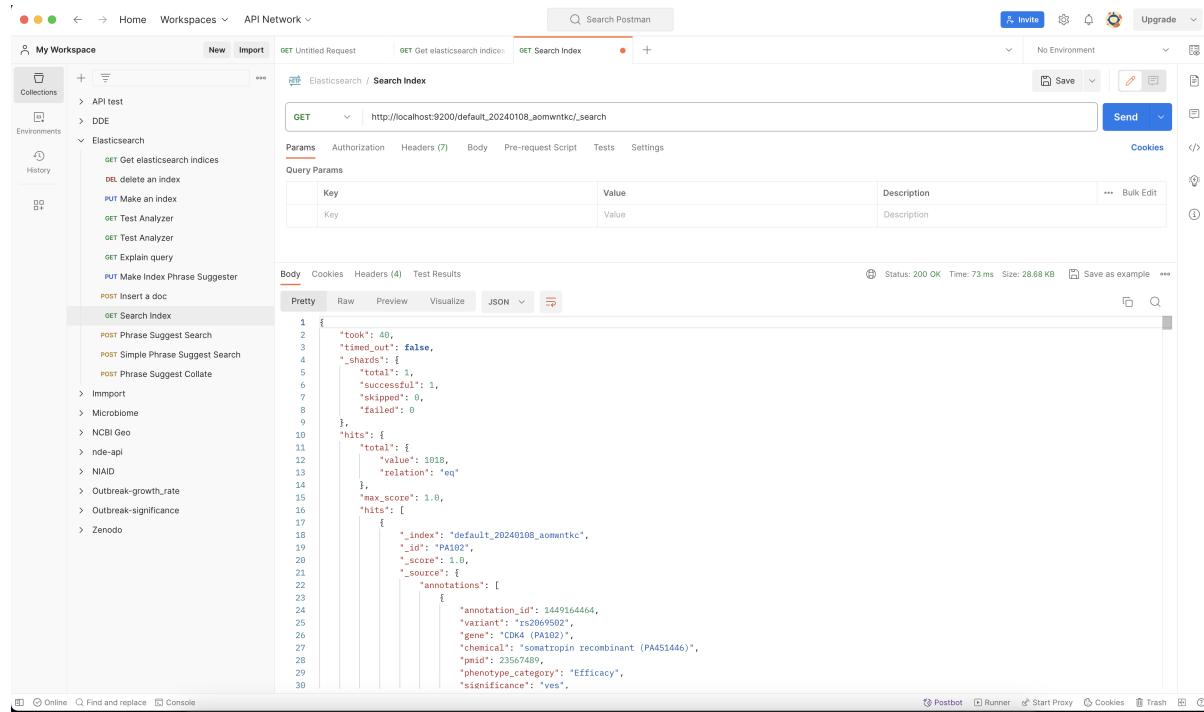
(continued from previous page)

```

    "pmid": 28470127,
    "sentence": "Genotypes AG + GG is associated with increased response to etanercept in people with Psoriasis as compared to genotype AA.",
    "significance": "yes",
    "studyparameters": "1448631688",
    "variant": "rs928655"
}
]
}

```

Most of the time, the API testing is not necessary. Unless you are specifically testing out a custom api feature. You can learn more about customizing api web components in the [Biothings Web](#). In our use case, you can just query the Elasticsearch instance directly. In this example, we will be using [postman](#), to query the Elasticsearch Index. Once you have postman installed you can make this query http://localhost:9200/MY_BUILD_NAME/_search. Check a few of the hits to make sure if your parser has correctly formatted the data. You can also make more detailed search queries in the elasticsearch index if needed.



The screenshot shows the Postman application interface. On the left, there's a sidebar with 'My Workspace' containing various collections and environments. The main area shows a 'GET Untitled Request' for 'Elasticsearch / Search Index'. The URL is set to http://localhost:9200/default_20240108_aomntkc/_search. The 'Params' tab is selected, showing a single parameter 'Query Params' with a key 'Key' and a value 'Value'. Below this, the 'Body' tab is selected, showing a JSON response. The response is a multi-line JSON object:

```

1 {
2   "took": 40,
3   "timed_out": false,
4   "_shards": {
5     "total": 1,
6     "successful": 1,
7     "skipped": 0,
8     "failed": 0
9   },
10  "hits": [
11    {
12      "total": {
13        "value": 1018,
14        "relation": "eq"
15      },
16      "max_score": 1.0,
17      "hits": [
18        {
19          "_index": "default_20240108_aomntkc",
20          "_id": "PA482",
21          "_score": 1.0,
22          "_source": {
23            "annotations": [
24              {
25                "annotation_id": 1449164646,
26                "variant": "rs2089502",
27                "gene": "CDKA (PA102)",
28                "chemical": "somatotropin recombinant (PA451446)",
29                "lead": 23501489,
30                "phenotype_category": "Efficacy",
31                "significance": "yes"
32              }
33            ]
34          }
35        }
36      ]
37    }
38  ]
39 }

```

4.9. Conclusions

We've been able to easily convert a remote flat file to a fully operational BioThings API:

- by defining a data plugin, we told the **BioThings Hub** where the remote data was and what the parser function was
- **BioThings Hub** then generated a *dumper* to download data locally on the server
- It also generated an *uploader* to run the parser and store resulting JSON documents
- We defined a build configuration to include the newly integrated datasource and then trigger a new build
- Data was indexed internally on local ElasticSearch by creating a full release

- Then we created a BioThings API instance pointing to that new index

The next step is to enrich that existing API with more datasources.

4.10. Multiple sources data plugin

In the previous part, we generated an API from a single flat file. This API serves data about gene annotations, but we need more: as mentioned earlier in **Input data**, we also downloaded drug labels and publications information. Integrating those unused files, we'll be able to enrich our API even more, that's the goal of this part.

In our case, we have one *dumper* responsible for downloading three different files, and we now need three different *uploaders* in order to process these files. With above data plugin (4.3), only one file is parsed. In order to proceed further, we need to specify multiple *uploaders* on the *manifest.json* file, the full example can be found in branch `pharmgkb_v5` available at https://github.com/biothings/tutorials/tree/pharmgkb_v5.

Note: You can learn more about data plugin in the section **B.4. Data plugin architecture and specifications**

5. Regular data source

5.1. Data plugin limitations

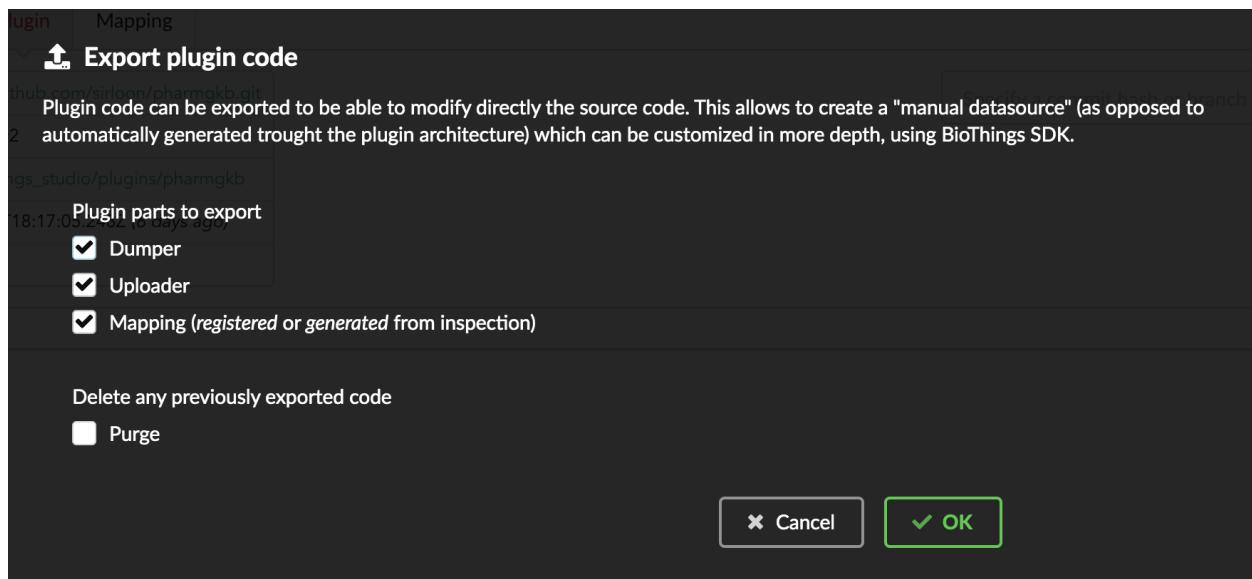
The **data plugin architecture** provided by **BioThings Studio** allows to quickly integrate a new datasource, describing where the data is located, and how the data should be parsed. It provides a simple and generic way to do so, but also comes with some limitations. Indeed, in many advanced use cases, you need to use a custom data builder instead of `LinkDataBuilder` (that you used at the point 4.5). But you can not define a custom builder on the data plugin.

Luckily, **BioThings Studio** provides an easy to export python code that has been generated during data plugin registration. Indeed, code is generated from the manifest file, compiled and injected into **Hub**'s memory. Exporting the code consists in writing down that dynamically generated code. After successful export, we have a new folder stays in `hub/dataload/sources` and contains exported python files - that is a **Regular data source** (or a regular dumper/uploader based data sources) Following below steps, you will learn about how to deal with a regular data source.

5.2. Code export

Note: You MUST to update above `pharmgkb` data plugin to the version `pharmgkb_v2`.

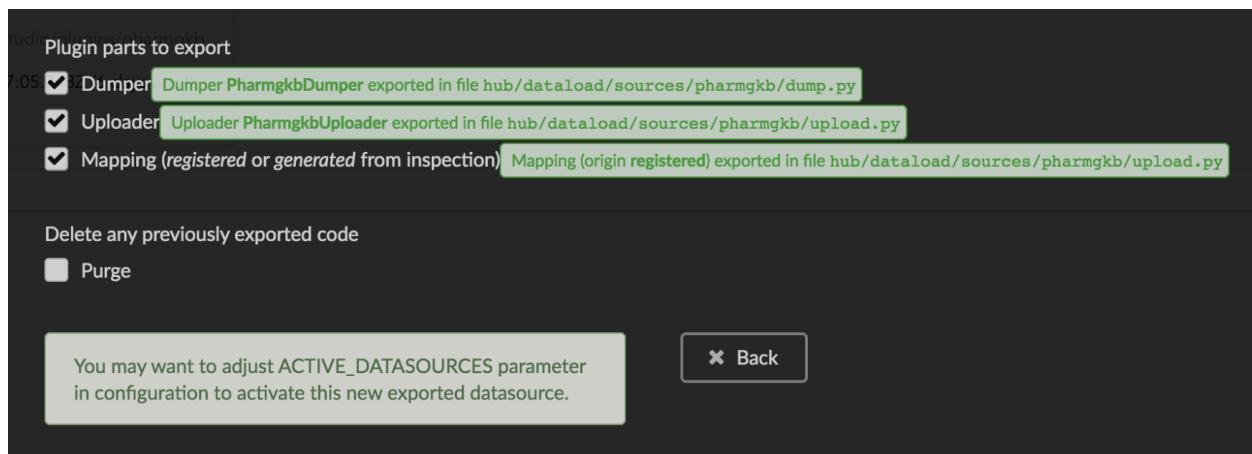
Let's go back to our datasource, **Plugin** tab. Clicking on  **Export code** brings the following form:



We have different options regarding the parts we can export:

- **Dumper**: exports code responsible for downloading the data, according to URLs defined in the manifest.
- **Uploader**: exports code responsible for data integration, using our parser code.
- **Mapping**: any mapping generated from inspection, and registered (commit) can also be exported. It'll be part of the uploader.

We'll export all these parts, let's validate the form. Export results are displayed (though quickly as Hub will detect changes in the code and will want to restart)



We can see the relative paths where code was exported. A message about ACTIVE_DATASOURCES is also displayed explaining how to activate our newly exported datasource. That said, **BioThings Studio** by default monitors specific locations for code changes, including where code is exported, so we don't need to manually activate it. That's also the reason why **Hub** has been restarted.

Once reconnected, if we go back on Sources, we'll see an error!

pharmgkb

2020-01-05 Updated 7 days ago

979 documents

⚠ Error loading manifest: Can't register <class 'biothings.hub.dataplugin.assistant.AssistedDumper_pharmgkb'> for source 'pharmgkb', dumper already registered: [<class 'hub.dataload.sources.pharmgkb.dumper.PharmgkbDumper'>]



Our original data plugin can't be registered (ie. activated) because another datasource with the same name is already registered. That's our new exported datasource! When the **Hub** starts, it first loads datasources which have been manually coded (or exported), and then data plugins. Both our plugin and exported code is active, but the **Hub** can't know which one to use. Let's delete the plugin, by clicking on , and confirm the deletion.

Hub will restart again (reload page if not) and this time, our datasource is active. If we click on **tutorials**, we'll see the same details as before except the **Plugin** tab which disappeared. So far, our exported code runs, and we're in the exact same state as before, the **Hub** even kept our previously dumped/uploaded data.

Let's explore the source code that has been generated through out this process. Let's enter our docker container, and become user **biothings** (from which everything runs):

```
$ docker exec -ti biothings /bin/bash  
$ sudo su - biothings
```

Paths provided as export results (`hub/dataload/sources/*`) are relative to the started folder named `biothings_studio`. Let's move there:

```
$ cd ~/biothings_studio/hub/dataload/sources/  
$ ls -la  
total 0  
-rw-rw-r-- 1 biothings biothings 0 Jan 15 23:41 __init__.py  
drwxrwxr-x 2 biothings biothings 45 Jan 15 23:41 __pycache__  
drwxr-xr-x 1 biothings biothings 75 Jan 15 23:41 ..  
drwxr-xr-x 1 biothings biothings 76 Jan 22 19:32 .  
drwxrwxr-x 3 biothings biothings 154 Jan 22 19:32 tutorials
```

A `tutorials` folder can be found and contains the exported code:

```
$ cd tutorials  
$ ls  
total 32  
drwxrwxr-x 3 biothings biothings 154 Jan 22 19:32 .  
drwxr-xr-x 1 biothings biothings 76 Jan 22 19:32 ..  
-rw-rw-r-- 1 biothings biothings 1357 Jan 22 19:32 LICENSE
```

(continues on next page)

(continued from previous page)

```
-rw-rw-r-- 1 biothings biothings 225 Jan 22 19:32 README
-rw-rw-r-- 1 biothings biothings 70 Jan 22 19:32 __init__.py
drwxrwxr-x 2 biothings biothings 142 Jan 22 19:45 __pycache__
-rw-rw-r-- 1 biothings biothings 868 Jan 22 19:32 dump.py
-rw-rw-r-- 1 biothings biothings 1190 Jan 22 19:32 parser.py
-rw-rw-r-- 1 biothings biothings 2334 Jan 22 19:32 upload.py
```

Some files were copied from data plugin repository (LICENCE, README and parser.py), the others are the exported ones: dump.py for the dumper, upload.py for the uploader and the mappings, and __init__.py so the **Hub** can find these components upon start. We'll go in further details later, specially when we'll add more uploaders.

For convenience, the exported code can be found in branch pharmgkb_v3 available at https://github.com/biothings/tutorials/tree/pharmgkb_v3. One easy way to follow this tutorial without having to type too much is to replace folder tutorials with a clone from Git repository. The checked out code is exactly the same as code after export.

```
$ cd ~/biothings_studio/hub/dataload/sources/
$ rm -fr tutorials
$ git clone https://github.com/biothings/tutorials.git
$ cd tutorials
$ git checkout pharmgkb_v3
```

5.3. More uploaders

Now that we have exported the code, we can start the modifications. The final code can be found on branch https://github.com/biothings/tutorials/tree/pharmgkb_v4.

Note: We can directly point to that branch using `git checkout pharmgkb_v4` within the datasource folder previously explored.

First we'll write two more parsers, one for each addition files. Within `parser.py`:

- at the beginning, `load_annotations` is the first parser we wrote, no changes required
- `load_druglabels` function is responsible for parsing file named `drugLabels.byGene.tsv`
- `load_occurrences` function is parsing file `occurrences.tsv`

Writing parsers is not the main purpose of this tutorial, which focuses more on how to use **BioThings Studio**, so we won't go into further details.

Next is about defining new uploaders. In `upload.py`, we currently have one uploader definition, which looks like this:

```
class PharmgkbUploader(biothings.hub.dataload.uploader.BaseSourceUploader):

    name = "pharmgkb"
    __metadata__ = {"src_meta": {}}
    idconverter = None
    ...
```

The important pieces of information here is `name`, which gives the name of the uploader we define. Currently uploader is named `pharmgkb`. That's how this name is displayed in the “Upload” tab of the datasource. We know we need three uploaders in the end so we need to adjust names. In order to do so, we'll define a main source, `pharmgkb`, then three different other “sub” sources: `annotations`, `druglabels` and `occurrences`. For clarity, we'll put these uploaders in three different files. As a result, we now have:

- file upload_annotations.py, originally coming from the code export. Class definition is:

```
class AnnotationsUploader(biothings.hub.dataloader.uploader.BaseSourceUploader):  
  
    main_source = "pharmgkb"  
    name = "annotations"
```

Note: We renamed the class itself, pharmgkb is now set as field main_source. This name matches the dumper name as well, which is how the **Hub** knows how dumpers and uploaders relates to each others. Finally, the sub-source named annotation is set as field name.

- doing the same for upload_druglabels.py:

```
from .parser import load_druglabels  
  
class DrugLabelsUploader(biothings.hub.dataloader.uploader.BaseSourceUploader):  
  
    main_source = "pharmgkb"  
    name = "druglabels"  
  
    def load_data(self, data_folder):  
        self.logger.info("Load data from directory: '%s'" % data_folder)  
        return load_druglabels(data_folder)  
  
    @classmethod  
    def get_mapping(klass):  
        return {}
```

Note: In addition to adjusting the names, we need to import our dedicated parser, load_druglabels. Following what the **Hub** did during code export, we “connect” that parser to this uploader in method load_data. Finally, each uploader needs to implement class method get_mapping, currently an empty dictionary, that is, no mapping at all. We’ll fix this soon.

- finally, upload_occurrences.py will deal with occurrences uploader. Code is similar as previous one.

```
from .parser import load_occurrences  
  
class OccurrencesUploader(biothings.hub.dataloader.uploader.BaseSourceUploader):  
  
    main_source = "pharmgkb"  
    name = "occurrences"  
  
    def load_data(self, data_folder):  
        self.logger.info("Load data from directory: '%s'" % data_folder)  
        return load_occurrences(data_folder)  
  
    @classmethod  
    def get_mapping(klass):  
        return {}
```

The last step to activate those components is to expose them through the __init__.py:

```
from .dump import PharmgkbDumper
from .upload_annotations import AnnotationsUploader
from .upload_druglabels import DrugLabelsUploader
from .upload_occurrences import OccurrencesUploader
```

Upon restart, the “Upload” tab now looks like this:

The screenshot shows the 'Uploader' tab selected in the top navigation bar. Below it, a sub-tab for 'pharmgkb' is active, indicated by a green border. The main content area displays a table with the following data:

Release	2020-01-05
Data folder	/data/biothings_studio/datasources/pharmgkb/2020-01-05
Status	success
Last upload	2020-01-23T23:07:03.813Z (a few seconds ago)
Duration	0.73s
Documents uploaded	979
Uploader	No uploader found, datasource may be broken

We still have an uploader named pharmgkb, but that component has been deleted! **Hub** indeed kept information within its internal database, but also detected that the actual uploader class doesn't exist anymore (see message `No uploader found, datasource may be broken`). In that specific case, an option to delete that internal information is provided, let's click on the closing button on that tab to remove that information.

If we look at the other uploader tabs, we don't see much information, that's because they haven't been launched yet. For each one of them, let's click on “Upload” button.

Note: Another way to trigger all uploaders at once is to click on Sources to list all datasources, then click on for that datasource in particular.

After a while, all uploaders have run, data is populated, as shown in the different tabs.

5.4. More data inspection

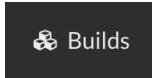
Data is ready, it's now time to inspect the data for the new uploaders. Indeed, if we check the "Mapping" tab, we still have the old mapping from the original pharmgkb uploader (we can remove that "dead" mapping by clicking on the closing button of the tab), but nothing for uploaders druglabels and occurrences.

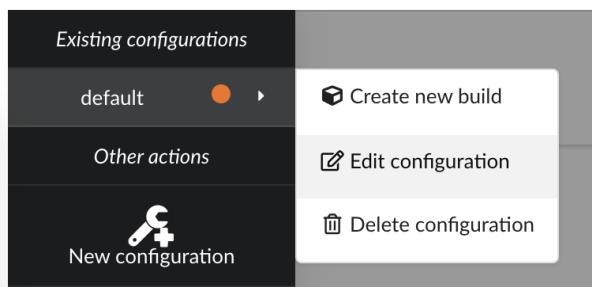
Looking back at the uploaders' code, `get_mapping` class method was defined such as it returns an empty mapping. That's the reason why we don't have anything shown here, let's fix that by click on  . After few seconds, mappings are generated, we can review them, and click on  to validate and register those mappings, for each tab.

5.5. Modifying build configuration

All data is now ready, as well as mappings, it's time to move forward and build the merged data. We now have three different source for documents, and we need to merge them together. **Hub** will do so according to field `_id`: if two documents from different sources share the same `_id`, they are merged together (think about dictionary merge).

In order to proceed further, we need to update our build configuration, as there's currently only datasource involved in

the merge. Clicking on  , then  we can edit existing configuration.



There several parameters we need to adjust:

- first, since original pharmgkb uploader doesn't anymore, that datasource isn't listed anymore
- in the other hand, we now have our three new datasources, and we need to select all of them
- our main data is coming from annotations, and we want to enrich this data with druglabels and litterature occurrences. But only if data first exists in annotations. Behing this requirement is the notion of *root documents*. When selection annotations as a source for root documents, we tell the **Hub** to first merge that data, then merge the other sources **only** if a document from annotations with the same `_id` exists. If not, documents are silently ignored.
- finally, we were previously using a `LinkDataBuilder` because we only had one datasource (data wasn't copied, but referred, or linked to the original datasource collection). We now have three datasources involved in the merge so we can't use that builder anymore and need to switch to the default `DataBuilder` one. If not, **Hub** will complain and deactivate the build configuration until it's fixed. Since we already had a previous build, we want to specify an incremental build `diff`.

The next configuration is summarized in the following picture:

Create/edit build configuration

Enter a build configuration name
default

Enter a name for the type of stored documents ("gene", "variant", ...)
gene

Select the sources used to create merged data
annotations x druglabels x occurrences x

Once sources are selected, choose sources providing root documents
annotations x

Select a builder type
biothings.hub.databuild.builder.DataBuilder (default)

About this builder
Generic data builder.

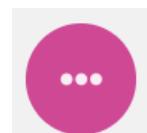
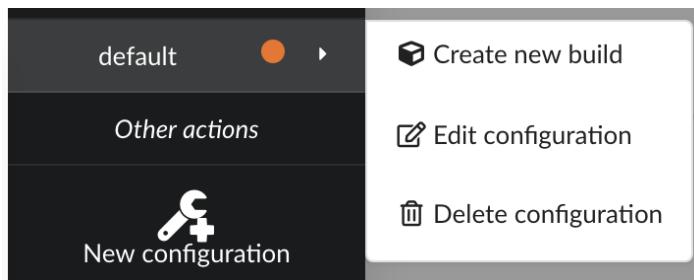
Optional parameters can be added to the configuration (usefull to customize builder behavior). Enter a JSON object structure

```
{
  "autobuild": {
    "type": "diff"
  }
}
```

Upon validation, build configuration is ready to be used.

5.6. Incremental release

Configuration reflects our changes and is up-to-date, let's create a new build. Click on then “Create a new build”



if not already open,

After few seconds, we have a new build listed. Clicking on “Logs” will show how the **Hub** created it. We can see it first merged annotations in the “merge-root” step (for *root documents*), then druglabels and occurrences sources. The remaining steps, (*diff, release note*) were automatically triggered by the **Hub**. Let’s explore these further.

default

default_20200128_cmrdvjml

979 documents 20200128
Build time: 03s Built a few seconds ago

Sources Stats **Logs**

Build starts Jan 28th 2020, 10:19:26 am

✓ merge-root 0.43s
annotations

✓ merge-others 1.1s
druglabels, occurrences

✓ finalizing 0.0s

✓ post-merge 0.01s

✓ metadata 0.05s

✓ diff-mapping 0.03s

✓ diff-content 2.27s

✓ diff-reduce 0.02s

✓ diff-post 0.02s

✓ release_note 0.18s

If we open the build and click on “Releases” tab, we have a *diff* release, or *incremental* release, as mentioned in the “Logs”. Because a previous release existed for that build configuration (the one we did in part one), the **Hub** tries to compute an release comparing the two together, identifying new, deleted and updated documents. The result is a *diff* release, based on **json diff** format.

The screenshot shows the BioThings Studio interface with the 'Releases' tab selected. The main content area displays a diff report. Key information includes:

- A diff icon followed by the text: "Diff with **default_20200128_xitqlcz** has been computed."
- The old version is listed as **20200128**, and the current version is **20200128**.
- The report was created a minute ago (on January 28th 2020, 10:19:32 am).
- Statistics: 1 diff file(s) created (2 MB), with buttons for **Apply** and **Publish**.
- A note: "971 updated, 0 added, 0 deleted. Mapping has changed."

In our case, one diff file has been generated, its size is 2 MiB, and contains information to update 971 documents. This is expected since we enriched our existing data. **Hub** also mention the mapping has been changed, and these will be reported to the index as we “apply” that diff release.

Note: Because we added new datasources, without modifying existing mapping in the first annotations source, the differences between previous and new mappings correspond to “add” json-diff operations. This means we strictly only add **more** information to the existing mapping. If we’d removed, and modify existing mapping fields, the **Hub** would have reported an error and aborted the generation of that diff release, to prevent an error during the update of the ElasticSearch index, or to avoid data inconsistency.

The other document that has been automatically generated is a *release note*.

The screenshot shows the 'Release Note' section. It indicates 1 release note available and compares it with the previous version, **default_20200128_xitqlcz**. A 'View' link is provided to see the details.

Compared with	Notes
default_20200128_xitqlcz	View

If we click on “View”, we can see the results: the **Hub** compared previous data versions and counts, deleted and added datasources and field, etc... In other words, a “change log” summarizing what happened between previous and new releases. These release notes are informative, but also can be published when deploying data releases (see part 3).

Release note

Releases

Build version: '20200128'
 ======
 Previous build version: '20200128'
 Generated on: 2020-01-28 at 18:19:40
 Current version: 20200128

Updated datasource	prev. release	new release	prev. # of docs	new # of docs	Compared with	Notes
pharmgkb.annotations	-	2020-01-05	-	979	default_20200128_122	View
pharmgkb.druglabels	-	2020-01-05	-	5503		
pharmgkb.occurrences	-	2020-01-05	-			
pharmgkb	2020-01-05	-	979	-		

New datasource(s): pharmgkb
 Deleted datasource(s): pharmgkb

New field(s): drug_labels, occurrences

Overall, 979 documents in this release
 0 document(s) added, 0 document(s) deleted, 971 document(s) updated

Let's apply that diff release, by clicking on [Apply](#)

We can select which index to update, from a dropdown list. We only have index, the one we created earlier in part 1. That said, **Hub** will do its best to filter out any incompatible indices, such those not coming from the same build configuration, or not having the same document type.

Apply increment update (diff)

Select a backend to apply the diff to

localhub (localhost:9200 | default_20200128_xitqlizc)

Once confirmed, the synchronization process begins, diff files are applied to the index, just as if we were “patching” data. We can track the command execution from the command list, and also from the notification popups when it’s done.



Our index, currently served by our API defined in the part 1, has been updated, using a diff, or incremental, release. It's time to have a look at the data.

5.7. Testing final API

Because we directly apply a diff, or patch our data, on ElasticSearch index, we don't need to re-create an API. Querying the API should just transparently reflect that "live" update. Time to try our new enriched API. We'll use curl again, here few query examples:

```
$ curl localhost:8000/metadata
{
  "biotesting_type": "gene",
  "build_date": "2020-01-24T00:14:28.112289",
  "build_version": "20200124",
  "src": {
    "pharmgkb": {
      "stats": {
        "annotations": 979,
        "druglabels": 122,
        "occurrences": 5503
      },
      "version": "2020-01-05"
    }
  },
  "stats": {
    "total": 979
  }
}
```

Metadata has changed, as expected. If we compare this result with previous one, we now have three different sources:

annotations, druglabels and occurrences, reflecting our new uploaders. For each of them, we have the total number of documents involved during the merge. Interestingly, the total number of documents is in our case 979 but, for instance, occurrences shows 5503 documents. Remember, we set annotations as a *root documents* source, meaning documents from others are merged only if they matched (based on `_id` field) an existing documents in this *root document* source. In other words, with this specific build configuration, we can't have more documents in the final API than the number of documents in *root document* sources.

Let's query by symbol name, just as before:

```
$ curl localhost:8000/query?q=ABL1
{
  "max_score": 7.544187,
  "took": 2,
  "total": 1,
  "hits": [
    {
      "_id": "PA24413",
      "_score": 7.544187,
      "annotations": [
        {
          "alleles": "T",
          "annotation_id": 1447814556,
          "chemical": "homoharringtonine (PA166114929)",
          "chromosome": "chr9",
          "gene": "ABL1 (PA24413)",
          "notes": "Patient received received omacetaxine, treatment had been stopped\u2192 after two cycles because of clinical intolerance, but a major molecular response and\u2192 total disappearance of the T315I clone was obtained. Treatment with dasatinib was then\u2192 started and after 34-month follow-up the patient is still in major molecular response.\u2192",
          "phenotype_category": "efficacy",
          "pmid": 25950190,
          "sentence": "Allele T is associated with response to homoharringtonine in people\u2192 with Leukemia, Myelogenous, Chronic, BCR-ABL Positive as compared to allele C.",
          "significance": "no",
          "studyparameters": "1447814558",
          "variant": "rs121913459"
        },
        ...
      ],
      "drug_labels": [
        {
          "id": "PA166117941",
          "name": "Annotation of EMA Label for bosutinib and ABL1,BCR"
        },
        {
          "id": "PA166104914",
          "name": "Annotation of EMA Label for dasatinib and ABL1,BCR"
        },
        {
          "id": "PA166104926",
          "name": "Annotation of EMA Label for imatinib and ABL1,BCR,FIP1L1,KIT,PDGFRA,\u2192 PDGFRB"
        }
      ]
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```

...
]
"occurrences": [
{
  "object_id": "PA24413",
  "object_name": "ABL1",
  "object_type": "Gene",
  "source_id": "PMID:18385728",
  "source_name": "The cancer biomarker problem.",
  "source_type": "Literature"
},
{
  "object_id": "PA24413",
  "object_name": "ABL1",
  "object_type": "Gene",
  "source_id": "PMC443563",
  "source_name": "Two different point mutations in ABL gene ATP-binding domain ↵
  ↵ conferring Primary Imatinib resistance in a Chronic Myeloid Leukemia (CML) patient: A ↵
  ↵ case report.",
  "source_type": "Literature"
},
...
]
}

```

We now have much information associated (much have been removed for clarity), including keys `drug_labels` and `occurrences` coming from the two new uploaders.

5.8. Conclusions

Moving from a single datasource based API, previously defined as a data plugin, we've been able to export this data plugin code. This code was used as a base to extend our API, specifically:

- we implemented two more parsers, and their counter-part uploaders.
- we updated the build configuration to add these new datasources
- we created a new index (*full release*) and created a new API serving this new data.

So far APIs are running from within **BioThings Studio**, and data still isn't exposed to the public. The next step to publish this data and make the API available for everyone.

Note: **BioThings Studio** is a backend service, aimed to be used internally to prepare, test and release APIs. It is not intended to be facing public internet, in other words, it's not recommended to expose any ports, including API ports, to public-facing internet.

6. Data Plugins

In our last section, we learned how to create a **Regular Data Source**. Now that we have learned how to dynamically generate code for our plugin, we can discuss the different classes that we can use for our dumpers. Using these classes, we can have an easier time creating dumpers to fit our needs when downloading different types of data on the interweb. Here is a short summary of some of the important classes that you may typically use.

- APIDumper: This will run API calls in a clean process and write its results in one or more NDJSON documents.
- DockerContainerDumper: Starts a docker container (typically runs on a different server) to prepare the data file on the remote container, and then download this file to the local data source folder.
- LastModifiedFTPDumper: SRC_URLS containing a list of URLs pointing to files to download, uses FTP's MDTM command to check whether files should be downloaded.
- LastModifiedHTTPDumper: Given a list of URLs, check Last-Modified header to see whether the file should be downloaded.

All of data plugin types can all be reviewed in our biothings.api on github <https://github.com/biothings/biothings.api/blob/master/biothings/hub/dataloader/dumper.py>.

6.1. DockerContainer Plugin

Note: For this section, you will need to know how to use our Biothings CLI, as our docker compose is not built to handle this type of plugin. Please refer back to our Biothings CLI tutorial before starting this section.

The DockerContainer plugin allows us to remotely start and control a docker container from another server, using our Biothings Hub. Using another server to run the bulk process of our dumper can have many different use cases. For example, if we are using a public api to create a source, we may need to call their api multiple times for testing. This may inadvertently cause an accidental ban. Even if we follow their rate limiting, the api may flag our IP address as a bot. By using another server, we can minimize this issue by setting a different IP address separate from the Scripps Network, so if there is a ban it only be contained to our one server instead of our whole network.

The steps to the DockerContainerDumper Plugin look like this:

- It boots up a container from provided parameters: image, tag, container_name.
- Runs the dump_command inside this container. This command MUST block the dumper until the data file is completely prepared. This will guarantee that the remote file is ready for downloading.
- Optional: runs the get_version_cmd inside this container. Set this command out put as self.release.
- Download the remote file via Docker API, extract the downloaded .tar file.
- **When the downloading is complete:**
 - if keep_container=false: Remove the above container after.
 - if keep_container=true: leave this container running.
- If there are any errors during the data dump, the remote container and volumes won't be removed.

There are additional parameters that can also be added. All of them will be listed here with a short summary, but we will not be using all of the parameters for this tutorial:

- image: (Optional) the Docker image name
- tag: (Optional) the image tag

- container_name: (Required) Boots up an existing container. If the container does not exist, it will create a new container using the image and tag parameters.
- volumes: (Optional) Used specifically for local bind mounts. If used without a named_volume, the volume will not automatically be removed once the data finishes dumping.
- named_volumes: (Optional) Creates a named volume to be removed when the data finishes dumping.
- path: (Required) path to the remote file inside the Docker container.
- dump_command: (Required) This command will be run inside the Docker container in order to create the remote file.
- **keep_container: (Optional) accepted values: true/false, default: false.**
 - If keep_container=true, the remote container will be persisted.
 - If keep_container=false, the remote container will be removed in the end of dump step.
- get_version_cmd: (Optional) The custom command for checking release version of local and remote file.

Now we will need to clone the tutorial onto your local computer and switch to pharmgkb_v7.

Now we will need to install the requirements to run our Biothings CLI. We will first create a virtual environment and then install a Biothings Hub dev environemt.

Just like our original pharmgkb plugin, we have a manifest and a parser file with the new addition of a Dockerfile. Lets have a quick look at the manifest file.

Note: If you notice, the manifest file is in a yaml format while the previous manifest files have all been in a json format. This is because our Hub can parse both yaml and json formatted files!

As you can see, the manifest file is in a very similar format as the manifest file in our Data Plugin <studio.html#id1>_ section. The only difference is we have included a new data_url section within the dumper. The data_url should match the following format:

```
docker://CONNECTION_NAME?image=DOCKER_IMAGE&tag=TAG&path=/path/to/
remote_file&dump_command="this is custom command"&container_name=CONTAINER_NAME&keep_container=true&get...
```

There seems to be an issue though looking at our listed parameters, the dump_command, path, and countainer_name are all required, but they seem to be missing from the data_url. Lets try to dump using this manifest file to see what happens!

Lets first build our docker file. As shown in our manifest, the data url has `image=annotations` so when we build we have to make sure to name our image accordingly.

Now we can finally test our source using the Biothings CLI. Since our document size is small we can directly use the dump_and_upload. If you are working with a larger source you will need need to use them separately and specify the --batch-limit flag when uploading:

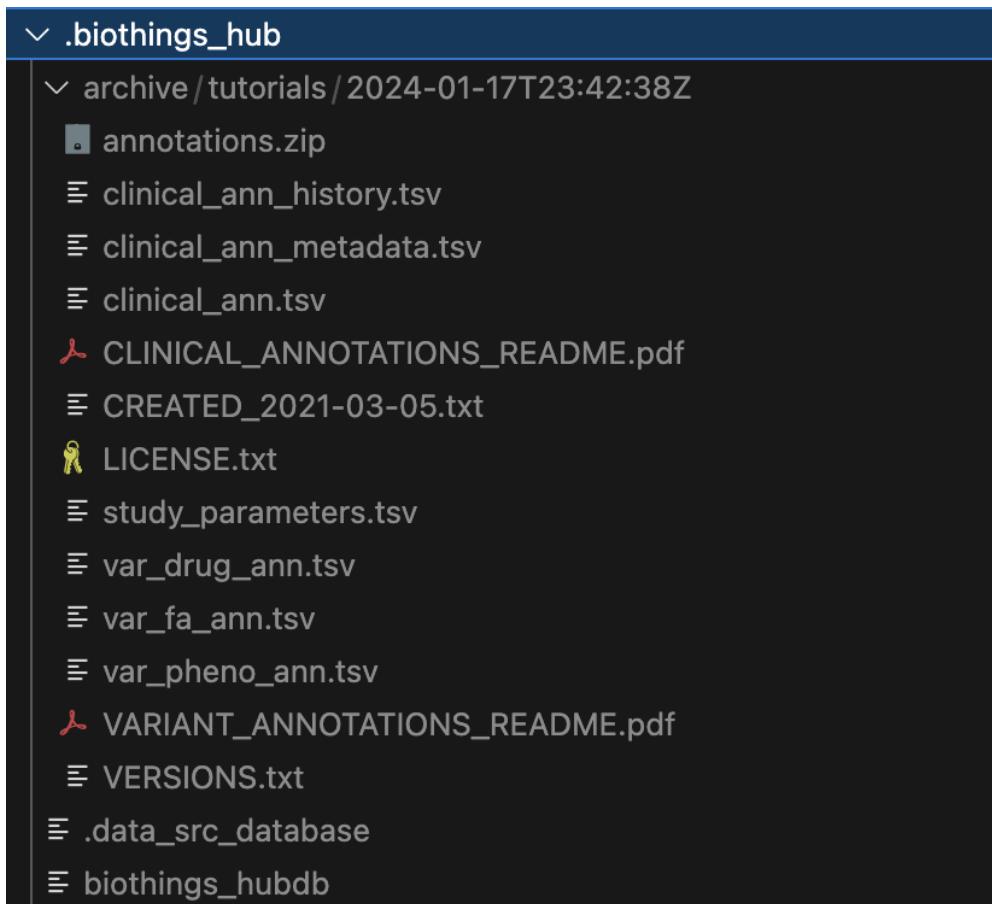
We have now successfully dumped and uploaded our source. Why did we not see any of the expected errors for missing parameters?

```
INFO Success! 🎉
Dump —————
Source: tutorials
Data Folder: .biothings_hub/archive/tutorials/2024-01-17T23:42:38Z:
- var_fa_ann.tsv
- study_parameters.tsv
- var_pheno_ann.tsv
- VERSIONS.txt
- var_drug_ann.tsv
- CREATED_2021-03-05.txt
- clinical_ann_metadata.tsv
- clinical_ann_history.tsv
- clinical_ann.tsv
- CLINICAL_ANNOTATIONS_README.pdf
- VARIANT_ANNOTATIONS_README.pdf
- LICENSE.txt
- annotations.zip

Upload —————
Source: tutorials
DB path: /Users/jalin/Desktop/Scripps_Work/tutorials/.biothings_hub/.data_src_database
- Database: .data_src_database
  - Collections:
    - tutorials
  - Archived collections:
    - tutorials_archive_20240117_2UoS5akt
  - Temporary collections:
```

To answer this question, we have to take a look at the Dockerfile.

To keep the data url from becoming too difficult to read, we can directly set the parameters into a Docker **LABEL** object. This is why we are able to run the dump and upload process without encountering any errors. A new directory has been created by the CLI. Lets take a look at it.



Within the `.biothings_hub/archive/tutorials/2024-01-17T23:42:38Z` directory, there is an `annotations.zip` that was dumped along with the uncompressed contents that we specified in the manifest. Since we did not specify a version, the current datetime is automatically used for the directory name .

`biothings_hub/archive/tutorials/2024-01-17T23:42:38Z` The `.biothings_hub/biothings_hubdb` and `.biothings_hub/.data_src_database` are both sqlite databases that hold the information that would normally be held in the mongodb. The former being the datasource settings and the latter holding our uploaded data.

With our uploaded data in our database, we can finally serve this data on our localhost.

The result should look something similar to this:

```
(.venv) (base) jaling@MacBook-Pro tutorials % biothings-cli dataplugin serve
[...]
INFO: [INFO] Starting requirement server
Requirement already satisfied: pandas in ./venv/lib/python3.11/site-packages (2.1.4)
Requirement already satisfied: numpy<2,>=1.23.2 in ./venv/lib/python3.11/site-packages (from pandas) (1.26.3)
Requirement already satisfied: python-dateutil>=2.8.2 in ./venv/lib/python3.11/site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in ./venv/lib/python3.11/site-packages (from pandas) (2023.3.post1)
Requirement already satisfied: tzdata<2022.1 in ./venv/lib/python3.11/site-packages (from pandas) (2023.4)
Requirement already satisfied: tzlocal<2.1.0 in ./venv/lib/python3.11/site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
[...]
INFO: [INFO] Installed requirement: numpy
Requirement already satisfied: numpy in ./venv/lib/python3.11/site-packages (1.26.3)
Serving data plugin source: tutorials
Found collection: tutorials; counting...Done (1018 documents)
Listening on http://localhost:9999
View all available routes: http://localhost:9999/
- http://localhost:9999/tutorials
  ↗ Get a document by id:
    http://localhost:9999/tutorials/_doc_{id}
  Examples:
    http://localhost:9999/tutorials/PA33523
    http://localhost:9999/tutorials/PA27681
  ↗ Query documents by fields:
    http://localhost:9999/tutorials?_query={query}
  Examples:
    http://localhost:9999/tutorials?from=0&size=10
    http://localhost:9999/tutorials?q=annotations.alleles:G
    http://localhost:9999/tutorials?q=annotations.variant:rs1611115 AND annotations.significance:yes
```

Congratulations you have learned how to build a Docker based plugin! Remember to make sure to always test out some of the queries before submitting your plugin.

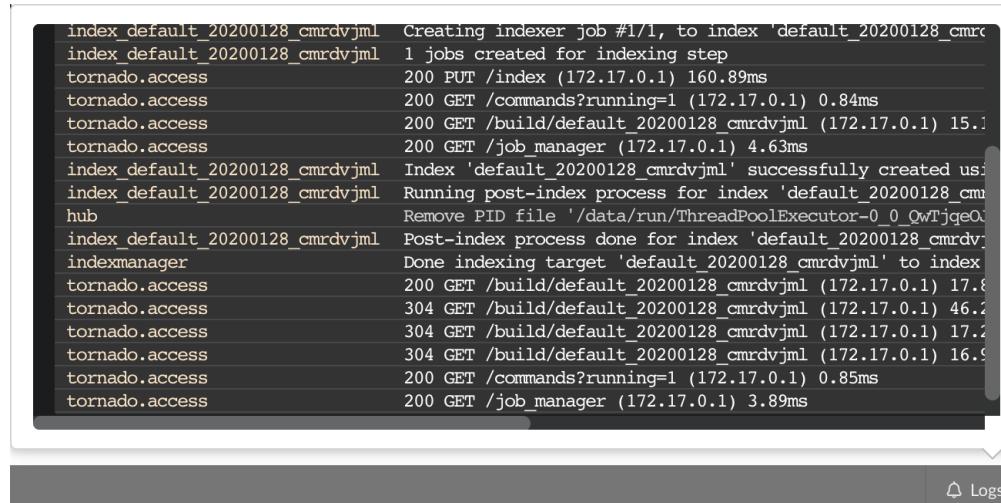
1. API cloud deployments and hosting

This part is still under development... Stay tuned and join Biothings Google Groups (<https://groups.google.com/forum/#!forum/biothings>) for more.

8. Troubleshooting

We test and make sure, as much as we can, that the **BioThings Studio** image is up-to-date and running properly. But things can still go wrong...

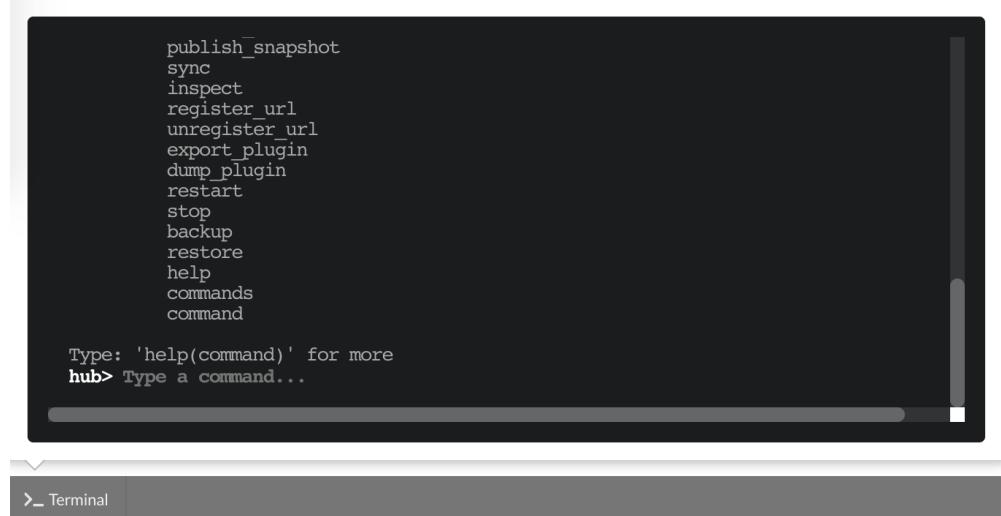
A good starting point investigating an issue is to look at the logs from the **BioThings Studio**. Make sure it's connected (green power button on the top right), then click "Logs" button, on the bottom right. You will see logs in real-time (if not connected, it will complain about a disconnected websocket). As you click and perform actions throughout the web application, you will see log message in that window, and potentially errors not displayed (or with less details) in the application.



The screenshot shows a terminal-like interface for viewing logs. The logs are displayed in a scrollable window, showing various requests and responses from the application. At the bottom right of the log window, there is a button labeled "Logs".

```
index_default_20200128_cmrdvjml Creating indexer job #1/1, to index 'default_20200128_cmrc
index_default_20200128_cmrdvjml 1 jobs created for indexing step
tornado.access 200 PUT /index (172.17.0.1) 160.89ms
tornado.access 200 GET /commands?running=1 (172.17.0.1) 0.84ms
tornado.access 200 GET /build/default_20200128_cmrdvjml (172.17.0.1) 15.1ms
tornado.access 200 GET /job_manager (172.17.0.1) 4.63ms
index_default_20200128_cmrdvjml Index 'default_20200128_cmrdvjml' successfully created us...
index_default_20200128_cmrdvjml Running post-index process for index 'default_20200128_cm...
hub Remove PID file '/data/run/ThreadPoolExecutor-0_0_QwTjgeO...
index_default_20200128_cmrdvjml Post-index process done for index 'default_20200128_cmrdv...
indexmanager Done indexing target 'default_20200128_cmrdvjml' to index
tornado.access 200 GET /build/default_20200128_cmrdvjml (172.17.0.1) 17.8ms
tornado.access 304 GET /build/default_20200128_cmrdvjml (172.17.0.1) 46.1ms
tornado.access 304 GET /build/default_20200128_cmrdvjml (172.17.0.1) 17.2ms
tornado.access 304 GET /build/default_20200128_cmrdvjml (172.17.0.1) 16.9ms
tornado.access 200 GET /commands?running=1 (172.17.0.1) 0.85ms
tornado.access 200 GET /job_manager (172.17.0.1) 3.89ms
```

The “Terminal” (click on the bottom left button) gives access to commands you can manually type from the web application. Basically, any action performed clicking on the application is converted into a command call. You can even see what commands were launched and which ones are running. This terminal also gives access to more commands, and advanced options that may be useful to troubleshoot an issue. Typing `help()`, or even passing a command name such as `help(dump)` will print documentation on available commands and how to use them.



On a lower level, make sure all services are running in the docker container. Enter the container with `docker exec -ti studio /bin/bash` and type `netstat -tnlp`, you should see services running on ports (see usual running services). If services on ports 7080 and 7022 aren't running, it means the **Hub** has not started. If you just started the instance, wait a little more as services may take a while before they're fully started and ready.

If after ~1 min, you still don't see the **Hub** running, log in as user `biothings` and check the starting sequence.

Note: **Hub** is running in a tmux session, under user `biothings`.

```
# sudo su - biothings
$ tmux a # recall tmux session

$ python bin/hub.py
DEBUG:asyncio:Using selector: EpollSelector
INFO:root:Hub DB backend: {'uri': 'mongodb://localhost:27017', 'module': 'biothings.utils.mongo'}
INFO:root:Hub database: biothings_src
DEBUG:hub:Last launched command ID: 14
INFO:root:Found sources: []
INFO:hub:Loading data plugin 'https://github.com/sirlooin/mvcgi.git' (type: github)
DEBUG:hub:Creating new GithubAssistant instance
DEBUG:hub:Loading manifest: {'dumper': {'data_url': 'https://www.cancergenomeinterpreter.org/data/cgi biomarkers latest.zip', 'uncompress': True}, 'uploader': {'ignore_duplicates': False, 'parser': 'parser:load_data'}, 'version': '0.1'}
INFO:indexmanager:{}
INFO:indexmanager:{'test': {'max_retries': 10, 'retry_on_timeout': True, 'es_host': 'localhost:9200', 'timeout': 300}}
DEBUG:hub:for managers [<SourceManager [0 registered]>, <AssistantManager [1]
```

(continues on next page)

(continued from previous page)

```

→registered]: ['github']>]
INFO:root:route: ['GET'] /job_manager => <class 'biothings.hub.api.job_manager_handler'>
INFO:root:route: ['GET'] /command/([\w\.]*)? => <class 'biothings.hub.api.command_handler'
→'>
...
INFO:root:route: ['GET'] /api/list => <class 'biothings.hub.api.list_handler'>
INFO:root:route: ['PUT'] /restart => <class 'biothings.hub.api.restart_handler'>
INFO:root:route: ['GET'] /status => <class 'biothings.hub.api.status_handler'>
DEBUG:tornado.general:sockjs.tornado will use json module
INFO:hub:Monitoring source code in, ['/home/biothings/biothings_studio/hub/dataload/
→sources', '/home/biothings/biothings_studio/plugins']:
['/home/biothings/biothings_studio/hub/dataload/sources',
 '/home/biothings/biothings_studio/plugins']

```

You should see something like above. If not, you should see the actual error, and depending on the error, you may be able to fix it (not enough disk space, etc...). **BioThings Hub** can be started again using `python bin/hub.py` from within the application directory (in our case, `/home/biothings/biothings_studio`)

Note: Press Control-B then D to detach the tmux session and let the **Hub** run in background.

By default, logs are available in `/data/biothings_studio/logs/`.

Finally, you can report issues and request for help, by joining Biothings Google Groups (<https://groups.google.com/forum/#!forum/biothings>).

6.1.2 B. Developer's guide

This section provides both an overview and detailed information about **BioThings Studio**, and is specifically aimed to developers who like to know more about internals.

A complementary [tutorial](#) is also available, explaining how to set up and use **BioThings Studio**, step-by-step, by building an API from a flat file.

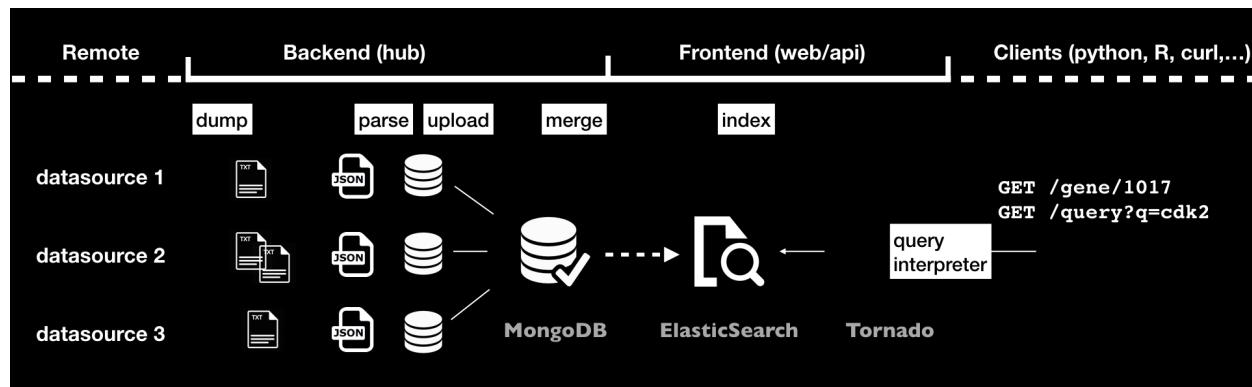
1. What is BioThings Studio

BioThings Studio is a pre-configured, ready-to-use application. At its core is **BioThings Hub**, the backend service behind all BioThings APIs.

1.1. BioThings Hub: the backend service

Hub is responsible for maintaining data up-to-date, and creating data releases for the BioThings frontend.

The process of integrating data and creating releases involves different steps, as shown in the following diagram:



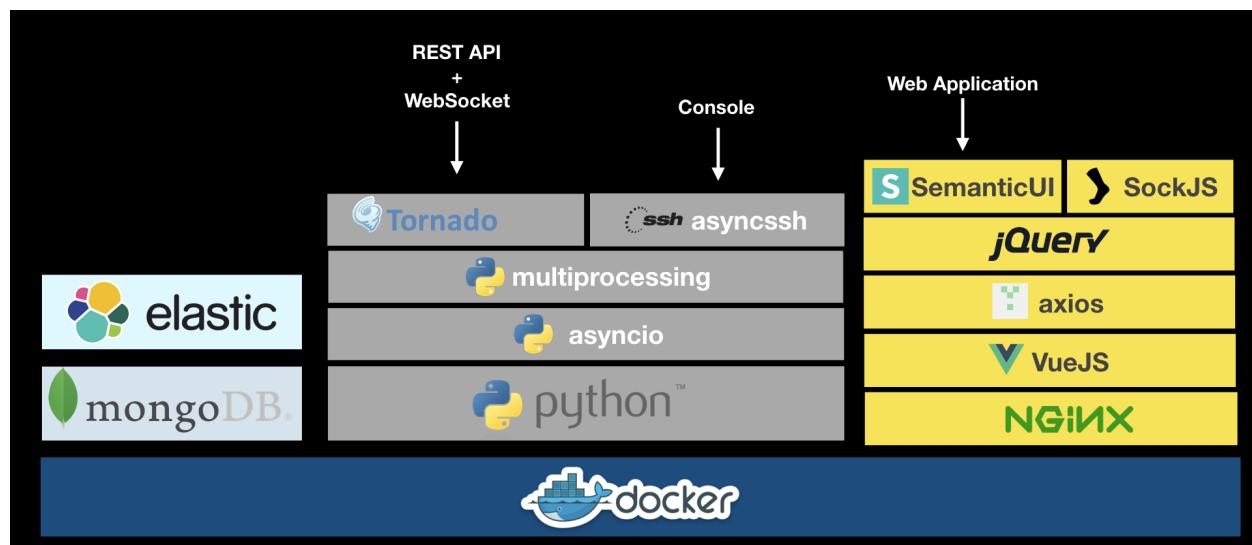
- data is first downloaded locally using *dumpers*
- *parsers* will then convert data into JSON documents, those will be stored in a Mongo database using *uploaders*
- when using multiple sources, data can be combined together using *mergers*
- data releases are then created either by indexing data to an ElasticSearch cluster with *indexers*, or by computing the differences between the current release and previous one, using *differs*, and applying these differences using *syncers*

The final index along with the Tornado application represents the frontend that is actually queried by the different available clients, and is out of this document's scope.

1.2. BioThings Studio

The architecture and different software involved in this system can be quite intimidating. To help, the whole service is packaged as a pre-configured application, **BioThings Studio**. A docker image is available Docker Hub registry, and can be pulled using:

```
$ docker pull biothings/biothings-studio:0.2a
```



A **BioThings Studio** instance exposes several services on different ports:

- **8080:** **BioThings Studio** web application port
- **7022:** **BioThings Hub** SSH port

- **7080:** BioThings Hub REST API port
- **7081:** BioThings Hub read-only REST API port
- **9200:** ElasticSearch port
- **27017:** MongoDB port
- **8000:** BioThings API, once created, it can be any non-privileged (>1024) port
- **9000:** Cerebro, a webapp used to easily interact with ElasticSearch clusters
- **60080:** Code-Server, a webapp used to directly edit code in the container

BioThings Hub and the whole backend service can be accessed through different options according to some of these services:

- a web application allows interaction with the most used elements of the service (port 8080)
- a console, accessible through SSH, gives access to more commands, for advanced usage (port 7022)
- a REST API and a websocket (port 7080) can be used to interact with the **Hub**, query the different objects inside, and get real-time notifications when processes are running. This interface is a good choice for third-party integration.

1.3. Who should use BioThings Studio ?

BioThings Studio can be used in different scenarios:

- you want to contribute to an existing BioThings API by integrating a new data source
- you want to run your own BioThings API but don't want to install all the dependencies and learn how to configure all the sub-systems

1.4. Filesystem overview

Several locations on the filesystem are important, when it comes to change default configuration or troubleshoot the application:

- **Hub** (backend service) is running under `biothings` user, running code is located in `/home/biothings/biothings_studio`. It heavily relies on BioThings SDK located in `/home/biothings/biothings.api`.
- Several scripts/helpers can be found in `/home/biothings/bin`:
 - `run_studio` is used to run the Hub in a tmux session. If a session is already running, it will first kill the session and create a new one. We don't have to run this manually when the studio first starts, it is part of the starting sequence.
 - `update_studio` is used to fetch the latest code for **BioThings Studio**
 - `update_biothings`, same as above but for BioThings SDK
- `/data` contains several important folders:
 - `mongodb` folder, where MongoDB server stores its data
 - `elasticsearch` folder, where ElasticSearch stores its data
 - `biothings_studio` folder, containing different sub-folders used by the **Hub**:
 - * `datasources` contains data downloaded by the different `dumpers`, it contains sub-folders named according to the datasource's name. Inside the datasource folder can be found the different releases, one per folder.

- * `dataupload` is where data is stored when uploaded to the Hub (see below dedicated section for more).
- * `logs` contains all log files produced by the **Hub**
- * `plugins` is where data plugins can be found (one sub-folder per plugin's name)

Note: Instance will store MongoDB data in `/data/mongodb`, ElasticSearch data in `/data/elasticsearch/` directory, and downloaded data and logs in `/data/biotothings_studio`. Those locations could require extra disk space; if necessary, Docker option `-v` can be used to mount a directory from the host, inside the container. Please refer to Docker documentation. It's also important to give enough permissions so the different services (MongoDB, ElasticSearch, NGNIX, BioThings Hub, ...) can actually write data on the docker host.

1.5. Configuration files

BioThings Hub expects some configuration variables to be defined first, in order to properly work. In most **BioThings Studio**, a `config_hub.py` defines those parameters, either by providing default value(s), or by setting them as `ConfigurationError` exception. In the latter case, it means no defaults can be used and user/developer has to define it. A final `config.py` file must be defined, it usually imports all parameters from `config_hub.py` (`from config_hub import *`). That `config.py` *has* to be defined before the **Hub** can run.

Note: This process is only required when implementing or initializing a Hub from scratch. All **BioThings Studio** applications come with that file defined, and the **Hub** is ready to be used.

It's also possible to override parameters directly from the webapp/UI. In that case, new parameters' values are stored in the internal Hub database. Upon start, **Hub** will check that database and supersede any values that are defined directly in the python configuration files. This process is handled by class `biotothings.ConfigurationManager`.

Finally, a special (simple) dialect can be used while defining configuration parameters, using special markup within comments above declaration. This allows to:

- provide documentation for parameters
- put parameters under different categories
- mark a parameter as read-only
- set a parameter as “invisible” (not exposed)

This process is used to expose **Hub** configuration through the UI, automatically providing documentation in the webapp without having to duplicate code, parameters and documentation. For more information, see class `biotothings.ConfigurationParser`, as well as existing configuration files in the different studios.

1.6. Services check

Let's enter the container to check everything is running fine. Services may take a while, up to 1 min, before fully started. If some services are missing, the troubleshooting section may help.

```
$ docker exec -ti studio /bin/bash
root@301e6a6419b9:/tmp# netstat -tnlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address          Foreign Address        State      PID/
→Program name
```

(continues on next page)

(continued from previous page)

tcp	0	0 0.0.0.0:7080	0.0.0.0:*	LISTEN	-
tcp	0	0 0.0.0.0:9000	0.0.0.0:*	LISTEN	-
tcp	0	0 127.0.0.1:27017	0.0.0.0:*	LISTEN	-
tcp	0	0 0.0.0.0:7022	0.0.0.0:*	LISTEN	-
tcp	0	0 0.0.0.0:9200	0.0.0.0:*	LISTEN	-
tcp	0	0 0.0.0.0:8080	0.0.0.0:*	LISTEN	166/
<code>↳ nginx: master p</code>					
tcp	0	0 0.0.0.0:9300	0.0.0.0:*	LISTEN	-
tcp	0	0 0.0.0.0:22	0.0.0.0:*	LISTEN	416/sshd
tcp6	0	0 :::7080	:::*	LISTEN	-
tcp6	0	0 :::7022	:::*	LISTEN	-
tcp6	0	0 :::22	:::*	LISTEN	416/sshd

Specifically, BioThings Studio services' ports are: 7080, 7022 and 8080.

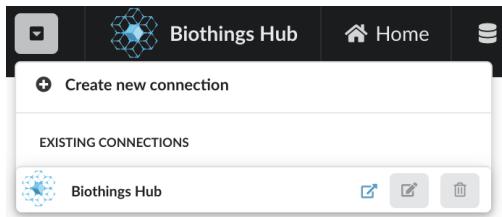
2. Overview of BioThings Studio web application

BioThings Studio web application can simply be accessed using any browser pointing to port 8080. The home page shows a summary of current data and recent updates. For now, it's pretty quiet since we didn't integrate any data yet.

Let's have a quick overview of the different elements accessible through the webapp. On the top left is the connection widget. By default, **BioThings Studio** webapp will connect to the hub API through port 7080, the one running within docker. But the webapp is a static web page, so you can access any other Hub API by configuring a new connection:

Enter the Hub API URL, `http://<host>:<port>` (you can omit `http://`, the webapp will use that scheme by default):

The new connection is now listed and can be accessed quickly later simply by selecting it. Note the connection can be deleted with the “trash” icon, but cannot be edited.



Following are several tabs giving access to the main steps involved in building a BioThings API. We'll get into those in more details while creating our new API. On the right, we have different information about jobs and resources:

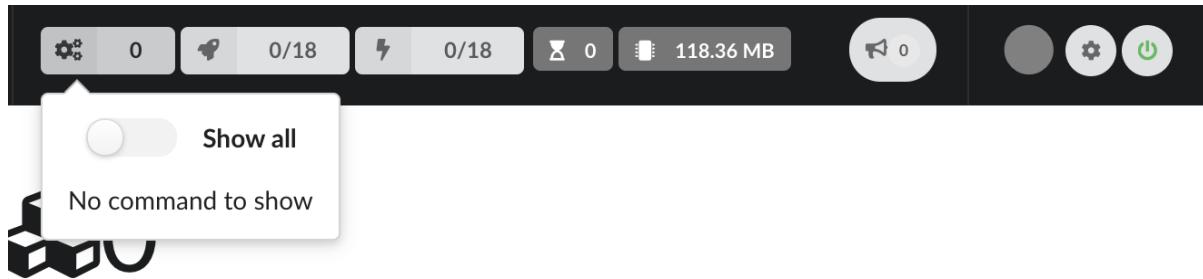


Fig. 1: Running commands are shown in this popup, as well as as commands that have been running before, when switching to “Show all”

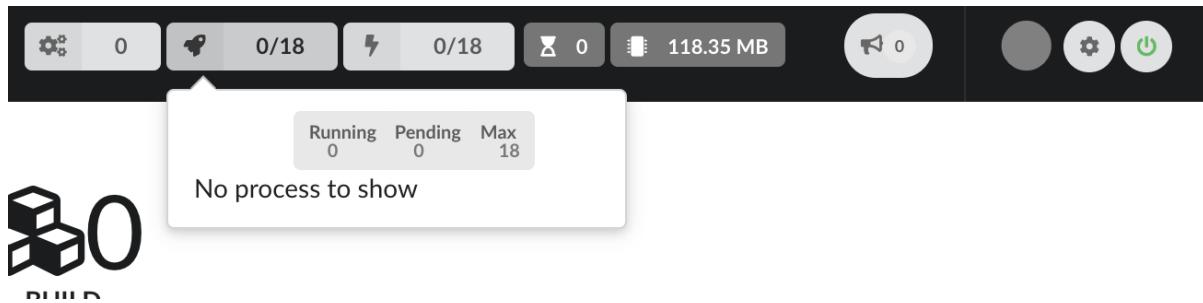


Fig. 2: When jobs are using parallelization, processes will show information about what is running and how much resources each process takes. Notice we only have 1 process available, as we're running a t2.medium instance which only has 2 CPU, Hub has automatically assigned half of them.

3. Configuration

By clicking on the cog icon in the bar on the right, **Hub** configuration can be accessed. The configuration parameters, documentation, sections are defined in python configuration files (see *Configuration files*). Specifically, if a parameter is hidden, redacted or/and read-only, it's because of how it was defined in the python configuration files.

All parameters must be entered in a JSON format. Ex: double quotes for strings, square brackets to define lists, etc. A changed parameter can be saved using the “Save” button, available for each parameter. The “Reset” button can be used to switch it back to the original default value that was defined in the configuration files.

Ex: Update Hub's name

First enter the new name, for parameter HUB_NAME. Because the value has changed, the “Save” button is available.

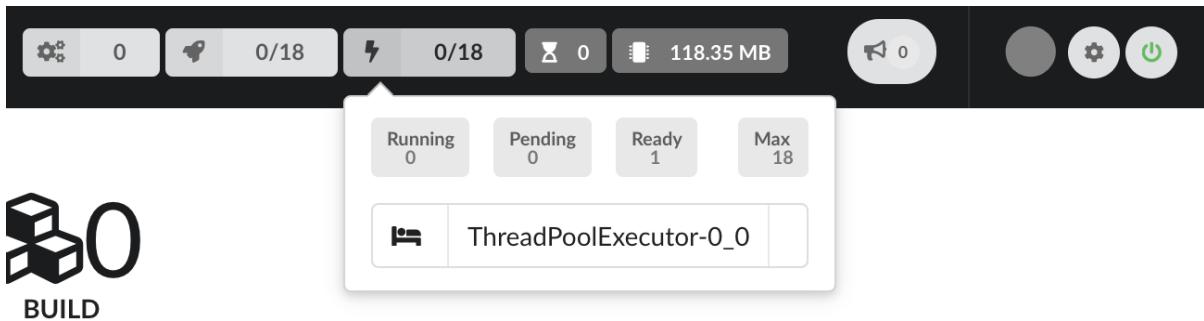


Fig. 3: **BioThings Hub** also uses threads for parallelization, their activity will be shown here. Number of queued jobs, waiting for a free process or thread, is shown as well as the total amount of memory the **Hub** is currently using

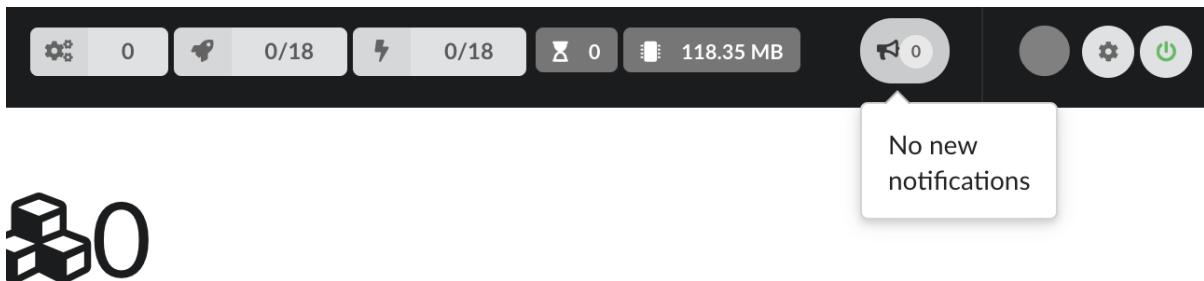


Fig. 4: In this popup are shown all notifications coming from the **Hub**, in real-time, allowing to follow all jobs and activity.

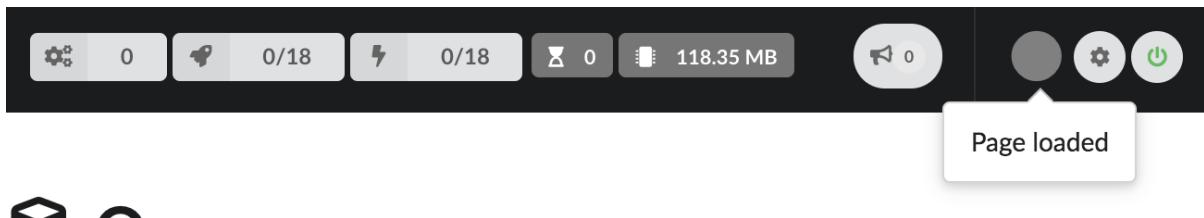


Fig. 5: The first circle shows the page loading activity. Gray means nothing active, flashing blue means webapp is loading information from the Hub, and red means an error occurred (error should be found either in notifications or by opening the logs from the bottom right corner).

The next button with a cog icon gives access to the configuration and is described in the next section.

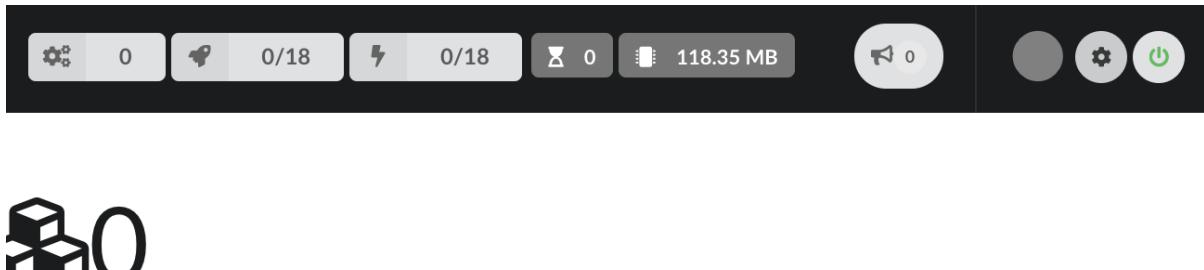


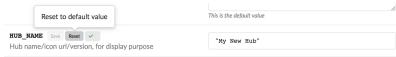
Fig. 6: Finally, a logo shows the websocket connection status (green power button means “connected”, red plug means “not connected”).

The screenshot shows the 'Hub Configuration' page with a dark theme. At the top, there are tabs for Home, Services, Builds, API, and several status indicators (0 builds, 0/18 releases, 0/18 jobs, 0 tasks, 118.35 MB storage). A prominent warning message in a box states: "Warning: From this page, you can restart or stop the Hub, access and modify its internal configuration in order to customize the behavior and appearance. Be careful though, as incorrect values could lead to a non-working system." Below the warning is a red 'Restart' button. The main content area has a tab navigation bar with '1. General' selected (highlighted in red), followed by tabs for 2. Datasources, 3. Folders, 4. Index & Diff, 5. Release, 6. Job Manager, 7. Hub Internals, and Misc. A note at the top of the configuration section says "All values must be in JSON format". The 'HUB_API_PORT' field is set to 7080 (read-only). The 'HUB_ICON' field contains the value "null". The 'HUB_NAME' field is set to "Biothings Hub". Each field has a 'Save' and 'Reset' button.

HUB_NAME	Save	Reset
Hub name/icon url/version, for display purpose		
This is the default value		

HUB_NAME	Save	Reset
Hub name/icon url/version, for display purpose		
"My New Hub"		

Upon validation, a green check mark is shown, and because the value is not the default one, the “Reset” button is now available. Clicking on it will switch the parameter’s value back to its original default one.



Note each time a parameter is changed, **Hub** needs to be restarted, as shown on the top.



4. Data plugin architecture and specifications

BioThings Studio allows to easily define and register datasources using *data plugins*. As of **BioThings Studio 0.2b**, there are two different types of data plugin.

4.1. Manifest plugins

- a *manifest.json* file
- other python files supporting the declaration in the manifest.

The plugin name, that is, the folder name containing the manifest file, gives the name to the resulting datasource.

A manifest file is defined like this:

```
{
  "version": "0.2",
  "__metadata__": { # optional
    "url" : "<datasource website/url>",
    "license_url" : "<url>",
    "licence" : "<license name>",
  "author" : {
    "name" : "<author name>",
    "url" : "<link to github's author for instance>"
  }
  },
  "requires" : ["lib==1.3", "anotherlib"],
  "dumper" : {
    "data_url" : "<url> # (or list of url: [<url1>, <url1>]), # optional
    "uncompress" : true|false, # optional, default to false
    "release" : "<path.to.module>:<function_name>" # optional
    "schedule" : "@ 12 * * *" # optional
  },
  "uploader" : { # optional
    "parser" : "<path.to.module>:<function_name>",
    "on_duplicates" : "ignore|error|merge" # optional, default to "error"
  }
}
```

or with multiple uploader

```
{  
    "version": "0.2",  
    "__metadata__": { # optional  
        "url" : "<datasource website/url>",  
        "license_url" : "<url>",  
        "licence" : "<license name>",  
        "author" : {  
            "name" : "<author name>",  
            "url" : "<link to github's author for instance>"  
        }  
    },  
    "requires" : ["lib==1.3", "anotherlib"],  
    "dumper" : {  
        "data_url" : "<url> # (or list of url: [<url1>, <url1>]),  
        "uncompress" : true|false, # optional, default to false  
        "release" : "<path.to.module>:<function_name>" # optional  
        "schedule" : "@ 12 * * *" # optional  
    },  
    "uploaders" : [{ # optional  
        "parser" : "<path.to.module>:<function_name_1>",  
        "on_duplicates" : "ignore|error|merge" # optional, default to "error"  
    }, {  
        "parser" : "<path.to.module>:<function_name_2>",  
        "on_duplicates" : "ignore|error|merge" # optional, default to "error"  
    }, {  
        "parser" : "<path.to.module>:<function_name_3>",  
        "on_duplicates" : "ignore|error|merge" # optional, default to "error"  
    }]  
}
```

Note: it's possible to only have a dumper section, without any uploader specified. In that case, the data plugin will only download data and won't provide any way to parse and upload data.

- a *version* defines the specification version the manifest is using. Currently, version 0.2 should be used. This is not the version of the datasource itself.
- an optional (but highly recommended) *__metadata__* key provides information about the datasource itself, such as a website, a link to its license, the license name. This information, when provided, is displayed in the / *metadata* endpoint of the resulting API.
- a *requires* section, optional, describes dependencies that should be installed for the plugin to work. This uses *pip* behind the scene, and each element of that list is passed to *pip install* command line. If one dependency installation fails, the plugin is invalidated. Alternately, a single string can be passed, instead of a list.
- a *dumper* section specifies how to download the actual data:
 - *data_url* specifies where to download the data from. It can be a URL (string) or a list of URLs (list of strings). Currently supported protocols are **http(s)** and **ftp**. URLs must point to individual files (no wildcards), and only one protocol is allowed within a list of URLs (no mix of URLs using http and ftp are allowed). All files are download in a data folder, determined by *config*. **DATA_ARCHIVE_ROOT/<plugin_name>/<release>**

- *uncompress*: once data is downloaded, this flag, if set to true, will uncompress all supported archives found in the data folder. Currently supported formats are: *.zip, *.gz, *.tar.gz (includes untar step)

- *schedule* will trigger the scheduling of the dumper, so it automatically checks for new data on a regular basis. Format is the same as crontabs, with the addition of an optional sixth parameter for scheduling by the seconds.

Ex: * * * * */10 will trigger the dumper every 10 seconds (unless specific use case, this is not recommended).

For more information, **Hub** relies on [aiocron](#) for scheduling jobs.

- *release* optionally specifies how to determine the release number/name of the datasource. By default, if not present, the release will be set using:

- * `Last-Modified` header for an HTTP-based URL. Format: YYYY-MM-DD
- * `ETag` header for an HTTP-based URL if `Last-Modified` isn't present in headers. Format: the actual etag hash.
- * `MDTM` ftp command if URL is FTP-based.

If a list of URLs is specified in *data_url*, the last URL is the one used to determine the release. If none of those are available or satisfactory, a *release* section can be specified, and should point to a python module and a function name following this format: `module:function_name`. Within this module, function has the following signature and should return the release, as a string. `set_release` is a reserved name and must not be used.

The example about *release* can be found at <https://github.com/remoteeng00/FIRE.git>

- * In master branch, the manifest file does not contain *release* field, so you can see the “failed” when dump the data source.
- * When you checkout to the version “v2” (<https://github.com/remoteeng00/FIRE/tree/v2>) then you can dump the data source.

```
def function_name(self):
    # code
    return "..."
```

`self` refers to the actual dumper instance of either `biothings.hub.dataload.dumper.HTTPDumper` or `biothings.hub.dataload.dumper.FTPDumper`, depending on the protocol. All properties and methods from the instance are available, specifically:

- `self.client`, the actual underlying client used to download files, which is either a `request.Session` or a `ftplib.FTP` instance, and should be preferred over initializing a new connection/client.
- `self.SRC_URLS`, containing the list of URLs (if only one URL was specified in *data_url*, this will be a list of one element), which is commonly used to inspect and possibly determine the release.
- an *uploader* section specifies how to parse and store (upload):

- `parser` key defines a module and a function name within that module. Format: `module:function_name`. Function has the following signature and returns a list of dictionary

(or `yield` dictionaries) containing at least a `_id` key representing a unique identifier (string) for this document:

```
def function_name(data_folder):
    # code
    yield {"_id": "..."}
```

`data_folder` is the folder containing the previously downloaded (dumped) data, it is automatically set to the latest release available. Note the function doesn't take an filename as input, it should select the file(s) to parse.

- `on_duplicates` defines the strategy to use when duplicated records are found (according to the `_id` key):
 - `error` (default) will raise an exception if duplicates are found
 - `ignore` will skip any duplicates, only the first one found will be stored
 - `merge` will merge existing document with the duplicated one. Refer to `biothings.hub.dataloader.storage.MergerStorage` class for more.
- `parallelizer` points to a `module:function_name` that can be used when the uploader can be parallelized. If multiple input files exist, using the exact same parser, the uploader can be parallelized using that option. The parser should take an input file as parameter, not a path to a folder. The parallelizer function should return a list of tuples, where each tuple corresponds to the list of input parameters for the parser. `jobs` is a reserved name and must not be used.
- `mapping` points to a `module:classmethod_name` that can be used to specify a custom ElasticSearch mapping. Class method must return a python dictionary with a valid mapping. `get_mapping` is a reserved name and must not be used. There's no need to add `@classmethod` decorator, `Hub` will take care of it. The first and only argument is a class. Ex:

```
def custom_mapping(cls):
    return {
        "root_field": {
            "properties": {
                "subfield": {
                    "type": "text",
                }
            }
        }
    }
```

- If you want to use multiple uploader in your data plugin, you will need to use `uploaders` section, it's a list of above `uploader`.

Please see https://github.com/remoteeng00/pharmgkb/tree/pharmgkb_v5 for an example about multiple uploader definition.

Note: Please do not use both `uploaders` and `uploader` in your manifest file.

Note: Please see <https://github.com/sirlooin/mvcgi> for a simple plugin definition. <https://github.com/sirlooin/gwascatalog> will show how to use the `release` key; <https://github.com/remoteeng00/FIRE> will demonstrate the parallelization in the uploader section.

4.2. Advanced plugins

This type of plugins is more advanced in the sense that it's plain python code. They typically come from a code export of a manifest plugin but has slightly different (Following the [A.5.2. Code export](#) section, the exported python code is placed in `hub/dataload/sources/*` folder, but advanced plugins are placed in the same folder with manifest plugins at `config.DATA_PLUGIN_FOLDER`). The resulting python code defines dumpers and uploaders as python class, inheriting from BioThings SDK components. These plugins can be written from scratch, they're "advanced" because they require more knowledge about BioThings SDK.

In the root folder (local folder or remote git repository), a `__init__.py` is expected, and should contain imports for one dumper, and one or more uploaders.

An example of advanced data plugin can be found at https://github.com/sirlooin/mvcgi_advanced.git. It comes from "mvcgi" manifest plugin, where code was exported.

5. Hooks and custom commands

While it's possible to define custom commands for the Hub console by deriving class `biothings.hub.HubServer`, there's also an easy way to enrich existing commands using **hooks**. A **hook** is a python file located in `HOOKS_FOLDER` (defaulting to `./hooks/`). When the Hub starts, it inspects this folder and "injects" hook's namespace into its console. Everything available from within the hook file becomes available in the console. On the other hand, hook can use any commands available in the Hub console.

Hooks provide an easy way to "program" the Hub, based on existing commands. The following example defines a new command, which will archive any builds older than X days. Code can be found at https://github.com/sirlooin/auto_archive_hook.git. File `auto_archive.py` should be copied into `./hooks/` folder. Upon restart, a new command named `auto_archive` is now part of the Hub. It's also been scheduled automatically using `schedule(...)` command at the end of the hook.

The `auto_archive` function uses several existing Hub commands:

- `lsmerge`: when given a build config name, returns a list of all existing build names.
- `archive`: will delete underlying data but keep existing metadata for a given build name
- `bm.build_info`: `bm` isn't a command, but a shortcut for **build_manager** instance. From this instance, we can call `build_info` method which, given a build name, returns information about it, including the `build_date` field we're interested in.

Note: Hub console is actually a python interpreter. When connecting to the Hub using SSH, the connection "lands" into that interpreter. That's why it's possible to inject python code into the console.

Note: Be careful. User-defined hooks can be conflicting with existing commands and may break the Hub. Ex: if a hook defines a command "dump", it will replace, and potentially break existing one!

6.2 BioThings CLI

6.2.1 Introduction

The BioThings CLI (Command Line Interface) provides a set of convenience command line tools for developers to create and test data plugins locally. Compared to the option of setting up a local Hub running in docker containers, the CLI further lowers the entry barrier by NOT requiring docker or any external databases installed locally. It is particularly suitable for data plugin developers to build and test their data plugin independantly. When a data plugin is ready, they can then pass the data plugin to a running BioThings Hub to build the data plugin into a BioThings API.

This tutorial aims to provide a comprehensive guide to the BioThings CLI, covering its essential commands and functionalities. We will explore a range of topics including installation, initial setup, and core features such as data plugin dump, upload, inspect, and other utility commands. Additionally, we will delve into practical applications of the CLI, demonstrating how to work with the local API server for data inspection and parser debugging.

6.2.2 Prerequisites

To use the BioThings CLI, you need to have Python installed on your system, specifically version 3.7 or higher.

Ensure that your Python version meets this requirement by running:

```
python --version
```

If you need to install or upgrade Python, visit the official Python website at <https://www.python.org/downloads/> for the latest version.

In addition to Python 3.7 or higher, having Git installed on your system is essential for using the BioThings CLI, particularly if you need to clone repositories or manage version-controlled code.

To check if Git is installed on your system, run:

```
git --version
```

If Git is not installed, you can download and install it from the official Git website:

- For Windows and macOS: Visit [Git's official download page](#).
- For Linux: Use your distribution's package manager (e.g., *apt-get install git* for Ubuntu, *yum install git* for Fedora).

After installing Git, you can proceed with the setup and usage of the BioThings CLI.

6.2.3 Setting Up

Clone the tutorials repository on our BioThings group.

```
git clone https://github.com/biotothings/tutorials.git
cd tutorials
git checkout pharmgkb_v5
```

Now we will need to install the requirements to run our BioThings CLI. We will first create a virtual environment and then install a BioThings Hub CLI environment.

```
python -m venv .venv
source ./venv/bin/activate
pip install "biothings[cli]"
```

6.2.4 Run/Test a data plugin

Let's check out our command line inputs. Here is a quick summary of every command we will be using in this tutorial.

- `biothings-cli datapluging dump`: Download source data files to local
- `biothings-cli datapluging list`: Listing dumped files or uploaded sources
- `biothings-cli datapluging upload`: Convert downloaded data from dump step into JSON documents and upload them to the source database
- `biothings-cli datapluging serve`: *serve* command runs a simple API server for serving documents from the source database.
- `biothings-cli datapluging clean`: Delete all dumped files and drop uploaded sources tables

If you have any further questions on what other options are available in our `biothings-cli`. You can check out more using the `--help` or `-h` flag on any attribute. Examples:

- `biothings-cli --help`
- `biothings-cli datapluging --help`
- `biothings-cli datapluging dump -h`

The BioThings CLI can only be used for a manifest based plugin. Looking at our manifest file, we are using a JSON based manifest with multiple uploaders. Check out our [manifest section](#) to know more about the different types of manifest files that can be used with our Hub.

```
{
    "version": "0.3",
    "requires": [
        "pandas",
        "numpy"
    ],
    "dumper": {
        "data_url": [
            "https://s3.pgkb.org/data/annotations.zip",
            "https://s3.pgkb.org/data/drugLabels.zip",
            "https://s3.pgkb.org/data/occurrences.zip"
        ],
        "uncompress": true
    },
    "uploaders": [
        {
            "name": "annotations",
            "parser": "parser:load_annotations",
            "mapping": "parser:custom_annotations_mapping",
            "on_duplicates": "error"
        },
        {
            "name": "druglabels",
            "parser": "parser:load_drug_labels",
            "mapping": "parser:custom_drug_labels_mapping",
            "on_duplicates": "error"
        }
    ]
}
```

(continues on next page)

(continued from previous page)

```

        "parser": "parser:load_druglabels",
        "on_duplicates": "error"
    },
    {
        "name": "occurrences",
        "parser": "parser:load_occurrences",
        "on_duplicates": "error"
    }
]
}

```

- **version** specifies the manifest version (it's not the version of the datasource itself) and tells the CLI what to expect from the manifest.
- **parser** uses pandas and numpy library, we declare that dependency in **requires** section.
- the **dumper** section declares where the input files are, using **data_url** key. In the end, we'll use 3 different files so a list of URLs is specified there. A single string is also allowed if only one file (ie. one URL) is required. Since the input file is a ZIP file, we first need to uncompress the archive, using **uncompress** : true. We will see the uncompressed contents shortly after dumping.
- the **uploaders** section tells the CLI how to upload JSON documents to local SQLite database. **parser** has a special format, **module_name:function_name**. For example the first parsing function is named **load_annotations** and can be found in *parser.py* module. "on_duplicates" : "error" tells the CLI to raise an error if we have documents with the same **_id** (this would mean we have a bug in our parser).

Now we will run the dump process using the **dump** command:

```
biothings-cli dataplugin dump
```

```

● (.venv) (base) jalin@MacBook-Pro tutorials % biothings-cli dataplugin dump
[16:58:58] INFO  Install requirement 'pandas'
Requirement already satisfied: pandas in ./venv/lib/python3.11/site-packages (2.1.4)
Requirement already satisfied: numpy>=1.23.2
Requirement already satisfied: python-dateutil> Focus folder in explorer (cmd + click) :kages (from pandas) (1.26.3)
Requirement already satisfied: python-dateutil> te-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in ./venv/lib/python3.11/site-packages (from pandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in ./venv/lib/python3.11/site-packages (from pandas) (2023.4)
Requirement already satisfied: six>=1.5 in ./venv/lib/python3.11/site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
[16:58:59] INFO  Install requirement 'numpy'
Requirement already satisfied: numpy in ./venv/lib/python3.11/site-packages (1.26.3)
[16:59:02] INFO  Downloading remote data from "https://s3.pkb.org/data/annotations.zip"...
INFO  Downloaded locally as ".biothings_hub/archive/2024-01-05/annotations.zip"
INFO  Downloading remote data from "https://s3.pkb.org/data/drugLabels.zip"...
INFO  Downloaded locally as ".biothings_hub/archive/2024-01-05/drugLabels.zip"
INFO  Downloading remote data from "https://s3.pkb.org/data/occurrences.zip"...
INFO  Downloaded locally as ".biothings_hub/archive/2024-01-05/occurrences.zip"
INFO  Uncompress all archive files in '.biothings_hub/archive/2024-01-05'
INFO  unzipping '.biothings_hub/archive/2024-01-05/occurrences.zip'
INFO  done unzipping '.biothings_hub/archive/2024-01-05/occurrences.zip'
INFO  unzipping '.biothings_hub/archive/2024-01-05/drugLabels.zip'
INFO  done unzipping '.biothings_hub/archive/2024-01-05/drugLabels.zip'
INFO  unzipping '.biothings_hub/archive/2024-01-05/annotations.zip'
INFO  done unzipping '.biothings_hub/archive/2024-01-05/annotations.zip'
[16:59:03] INFO  Success! ✨

Dump
Source: tutorials
Data Folder: .biothings_hub/archive/2024-01-05:
- var_fa_ann.tsv
- study_parameters.tsv
- occurrences.zip
- var_pheno_ann.tsv
- drugLabels.tsv
- VERSIONS.txt
- var_drug_ann.tsv
- CREATED_2021-03-05.txt
- clinical_ann_metadata.tsv
- clinical_ann_history.tsv
- drugLabels.zip
- clinical_ann.tsv
- CLINICAL_ANNOTATIONS_README.pdf
- drugLabels.byGene.tsv
- CREATED_2024-01-05.txt
- VARIANT_ANNOTATIONS_README.pdf
- LICENSE.txt
- occurrences.tsv
- annotations.zip
- README.pdf

```

There should be a successful dump along with the dump contents in the `.biothings_hub/archive/<DATE_TIME>` directory.

Note: Remember since we set `uncompress` as `true` in the manifest the `.biothings_hub/archive/<DATE_TIME>` will contain both the zip files and the uncompressed contents.

In our `.biothings_hub` directory, there should be a SQLite database that was created called `biothings_hubdb`. Let's take a look at the contents using `biothings-cli datapluging list --hubdb`.

```
Hubdb
Collection: data_plugin
[{'_id': 'tutorials',
 'download': {'data_folder': '.'},
 'plugin': {'active': True,
            'loader': 'manifest',
            'type': 'local',
            'url': 'local://tutorials'}}]
Collection: src_dump
[{'_id': 'tutorials',
 'download': {'data_folder': '.biothings_hub/archive/2024-01-05',
              'last_success': '2024-01-19T16:59:03.015821-08:00',
              'logfile': None,
              'release': '2024-01-05',
              'started_at': '2024-01-19T16:59:03.015821-08:00',
              'status': 'success',
              'time': '1.47s'}}]
```

We can see two collections/tables that have been created during our dump.

The `data_plugin` collection contains the information of our “**tutorial**” dataplugin. The each entry within the `data_plugin` contains:

- `_id`: name of the plugin
- `download.data_folder`: where the plugin is located
- `plugin.active`: if the plugin is still being used
- `plugin.loader`: type of plugin, at the moment, we can only using manifest type plugins for the cli, but more features will be updated in the future to include other types
- `plugin.type`: local vs remote repository
- `plugin.url`: plugin source folder

The `src_dump` collection contains the information of our dumps:

- `_id`: name of the dataplugin
- `download.data_folder`: location of the dumped contents
- `download.last_success`: datetime of last successful dump
- `download.logfile`: location of generated log files
- `download.release`: name of release
- `download.started_at`: datetime of when the dump was started
- `download.status`: status of the dump
- `download.time`: how long the dump process took

Now that our dumper has been populated, we can continue to the upload process. Let's take a look at the upload command.

```
(.venv) (base) jalin@MacBook-Pro tutorials % biothings-cli datapluging upload -h  
Usage: biothings-cli datapluging upload [OPTIONS]  
Convert downloaded data from dump step into JSON documents and upload them to the source database  
Options  
--batch-limit INTEGER The maximum number of batches that should be uploaded. Batch size is 1000 docs [default: None]  
--help -h Show this message and exit.
```

Since our data is small, we do not need to use the --batch-limit tag for testing. Instead, we can directly run:

```
biothings-cli datapluging upload
```

```
INFO Renaming collection occurrences_temp_ZEJZ0M1Z to occurrences  
INFO Success! 🎉  
Upload  
Source: tutorials  
DB path: /Users/jalin/Desktop/Scripps_Work/tutorials/.biothings_hub/.data_src_database  
- Database: .data_src_database  
  - Collections:  
    annotations  
    druglabels  
    occurrences  
  - Archived collections:  
    annotations_archive_20240122_7G11chM1  
    druglabels_archive_20240122_IdSDxGEY  
    occurrences_archive_20240122_xsUsnlls  
  - Temporary collections:
```

After a successful upload, the SQLite database `.biothings_hub/.data_src_database` is created with three different collections. Each collection matches the corresponding uploader in our manifest file: `annotations`, `druglabels`, `occurrences`.

To view our data, we will need to use the `serve` command.

```
biothings-cli datapluging serve
```

```

View all available routes: http://localhost:9999/
  http://localhost:9999/annotations —
    ⚡ Get a document by id:
      http://localhost:9999/annotations/<doc_id>
      Examples:
        http://localhost:9999/annotations/PA25911
        http://localhost:9999/annotations/PA33129
    ⚡ Query documents by fields:
      http://localhost:9999/annotations?q=<query>
      Examples:
        http://localhost:9999/annotations?from=0&size=10
        http://localhost:9999/annotations?q=annotations.significance:yes
        http://localhost:9999/annotations?q=annotations.variant:rs12046844 AND annotations.annotation_id:1184086254

  http://localhost:9999/druglabels —
    ⚡ Get a document by id:
      http://localhost:9999/druglabels/<doc_id>
      Examples:
        http://localhost:9999/druglabels/PA27006
        http://localhost:9999/druglabels/PA134979668
    ⚡ Query documents by fields:
      http://localhost:9999/druglabels?q=<query>
      Examples:
        http://localhost:9999/druglabels?from=0&size=10
        http://localhost:9999/druglabels?q=drug_labels.id:PA166127702
        http://localhost:9999/druglabels?q=drug_labels.id:PA166104845 AND drug_labels.id:PA166104845

  http://localhost:9999/occurrences —
    ⚡ Get a document by id:
      http://localhost:9999/occurrences/<doc_id>
      Examples:
        http://localhost:9999/occurrences/PA27974
        http://localhost:9999/occurrences/PA142670475
    ⚡ Query documents by fields:
      http://localhost:9999/occurrences?q=<query>
      Examples:
        http://localhost:9999/occurrences?from=0&size=10
        http://localhost:9999/occurrences?q=occurrences.object_type:Gene
        http://localhost:9999/occurrences?q=occurrences.object_type:Gene AND occurrences.object_type:Gene

Found collection: annotations; counting...Done (1018 documents)

```

Once we have served the data, there should be 3 endpoints that are created. Go to <http://localhost:9999/> to view all of the available endpoints. For each endpoint we can query by **id**:

- http://localhost:9999/annotations/<DOC_ID>

or field:

- <http://localhost:9999/annotations?q=<QUERY>>

Try out a few of the examples for yourself listed in the serve output!

Note: You may have noticed that we are able to serve *occurrences* and *druglabels* without registering a mapping. The reason is because BioThings CLI does not check for correct mappings. If you want to know if your mapping is correctly registered, you will have to use our [BioThings Studio](#).

To review we can use the `biotools-cli dataplugin list` command. Using this command we can see all of our dump and upload information.

```
● (.venv) (base) jalin@MacBook-Pro tutorials % biothings-cli dataplugin list
[11:31:15] INFO    Install requirement 'pandas'
Requirement already satisfied: pandas in ./venv/lib/python3.11/site-packages (2.1.4)
Requirement already satisfied: numpy<2,>=1.23.2 in ./venv/lib/python3.11/site-packages (from pandas) (1.26.3)
Requirement already satisfied: python-dateutil>=2.8.2 in ./venv/lib/python3.11/site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in ./venv/lib/python3.11/site-packages (from pandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in ./venv/lib/python3.11/site-packages (from pandas) (2023.4)
Requirement already satisfied: six>=1.5 in ./venv/lib/python3.11/site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
[11:31:15] INFO    Install requirement 'numpy'
Requirement already satisfied: numpy in ./venv/lib/python3.11/site-packages (1.26.3)

Dump -----
Source: tutorials
Data Folder: .biothings_hub/archive/2024-01-05:
- var_fa_ann.tsv
- study_parameters.tsv
- occurrences.zip
- var_pheno_ann.tsv
- drugLabels.tsv
- VERSIONS.txt
- var_drug_ann.tsv
- CREATED_2021-03-05.txt
- clinical_ann_metadata.tsv
- clinical_ann_history.tsv
- drugLabels.zip
- clinical_ann.tsv
- CLINICAL_ANNOTATIONS_README.pdf
- drugLabels.byGene.tsv
- CREATED_2024-01-05.txt
- VARIANT_ANNOTATIONS_README.pdf
- LICENSE.txt
- occurrences.tsv
- annotations.zip
- README.pdf

Upload -----
Source: tutorials
DB path: /Users/jalin/Desktop/Scripps_Work/tutorials/.biothings_hub/.data_src_database
- Database: .data_src_database
  - Collections:
    annotations
    druglabels
    occurrences
  - Archived collections:
    annotations_archive_20240122_7GiIcHm1
    druglabels_archive_20240122_IdSDxGEY
    occurrences_archive_20240122_xsUsnl1s
  - Temporary collections:

● (.venv) (base) jalin@MacBook-Pro tutorials % █
```

Once we are finished with our plugin we can delete our unused data with `biothings-cli dataplugin clean --all`. This will delete all the dumped files and drop all the uploaded source data.

```
(.venv) (base) jalin@MacBook-Pro tutorials % biothings-cli dataplug clean --all
[11:36:36] INFO  Install requirement 'pandas'
Requirement already satisfied: pandas in ./venv/lib/python3.11/site-packages (2.1.4)
Requirement already satisfied: numpy<2,>=1.23.2 in ./venv/lib/python3.11/site-packages (from pandas) (1.26.3)
Requirement already satisfied: python-dateutil>=2.8.2 in ./venv/lib/python3.11/site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in ./venv/lib/python3.11/site-packages (from pandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in ./venv/lib/python3.11/site-packages (from pandas) (2023.4)
Requirement already satisfied: six>=1.5 in ./venv/lib/python3.11/site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
[11:36:37] INFO  Install requirement 'numpy'
Requirement already satisfied: numpy in ./venv/lib/python3.11/site-packages (1.26.3)
There are all files dumped by tutorials:
var_fa_ann.tsv
study_parameters.tsv
occurrences.zip
var_pheno_ann.tsv
drugLabels.tsv
VERSIONS.txt
var_drug_ann.tsv
CREATED_2021-03-05.txt
clinical_ann_metadata.tsv
clinical_ann_history.tsv
drugLabels.zip
clinical_ann.tsv
CLINICAL_ANNOTATIONS_README.pdf
drugLabelsByGene.tsv
CREATED_2024-01-05.txt
VARIANT_ANNOTATIONS_README.pdf
LICENSE.txt
occurrences.tsv
annotations.zip
README.pdf
Do you want to delete them? [y/N]: y
Deleted!
There are all sources uploaded by tutorials:
annotations_archive_20240122_7Gi1cHm1
annotations
druglabels_archive_20240122_IdSDxGEY
druglabels
occurrences_archive_20240122_xsUsnl1s
occurrences
Do you want to drop them? [y/N]: y
All collections are dropped!
```

We can check if all the data is deleted using `biothings-cli dataplug list`.

```
(.venv) (base) jalin@MacBook-Pro tutorials % biothings-cli dataplug list
[11:37:20] INFO  Install requirement 'pandas'
Requirement already satisfied: pandas in ./venv/lib/python3.11/site-packages (2.1.4)
Requirement already satisfied: numpy<2,>=1.23.2 in ./venv/lib/python3.11/site-packages (from pandas) (1.26.3)
Requirement already satisfied: python-dateutil>=2.8.2 in ./venv/lib/python3.11/site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in ./venv/lib/python3.11/site-packages (from pandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in ./venv/lib/python3.11/site-packages (from pandas) (2023.4)
Requirement already satisfied: six>=1.5 in ./venv/lib/python3.11/site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
INFO  Install requirement 'numpy'
Requirement already satisfied: numpy in ./venv/lib/python3.11/site-packages (1.26.3)
Requirement already satisfied: pandas in ./venv/lib/python3.11/site-packages (2.1.4)
Requirement already satisfied: numpy<2,>=1.23.2 in ./venv/lib/python3.11/site-packages (from pandas) (1.26.3)
Requirement already satisfied: python-dateutil>=2.8.2 in ./venv/lib/python3.11/site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in ./venv/lib/python3.11/site-packages (from pandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in ./venv/lib/python3.11/site-packages (from pandas) (2023.4)
Requirement already satisfied: six>=1.5 in ./venv/lib/python3.11/site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
INFO  Install requirement 'numpy'
Requirement already satisfied: numpy in ./venv/lib/python3.11/site-packages (1.26.3)

Dump
Source: tutorials
Data Folder: .biothings_hub/archive/2024-01-05:
Empty file!

Upload
Source: tutorials
DB path: /Users/jalin/Desktop/Scripps_Work/tutorials/.biothings_hub/.data_src_database
Database: .data_src_database
Empty source!
```

6.2.5 In Summary

We have successfully set up a BioThings CLI environment and created a test environment from a flat file using only the CLI. Here is what we have achieved:

- Create a data plugin: by defining a data plugin, we pointed the **BioThings CLI** to where the remote data is and which parser functions to process the remote data
- Dump remote data: we used the **BioThings CLI** to dump the data locally
- Parse remote data: we also generated an *uploader* to run the parser and store resulting JSON documents into a SQLite database
- Run the test API: we served the resulting data with a simple API server from the source database.

6.2.6 Next Steps

- Deploy to production:
 - After you successfully created and tested your data plugin locally, you are ready to host your data plugin as a BioThings API in the production environment (e.g. AWS cloud environment).
 - Please contact the Manager of one of our managed BioThings Hubs. The rest of the deployment process will be handled by the managed Hub.
- Managing multiple plugins:

If you need to manage multiple data plugins locally, there are different options to organize them:

- Option 1: Create a new directory for every plugin and use the `biothings-cli dataplug` command to manage one data plugin at a time as we described in the tutorial above.
- Option 2: Create a parent directory and organize multiple data plugins in subdirectories. You can then run the `biothings-cli dataplug-hub` command at the parent directory as a controller to manage all data plugins, with almost identical subcommands (e.g. `dump`, `upload` etc.) described above.
- Option 3: Follow our [BioThings Studio Tutorial](#) to install a full-featured web UI to manage multiple data plugins, which is the same interface we use to manage a BioThings dataplug hub in our production environment.

6.3 BioThings Standalone

This step-by-step guide shows how to use Biothings standalone instances. Standalone instances are based on Docker containers and provide a fully pre-configured, ready-to-use Biothings API that can easily be maintained and kept up-to-date. The idea is, for any user, to be able to run his/her own APIs locally and fulfill different needs:

- keep all API requests private and local to your own server
- enrich existing and publicly available data found on our APIs with some private data
- run API on your own architecture to perform heavy queries that would sometimes be throttled out from online services

6.3.1 Quick Links

If you already know how to run a BioThings standalone instance, you can download the latest available Docker images from the following tables.

Warning: Some data sources, managed and served by standalone instances (production, demo and old), have restricted licenses. Therefore these standalone instances must not be used for other than non-profit purposes. By clicking on the following links, you agree and understand these restrictions. If you're a for-profit company and would like to run a standalone instance, please [contact us](#).

Note: Images don't contain data but are ready to download and maintain data up-to-date running simple commands through the hub.

List of standalone instances



Production and old data require at least 30GiB disk space.

Production	Demo	Old
Contact us	Download	Download



Production and old data require at least 2TiB disk space.

Production	Demo	Old
Contact us	Download	Download



Production and old data require at least 150Gib disk space.

Production	Demo	Old
Contact us	Download	Soon !

6.3.2 Prerequisites

Using standalone instances requires to have a Docker server up and running, some basic knowledge about commands to run and use containers. Images have been tested on Docker >=17. Using AWS cloud, you can use our public AMI **biothings_demo_docker** (ami-44865e3c in Oregon region) with Docker pre-configured and ready for standalone demo instances deployment. We recommend using instance type with at least 8GiB RAM, such as t2.large. AMI comes with an extra 30GiB EBS volume, which should be enough to deploy any demo instances.

Alternately, you can install your own Docker server (on recent Ubuntu systems, `sudo apt-get install docker.io` is usually enough). You may need to point Docker images directory to a specific hard drive to get enough space, using `-g` option:

```
# /mnt/docker points to a hard drive with enough disk space
sudo echo 'DOCKER_OPTS="-g /mnt/docker"' >> /etc/default/docker
# restart to make this change active
sudo service docker restart
```

Demo instances use very little disk space, as only a small subset of data is available. For instance, myvariant demo only requires ~10GiB to run with demo data up-to-date, including the whole Linux system and all other dependencies.

Demo instances provide a quick and easy way to setup a running APIs, without having to deal with some advanced system configurations.

For deployment with production or old data, you may need a large amount of disk space. Refer to the [Quick Links](#) section for more information. Bigger instance types will also be required, and even a full cluster architecture deployment. We'll soon provide guidelines and deployment scripts for this purpose.

6.3.3 What you'll learn

Through this guide, you'll learn:

- how to obtain a Docker image to run your favorite API
- how to run that image inside a Docker container and how to access the web API
- how to connect to the *hub*, a service running inside the container used to interact with the API systems
- how to use that hub, using specific commands, in order to perform update and keep data up-to-date

6.3.4 Data found in standalone instances

All BioThings APIs (mygene.info, myvariant.info, ...) provide data release in different flavors:

- **Production data**, the actual data found on live APIs we, the BioThings team at [SuLab](#), are running and keeping up-to-date on a regular basis. Please contact us if you're interested in obtaining this type of data.
- **Demo data**, a small subset of production data, publicly available
- **Old production data**, an at least one year old production dataset (full), publicly available

The following guide applies to demo data only, though the process would be very similar for other types of data flavors.

6.3.5 Downloading and running a standalone instance

Standalone instances are available as Docker images. For the purpose of this guide, we'll setup an instance running mygene API, containing demo data. Links to standalone demo Docker images, can be found in [Quick links](#) at the beginning of this guide. Use one of these links, or use this [direct link](#) to mygene's demo instance, and download the Docker image file, using your favorite browser or wget:

```
$ wget http://biothings-containers.s3-website-us-west-2.amazonaws.com/demo_mygene/demo_mygene.docker
```

You must have a running Docker server in order to use that image. Typing docker ps should return all running containers, or at least an empty list as in the following example. Depending on the systems and configuration, you may have to add sudo in front of this command to access Docker server.

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND       CREATED          STATUS
<--> STATUS          PORTS              NAMES
```

Once downloaded, the image can be loaded into the server:

```
$ docker image load < demo_mygene.docker
$ docker image list
REPOSITORY          TAG      IMAGE ID            CREATED          SIZE
<--> IMAGE ID        CREATED          SIZE
```

(continues on next page)

(continued from previous page)

demo_mygene		latest	
15d6395e780c	6 weeks ago	1.78GB	

Image is now loaded, size is ~1.78GiB, it contains no data (yet). A docker container can now be instantiated from that image, to create a BioThings standalone instance, ready to be used.

A standalone instance is a pre-configured system containing several parts. BioThings hub is the system used to interact with BioThings backend and perform operations such as downloading data and create/update ElasticSearch indices. Those indices are used by the actual BioThings web API system to serve data to end-users. The hub can be accessed through a standard SSH connection or through REST API calls. In this guide, we'll use the SSH server.

A BioThings instance expose several services on different ports:

- **80**: BioThings web API port
- **7022**: BioThings hub SSH port
- **7080**: BioThings hub REST API port
- **9200**: ElasticSearch port

We will map and expose those ports to the host server using option `-p` so we can access BioThings services without having to enter the container (eg. hub ssh port here will accessible using port 19022).

```
$ docker run --name demo_mygene -p 19080:80 -p 19200:9200 -p 19022:7022 -p 19090:7080 -d

demo_mygene
```

Note: Instance will store ElasticSearch data in `/var/lib/elasticsearch/` directory, and downloaded data and logs in `/data/` directory. Those two locations could require extra disk space, if needed Docker option `-v` can be used to mount a directory from the host, inside the container. Please refer to Docker documentation.

Let's enter the container to check everything is running fine. Services may take a while, up to 1 min, before fully started. If some services are missing, the troubleshooting section may help.

```
$ docker exec -ti demo_mygene /bin/bash

root@a6a6812e2969:/tmp# netstat -tnlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address          Foreign Address        State      PID/

Program name
tcp      0      0 0.0.0.0:7080            0.0.0.0:*          LISTEN     -
tcp      0      0 0.0.0.0:7022            0.0.0.0:*          LISTEN     -
tcp      0      0 0.0.0.0:80             0.0.0.0:*          LISTEN     25/nginx
tcp      0      0 127.0.0.1:8881          0.0.0.0:*          LISTEN     -
tcp      0      0 127.0.0.1:8882          0.0.0.0:*          LISTEN     -
tcp      0      0 127.0.0.1:8883          0.0.0.0:*          LISTEN     -
tcp      0      0 127.0.0.1:8884          0.0.0.0:*          LISTEN     -
tcp      0      0 127.0.0.1:8885          0.0.0.0:*          LISTEN     -
tcp      0      0 127.0.0.1:8886          0.0.0.0:*          LISTEN     -
tcp      0      0 127.0.0.1:8887          0.0.0.0:*          LISTEN     -
tcp      0      0 127.0.0.1:8888          0.0.0.0:*          LISTEN     -
tcp6     0      0 :::7080              :::*                LISTEN     -
tcp6     0      0 :::7022              :::*                LISTEN     -
tcp6     0      0 :::9200              :::*                LISTEN     -
tcp6     0      0 :::9300              :::*                LISTEN     -
```

We can see the different BioThings services' ports: 7080, 7022 and 7080. All 888x ports correspond to Tornado instances running behing Nginx port 80. They shouldn't be accessed directly. Ports 9200 and 9300 are ElasticSearch standard ports (9200 one can be used to perform queries directly on ES, if needed)

At this point, the standalone instance is up and running. No data has been downloaded yet, let's see how to populate the BioThings API using the hub.

6.3.6 Updating data using Biothings hub

If the standalone instance has been freshly started, there's no data to be queried by the API. If we make a API call, such as fetching metadata, we'll get an error:

```
# from Docker host
$ curl -v http://localhost:19080/metadata
*   Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 19080 (#0)
> GET /metadata HTTP/1.1
> Host: localhost:19080
> User-Agent: curl/7.47.0
> Accept: */*
>
< HTTP/1.1 500 Internal Server Error
< Date: Tue, 28 Nov 2017 18:19:23 GMT
< Content-Type: text/html; charset=UTF-8
< Content-Length: 93
< Connection: keep-alive
< Server: TornadoServer/4.5.2
<
* Connection #0 to host localhost left intact
```

This 500 error reflects a missing index (ElasticSearch index, the backend used by BioThings web API). We can have a look at existing indices in ElasticSearch:

```
# from Docker host
$ curl http://localhost:19200/_cat/indices
yellow open hubdb 5 1 0 0 795b 795b
```

There's only one index, hubdb, which is an internal index used by the hub. No index containing actual biological data...

BioThings hub is a service running inside the instance, it can be accessed through a SSH connection, or using REST API calls. For the purpose of the guide, we'll use SSH. Let's connect to the hub (type yes to accept the key on first connection):

```
# from Docker host
$ ssh guest@localhost -p 19022
The authenticity of host '[localhost]:19022 ([127.0.0.1]:19022)' can't be established.
RSA key fingerprint is SHA256:j63IEgXc3yJqgv0F4wa35aGliH5YQux84xxABew5AS0.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[localhost]:19022' (RSA) to the list of known hosts.

Welcome to Auto-hub, guest!
hub>
```

We're now connected to the hub, inside a python shell where the application is actually running. Let's see what commands are available:

Warning: the hub console, though accessed through SSH, is **not** a Linux shell (such as *bash*), it's a python interpreter shell.

```
hub> help()
```

Available commands:

```
    versions
    check
    info
    download
    apply
    step_update
    update
    help
```

Type: 'help(command)' for more

- `versions()` will display all available data build versions we can download to populate the API
- `check()` will return whether a more recent version is available online
- `info()` will display current local API version, and information about the latest available online
- `download()` will download the data compatible with current local version (but without populating the Elastic-Search index)
- `apply()` will use local data previously downloaded to populate the index
- `step_update()` will bring data release to the next one (one step in versions), compatible with current local version
- `update()` will bring data to the latest available online (using a combination of `download` and `apply` calls)

Note: `update()` is the fastest, easiest and preferred way to update the API. `download`, `apply`, `step_update` are available when it's necessary to bring the API data to a specific version (not the latest one), are considered more advanced, and won't be covered in this guide.

Note: Because the hub console is actually a python interpreter, we call the commands using parenthesis, just like functions or methods. We can also pass arguments when necessary, just like standard python (remember: it **is** python...)

Note: After each command is typed, we need to press "enter" to get either its status (still running) or the result

Let's explore some more.

```
hub> info()
[2] RUN {0.0s} info()
hub>
```

(continues on next page)

(continued from previous page)

```
[2] OK info(): finished
>>> Current local version: 'None'
>>> Release note for remote version 'latest':
Build version: '20171126'

=====
Previous build version: '20171119'
Generated on: 2017-11-26 at 03:11:51

+-----+-----+-----+-----+
| Updated datasource | prev. release | new release | prev. # of docs / new # of docs |
+-----+-----+-----+-----+
| entrez.entrez_gene | 20171118 | 20171125 | 10,003 | 10,003 |
| entrez.entrez_refseq | 20171118 | 20171125 | 10,003 | 10,003 |
| entrez.entrez_unigene | 20171118 | 20171125 | 10,003 | 10,003 |
| entrez.entrez_go | 20171118 | 20171125 | 10,003 | 10,003 |
| entrez.entrez_genomic_pos | 20171118 | 20171125 | 10,003 | 10,003 |
| entrez.entrez_retired | 20171118 | 20171125 | 10,003 | 10,003 |
| entrez.entrez_accession | 20171118 | 20171125 | 10,003 | 10,003 |
| generif | 20171118 | 20171125 | 10,003 | 10,003 |
| uniprot | 20171025 | 20171122 | 10,003 | 10,003 |
+-----+-----+-----+-----+
Overall, 9,917 documents in this release
0 document(s) added, 0 document(s) deleted, 130 document(s) updated
```

We can see here we don't have any local data release (Current local version: 'None'), whereas the latest online (at that time) is from November 26th 2017. We can also see the release note with the different changes involved in the release (whether it's a new version, or the number of documents that changed).

```
hub> versions()
[1] RUN {0.0s} versions()
hub>
[1] OK versions(): finished
version=20171003          date=2017-10-05T09:47:59.413191 type=full
version=20171009          date=2017-10-09T14:47:10.800140 type=full
version=20171009.20171015  date=2017-10-19T11:44:47.961731 type=incremental
version=20171015.20171022  date=2017-10-25T13:33:16.154788 type=incremental
version=20171022.20171029  date=2017-11-14T10:34:39.445168 type=incremental
version=20171029.20171105  date=2017-11-06T10:55:08.829598 type=incremental
```

(continues on next page)

(continued from previous page)

<code>version=20171105.20171112</code>	<code>date=2017-11-14T10:35:04.832871 type=incremental</code>
<code>version=20171112.20171119</code>	<code>date=2017-11-20T07:44:47.399302 type=incremental</code>
<code>version=20171119.20171126</code>	<code>date=2017-11-27T10:38:03.593699 type=incremental</code>

Data comes in two distinct types:

- **full**: this is a full data release, corresponding to an ElasticSearch snapshot, containing all the data
- **incremental** : this is a differential/incremental release, produced by computing the differences between two consecutive versions. The diff data is then used to patch an existing, compatible data release to bring it to the next version.

So, in order to obtain the latest version, the hub will first find a compatible version. Since it's currently empty (no data), it will use the first **full** release from 20171009, and then apply **incremental** updates sequentially (20171009.20171015, then 20171015.20171022, then 20171022.20171029, etc... up to 20171119.20171126).

Let's update the API:

```
hub> update()
[3] RUN {0.0s} update()
hub>
[3] RUN {1.3s} update()
hub>
[3] RUN {2.07s} update()
```

After a while, the API is up-to-date, we can run command `info()` again (it also can be used to track update progress):

```
hub> info()
[4] RUN {0.0s} info()
hub>
[4] OK info(): finished
>>> Current local version: '20171126'
>>> Release note for remote version 'latest':
Build version: '20171126'

=====
Previous build version: '20171119'
Generated on: 2017-11-26 at 03:11:51

+-----+-----+-----+
| Updated datasource | prev. release | new release | prev. # of docs / new # of_
| docs / |
+-----+-----+-----+
| entrez.entrez_gene | 20171118 | 20171125 | 10,003 | 10,
| 003 |
| entrez.entrez_refseq | 20171118 | 20171125 | 10,003 | 10,
| 003 |
| entrez.entrez_unigene | 20171118 | 20171125 | 10,003 | 10,
| 003 |
| entrez.entrez_go | 20171118 | 20171125 | 10,003 | 10,
| 003 |
| entrez.entrez_genomic_pos | 20171118 | 20171125 | 10,003 | 10,
| 003 |
```

(continues on next page)

(continued from previous page)

entrez.entrez_retired	20171118	20171125	10,003	10,
003				
entrez.entrez_accession	20171118	20171125	10,003	10,
003				
generif	20171118	20171125	10,003	10,
003				
uniprot	20171025	20171122	10,003	10,
003				
+-----+-----+-----+-----+				
-----+				

Overall, 9,917 documents in this release
0 document(s) added, 0 document(s) deleted, 130 document(s) updated

Local version is 20171126, remote is 20171126, we're up-to-date. We can also use check():

```
hub> check()
[5] RUN {0.0s} check()
hub>
[5] OK check(): finished
Nothing to dump
```

Nothing to dump means there's no available remote version that can be downloaded. It would otherwise return a version number, meaning we would be able to update the API again using command update().

Press Control-D to exit from the hub console.

Querying ElasticSearch, we can see a new index, named biothings_current, has been created and populated:

```
$ curl http://localhost:19200/_cat/indices
green  open biothings_current 1 0 14903 0 10.3mb 10.3mb
yellow open hubdb           5 1      2 0 11.8kb 11.8kb
```

We now have a populated API we can query:

```
# from Docker host
# get metadata (note the build_version field)
$ curl http://localhost:19080/metadata
{
  "app_revision": "672d55f2deab4c7c0e9b7249d22ccca58340a884",
  "available_fields": "http://mygene.info/metadata/fields",
  "build_date": "2017-11-26T02:58:49.156184",
  "build_version": "20171126",
  "genome_assembly": {
    "rat": "rn4",
    "nematode": "ce10",
    "fruitfly": "dm3",
    "pig": "susScr2",
    "mouse": "mm10",
    "zebrafish": "zv9",
    "frog": "xenTro3",
    "human": "hg38"
  },
}
```

(continues on next page)

(continued from previous page)

```
# annotation endpoint
$ curl http://localhost:19080/v3/gene/1017?fields=alias,ec
{
  "_id": "1017",
  "_score": 9.268311,
  "alias": [
    "CDKN2",
    "p33(CDK2)"
  ],
  "ec": "2.7.11.22",
  "name": "cyclin dependent kinase 2"
}

# query endpoint
$ curl http://localhost:19080/v3/query?q=cdk2
{
  "max_score": 310.69254,
  "took": 37,
  "total": 10,
  "hits": [
    {
      "_id": "1017",
      "_score": 310.69254,
      "entrezgene": 1017,
      "name": "cyclin dependent kinase 2",
      "symbol": "CDK2",
      "taxid": 9606
    },
    {
      "_id": "12566",
      "_score": 260.58084,
      "entrezgene": 12566,
      "name": "cyclin-dependent kinase 2",
      "symbol": "Cdk2",
      "taxid": 10090
    },
    ...
  ]
}
```

6.3.7 BioThings API with multiple indices

Some APIs use more than one ElasticSearch index to run. For instance, myvariant.info uses one index for hg19 assembly, and one index for hg38 assembly. With such APIs, the available commands contain a suffix showing which index (thus, which data release) they relate to. Here's the output of `help()` from myvariant's standalone instance:

```
hub> help()
```

Available commands:

```
versions_hg19
check_hg19
info_hg19
```

(continues on next page)

(continued from previous page)

```
download_hg19
apply_hg19
step_update_hg19
update_hg19
versions_hg38
check_hg38
info_hg38
download_hg38
apply_hg38
step_update_hg38
update_hg38
help
```

For instance, `update()` command is now available as `update_hg19()` and `update_hg38()` depending on the assembly.

6.3.8 Troubleshooting

We test and make sure, as much as we can, that standalone images are up-to-date and hub is properly running for each data release. But things can still go wrong...

First make sure all services are running. Enter the container and type `netstat -tnlp`, you should see services running on ports (see usual running [services](#)). If services running on ports 7080 or 7022 aren't running, it means the hub has not started. If you just started the instance, wait a little more as services may take a while before they're fully started and ready.

If after ~1 min, you still don't see the hub running, log to user `biothings` and check the starting sequence.

Note: Hub is running in a tmux session, under user `biothings`

```
# sudo su - biothings
$ tmux a # recall tmux session

python -m biothings.bin.autohub
(pyenv) biothings@a6a6812e2969:~/mygene.info/src$ python -m biothings.bin.autohub
INFO:root:Hub DB backend: {'module': 'biothings.utils.es', 'host': 'localhost:9200'}
INFO:root:Hub database: hubdb
DEBUG:asyncio:Using selector: EpollSelector
start
```

You should see something looking like this above. If not, you should see the actual error, and depending on the error, you may be able to fix it (not enough disk space, etc...). The hub can be started again using `python -m biothings.bin.autohub` from within the application directory (in our case, `/home/biothings/mygene.info/src/`)

Note: Press Control-B then D to detach the tmux session and let the hub running in background.

Logs are available in `/data/mygene.info/logs/`. You can have a look at:

- `dump_*.log` files for logs about data download
- `upload_*.log` files for logs about index update in general (full/incremental)

- sync_*.log files for logs about incremental update only
- and hub_*.log files for general logs about the hub process

Finally, you can report issues and request for help, by joining Biothings Google Groups (<https://groups.google.com/forum/#!forum/biothings>)

6.4 BioThings Hub

Note: This tutorial uses an old/deprecated version of BioThings SDK. It will be updated very soon.

In this tutorial, we will build the whole process, or “hub”, which produces the data for Taxonomy BioThings API, accessible at t.biothings.io. This API serves information about species, lineage, etc... This “hub” is used to download, maintain up-to-date, process, merge data. At the end of this process, an Elasticsearch index is created containing all the data of interest, ready to be served as an API, using Biothings SDK Web component (covered in another tutorial). Taxonomy Biothings API code is available at <https://github.com/biothings/biothings.species>.

6.4.1 Prerequisites

BioThings SDK uses MongoDB as the “staging” storage backend for JSON objects before they are sent to Elasticsearch for indexing. You must have a working MongoDB instance you can connect to. We’ll also perform some basic commands.

You also have to install the latest stable BioThings SDK release, with pip from PyPI:

```
pip install biothings
```

You can install the latest development version of BioThings SDK directly from our github repository like:

```
pip install git+https://github.com/biothings/biothings.api.git#egg=biothings
```

Alternatively, you can download the source code, or clone the [BioThings SDK repository](#) and run:

```
python setup.py install
```

You may want to use `virtualenv` to isolate your installation.

Finally, BioThings SDK is written in python, so you must know some basics.

6.4.2 Configuration file

Before starting to implement our hub, we first need to define a configuration file. This `default_config.py` <https://github.com/biothings/biothings.api/blob/master/biothings/hub/default_config.py> file contains all the required and optional configuration variables, some **have** to be defined in your own application, other **can** be overridden as needed (see `config_hub.py` <https://github.com/biothings/biothings.species/blob/master/src/config_hub.py> for an example).

Typically we will have to define the following:

- MongoDB connections parameters, `DATA_SRC_*` and `DATA_TARGET_*` parameters. They define connections to two different databases, one will contain individual collections for each datasource (SRC) and the other will contain merged collections (TARGET).

- HUB_DB_BACKEND defines a database connection for hub purpose (application specific data, like sources status, etc...). Default backend type is MongoDB. We will need to provide a valid mongodb:// URI. Other backend types are available, like sqlite3 and ElasticSearch, but since we'll use MongoDB to store and process our data, we'll stick to the default.
- DATA_ARCHIVE_ROOT contains the path of the root folder that will contain all the downloaded and processed data. Other parameters should be self-explanatory and probably don't need to be changed.
- LOG_FOLDER contains the log files produced by the hub

Create a **config.py** and add `from config_common import *` then define all required variables above. **config.py** will look something like this:

```
from config_common import *

DATA_SRC_SERVER = "myhost"
DATA_SRC_PORT = 27017
DATA_SRC_DATABASE = "tutorial_src"
DATA_SRC_SERVER_USERNAME = None
DATA_SRC_SERVER_PASSWORD = None

DATA_TARGET_SERVER = "myhost"
DATA_TARGET_PORT = 27017
DATA_TARGET_DATABASE = "tutorial"
DATA_TARGET_SERVER_USERNAME = None
DATA_TARGET_SERVER_PASSWORD = None

HUB_DB_BACKEND = {
    "module" : "biothings.utils.mongo",
    "uri" : "mongodb://myhost:27017",
}

DATA_ARCHIVE_ROOT = "/tmp/tutorial"
LOG_FOLDER = "/tmp/tutorial/logs"
```

Note: Log folder must be created manually

6.4.3 hub.py

This script represents the main hub executable. Each hub should define it, this is where the different hub commands are going to be defined and where tasks are actually running. It's also from this script that a SSH server will run so we can actually log into the hub and access those registered commands.

Along this tutorial, we will enrich that script. For now, we're just going to define a JobManager, the SSH server and make sure everything is running fine.

```
import asyncio, asyncssh, sys
import concurrent.futures
from functools import partial

import config, biothings
biothings.config_for_app(config)

from biothings.utils.manager import JobManager
```

(continues on next page)

(continued from previous page)

```

loop = asyncio.get_event_loop()
process_queue = concurrent.futures.ProcessPoolExecutor(max_workers=2)
thread_queue = concurrent.futures.ThreadPoolExecutor()
loop.set_default_executor(process_queue)
jmanager = JobManager(loop,
                      process_queue, thread_queue,
                      max_memory_usage=None,
                      )

```

jmanager is our JobManager, it's going to be used everywhere in the hub, each time a parallelized job is created. Species hub is a small one, there's no need for many process workers, two should be fine.

Next, let's define some basic commands for our new hub:

```

from biothings.utils.hub import schedule, top, pending, done
COMMANDS = {
    "sch" : partial(schedule,loop),
    "top" : partial(top,process_queue,thread_queue),
    "pending" : pending,
    "done" : done,
}

```

These commands are then registered in the SSH server, which is linked to a python interpreter. Commands will be part of the interpreter's namespace and be available from a SSH connection.

```

passwords = {
    'guest': '', # guest account with no password
}

from biothings.utils.hub import start_server
server = start_server(loop, "Taxonomy hub",passwords=passwords,port=7022,
                      commands=COMMANDS)

try:
    loop.run_until_complete(server)
except (OSErr, asyncssh.Error) as exc:
    sys.exit('Error starting server: ' + str(exc))

loop.run_forever()

```

Let's try to run that script ! The first run, it will complain about some missing SSH key:

```

AssertionError: Missing key 'bin/ssh_host_key' (use: 'ssh-keygen -f bin/ssh_host_key' to
generate it)

```

Let's generate it, following instruction. Now we can run it again and try to connect:

```

$ ssh guest@localhost -p 7022
The authenticity of host '[localhost]:7022 ([127.0.0.1]:7022)' can't be established.
RSA key fingerprint is SHA256:USgdr9nlFVryr475+kQWllyPxwzIUREcn0CyctU1y1Q.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[localhost]:7022' (RSA) to the list of known hosts.

```

(continues on next page)

(continued from previous page)

```
Welcome to Taxonomy hub, guest!  
hub>
```

Let's try a command:

```
hub> top()  
0 running job(s)  
0 pending job(s), type 'top(pending)' for more
```

Nothing fancy here, we don't have much in our hub yet, but everything is running fine.

6.4.4 Dumpers

BioThings species API gathers data from different datasources. We will need to define different dumpers to make this data available locally for further processing.

Taxonomy dumper

This dumper will download taxonomy data from NCBI FTP server. There's one file to download, available at this location: <ftp://ftp.ncbi.nih.gov/pub/taxonomy/taxdump.tar.gz>.

When defining a dumper, we'll need to choose a base class to derive our dumper class from. There are different base dumper classes available in BioThings SDK, depending on the protocol we want to use to download data. In this case, we'll derive our class from `biothings.hub.dataloader.dumper.FTPDumper`. In addition to defining some specific class attributes, we will need to implement a method called `create_todump_list()`. This method fills `self.to_dump` list, which is later going to be used to download data. One element in that list is a dictionary with the following structure:

```
{"remote": "<path to file on remote server>", "local": "<local path to file>"}
```

Remote information are relative to the working directory specified as class attribute. Local information is an absolute path, containing filename used to save data.

Let's start coding. We'll save that python module in `dataloader/sources/taxonomy/dumper.py`.

```
import biothings, config  
biothings.config_for_app(config)
```

Those lines are used to configure BioThings SDK according to our own configuration information.

```
from config import DATA_ARCHIVE_ROOT  
from biothings.hub.dataloader.dumper import FTPDumper
```

We then import a configuration constant, and the `FTPDumper` base class.

```
class TaxonomyDumper(FTPDumper):  
  
    SRC_NAME = "taxonomy"  
    SRC_ROOT_FOLDER = os.path.join(DATA_ARCHIVE_ROOT, SRC_NAME)  
    FTP_HOST = 'ftp.ncbi.nih.gov'  
    CWD_DIR = '/pub/taxonomy'  
    SUFFIX_ATTR = "timestamp"  
    SCHEDULE = "0 9 * * *"
```

- SRC_NAME will used as the registered name for this datasource (more on this later).
- SRC_ROOT_FOLDER is the folder path for this resource, without any version information (dumper will create different sub-folders for each version).
- FTP_HOST and CWD_DIR gives information to connect to the remove FTP server and move to appropriate remote directory (FTP_USER and FTP_PASSWD constants can also be used for authentication).
- SUFFIX_ATTR defines the attributes that's going to be used to create folder for each downloaded version. It's basically either "release" or "timestamp", depending on whether the resource we're trying to dump has an actual version. Here, for taxdump file, there's no version, so we're going to use "timestamp". This attribute is automatically set to current date, so folders will look like that: .../taxonomy/20170120, .../taxonomy/20170121, etc...
- Finally SCHEDULE, if defined, will allow that dumper to regularly run within the hub. This is a cron-like notation (see aiocron documentation for more).

We now need to tell the dumper what to download, that is, create that self.to_dump list:

```
def create_todump_list(self, force=False):
    file_to_dump = "taxdump.tar.gz"
    new_localfile = os.path.join(self.new_data_folder, file_to_dump)
    try:
        current_localfile = os.path.join(self.current_data_folder, file_to_dump)
    except TypeError:
        # current data folder doesn't even exist
        current_localfile = new_localfile
    if force or not os.path.exists(current_localfile) or self.remote_is_better(file_to_
dump, current_localfile):
        # register new release (will be stored in backend)
        self.to_dump.append({"remote": file_to_dump, "local":new_localfile})
```

That method tries to get the latest downloaded file and then compare that file with the remote file using `self.remote_is_better(file_to_dump, current_localfile)`, which compares the dates and returns True if the remote is more recent. A dict is then created with required elements and appended to `self.to_dump` list.

When the dump is running, each element from that `self.to_dump` list will be submitted to a job and be downloaded in parallel. Let's try our new dumper. We need to update `hub.py` script to add a DumperManager and then register this dumper:

In `hub.py`:

```
import dataload
import biothings.hub.dataload.dumper as dumper

dmanager = dumper.DumperManager(job_manager=jmanager)
dmanager.register_sources(dataload.__sources__)
dmanager.schedule_all()
```

Let's also register new commands in the hub:

```
COMMANDS = {
    # dump commands
    "dm" : dmanager,
    "dump" : dmanager.dump_src,
    ...}
```

`dm` will a shortcut for the dumper manager object, and `dump` will actually call manager's `dump_src()` method.

Manager is auto-registering dumpers from list defines in dataload package. Let's define that list:

In `dataload/__init__.py`:

```
__sources__ = [
    "dataload.sources.taxonomy",
]
```

That's it, it's just a string pointing to our taxonomy package. We'll expose our dumper class in that package so the manager can inspect it and find our dumper (note: we could use give the full path to our dumper module, `dataload.sources.taxonomy.dumper`, but we'll add uploaders later, it's better to have one single line per resource).

In `dataload/sources/taxonomy/__init__.py`

```
from .dumper import TaxonomyDumper
```

Let's run the hub again. We can see on the logs that our dumper has been found:

```
Found a class based on BaseDumper: '<class \'dataload.sources.taxonomy.dumper.TaxonomyDumper\'>'
```

Also, manager has found scheduling information and created a task for this:

```
Scheduling task functools.partial(<bound method DumperManager.create_and_dump of
<DumperManager [1 registered]: ['taxonomy']>, <class 'dataload.sources.taxonomy.dumper.TaxonomyDumper'>, job_manager=<biotools.utils.manager.JobManager object at 0x7f88fc5346d8>, force=False): 0 9 * * *
```

We can double-check this by connecting to the hub, and type some commands:

```
Welcome to Taxonomy hub, guest!
hub> dm
<DumperManager [1 registered]: ['taxonomy']>
```

When printing the manager, we can check our taxonomy resource has been registered properly.

```
hub> sch()
DumperManager.create_and_dump(<class 'dataload.sources.taxonomy.dumper.TaxonomyDumper'>,
  [0 9 * * *] {run in 00h:39m:09s}
```

Dumper is going to run in 39 minutes ! We can trigger a manual upload too:

```
hub> dump("taxonomy")
[1] RUN {0.0s} dump("taxonomy")
```

OK, dumper is running, we can follow task status from the console. At some point, task will be done:

```
hub>
[1] OK dump("taxonomy"): finished, [None]
```

It successfully ran (OK), nothing was returned by the task ([None]). Logs show some more details:

```
DEBUG:taxonomy.hub:Creating new TaxonomyDumper instance
INFO:taxonomy_dump:1 file(s) to download
DEBUG:taxonomy_dump:Downloading 'taxdump.tar.gz'
INFO:taxonomy_dump:taxdump successfully downloaded
INFO:taxonomy_dump:success
```

Alright, now if we try to run the dumper again, nothing should be downloaded since we got the latest file available. Let's try that, here are the logs:

```
DEBUG:taxonomy.hub:Creating new TaxonomyDumper instance
DEBUG:taxonomy_dump:'taxdump.tar.gz' is up-to-date, no need to download
INFO:taxonomy_dump:Nothing to dump
```

So far so good! The actual file, depending on the configuration settings, it's located in `/data/taxonomy/20170125/taxdump.tar.gz`. We can notice the timestamp used to create the folder. Let's also have a look at in the internal database to see the resource status. Connect to MongoDB:

```
> use hub_config
switched to db hub_config
> db.src_dump.find()
{
    "_id" : "taxonomy",
    "release" : "20170125",
    "data_folder" : "./data/taxonomy/20170125",
    "pending_to_upload" : true,
    "download" : {
        "logfile" : "./data/taxonomy/taxonomy_20170125_dump.log",
        "time" : "4.52s",
        "status" : "success",
        "started_at" : ISODate("2017-01-25T08:32:28.448Z")
    }
}
>
```

We have some information about the download process, how long it took to download files, etc... We have the path to the `data_folder` containing the latest version, the `release` number (here, it's a timestamp), and a flag named `pending_to_upload`. That will be used later to automatically trigger an upload after a dumper has run.

So the actual file is currently compressed, we need to uncompress it before going further. We can add a post-dump step to our dumper. There are two options there, by overriding one of those methods:

```
def post_download(self, remotefile, localfile): triggered for each downloaded file
def post_dump(self): triggered once all files have been downloaded
```

We could use either, but there's a utility function available in BioThings SDK that uncompress everything in a directory, let's use it in a global post-dump step:

```
from biothings.utils.common import untargzall
...
def post_dump(self):
    untargzall(self.new_data_folder)
```

`self.new_data_folder` is the path to the folder freshly created by the dumper (in our case, `/data/taxonomy/20170125`)

Let's try this in the console (restart the hub to make those changes alive). Because file is up-to-date, dumper will not run. We need to force it:

```
hub> dump("taxonomy",force=True)
```

Or, instead of downloading the file again, we can directly trigger the post-dump step:

```
hub> dump("taxonomy", steps="post")
```

There are 2 steps steps available in a dumper:

1. **dump** : will actually download files
2. **post** : will post-process downloaded files (post_dump)

By default, both run sequentially.

After typing either of these commands, logs will show some information about the uncompressing step:

```
DEBUG:taxonomy.hub:Creating new TaxonomyDumper instance
INFO:taxonomy_dump:success
INFO:root:untargz '/opt/slelong/Documents/Projects/biothings.species/src/data/taxonomy/
˓→20170125/taxdump.tar.gz'
```

Folder contains all uncompressed files, ready to be process by an uploader.

UniProt species dumper

Following guideline from previous taxonomy dumper, we're now implementing a new dumper used to download species list. There's just one file to be downloaded from ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/complete/docs/speclist.txt. Same as before, dumper will inherits FTPDumper base class. File is not compressed, so except this, this dumper will look the same.

Code is available on github for further details: [ee674c55bad849b43c8514fcc6b7139423c70074](#) for the whole commit changes, and [dataload/sources/uniprot/dumper.py](#) for the actual dumper.

Gene information dumper

The last dumper we have to implement will download some gene information from NCBI (ftp://ftp.ncbi.nlm.nih.gov/gene/DATA/gene_info.gz). It's very similar to the first one (we could even have merged them together).

Code is available on github: [d3b3486f71e865235efd673d2f371b53eaa0bc5b](#) for whole changes and [dataload/sources/geneinfo/dumper.py](#) for the dumper.

Additional base dumper classes

The previous examples utilized the *FTPDumper* base dumper class. The list of available base dumper classes include:

- **FTPDumper** - Downloads content from ftp source
- **LastModifiedFTPDumper** - A wrapper over FTPDumper, one URL will give one FTPDumper instance. SRC_URLS containing a list of URLs pointing to files to download, use FTP's MDTM command to check whether files should be downloaded. The release is generated from the last file's MDTM in SRC_URLS, and formatted according to RELEASE_FORMAT.
- **HTTPDumper** - Dumper using HTTP protocol and “requests” library
- **LastModifiedHTTPDumper** - Similar to LastModifiedFTPDumper, but for http
- **WgetDumper** - Fill self.to_dump list with dict(“remote”:remote_path,”local”:local_path) elements. This is the todo list for the dumper
- **FilesystemDumper** - works locally and copy (or move) files to datasource folder
- **DummyDumper** - Does nothing

- **ManualDumper** - Assists user to dump a resource. This will usually expect the files to be downloaded first (sometimes there's no easy way to automate this process). Once downloaded, a call to dump() will make sure everything is fine in terms of files and metadata
- **GoogleDriveDumper** - Dumps files from google drive
- **GitDumper** - Gets data from a git repo. Repo is stored in SRC_ROOT_FOLDER (without versioning) and then versions/releases are fetched in SRC_ROOT_FOLDER/<release>

Additional details on the available base dumper classes can be found at: <https://github.com/biothings/biothings.api/blob/master/biothings/hub/dataloader/dumper.py>

6.4.5 Uploaders

Now that we have local data available, we can process them. We're going to create 3 different uploaders, one for each datasource. Each uploader will load data into MongoDB, into individual/single collections. Those will then be used in the last merging step.

Before going further, we'll first create an UploaderManager instance and register some of its commands in the hub:

```
import biothings.hub.dataloader.uploader as uploader
# will check every 10 seconds for sources to upload
umanager = uploader.UploaderManager(poll_schedule = '* * * * */10', job_
manager=jmanager)
umanager.register_sources(dataloader.__sources__)
umanager.poll()

COMMANDS = {
    ...
    # upload commands
    "um" : umanager,
    "upload" : umanager.upload_src,
    ...
}
```

Running the hub, we'll see the kind of log statements:

```
INFO:taxonomy.hub:Found 2 resources to upload (['species', 'geneinfo'])
INFO:taxonomy.hub:Launch upload for 'species'
ERROR:taxonomy.hub:Resource 'species' needs upload but is not registered in manager
INFO:taxonomy.hub:Launch upload for 'geneinfo'
ERROR:taxonomy.hub:Resource 'geneinfo' needs upload but is not registered in manager
...
```

Indeed, datasources have been dumped, and a pending_to_upload flag has been set to True in src_dump. UploadManager polls this src_dump internal collection, looking for this flag. If set, it runs automatically the corresponding uploader(s). Since we didn't implement any uploaders yet, manager complains... Let's fix that.

Taxonomy uploader

The taxonomy files we downloaded need to be parsed and stored into a MongoDB collection. We won't go in too much details regarding the actual parsing, there are two parsers, one for **nodes.dmp** and another for **names.dmp** files. They yield dictionaries as the result of this parsing step. We just need to "connect" those parsers to uploaders.

Following the same approach as for dumpers, we're going to implement our first uploaders by inheriting one the base classes available in BioThings SDK. We have two files to parse, data will stored in two different MongoDB collections, so we're going to have two uploaders. Each inherits from `biothings.hub.dataload.uploader.BaseSourceUploader`, `load_data` method has to be implemented, this is where we "connect" parsers.

Beside this method, another important point relates to the storage engine. `load_data` will, through the parser, yield documents (dictionaries). This data is processed internally by the base uploader class (`BaseSourceUploader`) using a storage engine. `BaseSourceUploader` uses `biothings.hub.dataload.storage.BasicStorage` as its engine. This storage inserts data in MongoDB collection using bulk operations for better performances. There are other storages available, depending on how data should be inserted (eg. `IgnoreDuplicatedStorage` will ignore any duplicated data error). While choosing a base uploader class, we need to consider which storage class it's actually using behind-the-scene (an alternative way to do this is using `BaseSourceUploader` and set the class attribute `storage_class`, such as in this uploader: `biothings/dataload/uploader.py#L447`).

The first uploader will take care of `nodes.dmp` parsing and storage.

```
import biothings.hub.dataload.uploader as uploader
from .parser import parse_refseq_names, parse_refseq_nodes

class TaxonomyNodesUploader(uploader.BaseSourceUploader):

    main_source = "taxonomy"
    name = "nodes"

    def load_data(self,data_folder):
        nodes_file = os.path.join(data_folder,"nodes.dmp")
        self.logger.info("Load data from file '%s'" % nodes_file)
        return parse_refseq_nodes(open(nodes_file))
```

- `TaxonomyNodesUploader` derives from `BaseSourceUploader`
- `name` gives the name of the collection used to store the data. If `main_source` is *not* defined, it must match `SRC_NAME` in dumper's attributes
- `main_source` is optional and allows to define main sources and sub-sources. Since we have 2 parsers here, we're going to have 2 collections created. For this one, we want the collection named "nodes". But this parser relates to *taxonomy* datasource, so we define a `main_source` called **taxonomy**, which matches `SRC_NAME` in dumper's attributes.
- `load_data()` has `data_folder` as parameter. It will be set accordingly, to the path of the last version dumped. Also, that method gets data from parsing function `parse_refseq_nodes`. It's where we "connect" the parser. We just need to return parser's result so the storage can actually store the data.

The other parser, for `names.dmp`, is almost the same:

```
class TaxonomyNamesUploader(uploader.BaseSourceUploader):

    main_source = "taxonomy"
    name = "names"

    def load_data(self,data_folder):
```

(continues on next page)

(continued from previous page)

```
names_file = os.path.join(data_folder, "names.dmp")
self.logger.info("Load data from file '%s'" % names_file)
return parse_refseq_names(open(names_file))
```

We then need to “expose” those parsers in taxonomy package, in `dataload/sources/taxonomy/__init__.py`:

```
from .uploader import TaxonomyNodesUploader, TaxonomyNamesUploader
```

Now let's try to run the hub again. We should see uploader manager has automatically triggered some uploads:

```
INFO:taxonomy.hub:Launch upload for 'taxonomy'
...
INFO:taxonomy.names_upload:Uploading 'names' (collection: names)
INFO:taxonomy.nodes_upload:Uploading 'nodes' (collection: nodes)
INFO:taxonomy.nodes_upload:Load data from file './data/taxonomy/20170125/nodes.dmp'
INFO:taxonomy.names_upload:Load data from file './data/taxonomy/20170125/names.dmp'
INFO:root:Uploading to the DB...
INFO:root:Uploading to the DB...
```

While running, we can check what jobs are running, using `top()` command:

```
hub> top()
      PID |           SOURCE |          CATEGORY |          STEP | 
  ↵DESCRIPTION |     MEM |     CPU | STARTED_AT | DURATION | 
  ↵5795 | taxonomy.nodes | uploader | update_data | 
  ↵         | 49.7MiB | 0.0% | 2017/01/25 14:58:40 | 15.49s | 
  ↵5796 | taxonomy.names | uploader | update_data | 
  ↵         | 54.6MiB | 0.0% | 2017/01/25 14:58:40 | 15.49s | 
2 running job(s)
0 pending job(s), type 'top(pending)' for more
16 finished job(s), type 'top(done)' for more
```

We can see two uploaders running at the same time, one for each file. `top(done)` can also display jobs that are done and finally `top(pending)` can give an overview of jobs that are going to be launched when a worker is available (it happens when there are more jobs created than the available number of workers overtime).

In `src_dump` collection, we can see some more information about the resource and its upload processes. Two jobs were created, we have information about the duration, log files, etc...

```
> db.src_dump.find({_id:"taxonomy"})
{
    "_id" : "taxonomy",
    "download" : {
        "started_at" : ISODate("2017-01-25T13:09:26.423Z"),
        "status" : "success",
        "time" : "3.31s",
        "logfile" : "./data/taxonomy/taxonomy_20170125_dump.log"
    },
    "data_folder" : "./data/taxonomy/20170125",
    "release" : "20170125",
    "upload" : {
        "status" : "success",
    }
}
```

(continues on next page)

(continued from previous page)

```

"jobs" : {
    "names" : {
        "started_at" : ISODate("2017-01-25T14:58:40.034Z"),
        "pid" : 5784,
        "logfile" : "./data/taxonomy/taxonomy.names_20170125_
↪upload.log",
        "step" : "names",
        "temp_collection" : "names_temp_eJUdh1te",
        "status" : "success",
        "time" : "26.61s",
        "count" : 1552809,
        "time_in_s" : 27
    },
    "nodes" : {
        "started_at" : ISODate("2017-01-25T14:58:40.043Z"),
        "pid" : 5784,
        "logfile" : "./data/taxonomy/taxonomy.nodes_20170125_
↪upload.log",
        "step" : "nodes",
        "temp_collection" : "nodes_temp_T5VnzRQC",
        "status" : "success",
        "time" : "22.4s",
        "time_in_s" : 22,
        "count" : 1552809
    }
}
}
}

```

In the end, two collections were created, containing parsed data:

```

> db.names.count()
1552809
> db.nodes.count()
1552809

> db.names.find().limit(2)
{
    "_id" : "1",
    "taxid" : 1,
    "other_names" : [
        "all"
    ],
    "scientific_name" : "root"
}
{
    "_id" : "2",
    "other_names" : [
        "bacteria",
        "not bacteria haeckel 1894"
    ],
    "genbank_common_name" : "eubacteria",
}

```

(continues on next page)

(continued from previous page)

```

"in-part" : [
    "monera",
    "procaryotae",
    "prokaryota",
    "prokaryotae",
    "prokaryote",
    "prokaryotes"
],
"taxid" : 2,
"scientific_name" : "bacteria"
}

> db.nodes.find().limit(2)
{ "_id" : "1", "rank" : "no rank", "parent_taxid" : 1, "taxid" : 1 }
{
    "_id" : "2",
    "rank" : "superkingdom",
    "parent_taxid" : 131567,
    "taxid" : 2
}

```

UniProt species uploader

Following the same guideline, we're going to create another uploader for species file.

```

import biothings.hub.dataloader.uploader as uploader
from .parser import parse_uniprot_speclist

class UniprotSpeciesUploader(uploader.BaseSourceUploader):

    name = "uniprot_species"

    def load_data(self,data_folder):
        nodes_file = os.path.join(data_folder,"speclist.txt")
        self.logger.info("Load data from file '%s'" % nodes_file)
        return parse_uniprot_speclist(open(nodes_file))

```

In that case, we need only one uploader, so we just define “name” (no need to define main_source here).

We need to expose that uploader from the package, in `dataloader/sources/uniprot/__init__.py`:

```
from .uploader import UniprotSpeciesUploader
```

Let's run this through the hub. We can use the “upload” command there (though manager should trigger the upload itself):

```
hub> upload("uniprot_species")
[1] RUN {0.0s} upload("uniprot_species")
```

Similar to dumper, there are different steps we can individually call for an uploader:

- **data**: will take care of storing data

- **post**: calls post_update() method, once data has been inserted. Useful to post-process data or create an index for instance
- **master**: will register the source in src_master collection, which is used during the merge step. Uploader method get_mapping() can optionally returns an ElasticSearch mapping, it will be stored in src_master during that step. We'll see more about this later.
- **clean**: will clean temporary collections and other leftovers...

Within the hub, we can specify these steps manually (they're all executed by default).

```
hub> upload("uniprot_species", steps="clean")
```

Or using a list:

```
hub> upload("uniprot_species", steps=["data", "clean"])
```

Gene information uploader

Let's move forward and implement the last uploader. The goal for this uploader is to identify whether, for a taxonomy ID, there are existing/known genes. File contains information about genes, first column is the `taxid`. We want to know all taxonomy IDs present in the file, and the merged document, we want to add key such as `{'has_gene' : True/False}`.

Obviously, we're going to have a lot of duplicates, because for one taxid we can have many genes present in the files. We have options here 1) remove duplicates before inserting data in database, or 2) let the database handle the duplicates (rejecting them). Though we could process data in memory – processed data is rather small in the end –, for demo purpose, we'll go for the second option.

```
import biothings.hub.dataload.uploader as uploader
import biothings.hub.dataload.storage as storage
from .parser import parse_geneinfo_taxid

class GeneInfoUploader(uploader.BaseSourceUploader):

    storage_class = storage.IgnoreDuplicatedStorage

    name = "geneinfo"

    def load_data(self, data_folder):
        gene_file = os.path.join(data_folder, "gene_info")
        self.logger.info("Load data from file '%s'" % gene_file)
        return parse_geneinfo_taxid(open(gene_file))
```

- `storage_class`: this is the most important setting in this case, we want to use a storage that will ignore any duplicated records.
- `parse_geneinfo_taxid`: is the parsing function, yield documents as `{"_id" : "taxid"}`

The rest is closed to what we already encountered. Code is available on github in `dataload/sources/geneinfo/uploader.py`

When running the uploader, logs show statements like these:

```
INFO:taxonomy.hub:Found 1 resources to upload (['geneinfo'])
INFO:taxonomy.hub:Launch upload for 'geneinfo'
INFO:taxonomy.hub:Building task: functools.partial(<bound method UploaderManager.create_
(continues on next page)
```

(continued from previous page)

```

↳ and_load of <UploaderManager [3 registered]: ['geneinfo', 'species', 'taxonomy']>>,
↳ <class 'dataload.sources.gen
einfo.uploader.GeneInfoUploader'>, job_manager=<biotools.utils.manager.JobManager>
↳ object at 0x7fbf5f8c69b0>
INFO:geneinfo_upload:Uploading 'geneinfo' (collection: geneinfo)
INFO:geneinfo_upload:Load data from file './data/geneinfo/20170125/gene_info'
INFO:root:Uploading to the DB...
INFO:root:Inserted 62 records, ignoring 9938 [0.3s]
INFO:root:Inserted 15 records, ignoring 9985 [0.28s]
INFO:root:Inserted 0 records, ignoring 10000 [0.23s]
INFO:root:Inserted 31 records, ignoring 9969 [0.25s]
INFO:root:Inserted 16 records, ignoring 9984 [0.26s]
INFO:root:Inserted 4 records, ignoring 9996 [0.21s]
INFO:root:Inserted 4 records, ignoring 9996 [0.25s]
INFO:root:Inserted 1 records, ignoring 9999 [0.25s]
INFO:root:Inserted 26 records, ignoring 9974 [0.23s]
INFO:root:Inserted 61 records, ignoring 9939 [0.26s]
INFO:root:Inserted 77 records, ignoring 9923 [0.24s]

```

While processing data in batch, some are inserted, others (duplicates) are ignored and discarded. The file is quite big, so the process can be long...

Note: should we want to implement the first option, the parsing function would build a dictionary indexed by taxid and would read the whole, extracting taxid. The whole dict would then be returned, and then processed by storage engine.

So far, we've defined dumpers and uploaders, made them working together through some managers defined in the hub. We're now ready to move the last step: merging data.

6.4.6 Mergers

Merging will be the last step in our hub definition. So far we have data about species, taxonomy and whether a taxonomy ID has known genes in NCBI. In the end, we want to have a collection where documents look like this:

```
{
  _id: "9606",
  authority: ["homo sapiens linnaeus, 1758"],
  common_name: "man",
  genbank_common_name: "human",
  has_gene: true,
  lineage: [9606, 9605, 207598, 9604, 314295, 9526, ...],
  other_names: ["humans"],
  parent_taxid: 9605,
  rank: "species",
  scientific_name: "homo sapiens",
  taxid: 9606,
  uniprot_name: "homo sapiens"
}
```

- `_id`: the taxid, the ID used in all of our individual collection, so the key will be used to collect documents and merge them together (it's actually a requirement, documents are merged using `_id` as the common key).
- `authority`, `common_name`, `genbank_common_name`, `other_names`, `scientific_name` and `taxid` come from `taxonomy.names` collection.

- uniprot_name comes from species collection.
- has_gene is a flag set to true, because taxid 9606 has been found in collection geneinfo.
- parent_taxid and rank come from taxonomy.nodes collection.
- (there can be other fields available, but basically the idea here is to merge all our individual collections...)
- finally, lineage... it's a little tricky as we need to query nodes in order to compute that field from _id and parent_taxid.

A first step would be to merge **names**, **nodes** and **species** collections together. Other keys need some post-merge processing, they will be handled in a second part.

Let's first define a BuilderManager in the hub.

```
import biothings.hub.databuild.builder as builder
bmanager = builder.BuilderManager(poll_schedule='* * * * */10', job_manager=jmanager)
bmanager.configure()
bmanager.poll()

COMMANDS = {
...
    # building/merging
    "bm" : bmanager,
    "merge" : bmanager.merge,
...
}
```

Merging configuration

BuilderManager uses a builder class for merging. While there are many different dumpers and uploaders classes, there's only one merge class (for now). The merging process is defined in a configuration collection named src_build. Usually, we have as many configurations as merged collections, in our case, we'll just define one configuration.

When running the hub with a builder manager registered, manager will automatically create this src_build collection and create configuration placeholder.

```
> db.src_build.find()
{
    "_id" : "placeholder",
    "name" : "placeholder",
    "sources" : [ ],
    "root" : [ ]
}
```

We're going to use that template to create our own configuration:

- **_id** and name are the name of the configuration (they can be different but really, _id is the one used here)... We'll set these as: `{"_id": "mytaxonomy", "name": "mytaxonomy" }`.
- **sources** is a list of collection names used for the merge. A element in this can also be a regular expression matching collection names. If we have data spread across different collection, like one collection per chromosome data, we could use a regex such as `data_chr.*`. We'll set this as: `{"sources": ["names", "species", "nodes", "geneinfo"]}`
- **root** defines root datasources, that is, datasources that can be used to initiate document creation. Sometimes, we want data to be merged only if an existing document previously exists in the merged collection. If root sources are defined, they will be merged first, then the other remaining in sources will be merged with existing documents.

If root doesn't exist (or list is empty), all sources can initiate documents creation. root can be a list of collection names, or a negation (not a mix of both). So, for instance, if we want all datasources to be root, except source10, we can specify: "root" : ["!source10"]. Finally, all root sources must all be declared in sources (root is a subset of sources). That said, it's interesting in our case because we have taxonomy information coming from NCBI and UniProt, but we want to make sure a document built from UniProt only doesn't exist (it's because we need parent_taxid field which only exists in NCBI data, so we give priority to those sources first). So root sources are going to be **names** and **nodes**, but because we're lazy typist, we're going to set this to: {"root" : ["!species"]}

The resulting document should look like this. Let's save this in src_build (and also remove the placeholder, not useful anymore):

```
> conf
{
    "_id" : "mytaxonomy",
    "name" : "mytaxonomy",
    "sources" : [
        "names",
        "uniprot_species",
        "nodes",
        "geneinfo"
    ],
    "root" : ["!uniprot_species"]
}
> db.src_build.save(conf)
> db.src_build.remove({_id:"placeholder"})
```

Note: **geneinfo** contains only IDs, we could ignore it while merging but we'll need it to be declared as a source when we'll create the index later.

Restarting the hub, we can then check that configuration has properly been registered in the manager, ready to be used. We can list the sources specified in configuration.

```
hub> bm
<BuilderManager [1 registered]: ['mytaxonomy']>
hub> bm.list_sources("mytaxonomy")
['names', 'species', 'nodes']
```

OK, let's try to merge !

```
hub> merge("mytaxonomy")
[1] RUN {0.0s} merge("mytaxonomy")
```

Looking at the logs, we can see builder will first root sources:

```
INFO:mytaxonomy_build:Merging into target collection 'mytaxonomy_20170127_pn1ygtqp'
...
INFO:mytaxonomy_build:Sources to be merged: ['names', 'nodes', 'species', 'geneinfo']
INFO:mytaxonomy_build:Root sources: ['names', 'nodes', 'geneinfo']
INFO:mytaxonomy_build:Other sources: ['species']
INFO:mytaxonomy_build:Merging root document sources: ['names', 'nodes', 'geneinfo']
```

Then once root sources are processed, **species** collection merged on top on existing documents:

```
INFO:mytaxonomy_build:Merging other resources: ['species']
DEBUG:mytaxonomy_build:Documents from source 'species' will be stored only if a previous
↳ document exists with same _id
```

After a while, task is done, merge has returned information about the amount of data that have been merge: 1552809 records from collections **names**, **nodes** and **geneinfo**, 25394 from **species**. Note: the figures show the number fetched from collections, not necessarily the data merged. For instance, merged data from **species** may be less since it's not a root datasource).

```
hub>
[1] OK  merge("mytaxonomy"): finished, [{"total_species": 25394, "total_nodes": 1552809,
↳ "total_names": 1552809}]
```

Builder creates multiple merger jobs per collection. The merged collection name is, by default, generating from the build name (**mytaxonomy**), and contains also a timestamp and some random chars. We can specify the merged collection name from the hub. By default, all sources defined in the configuration are merged., and we can also select one or more specific sources to merge:

```
hub> merge("mytaxonomy", sources="uniprot_species", target_name="test_merge")
```

Note: **sources** parameter can also be a list of string.

If we go back to **src_build**, we can have information about the different merges (or builds) we ran:

```
> db.src_build.find({_id:"mytaxonomy"}, {build:1})
{
    "_id" : "mytaxonomy",
    "build" : [
        ...
    {
        "src_versions" : {
            "geneinfo" : "20170125",
            "taxonomy" : "20170125",
            "uniprot_species" : "20170125"
        },
        "time_in_s" : 386,
        "logfile" : "./data/logs/mytaxonomy_20170127_build.log",
        "pid" : 57702,
        "target_backend" : "mongo",
        "time" : "6m26.29s",
        "step_started_at" : ISODate("2017-01-27T11:36:47.401Z"),
        "stats" : {
            "total_uniprot_species" : 25394,
            "total_nodes" : 1552809,
            "total_names" : 1552809
        },
        "started_at" : ISODate("2017-01-27T11:30:21.114Z"),
        "status" : "success",
        "target_name" : "mytaxonomy_20170127_pn1ygtqp",
        "step" : "post-merge",
        "sources" : [
            "uniprot_species"
        ]
    }
}
```

We can see the merged collection (auto-generated) is `mytaxonomy_20170127_pn1ygtqp`. Let's have a look at the content (remember, collection is in target database, not in src):

```
> use tutorial
switched to db tutorial
> db.mytaxonomy_20170127_pn1ygtqp.count()
1552809
> db.mytaxonomy_20170127_pn1ygtqp.find({_id:9606})
{
    "_id" : 9606,
    "rank" : "species",
    "parent_taxid" : 9605,
    "taxid" : 9606,
    "common_name" : "man",
    "other_names" : [
        "humans"
    ],
    "scientific_name" : "homo sapiens",
    "authority" : [
        "homo sapiens linnaeus, 1758"
    ],
    "genbank_common_name" : "human",
    "uniprot_name" : "homo sapiens"
}
}
```

Both collections have properly been merged. We now have to deal with the other data.

Mappers

The next bit of data we need to merge is `geneinfo`. As a reminder, this collection only contains taxonomy ID (as `_id` key) which have known NCBI genes. We'll create a mapper, containing this information. A mapper basically acts as an object that can pre-process documents while they are merged.

Let's define that mapper in `databuild/mapper.py`

```
import biotools, config
biotools.config_for_app(config)
from biotools.utils.common import loadobj
import biotools.utils.mongo as mongo
import biotools.hub.databuild.mapper as mapper
# just to get the collection name
from dataload.sources.geneinfo.uploader import GeneInfoUploader

class HasGeneMapper(mapper.BaseMapper):

    def __init__(self, *args, **kwargs):
        super(HasGeneMapper, self).__init__(*args, **kwargs)
        self.cache = None

    def load(self):
        if self.cache is None:
            # this is a whole dict containing all taxonomy_ids
```

(continues on next page)

(continued from previous page)

```

col = mongo.get_src_db()[GeneInfoUploader.name]
self.cache = [d["_id"] for d in col.find({}, {"_id":1})]

def process(self,docs):
    for doc in docs:
        if doc["_id"] in self.cache:
            doc["has_gene"] = True
        else:
            doc["has_gene"] = False
    yield doc

```

We derive our mapper from `biothings.hub.databuild.mapper.BaseMapper`, which expects `load` and `process` methods to be defined. `load` is automatically called when the mapper is used by the builder, and `process` contains the main logic, iterating over documents, optionally enrich them (it can also be used to filter documents, by not yielding them). The implementation is pretty straightforward. We get and cache the data from geneinfo collection (the whole collection is very small, less than 20'000 IDs, so it can fit nicely and efficiently in memory). If a document has its `_id` found in the cache, we enrich it.

Once defined, we register that mapper into the builder. In `bin/hub.py`, we modify the way we define the builder manager:

```

import biothings.hub.databuild.builder as builder
from databuild.mapper import HasGeneMapper
hasgene = HasGeneMapper(name="has_gene")
pbuilder = partial(builder.DataBuilder,mappers=[hasgene])
bmanager = builder.BuilderManager(
    poll_schedule='* * * * */10',
    job_manager=jmanager,
    builder_class=pbuilder)
bmanager.configure()
bmanager.poll()

```

First we instantiate a mapper object and give it a name (more on this later). While creating the manager, we need to pass a builder class. The problem here is we also have to give our mapper to that class while it's instantiated. We're using `partial` (from `functools`), which allows to partially define the class instantiation. In the end, `builder_class` parameter is expected to a callable, which is the case with `partial`.

Let's try if our mapper works (restart the hub). Inside the hub, we're going to manually get a builder instance. Remember through the SSH connection, we can access python interpreter's namespace, which is very handy when it comes to develop and debug as we can directly access and "play" with objects and their states:

First we get a builder instance from the manager:

```

hub> builder = bm["mytaxonomy"]
hub> builder
<biothings.hub.databuild.builder.DataBuilder object at 0x7f278aecf400>

```

Let's check the mappers and get ours:

```

hub> builder.mappers
{None: <biothings.hub.databuild.mapper.TransparentMapper object at 0x7f278aecf4e0>, 'has_
→gene': <databuild.mapper.HasGeneMapper object at 0x7f27ac6c0a90>}

```

We have our `has_gene` mapper (it's the name we gave). We also have a `TransparentMapper`. This mapper is automatically added and is used as the default mapper for any document (there has to be one...).

```
hub> hasgene = builder.mappers["has_gene"]
hub> len(hasgene.cache)
Error: TypeError("object of type 'NoneType' has no len()",)
```

Oops, cache isn't loaded yet, we have to do it manually here (but it's done automatically during normal execution).

```
hub> hasgene.load()
hub> len(hasgene.cache)
19201
```

OK, it's ready. Let's now talk more about the mapper's name. A mapper can be applied to different sources, and we have to define which sources' data should go through that mapper. In our case, we want **names** and **species** collection's data to go through. In order to do that, we have to instruct the uploader with a special attribute. Let's modify `dataloader.sources.species.uploader.UniprotSpeciesUploader` class

```
class UniprotSpeciesUploader(uploader.BaseSourceUploader):

    name = "uniprot_species"
    __metadata__ = {"mapper": "has_gene"}
```

`__metadata__` dictionary is going to be used to create a master document. That document is stored in `src_master` collection (we talked about it earlier). Let's add this metadata to `dataloader.sources.taxonomy.uploader.TaxonomyNamesUploader`

```
class TaxonomyNamesUploader(uploader.BaseSourceUploader):

    main_source = "taxonomy"
    name = "names"
    __metadata__ = {"mapper": "has_gene"}
```

Before using the builder, we need to refresh master documents so these metadata are stored in `src_master`. We could trigger a new upload, or directly tell the hub to only process master steps:

```
hub> upload("uniprot_species", steps="master")
[1] RUN {0.0s} upload("uniprot_species", steps="master")
hub> upload("taxonomy.names", steps="master")
[1] OK upload("uniprot_species", steps="master"): finished, [None]
[2] RUN {0.0s} upload("taxonomy.names", steps="master")
```

(you'll notice for taxonomy, we only trigger upload for sub-source **names**, using “dot-notation”, corresponding to “`main_source.name`”. Let's now have a look at those master documents:

```
> db.src_master.find({_id:{$in:["uniprot_species","names"]}})
{
    "_id" : "names",
    "name" : "names",
    "timestamp" : ISODate("2017-01-26T16:21:32.546Z"),
    "mapper" : "has_gene",
    "mapping" : {

    }
}
{
    "_id" : "uniprot_species",
```

(continues on next page)

(continued from previous page)

```
"name" : "uniprot_species",
"timestamp" : ISODate("2017-01-26T16:21:19.414Z"),
"mapper" : "has_gene",
"mapping" : {

}

}
```

We have our `mapper` key stored. We can now trigger a new merge (we specify the target collection name):

```
hub> merge("mytaxonomy",target_name="mytaxonomy_test")
[3] RUN {0.0s} merge("mytaxonomy",target_name="mytaxonomy_test")
```

In the logs, we can see our mapper has been detected and is used:

```
INFO:mytaxonomy_build:Creating merger job #1/16, to process 'names' 1000000/1552809 (6.4%)
INFO:mytaxonomy_build:Found mapper '<databuild.mapper.HasGeneMapper object at
˓→0x7f47ef3bbac8>' for source 'names'
INFO:mytaxonomy_build:Creating merger job #1/1, to process 'species' 25394/25394 (100.0%)
INFO:mytaxonomy_build:Found mapper '<databuild.mapper.HasGeneMapper object at
˓→0x7f47ef3bbac8>' for source 'species'
```

Once done, we can query the merged collection to check the data:

```
> use tutorial
switched to db tutorial
> db.mytaxonomy_test.find({_id:9606})
{
    "_id" : "9606",
    "has_gene" : true,
    "taxid" : 9606,
    "uniprot_name" : "homo sapiens",
    "other_names" : [
        "humans"
    ],
    "scientific_name" : "homo sapiens",
    "authority" : [
        "homo sapiens linnaeus, 1758"
    ],
    "genbank_common_name" : "human",
    "common_name" : "man"
}
```

OK, there's a `has_gene` flag that's been set. So far so good !

Post-merge process

We need to add lineage and parent taxid information for each of these documents. We'll implement that last part as a post-merge step, iterating over each of them. In order to do so, we need to define our own builder class to override proper methods there. Let's define it in `databuild/builder.py`.

```
import biothings.hub.databuild.builder as builder
import config

class TaxonomyDataBuilder(builder.DataBuilder):

    def post_merge(self, source_names, batch_size, job_manager):
        pass
```

The method we have to implement in `post_merge`, as seen above. We also need to change `hub.py` to use that builder class:

```
from databuild.builder import TaxonomyDataBuilder
pbuilder = partial(TaxonomyDataBuilder,mappers=[hasgene])
```

For now, we just added a class level in the hierarchy, everything runs the same as before. Let's have a closer look to that post-merge process. For each document, we want to build the lineage. Information is stored in `nodes` collection. For instance, taxid 9606 (homo sapiens) has a parent_taxid 9605 (homo), which has a parent_taxid 207598 (homininae), etc... In the end, the homo sapiens lineage is:

```
9606, 9605, 207598, 9604, 314295, 9526, 314293, 376913, 9443, 314146, 1437010, 9347, ↵
↪ 32525, 40674, 32524, 32523, 1338369,
8287, 117571, 117570, 7776, 7742, 89593, 7711, 33511, 33213, 6072, 33208, 33154, 2759, ↵
↪ 131567 and 1
```

We could recursively query `nodes` collections until we reach the top of the tree, but that would be a lot of queries. We just need `taxid` and `parent_taxid` information to build the lineage, maybe it's possible to build a dictionary that could fit in memory. `nodes` has 1552809 records. A dictionary would use $2 * 1552809 * \text{sizeof(integer)} + \text{index overhead}$. That's probably few megabytes, let's assume that ok... (note: using `pympler` lib, we can actually know that dictionary size will be closed to 200MB...)

We're going to use another mapper here, but no sources will use it. We'll just instantiate it from `post_merge` method. In `databuild/mapper.py`, let's add another class:

```
from dataload.sources.taxonomy.uploader import TaxonomyNodesUploader
```

```
class LineageMapper(mapper.BaseMapper):

    def __init__(self, *args, **kwargs):
        super(LineageMapper, self).__init__(*args, **kwargs)
        self.cache = None

    def load(self):
        if self.cache is None:
            col = mongo.get_src_db()[TaxonomyNodesUploader.name]
            self.cache = {}
            [self.cache.setdefault(d["_id"], d["parent_taxid"]) for d in col.find({}, {
                "parent_taxid": 1})]

    def get_lineage(self, doc):
```

(continues on next page)

(continued from previous page)

```

if doc['taxid'] == doc['parent_taxid']: #take care of node #1
    # we reached the top of the taxonomy tree
    doc['lineage'] = [doc['taxid']]
    return doc
# initiate lineage with information we have in the current doc
lineage = [doc['taxid'], doc['parent_taxid']]
while lineage[-1] != 1:
    parent = self.cache[lineage[-1]]
    lineage.append(parent)
doc['lineage'] = lineage
return doc

def process(self,docs):
    for doc in docs:
        doc = self.get_lineage(doc)
    yield doc

```

Let's use that mapper in TaxonomyDataBuilder's `post_merge` method. The signature is the same as `merge()` method (what's actually called from the hub) but we just need the `batch_size` one: we're going to grab documents from the merged collection in batch, process them and update them in batch as well. It's going to be much faster than dealing one document at a time. To do so, we'll use `doc_feeder` utility function:

```

from biothings.utils.mongo import doc_feeder, get_target_db
from biothings.hub.databuild.builder import DataBuilder
from biothings.hub.dataload.storage import UpsertStorage

from databuild.mapper import LineageMapper
import config
import logging

class TaxonomyDataBuilder(DataBuilder):

    def post_merge(self, source_names, batch_size, job_manager):
        # get the lineage mapper
        mapper = LineageMapper(name="lineage")
        # load cache (it's being loaded automatically
        # as it's not part of an upload process
        mapper.load()

        # create a storage to save docs back to merged collection
        db = get_target_db()
        col_name = self.target_backend.target_collection.name
        storage = UpsertStorage(db, col_name)

        for docs in doc_feeder(self.target_backend.target_collection, step=batch_size,
                               inbatch=True):
            docs = mapper.process(docs)
            storage.process(docs, batch_size)

```

Since we're using the mapper manually, we need to load the cache

- `db` and `col_name` are used to create our storage engine. Builder has an attribute called `target_backend` (a `biothings.hub.dataload.backend.TargetDocMongoBackend` object) which can be used to reach the col-

lection we want to work with.

- **doc_feeder** iterates over all the collection, fetching documents in batch. `inbatch=True` tells the function to return data as a list (default is a dict indexed by `_id`).
- those documents are processed by our mapper, setting the lineage information and then are stored using our `UpsertStorage` object.

Note: `post_merge` actually runs within a thread, so any calls here won't block the execution (ie. won't block the `asyncio` event loop execution)

Let's run this on our merged collection. We don't want to merge everything again, so we specify the step we're interested in and the actual merged collection (`target_name`)

```
hub>     merge("mytaxonomy",steps="post",target_name="mytaxonomy_test")      [1]      RUN      {0.0s}
merge("mytaxonomy",steps="post",target_name="mytaxonomy_test")
```

After a while, process is done. We can test our updated data:

```
> use tutorial
switched to db tutorial
> db.mytaxonomy_test.find({_id:9606})
{
    "_id" : 9606,
    "taxid" : 9606,
    "common_name" : "man",
    "other_names" : [
        "humans"
    ],
    "uniprot_name" : "homo sapiens",
    "rank" : "species",
    "lineage" : [9606,9605,207598,9604,...,131567,1],
    "genbank_common_name" : "human",
    "scientific_name" : "homo sapiens",
    "has_gene" : true,
    "parent_taxid" : 9605,
    "authority" : [
        "homo sapiens linnaeus, 1758"
    ]
}
```

OK, we have new lineage information (truncated for sanity purpose). Merged collection is ready to be used. It can be used for instance to create and send documents to an ElasticSearch database. This is what's actually occurring when creating a BioThings web-service API. That step will be covered in another tutorial.

6.4.7 Indexers

Coming soon!

Full updated and maintained code for this hub is available here: <https://github.com/biotools/biotools.species>

Also, taxonomy BioThings API can be queried as this URL: [http://t.biotoools.io](http://t.biotools.io)

6.5 BioThings Web

In this tutorial we will start a Biothings API and learn to customize it, overriding the default behaviors and adding new features, using increasingly more advanced techniques step by step. In the end, you will be able to make your own Biothings API, run other production APIs, like Mygene.info, and additionally, customize and add more features to those projects.

Attention: Before starting the tutorial, you should have the `biothings` package installed, and have an `Elasticsearch` running with only one index populated with this `dataset` using this `mapping`. You may also need a JSON Formatter browser extension for the best experience following this tutorial. (For [Chrome](#))

6.5.1 1. Starting an API server

First, assuming your Elasticsearch service is running on the default port 9200, we can run a Biothings API with all default settings to explore the data, simply by creating a `config.py` under your project folder. After creating the file, run `python -m biothings.web` to start the API server. You should be able to see the following console output:

```
[I 211130 22:21:57 launcher:28] Biothings API 0.10.0
[I 211130 22:21:57 configs:86] <module 'config' from 'C:\\\\Users\\\\Jerry\\\\code\\\\biothings.
˓tutorial\\\\config.py'>
[INFO biothings.web.connections:31] <Elasticsearch([{'host': 'localhost', 'port': 9200}
˓])>
[INFO biothings.web.connections:31] <AsyncElasticsearch([{'host': 'localhost', 'port': 9200}])>
[INFO biothings.web.applications:137] API Handlers:
  [('/', <class 'biothings.web.handlers.services.FrontPageHandler'>, {}),
   ('/status', <class 'biothings.web.handlers.services.StatusHandler'>, {}),
   ('/metadata/fields/?', <class 'biothings.web.handlers.query.MetadataFieldHandler'>,
˓{}),
   ('/metadata/?', <class 'biothings.web.handlers.query.MetadataSourceHandler'>, {}),
   ('/v1/spec/?', <class 'biothings.web.handlers.services.APISpecificationHandler'>, {}
˓),
   ('/v1/doc(?:/([^\?]+))/??', <class 'biothings.web.handlers.query.BiothingHandler'>, {
˓'biothing_type': 'doc'},
   ('/v1/metadata/fields/?', <class 'biothings.web.handlers.query.MetadataFieldHandler'>,
˓{}),
   ('/v1/metadata/?', <class 'biothings.web.handlers.query.MetadataSourceHandler'>, {}),
   ('/v1/query/?', <class 'biothings.web.handlers.query.QueryHandler'>, {})]
[INFO biothings.web.launcher:99] Server is running on "0.0.0.0:8000"...
[INFO biothings.web.connections:25] Elasticsearch Package Version: 7.13.4
[INFO biothings.web.connections:27] Elasticsearch DSL Package Version: 7.3.0
[INFO biothings.web.connections:51] localhost:9200: docker-cluster 7.9.3
```

Note the console log shows the API version, the config file it uses, its database connections, HTTP routes, service port, important python dependency package versions, as well as the database cluster details.

Note: The cluster detail appears as the last line, sometimes with a delay, because it is scheduled asynchronously at start time, but executed later after the main program has launched. The default implementation of our application is `asynchronous` and `non-blocking` based on `asyncio` and `tornado.ioloop` interface. The specific logic in this case is implemented in the `biothings.web.connections` module.

Of all the information provided, note that it says the server is running on port 8000, this is the default port we use when we start a Biothings API. It means you can access the API by opening <http://localhost:8000/> in your browser in most of the cases.

Note: If this port is occupied, you can pass the “port” parameter during startup to change it, for example, running `python -m biothings.web --port=9000`. The links in the tutorial assume the service is running on the default port 8000. If you are running the service on a different port. You need to modify the URLs provided in the tutorial before opening in the browser.

Now open the browser and access localhost:8000, we should be able to see the biothings welcome page, showing the public routes in `regex` formats reading like:

```
/  
/status  
/metadata/fields/?  
/metadata/?  
/v1/spec/?  
/v1/doc(?:/([^\/]+))?/?  
/v1/metadata/fields/?  
/v1/metadata/?  
/v1/query/?
```

6.5.2 2. Exploring an API endpoint

The last route on the welcome page shows the URL pattern of the query API. Let’s use this pattern to access the query endpoint. Accessing <http://localhost:8000/v1/query/> returns a JSON document containing 10 results from our elasticsearch index.

Let’s explore some Biothings API features here, adding a query parameter “fields” to limit the fields returned by the API, and another parameter “size” to limit the returned document number. If you used the dataset mentioned at the start of the tutorial, accessing <http://localhost:8000/v1/query?fields=symbol,alias,name&size=1> should return a document like this:

```
{  
    "took": 15,  
    "total": 1030,  
    "max_score": 1,  
    "hits": [  
        {  
            "_id": "1017",  
            "_score": 1,  
            "alias": [  
                "CDKN2",  
                "p33(CDK2)"  
            ],  
            "name": "cyclin dependent kinase 2",  
            "symbol": "CDK2"  
        }  
    ]  
}
```

The most commonly used parameter is the “q” parameter, try <http://localhost:8000/v1/query?q=cdk2> and see all the returned results contain “cdk2”, the value specified for the “q” parameter.

Note: For a list of the supporting parameters, visit [Biothings API Specifications](#). The documentation for our most popular service <https://mygene.info/> also covers a lot of features also available in all biothings applications. Read more on [Gene Query Service](#) and [Gene Annotation Service](#).

6.5.3 3. Customizing an API through the config file

In the previous step, we tested document exploration by search its content. Is there a way to access individual documents directly by their “_id” or other id fields? We can look at the annotation endpoint doing exactly that.

By default, this endpoint is accessible by an URL pattern like this: /<ver>/doc/<_id> where “ver” refers to the API version. In our case, if we want to access a document with an id of “1017”, one of those doc showing up in the previous example, we can try: <http://localhost:8000/v1/doc/1017>

Note: To configure a different API version other than “v1” for your program, add a prefix to all API patterns, like /api/<ver>/..., or remove these patterns, make changes in the config file modifying the settings prefixed with “APP”, as those control the web application behavior. A web application is basically a collection of routes and settings that can be understood by a web server. See [biothings.web.settings.default](#) source code to look at the current configuration and refer to [biothings.web.applications](#) to see how the settings are turned to routes in different web frameworks.

In this dataset, we know the document type can be best described as “gene”s. We can enable a widely-used feature, document type URL templating, by providing more information to the biothings app in the config.py file. Write the following lines to the config file:

```
1 ES_HOST = "localhost:9200" # optional
2 ES_INDICES = {"gene": "<your index name>"}
3
4 ANNOTATION_DEFAULT_SCOPES = ["_id", "symbol"]
```

Note: The ES_HOST setting is a common parameter that you see in the config file. Although it is not making a difference here, you can configure the value of this setting to ask biothings.web to connect to a different Elasticsearch server, maybe hosted remotely on the cloud. The ANNOTATION_DEFAULT_SCOPES setting specifies the document fields we consider as the id fields. By default, only the “_id” field in the document, a must-have field in Elasticsearch, is considered the biothings id field. We additionally added the “symbol” field, to allow the user to it to find documents in this demo API.

Restart your program and see the annotation route is now prefixed with /v1/gene if you pay close attention to the console log. Now try the following URL:

<http://localhost:8000/v1/gene/1017>
<http://localhost:8000/v1/gene/CDK2>

See that using both of the URLs can take you straight to the document previously mentioned. Note using the symbol field “CDK2” may yield multiple documents because multiple documents may have the same key-value pair. This also means “symbol” may not be a good choice of the key field we want to support in the URL.

These two endpoints, annotation and query, are the pillars for Biothings API. You can additionally customize these endpoints to work better with your data.

For example, if you think our returned result by default from the query endpoint is too verbose and we want to only include limited information unless the user specifically asked for more, we can set a default “fields” value, for this parameter used in the previous example. Open config.py and add:

```
from biothings.web.settings.default import QUERY_KWARGS
QUERY_KWARGS['*']['_source']['default'] = ['name', 'symbol', 'taxid', 'entrezgene']
```

Restart your program after changing the config file and visit <http://localhost:8000/v1/query>, see the effect of specifying default fields to return. Like this:

```
{
    "took": 9,
    "total": 100,
    "max_score": 1,
    "hits": [
        {
            "_id": "1017",
            "_score": 1,
            "entrezgene": "1017",
            "name": "cyclin dependent kinase 2",
            "symbol": "CDK2",
            "taxid": 9606
        },
        {
            "_id": "12566",
            "_score": 1,
            "entrezgene": "12566",
            "name": "cyclin-dependent kinase 2",
            "symbol": "Cdk2",
            "taxid": 10090
        },
        {
            "_id": "362817",
            "_score": 1,
            "entrezgene": "362817",
            "name": "cyclin dependent kinase 2",
            "symbol": "Cdk2",
            "taxid": 10116
        },
        ...
    ]
}
```

6.5.4 4. Customizing an API through pipeline stages

In the previous example, the numbers in the “entrezgene” field are typed as strings. Let’s modify the internal logic called the query pipeline to convert these values to integers just to show what we can do in customization.

Note: The pipeline is one of the *biothings.web.services*. It defines the intermediate steps or stages we take to execute a query. See *biothings.web.query* to learn more about the individual stages.

Add to config.py:

```
ES_RESULT_TRANSFORM = "pipeline.MyFormatter"
```

And create a file `pipeline.py` to include:

```
1  from biothings.web.query import ESResultFormatter
2
3
4  class MyFormatter(ESResultFormatter):
5
6      def transform_hit(self, path, doc, options):
7
8          if path == '' and 'entrezgene' in doc: # root level
9              try:
10                  doc['entrezgene'] = int(doc['entrezgene'])
11              except:
12                  ...
13
```

Commit your changes and restart the webserver process. Run some queries and you should be able to see the “entrezgene” field now showing as integers:

```
{  
    "_id": "1017",  
    "_score": 1,  
    "entrezgene": 1017, # instead of the quoted "1017" (str)  
    "name": "cyclin dependent kinase 2",  
    "symbol": "CDK2",  
    "taxid": 9606  
}
```

In this example, we made changes to the query transformation stage, controlled by the `biothings.web.query.formatter.ESResultFormatter` class, this is one of the three stages that defined the query pipeline. The two stages coming before it are represented by `biothings.web.query.engine.AsyncESQueryBackend` and `biothings.web.query.builder.ESQueryBuilder`.

Let’s try to modify the query builder stage to add another feature. We’ll incorporate domain knowledge here to deliver more user-friendly search result by scoring the documents with a few rules to increase result relevancy. Additionally add to the `pipeline.py` file:

```
from biothings.web.query import ESQueryBuilder
from elasticsearch_dsl import Search

class MyQueryBuilder(ESQueryBuilder):

    def apply_extras(self, search, options):
        search = Search().query(
            "function_score",
            query=search.query,
            functions=[{"filter": {"term": {"name": "pseudogene"}}, "weight": "0.5"}, #_
        ↵downgrade
            {"filter": {"term": {"taxid": 9606}}, "weight": "1.55"},  

            {"filter": {"term": {"taxid": 10090}}, "weight": "1.3"},  

            {"filter": {"term": {"taxid": 10116}}, "weight": "1.1"},  

        ]
)
```

(continues on next page)

(continued from previous page)

```
    ], score_mode="first")

    return super().apply_extras(search, options)
```

Make sure our application can pick up the change by adding this line to `config.py`:

```
ES_QUERY_BUILDER = "pipeline.MyQueryBuilder"
```

Note: We wrapped our original query logic in an Elasticsearch compound query function `score query`. For more on writing python-friendly Elasticsearch queries, see [Elasticsearch DSL](#) package, one of the dependencies used in `biothings.web`.

Save the file and restart the webserver process. Search something and if you compare with the application before, you may notice some result rankings have changed. It is not easy to pick up this change if you are not familiar with the data, visit <http://localhost:8000/v1/query?q=kinase&rawquery> instead and see that our code was indeed making a difference and get passed to elasticsearch, affecting the query result ranking. Notice the “rawquery” is a feature in our program to intercept the raw query we sent to elasticsearch for debugging.

6.5.5 5. Customizing an API through pipeline services

Taking it one more step further, we can add more procedures or stages to the pipeline by overwriting the Pipeline class. Add to the config file:

```
ES_QUERY_PIPELINE = "pipeline.MyQueryPipeline"
```

and add the following code to `pipeline.py`:

```
class MyQueryPipeline(AsyncESQueryPipeline):

    async def fetch(self, id, **options):

        if id == "tutorial":
            res = {"_welcome": "to the world of biothings.api"}
            res.update(await super().fetch("1017", **options))
            return res

        res = await super().fetch(id, **options)
        return res
```

Now we made ourselves a tutorial page to show what annotation results can look like, by visiting <http://localhost:8000/v1/gene/tutorial>, you can see what <http://localhost:8000/v1/gene/1017> would typically give you, and the additional welcome message:

```
{
    "_welcome": "to the world of biothings.api",
    "_id": "1017",
    "_version": 1,
    "entrezgene": 1017,
    "name": "cyclin dependent kinase 2",
    "symbol": "CDK2",
```

(continues on next page)

(continued from previous page)

```
"taxid": 9606  
}
```

Note: In this example, we modified the query pipeline's "fetch" method, the one used in the annotation endpoint, to include some additional logic before executing what we would typically do. The call to the "super" function executes the typical query building, executing and formatting stages.

6.5.6 6. Customizing an API through the web app

The examples above demonstrated the customizations you can make on top of our pre-defined APIs, for the most demanding tasks, you can additionally add your own API routes to the web app.

Modify the config file as a usual first step. Declare a new route by adding:

```
from biothings.web.settings.default import APP_LIST  
  
APP_LIST = [  
    *APP_LIST, # keep the original ones  
    (r"/{ver}/echo/(.+)", "handlers.EchoHandler"),  
]
```

Let's make an echo handler that just echos what the user puts in the URL. Create a `handlers.py` and add:

```
1 from biothings.web.handlers import BaseAPIHandler  
2  
3  
4 class EchoHandler(BaseAPIHandler):  
5  
6     def get(self, text):  
7         self.write({  
8             "status": "ok",  
9             "result": text  
10        })
```

Now we have added a completely new feature not based on any of the existing biothings offerings, which can be as simple and as complex as you need. Visiting <http://localhost:8000/v1/echo/hello> would give you:

```
{  
    "status": "ok",  
    "result": "hello"  
}
```

in which case, the "hello" in "result" field is the input we give the application in the URL.

6.5.7 7. Customizing an API through the app launcher

Another convenient place to customize the API is to have a launching module, typically called `index.py`, and pass parameters to the starting function, provided as `biothings.web.launcher.main()`. Create an `index.py` in your project folder:

```

1 from biothings.web.launcher import main
2 from tornado.web import RedirectHandler
3
4 if __name__ == '__main__':
5     main([
6         (r"/v2/query(.*)", RedirectHandler, {"url": "/v1/query{0}"})
7     ], {
8         "static_path": "static"
9     })

```

Create another folder called “static” and add a file of random content named “file.txt” under the newly created static folder. In this step, we added a redirection of a later-to-launch v2 query API, that we temporarily set to redirect to the v1 API and passed a static file configuration that asks tornado to serve files under the static folder we specified to the tornado webserver, the default webserver we use. The static folder is named “static” and contains only one file in this example.

Note: For more on configuring route redirections and other application features in tornado, see [RedirectHandler](#) and [Application configuration](#).

After making the changes, visiting `http://localhost:8000/v2/query/?q=cdk2` would direct you back to `http://localhost:8000/v1/query/?q=cdk2` and by visiting `http://localhost:8000/static/file.txt` you should see the random content you previously created. Note in this step, you should run the python launcher module directly by calling something like `python index.py` instead of running the first command we introduced. Running the launcher directly is also how we start most of our user-facing products that require complex configurations, like `http://mygene.info/`. ts code is publicly available at <https://github.com/biothings/mygene.info> under the Biothings Organization.

6.5.8 The End

Finishing this tutorial, you have completed the most common steps to customize `biothings.api`. The customization starts from passing a different parameter at launch time and evolve to modifying the app code at different levels. I hope you feel confident running `biothings API` now and please check out the documentation page for more details on customizing APIs.

6.6 DataTransform Module

A key problem when merging data from multiple data sources is finding a common identifier. To ameliorate this problem, we have written a `DataTransform` module to convert identifiers from one type to another. Frequently, this conversion process has multiple steps, where an identifier is converted to one or more intermediates before having its final value. To describe these steps, the user defines a graph where each node represents an identifier type and each edge represents a conversion. The module processes documents using the network to convert their identifiers to their final form.

A graph is a mathematical model describing how different things are connected. Using our model, our module is connecting different identifiers together. Each connection is an identifier conversion or lookup process. For example, a simple graph could describe how `pubchem` identifiers could be converted to `drugbank` identifiers using `MyChem.info`.

6.6.1 Graph Definition

The following graph facilitates conversion from *inchi* to *inchikyey* using *pubchem* as an intermediate:

```
from biothings.hub.datatransform import MongoDBEdge
import networkx as nx

graph_mychem = nx.DiGraph()

#####
# DataTransform Nodes and Edges
#####
graph_mychem.add_node('inchi')
graph_mychem.add_node('pubchem')
graph_mychem.add_node('inchikyey')

graph_mychem.add_edge('inchi', 'pubchem',
                      object=MongoDBEdge('pubchem', 'pubchem.inchi', 'pubchem.cid'))

graph_mychem.add_edge('pubchem', 'inchikyey',
                      object=MongoDBEdge('pubchem', 'pubchem.cid', 'pubchem.inchi_key'))
```

To setup a graph, one must define nodes and edges. There should be a node for each type of identifier and an edge which describes how to convert from one identifier to another. Node names can be arbitrary; the user is allowed to chose what an identifier should be called. Edge classes, however, must be defined precisely for conversion to be successful.

6.6.2 Edge Classes

The following edge classes are supported by the *DataTransform* module. One of these edge classes must be selected when defining an edge connecting two nodes in a graph.

MongoDBEdge

```
class biothings.hub.datatransform.MongoDBEdge(collection_name, lookup, field, weight=1, label=None,
                                               check_index=True)
```

The MongoDBEdge uses data within a MongoDB collection to convert one identifier to another. The input identifier is used to search a collection. The output identifier values are read out of that collection:

Parameters

- **collection_name** (*str*) – The name of the MongoDB collection.
- **lookup** (*str*) – The field that will match the input identifier in the collection.
- **field** (*str*) – The output identifier field that will be read out of matching documents.
- **weight** (*int*) – Weights are used to prefer one path over another. The path with the lowest weight is preferred. The default weight is 1.

The example above uses the *MongoDBEdge* class to convert from *inchi* to *inchikyey*.

MyChemInfoEdge

```
class biothings.hub.datatransform.MyChemInfoEdge(lookup, field, weight=1, label=None, url=None)
```

The MyChemInfoEdge uses the MyChem.info API to convert identifiers.

Parameters

- **lookup (str)** – The field in the API to search with the input identifier.
- **field (str)** – The field in the API to convert to.
- **weight (int)** – Weights are used to prefer one path over another. The path with the lowest weight is preferred. The default weight is 1.

This example graph uses the *MyChemInfoEdge* class to convert from *pubchem* to *inchikey*. The *pubchem.cid* and *pubchem.inchi_key* fields are returned by *MyChem.info* and are listed by [/metadata/fields](#).

```
from biothings.hub.datatransform import MyChemInfoEdge
import networkx as nx

graph_mychem = nx.DiGraph()

#####
# DataTransform Nodes and Edges
#####
graph_mychem.add_node('pubchem')
graph_mychem.add_node('inchikey')

graph_mychem.add_edge('pubchem', 'inchikey',
                      object=MyChemInfoEdge('pubchem.cid', 'pubchem.inchi_key'))
```

MyGeneInfoEdge

```
class biothings.hub.datatransform.MyGeneInfoEdge(lookup, field, weight=1, label=None, url=None)
```

The MyGeneInfoEdge uses the MyGene.info API to convert identifiers.

Parameters

- **lookup (str)** – The field in the API to search with the input identifier.
- **field (str)** – The field in the API to convert to.
- **weight (int)** – Weights are used to prefer one path over another. The path with the lowest weight is preferred. The default weight is 1.

RegExEdge

```
class biothings.hub.datatransform.RegExEdge(from_regex, to_regex, weight=1, label=None)
```

The RegExEdge allows an identifier to be transformed using a regular expression. POSIX regular expressions are supported.

Parameters

- **from_regex (str)** – The first parameter of the regular expression substitution.
- **to_regex (str)** – The second parameter of the regular expression substitution.

- **weight (int)** – Weights are used to prefer one path over another. The path with the lowest weight is preferred. The default weight is 1.

This example graph uses the *RegExEdge* class to convert from *pubchem* to a shorter form. The *CID:* prefix is removed by the regular expression substitution:

```
from biotools.hub.datatransform import RegExEdge
import networkx as nx

graph = nx.DiGraph()

#####
# DataTransform Nodes and Edges
#####
graph.add_node('pubchem')
graph.add_node('pubchem-short')

graph.add_edge('pubchem', 'pubchem-short',
               object=RegExEdge('CID:', ''))
```

6.6.3 Example Usage

A complex graph developed for use with [MyChem.info](#) is shown [here](#). This file includes a definition of the *MyChemKeyLookup* class which is used to call the module on the data source. In general, the graph and class should be supplied to the user by the BioThings.api maintainers.

To call the *DataTransform* module on the *Biothings Uploader*, the following definition is used:

```
keylookup = MyChemKeyLookup(
    [('inchi', 'pharmgkb.inchi'),
     ('pubchem', 'pharmgkb.xrefs.pubchem.cid'),
     ('drugbank', 'pharmgkb.xrefs.drugbank'),
     ('chebi', 'pharmgkb.xrefs.chebi')])

def load_data(self,data_folder):
    input_file = os.path.join(data_folder,"drugs.tsv")
    return self.keylookup(load_data)(input_file)
```

The parameters passed to *MyChemKeyLookup* are a list of input types. The first element in an input type is the node name that must match the graph. The second element is the field in *dotstring* notation which should describe where the identifier should be read from in a document.

The following report was reported when using the *DataTransform* module with PharmGKB. Reports have a section for document conversion and a section describing conversion along each edge. The document section shows which inputs were used to produce which outputs. The edge section is useful in debugging graphs, ensuring that different conversion edges are working properly.

```
{
  'doc_report': {
    "('inchi', 'pharmgkb.inchi')-->inchikyey": 1637,
    "('pubchem', 'pharmgkb.xrefs.pubchem.cid')-->inchikyey": 46
    "('drugbank', 'pharmgkb.xrefs.drugbank')-->inchikyey": 41,
    "('drugbank', 'pharmgkb.xrefs.drugbank')-->drugbank": 25,
  }
}
```

(continues on next page)

(continued from previous page)

```
'edge_report': {
    'inchi-->chembl': 1109,
    'inchi-->drugbank': 319,
    'inchi-->pubchem': 209,
    'chembl-->inchikey': 1109,
    'drugbank-->inchikey': 360,
    'pubchem-->inchikey': 255
    'drugbank-->drugbank': 25,
},
}
```

As an example, the number identifiers converted from *inchi* to *inchikey* is 1637. However, these conversions are done via intermediates. One of these intermediates is *chembl* and the number of identifiers converted from *inchi* to *chembl* is 319. Some identifiers are converted directly from *pubchem* and *drugbank*. The *inchi* field is used to lookup several intermediates (*chembl*, *drugbank*, and *pubchem*). Eventually, most of these intermediates are converted to *inchikey*.

6.6.4 Advanced Usage - DataTransform MDB

The *DataTransformMDB* module was written as a decorator class which is intended to be applied to the *load_data* function of a *Biothings Uploader*. This class can be sub-classed to simplify application within a Biothings service.

```
class biothings.hub.datatransform.DataTransformMDB(graph, *args, **kwargs)
```

Convert document identifiers from one type to another.

The *DataTransformNetworkX* module was written as a decorator class which should be applied to the *load_data* function of a Biothings Uploader. The *load_data* function yields documents, which are then post processed by *call* and the ‘*id*’ key conversion is performed.

Parameters

- **graph** – nx.DiGraph (networkx 2.1) configuration graph
- **input_types** – A list of input types for the form (identifier, field) where identifier matches a node and field is an optional dotstring field for where the identifier should be read from (the default is ‘*_id*’).
- **output_types** (*list(str)*) – A priority list of identifiers to convert to. These identifiers should match nodes in the graph.
- **id_priority_list** (*list(str)*) – A priority list of identifiers to sort input and output types by.
- **skip_on_failure** (*bool*) – If True, documents where identifier conversion fails will be skipped in the final document list.
- **skip_w_regex** (*bool*) – Do not perform conversion if the identifier matches the regular expression provided to this argument. By default, this option is disabled.
- **skip_on_success** (*bool*) – If True, documents where identifier conversion succeeds will be skipped in the final document list.
- **idstruct_class** (*class*) – Override an internal data structure used by the this module (advanced usage)
- **copy_from_doc** (*bool*) – If true then an identifier is copied from the input source document regardless as to weather it matches an edge or not. (advanced usage)

An example of how to apply this class is shown below:

```
keylookup = DataTransformMDB(graph, input_types, output_types,
                             skip_on_failure=False, skip_w_regex=None,
                             idstruct_class=IDStruct, copy_from_doc=False)
def load_data(self,data_folder):
    input_file = os.path.join(data_folder,"drugs.tsv")
    return self.keylookup(load_data)(input_file)
```

It is possible to extend the *DataTransformEdge* type and define custom edges. This could be useful for example if the user wanted to define a computation that transforms one identifier to another. For example *inchikey* may be computed directly by performing a hash on the *inchi* identifier.

6.6.5 Document Maintainers

- Greg Taylor (@gretaylor)
- Chunlei Wu (@chunleiwu)

6.7 biothings.web

Generate a customized BioThings API given a supported database.

biothings.web.launcher: Launch web applications in different environments. **biothings.web.applications:** HTTP web application over data services below. **biothings.web.services & query:** Data services built on top of connections. **biothings.web.connections:** Elasticsearch, MongoDB and SQL database access.

6.7.1 Layers

biothings.web.launcher

Biothings API Launcher

In this module, we have three framework-specific launchers and a command-line utility to provide both programmatic and command-line access to start Biothings APIs.

```
class biothings.web.launcher.BiothingsAPILauncher(config=None)
```

Bases: object

```
get_app()
```

```
get_server()
```

```
start(port=8000)
```

```
biothings.web.launcher.BiothingsAPILauncher
```

alias of *TornadoAPILauncher*

```
class biothings.web.launcher.FastAPILauncher(config=None)
```

Bases: *BiothingsAPILauncher*

```
get_app()
```

```
class biothings.web.launcher.FlaskAPILauncher(config=None)
```

Bases: *BiothingsAPILauncher*

```

get_app()
get_server()
start(port=8000, dev=True)

class biothings.web.launcher.TornadoAPILauncher(config=None)
    Bases: BiothingsAPIBaseLauncher
        get_app()
        get_server()
        start(port=8000)
        static use_curl()
            Use curl implementation for tornado http clients. More on https://www.tornadoweb.org/en/stable/httpclient.html
biothings.web.launcher.main(app_handlers=None, app_settings=None, use_curl=False)
    Start a Biothings API Server

```

biothings.web.applications

Biothings Web Applications -

define the routes and handlers a supported web framework would consume basing on a config file, typically named *config.py*, enhanced by *biothings.web.settings.configs*.

The currently supported web frameworks are Tornado, Flask, and FastAPI.

The *biothings.web.launcher* can start the compatible HTTP servers basing on their interface. And the web applications delegate routes defined in the config file to handlers typically in *biothings.web.handlers*.

Web Framework	Interface	Handlers
Tornado	Tornado	<i>biothings.web.handlers.*</i>
Flask	WSGI	<i>biothings.web.handlers._flask</i>
FastAPI	ASGI	<i>biothings.web.handlers._fastapi</i>

biothings.web.applications.BiothingsAPI

alias of *TornadoBiothingsAPI*

class biothings.web.applications.FastAPIBiothingsAPI

Bases: *object*

classmethod get_app(config)

class biothings.web.applications.FlaskBiothingsAPI

Bases: *object*

classmethod get_app(config)

class biothings.web.applications.TornadoBiothingsAPI(*args, **kwargs)

Bases: *Application*

```
classmethod get_app(config, settings=None, handlers=None)
```

Return the tornado.web.Application defined by this config. **Additional** settings and handlers are accepted as parameters.

```
biothings.web.applications.load_class(kls)
```

biothings.web.services

biothings.web.services.query

A Programmatic Query API supporting Biothings Query Syntax.

From an architecture perspective, `biothings.web.query` is one of the data services, built on top of the `biothings.web.connections` layer, however, due to the complexity of the module, it is escalated one level in organization to simplify the overall folder structure. The features are available in `biothings.web.services.query` namespace via import.

biothings.web.services.health

```
class biothings.web.services.health.DBHealth(client)
```

Bases: object

```
check(**kwargs)
```

```
class biothings.web.services.health.ESHealth(client, payload=None)
```

Bases: `DBHealth`

```
async async_check(verbose=False)
```

```
check()
```

```
class biothings.web.services.health.MongoHealth(client)
```

Bases: `DBHealth`

```
check(**kwargs)
```

```
class biothings.web.services.health.SQLHealth(client)
```

Bases: `DBHealth`

```
check(**kwargs)
```

biothings.web.services.metadata

```
class biothings.web.services.metadata.BiothingHubMeta(**metadata)
```

Bases: `BiothingMetaProp`

```
to_dict()
```

```
class biothings.web.services.metadata.BiothingLicenses(licenses)
```

Bases: `BiothingMetaProp`

```
to_dict()
```

```
class biothings.web.services.metadata.BiothingMappings(properties)
```

Bases: `BiothingMetaProp`

```

to_dict()

class biothings.web.services.metadata.BiothingMetaProp
    Bases: object

to_dict()

class biothings.web.services.metadata.BiothingsESMetadata(indices, client)
    Bases: BiothingsMetadata
    refresh(biothing_type=None)

property types

update(biothing_type, info, count)
    Read ES index mappings for the corresponding biothing_type, Populate datasource info and field properties from mappings.

class biothings.web.services.metadata.BiothingsMetadata
    Bases: object

get_licenses(biothing_type)
get_mappings(biothing_type)
get_metadata(biothing_type)
async refresh(biothing_type)

class biothings.web.services.metadata.BiothingsMongoMetadata(collections, client)
    Bases: BiothingsMetadata
    get_licenses(biothing_type)
    get_mappings(biothing_type)
    async refresh(biothing_type)

property types

class biothings.web.services.metadata.BiothingsSQLMetadata(tables, client)
    Bases: BiothingsMetadata
    async refresh(biothing_type)

property types

biothings.web.services.namespace

class biothings.web.services.namespace.BiothingsDBProxy
    Bases: object
    Provide database-agnostic access to common biothings service components, for single database application.

configure(db)

class biothings.web.services.namespace.BiothingsNamespace(config)
    Bases: object
    biothings.web.services.namespace.load_class(kls)

```

`biothings.web.connections`

```
biothings.web.connections.get_es_client(hosts=None, async_=False, **settings)
```

Enhanced ES client initialization.

Additionally support these parameters:

async_: use AsyncElasticserach instead of Elasticsearch. aws: setup request signing and provide reasonable ES settings

to access AWS OpenSearch, by default assuming it is on HTTPS.

sniff: provide resonable default settings to enable client-side

LB to an ES cluster. this param itself is not an ES param.

```
biothings.web.connections.get_mongo_client(uri, **settings)
```

```
biothings.web.connections.get_sql_client(uri, **settings)
```

Additionally, you can reuse connecions initialized with the same parameters by getting it from the connection pools every time. Here's the connection pool interface signature:

```
class biothings.web.connections._ClientPool(client_factory, async_factory, callback=None)
```

Bases: object

```
get_async_client(uri, **settings)
```

```
get_client(uri, **settings)
```

```
static hash(config)
```

The module has already initialized connection pools for each supported databases. Directly access these pools without creating by yourselves.

```
biothings.web.connections.es = <biothings.web.connections._ClientPool object>
```

```
biothings.web.connections.sql = <biothings.web.connections._ClientPool object>
```

```
biothings.web.connections.mongo = <biothings.web.connections._ClientPool object>
```

6.7.2 Components

`biothings.web.analytics`

`biothings.web.analytics.channels`

```
class biothings.web.analytics.channels.Channel
```

Bases: object

```
handles(event)
```

```
send(event)
```

```
class biothings.web.analytics.channels.GA4Channel(measurement_id, api_secret, uid_version=1)
```

Bases: `Channel`

```
handles(event)
```

```

send(payload)
Limitations: https://developers.google.com/analytics/devguides/collection/protocol/ga4/sending-events?client\_type=gtag

class biothings.web.analytics.channels.GAChannel(tracking_id, uid_version=1)
    Bases: Channel
        handles(event)
            send(payload)
class biothings.web.analytics.channels.SlackChannel(hook_urls)
    Bases: Channel
        handles(event)
            send(message)

```

biothings.web.analytics.events

```

class biothings.web.analytics.events.Event(dict=None, /, **kwargs)
    Bases: UserDict
        to_GA4_payload(measurement_id, cid_version=1)
        to_GA_payload(tracking_id, cid_version=1)
class biothings.web.analytics.events.GAEVENT(dict=None, /, **kwargs)
    Bases: Event
        to_GA4_payload(measurement_id, cid_version=1)
        to_GA_payload(tracking_id, cid_version=1)
class biothings.web.analytics.events.Message(dict=None, /, **kwargs)
    Bases: Event
        Logical document that can be sent through services. Processable fields: title, body, url, url_text, image, image_altext Optionally define default field values below.
        DEFAULTS = {'image_altext': '<IMAGE>', 'title': 'Notification Message', 'url_text': 'View Details'}
        to_ADF()
            Generate ADF for Atlassian Jira payload. Overwrite this to build differently. https://developer.atlassian.com/cloud/jira/platform/apis/document/playground/
        to_email_payload(sendfrom, sendto)
            Build a MIMEMultipart message that can be sent as an email. https://docs.aws.amazon.com/ses/latest/DeveloperGuide/examples-send-using-smtp.html
        to_jira_payload(profile)
            Combine notification message with project profile to generate jira issue tracking ticket request payload.
        to_slack_payload()
            Generate slack webhook notification payload. https://api.slack.com/messaging/composing/layouts

```

biothings.web.analytics.notifiers

```
class biothings.web.analytics.notifiers.AnalyticsMixin(application: Application, request:  
HTTPServerRequest, **kwargs: Any)
```

Bases: RequestHandler

on_finish()

Called after the end of a request.

Override this method to perform cleanup, logging, etc. This method is a counterpart to *prepare*. **on_finish** may not produce any output, as it is called after the response has been sent to the client.

```
class biothings.web.analytics.notifiers.Notifier(settings)
```

Bases: object

broadcast(event)

biothings.web.handlers

biothings.web.handlers.base

Biothings Web Handlers

biothings.web.handlers.BaseHandler

Supports: - access to biothings namespace - monitor exceptions with Sentry

biothings.web.handlers.BaseAPIHandler

Additionally supports: - JSON and YAML payload in the request body - request arguments standardization
- multi-type output (json, yaml, html, msgpack) - standardized error response (exception -> error template)
- analytics and usage tracking (Google Analytics and AWS) - default common http headers (CORS and Cache Control)

```
class biothings.web.handlers.base.BaseAPIHandler(application: Application, request:  
HTTPServerRequest, **kwargs: Any)
```

Bases: *BaseHandler*, *AnalyticsMixin*

cache = None

cache_control_template = 'max-age={cache}, public'

format = 'json'

get_template_path()

Override to customize template path for each handler.

By default, we use the `template_path` application setting. Return `None` to load templates relative to the calling file.

initialize(cache=None)

```
kwargs = {'*': {'format': {'default': 'json', 'enum': ('json', 'yaml', 'html',  
'msgpack'), 'type': <class 'str'>}}}
```

name = '__base__'

on_finish()

This is a tornado lifecycle hook. Override to provide tracking features.

options(*args, **kwargs)**prepare()**

Called at the beginning of a request before *get/post/etc*.

Override this method to perform common initialization regardless of the request method.

Asynchronous support: Use `async def` or decorate this method with `.gen.coroutine` to make it asynchronous. If this method returns an `Awaitable` execution will not proceed until the `Awaitable` is done.

New in version 3.1: Asynchronous support.

set_cache_header(cache_value)**set_default_headers()**

Override this to set HTTP headers at the beginning of the request.

For example, this is the place to set a custom `Server` header. Note that setting such headers in the normal flow of request processing may not do what you want, since headers may be reset during error handling.

write(chunk)

Writes the given chunk to the output buffer.

To write the output to the network, use the `flush()` method below.

If the given chunk is a dictionary, we write it as JSON and set the Content-Type of the response to be `application/json`. (if you want to send JSON as a different Content-Type, call `set_header` after calling `write()`).

Note that lists are not converted to JSON because of a potential cross-site security vulnerability. All JSON output should be wrapped in a dictionary. More details at <http://haacked.com/archive/2009/06/25/json-hijacking.aspx/> and <https://github.com/facebook/tornado/issues/1009>

write_error(status_code, **kwargs)

```
from tornado.web import Finish, HTTPError
```

```
raise HTTPError(404) raise HTTPError(404, reason="document not found") raise HTTPError(404, None, {"id": "-1"}, reason="document not found") -> {
```

```
    "code": 404, "success": False, "error": "document not found" "id": "-1"
```

```
}
```

```
class biothings.web.handlers.base.BaseHandler(application: Application, request: HTTPServerRequest, **kwargs: Any)
```

Bases: `RequestHandler`

property biothings

biothings.web.handlers.query

Elasticsearch Handlers

`biothings.web.handlers.BaseESRequestHandler`

Supports: (all features above and) - access to `biothing_type` attribute - access to ES query pipeline stages
- pretty print elasticsearch exceptions - common control option `out_format`

Subclasses: - `biothings.web.handlers.MetadataSourceHandler` - `biothings.web.handlers.MetadataFieldHandler` - `myvariant.web.beacon.BeaconHandler` - `biothings.web.handlers.ESRequestHandler`

Supports: (all features above and) - common control options (`raw`, `rawquery`) - common transform options (`dotfield`, `always_list...`) - query pipeline customization hooks - single query through GET - multiple queries through POST

Subclasses: - `biothings.web.handlers.BiothingHandler` - `biothings.web.handlers.QueryHandler`

class `biothings.web.handlers.query.BaseQueryHandler`(*application: Application, request: HTTPServerRequest, **kwargs: Any*)

Bases: `BaseAPIHandler`

initialize(`biothing_type=None`, *args, **kwargs)

prepare()

Called at the beginning of a request before `get/post/etc.`

Override this method to perform common initialization regardless of the request method.

Asynchronous support: Use `async def` or decorate this method with `.gen.coroutine` to make it asynchronous. If this method returns an `Awaitable` execution will not proceed until the `Awaitable` is done.

New in version 3.1: Asynchronous support.

write(`chunk`)

Writes the given chunk to the output buffer.

To write the output to the network, use the `flush()` method below.

If the given chunk is a dictionary, we write it as JSON and set the Content-Type of the response to be `application/json`. (if you want to send JSON as a different Content-Type, call `set_header` after calling `write()`).

Note that lists are not converted to JSON because of a potential cross-site security vulnerability. All JSON output should be wrapped in a dictionary. More details at <http://haacked.com/archive/2009/06/25/json-hijacking.aspx/> and <https://github.com/facebook/tornado/issues/1009>

class `biothings.web.handlers.query.BiothingHandler`(*application: Application, request: HTTPServerRequest, **kwargs: Any*)

Bases: `BaseQueryHandler`

Biothings Annotation Endpoint

URL pattern examples:

`/{pre}/{ver}/{typ}/? /{pre}/{ver}/{typ}/([^\+])/?`

queries a term against a pre-determined field that represents the id of a document, like `_id` and `db-snp.rsid`

GET -> `{...}` or `[{...}, ...]` POST -> `[{...}, ...]`

```

async get(**kwargs)
    name = 'annotation'

async post(**kwargs)

class biothings.web.handlers.query.MetadataFieldHandler(application: Application, request:
                                                               HTTPServerRequest, **kwargs: Any)
    Bases: BaseQueryHandler
    GET /metadata/fields

    async get()

    kwargs = {'*': {'format': {'default': 'json', 'enum': ('json', 'yaml', 'html',
        'msgpack'), 'type': <class 'str'>}}, 'GET': {'prefix': {'default': None, 'type':
        <class 'str'>}, 'raw': {'default': False, 'type': <class 'bool'>}, 'search':
        {'default': None, 'type': <class 'str'>}}}

    name = 'fields'

class biothings.web.handlers.query.MetadataSourceHandler(application: Application, request:
                                                               HTTPServerRequest, **kwargs: Any)
    Bases: BaseQueryHandler
    GET /metadata

    extras(_meta)
        Override to add app specific metadata.

    async get()

    kwargs = {'*': {'format': {'default': 'json', 'enum': ('json', 'yaml', 'html',
        'msgpack'), 'type': <class 'str'>}}, 'GET': {'dev': {'default': False, 'type':
        <class 'bool'>}, 'raw': {'default': False, 'type': <class 'bool'>}}}

    name = 'metadata'

class biothings.web.handlers.query.QueryHandler(application: Application, request:
                                                               HTTPServerRequest, **kwargs: Any)
    Bases: BaseQueryHandler
    Biothings Query Endpoint

    URL pattern examples:
        /{pre}/{ver}/{typ}/query/? /{pre}/{ver}//query/?

        GET -> {...} POST -> [...], ...]

    async get(**kwargs)
        name = 'query'

    async post(**kwargs)

```

biothings.web.handlers.services

```
class biothings.web.handlers.services.APISpecificationHandler(application: Application, request:  
HTTPServerRequest, **kwargs:  
Any)
```

Bases: *BaseAPIHandler*

get()

```
class biothings.web.handlers.services.FrontPageHandler(application: Application, request:  
HTTPServerRequest, **kwargs: Any)
```

Bases: *BaseHandler*

get()

get_template_path()

Override to customize template path for each handler.

By default, we use the `template_path` application setting. Return None to load templates relative to the calling file.

```
class biothings.web.handlers.services.StatusHandler(application: Application, request:  
HTTPServerRequest, **kwargs: Any)
```

Bases: *BaseHandler*

Web service health check

async get()

head()

biothings.web.options

biothings.web.options.manager

Request Argument Standardization

```
class biothings.web.options.manager.Converter(**kwargs)
```

Bases: *object*

A generic HTTP request argument processing unit. Only perform one level of validation at this moment. The strict switch controls the type conversion rules.

convert(value)

convert_to(value, to_type)

static str_to_bool(val)

Interpret string representation of bool values.

str_to_int(val)

Convert a numerical string to an integer.

static str_to_list(val)

Cast Biothings-style str to list.

classmethod subclasses(kwargs)

```
static to_type(val, type_)

Native type casting in Python. Fallback approach for type casting.

translate(value)

class biothings.web.options.manager.Existentialist(defdict)
Bases: object

Describes the requirement of the existance of an argument. {

    “default”: <object>, “required”: <bool>,
}

inquire(obj)

class biothings.web.options.manager.FormArgCvter(**kwargs)
Bases: Converter

Dedicated argument converter for HTTP body arguments. Additionally support JSON serialization format as values. Correspond to arguments received in tornado from

    RequestHandler.get_body_argument

See https://www.tornadoweb.org/en/stable/web.html

convert_to(value, to_type)

class biothings.web.options.manager.JsonArgCvter(**kwargs)
Bases: Converter

Dedicated argument converter for JSON HTTP bodys. Here it is used for dict JSON objects, with their first level keys considered as parameters and their values considered as arguments to process.

May correspond to this tornado implementation: https://www.tornadoweb.org/en/stable/web.html#input

convert_to(value, to_type)

to_type(val, type_)

Native type casting in Python. Fallback approach for type casting.

class biothings.web.options.manager.Locator(defdict)
Bases: object

Describes the location of an argument in ReqArgs. {

    “keyword”: <str>, “path”: <int or str>, “alias”: <str or [<str>, …]>
}

lookin(location)
lookin(path: Path)
lookin(dic: dict)

Find an argument in the specified location. Use directions indicated in this locator.

class biothings.web.options.manager.Option(*args, **kwargs)
Bases: UserDict

A parameter for end applications to consume. Find the value of it in the desired location.

For example: {

    “keyword”: “q”, “location”: (“query”, “form”, “json”), “default”: “__all__”, “type”: “str”
```

```
}

parse(reqargs)

exception biothings.web.options.manager.OptionError(reason=None, **kwargs)
    Bases: ValueError

simplify()

class biothings.web.options.manager.OptionSet(*args, **kwargs)
    Bases: UserDict

    A collection of options that a specific endpoint consumes. Divided into groups and by the request methods.  

    For example: {  

        “*”:{“raw”:{...},“size”:{...},“dotfield”:{...}},      “GET”:{“q”:{...},“from”:{...},“sort”:{...}},  

        “POST”:{“q”:{...},“scopes”:{...}}  

    }  

parse(method, reqargs)
    Parse a HTTP request, represented by its method and args, with this OptionSet and return an attribute dictionary.  

setup()
    Apply the wildcard method configurations dict. Must call this method after changes to this object.  

class biothings.web.options.manager.OptionsManager(dict=None, /, **kwargs)
    Bases: UserDict

    A collection of OptionSet(s) that makes up an application. Provide an interface to setup and serialize.  

    Example: {  

        “annotation”: {“*”: {...}, “GET”: {...}, “POST”: {...}}, “query”: {“*”: {...}, “GET”: {...},  

        “POST”: {...}}, “metadata”: {“GET”: {...}, “POST”: {...}}  

    }  

add(name, optionset, groups=())
log()

class biothings.web.options.manager.PathArgCvter(**kwargs)
    Bases: Converter

    Dedicated argument converter for path arguments. Correspond to arguments received in tornado for  

        RequestHandler.path_args RequestHandler.path_kwargs  

    See https://www.tornadoweb.org/en/stable/web.html  

class biothings.web.options.manager.QueryArgCvter(**kwargs)
    Bases: Converter

    Dedicated argument converter for url query arguments. Correspond to arguments received in tornado from  

        RequestHandler.get_query_argument  

    See https://www.tornadoweb.org/en/stable/web.html  

classmethod str_to_bool(val)
    Biothings-style str to bool interpretation
```

```

class biothings.web.options.manager.ReqArgs(path=None, query=None, form=None, json_=None)
    Bases: object

class Path(args=None, kwargs=None)
    Bases: object

        lookup(locator, order=None, src=False)

class biothings.web.options.manager.ReqResult
    Bases: dotdict

class biothings.web.options.manager.Validator(defdict)
    Bases: object

    Describes the requirement of the existance of an argument. {

        “enum”: <container>, “max”: <int>, “min”: <int>, “date_format”: <str>,
    }

    validate(obj)

```

biothings.web.options.openapi

```

class biothings.web.options.openapi.OpenAPIContactContext(parent)
    Bases: _ChildContext

    ATTRIBUTE_FIELDS = {'email': 'email', 'name': 'name', 'url': 'url'}

    EXTENSION = True

    email(v)
        Set email field

    name(v)
        Set name field

    subclasses: MutableMapping[str, Type[_ChildContext]] = {'OpenAPIContactContext':
<class 'biothings.web.options.openapi.OpenAPIContactContext'>, 'OpenAPIContext':
<class 'biothings.web.options.openapi.OpenAPIContext'>,
'OpenAPIExternalDocsContext': <class
'biothings.web.options.openapi.OpenAPIExternalDocsContext'>, 'OpenAPIInfoContext':
<class 'biothings.web.options.openapi.OpenAPIInfoContext'>, 'OpenAPILicenseContext':
<class 'biothings.web.options.openapi.OpenAPILicenseContext'>, 'OpenAPIOperation':
<class 'biothings.web.options.openapi.OpenAPIOperation'>, 'OpenAPIParameterContext':
<class 'biothings.web.options.openapi.OpenAPIParameterContext'>,
'OpenAPIPathItemContext': <class
'biothings.web.options.openapi.OpenAPIPathItemContext'>, '_BaseContext': <class
'biothings.web.options.openapi._BaseContext'>, '_ChildContext': <class
'biothings.web.options.openapi._ChildContext'>, '_HasDescription': <class
'biothings.web.options.openapi._HasDescription'>, '_HasExternalDocs': <class
'biothings.web.options.openapi._HasExternalDocs'>, '_HasParameters': <class
'biothings.web.options.openapi._HasParameters'>, '_HasSummary': <class
'biothings.web.options.openapi._HasSummary'>, '_HasTags': <class
'biothings.web.options.openapi._HasTags'>}

```

```
url(v)
Set url field

class biothings.web.options.openapi.OpenAPIContext
Bases: _HasExternalDocs
CHILD_CONTEXTS = {'info': ('OpenAPIInfoContext', 'info')}

EXTENSION = True

info(**kwargs)
Set info Create OpenAPIInfoContext and set info

path(path: str, summary: str | None = None, description: str | None = None)

server(url: str, description: str | None = None)

subclasses: MutableMapping[str, Type[_ChildContext]] = {'OpenAPIContactContext':
<class 'biothings.web.options.openapi.OpenAPIContactContext'>, 'OpenAPIContext':
<class 'biothings.web.options.openapi.OpenAPIContext'>,
'OpenAPIExternalDocsContext': <class
'biothings.web.options.openapi.OpenAPIExternalDocsContext'>, 'OpenAPIInfoContext':
<class 'biothings.web.options.openapi.OpenAPIInfoContext'>, 'OpenAPILicenseContext':
<class 'biothings.web.options.openapi.OpenAPILicenseContext'>, 'OpenAPIOperation':
<class 'biothings.web.options.openapi.OpenAPIOperation'>, 'OpenAPIParameterContext':
<class 'biothings.web.options.openapi.OpenAPIParameterContext'>,
'OpenAPIPathItemContext': <class
'biothings.web.options.openapi.OpenAPIPathItemContext'>, '_BaseContext': <class
'biothings.web.options.openapi._BaseContext'>, '_ChildContext': <class
'biothings.web.options.openapi._ChildContext'>, '_HasDescription': <class
'biothings.web.options.openapi._HasDescription'>, '_HasExternalDocs': <class
'biothings.web.options.openapi._HasExternalDocs'>, '_HasParameters': <class
'biothings.web.options.openapi._HasParameters'>, '_HasSummary': <class
'biothings.web.options.openapi._HasSummary'>, '_HasTags': <class
'biothings.web.options.openapi._HasTags'>}

biothings.web.options.openapi.OpenAPIDocumentBuilder
alias of OpenAPIContext

class biothings.web.options.openapi.OpenAPIExternalDocsContext(parent)
Bases: _ChildContext, _HasDescription
ATTRIBUTE_FIELDS = {'url': 'url'}

EXTENSION = True
```

```

subclasses: MutableMapping[str, Type[_ChildContext]] = {'OpenAPIContactContext':
<class 'biothings.web.options.openapi.OpenAPIContactContext'>, 'OpenAPIContext':
<class 'biothings.web.options.openapi.OpenAPIContext'>,
'OpenAPIExternalDocsContext': <class
'biothings.web.options.openapi.OpenAPIExternalDocsContext'>, 'OpenAPIInfoContext':
<class 'biothings.web.options.openapi.OpenAPIInfoContext'>, 'OpenAPILicenseContext':
<class 'biothings.web.options.openapi.OpenAPILicenseContext'>, 'OpenAPIOperation':
<class 'biothings.web.options.openapi.OpenAPIOperation'>, 'OpenAPIParameterContext':
<class 'biothings.web.options.openapi.OpenAPIParameterContext'>,
'OpenAPIPathItemContext': <class
'biothings.web.options.openapi.OpenAPIPathItemContext'>, '_BaseContext': <class
'biothings.web.options.openapi._BaseContext'>, '_ChildContext': <class
'biothings.web.options.openapi._ChildContext'>, '_HasDescription': <class
'biothings.web.options.openapi._HasDescription'>, '_HasExternalDocs': <class
'biothings.web.options.openapi._HasExternalDocs'>, '_HasParameters': <class
'biothings.web.options.openapi._HasParameters'>, '_HasSummary': <class
'biothings.web.options.openapi._HasSummary'>, '_HasTags': <class
'biothings.web.options.openapi._HasTags'>}

url(v)
    Set url field

class biothings.web.options.openapi.OpenAPIInfoContext(parent)
    Bases: _ChildContext, _HasDescription

    ATTRIBUTE_FIELDS = {'terms_of_service': 'termsOfService', 'title': 'title',
    'version': 'version'}

    CHILD_CONTEXTS = {'contact': ('OpenAPIContactContext', 'contact'), 'license':
    ('OpenAPILicenseContext', 'license')}

    EXTENSION = True

    contact(**kwargs)
        Set contact Create OpenAPIContactContext and set contact

    license(**kwargs)
        Set license Create OpenAPILicenseContext and set license

subclasses: MutableMapping[str, Type[_ChildContext]] = {'OpenAPIContactContext':
<class 'biothings.web.options.openapi.OpenAPIContactContext'>, 'OpenAPIContext':
<class 'biothings.web.options.openapi.OpenAPIContext'>,
'OpenAPIExternalDocsContext': <class
'biothings.web.options.openapi.OpenAPIExternalDocsContext'>, 'OpenAPIInfoContext':
<class 'biothings.web.options.openapi.OpenAPIInfoContext'>, 'OpenAPILicenseContext':
<class 'biothings.web.options.openapi.OpenAPILicenseContext'>, 'OpenAPIOperation':
<class 'biothings.web.options.openapi.OpenAPIOperation'>, 'OpenAPIParameterContext':
<class 'biothings.web.options.openapi.OpenAPIParameterContext'>,
'OpenAPIPathItemContext': <class
'biothings.web.options.openapi.OpenAPIPathItemContext'>, '_BaseContext': <class
'biothings.web.options.openapi._BaseContext'>, '_ChildContext': <class
'biothings.web.options.openapi._ChildContext'>, '_HasDescription': <class
'biothings.web.options.openapi._HasDescription'>, '_HasExternalDocs': <class
'biothings.web.options.openapi._HasExternalDocs'>, '_HasParameters': <class
'biothings.web.options.openapi._HasParameters'>, '_HasSummary': <class
'biothings.web.options.openapi._HasSummary'>, '_HasTags': <class
'biothings.web.options.openapi._HasTags'>}

```

```
terms_of_service(v)
    Set termsOfService field

title(v)
    Set title field

version(v)
    Set version field

class biothings.web.options.openapi.OpenAPILicenseContext(parent)
    Bases: _ChildContext
    ATTRIBUTE_FIELDS = {'name': 'name', 'url': 'url'}

    EXTENSION = True

    name(v)
        Set name field

    subclasses: MutableMapping[str, Type[_ChildContext]] = {'OpenAPIContactContext':
<class 'biothings.web.options.openapi.OpenAPIContactContext'>, 'OpenAPIContext':
<class 'biothings.web.options.openapi.OpenAPIContext'>,
'OpenAPIExternalDocsContext': <class
'biothings.web.options.openapi.OpenAPIExternalDocsContext'>, 'OpenAPIInfoContext':
<class 'biothings.web.options.openapi.OpenAPIInfoContext'>, 'OpenAPILicenseContext':
<class 'biothings.web.options.openapi.OpenAPILicenseContext'>, 'OpenAPIOperation':
<class 'biothings.web.options.openapi.OpenAPIOperation'>, 'OpenAPIParameterContext':
<class 'biothings.web.options.openapi.OpenAPIParameterContext'>,
'OpenAPIPathItemContext': <class
'biothings.web.options.openapi.OpenAPIPathItemContext'>, '_BaseContext': <class
'biothings.web.options.openapi._BaseContext'>, '_ChildContext': <class
'biothings.web.options.openapi._ChildContext'>, '_HasDescription': <class
'biothings.web.options.openapi._HasDescription'>, '_HasExternalDocs': <class
'biothings.web.options.openapi._HasExternalDocs'>, '_HasParameters': <class
'biothings.web.options.openapi._HasParameters'>, '_HasSummary': <class
'biothings.web.options.openapi._HasSummary'>, '_HasTags': <class
'biothings.web.options.openapi._HasTags'>}

    url(v)
        Set url field

class biothings.web.options.openapi.OpenAPIOperation(parent)
    Bases: _ChildContext, _HasSummary, _HasExternalDocs, _HasTags, _HasDescription,
_HasParameters
    ATTRIBUTE_FIELDS = {'operation_id': 'operationId'}

    EXTENSION = True

    operation_id(v)
        Set operationId field
```

```

subclasses: MutableMapping[str, Type[_ChildContext]] = {'OpenAPIContactContext':
<class 'biothings.web.options.openapi.OpenAPIContactContext'>, 'OpenAPIContext':
<class 'biothings.web.options.openapi.OpenAPIContext'>,
'OpenAPIExternalDocsContext': <class
'biothings.web.options.openapi.OpenAPIExternalDocsContext'>, 'OpenAPIInfoContext':
<class 'biothings.web.options.openapi.OpenAPIInfoContext'>, 'OpenAPILicenseContext':
<class 'biothings.web.options.openapi.OpenAPILicenseContext'>, 'OpenAPIOperation':
<class 'biothings.web.options.openapi.OpenAPIOperation'>, 'OpenAPIParameterContext':
<class 'biothings.web.options.openapi.OpenAPIParameterContext'>,
'OpenAPIPathItemContext': <class
'biothings.web.options.openapi.OpenAPIPathItemContext'>, '_BaseContext': <class
'biothings.web.options.openapi._BaseContext'>, '_ChildContext': <class
'biothings.web.options.openapi._ChildContext'>, '_HasDescription': <class
'biothings.web.options.openapi._HasDescription'>, '_HasExternalDocs': <class
'biothings.web.options.openapi._HasExternalDocs'>, '_HasParameters': <class
'biothings.web.options.openapi._HasParameters'>, '_HasSummary': <class
'biothings.web.options.openapi._HasSummary'>, '_HasTags': <class
'biothings.web.options.openapi._HasTags'>}

class biothings.web.options.openapi.OpenAPIParameterContext(parent, name: str, in_: str, required:
bool)

Bases: _ChildContext, _HasDescription

ATTRIBUTE_FIELDS = {'allow_empty': 'allowEmptyValue', 'allow_reserved':
'allowReserved', 'deprecated': 'deprecated', 'explode': 'explode', 'schema':
'schema', 'style': 'style'}

EXTENSION = True

allow_empty(v)
    Set allowEmptyValue field

allow_reserved(v)
    Set allowReserved field

deprecated(v)
    Set deprecated field

explode(v)
    Set explode field

schema(v)
    Set schema field

style(v)
    Set style field

```

```
subclasses: MutableMapping[str, Type[_ChildContext]] = {'OpenAPIContactContext':  
<class 'biothings.web.options.openapi.OpenAPIContactContext'>, 'OpenAPIContext':  
<class 'biothings.web.options.openapi.OpenAPIContext'>,  
'OpenAPIExternalDocsContext': <class  
'biothings.web.options.openapi.OpenAPIExternalDocsContext'>, 'OpenAPIInfoContext':  
<class 'biothings.web.options.openapi.OpenAPIInfoContext'>, 'OpenAPILicenseContext':  
<class 'biothings.web.options.openapi.OpenAPILicenseContext'>, 'OpenAPIOperation':  
<class 'biothings.web.options.openapi.OpenAPIOperation'>, 'OpenAPIParameterContext':  
<class 'biothings.web.options.openapi.OpenAPIParameterContext'>,  
'OpenAPIPathItemContext': <class  
'biothings.web.options.openapi.OpenAPIPathItemContext'>, '_BaseContext': <class  
'biothings.web.options.openapi._BaseContext'>, '_ChildContext': <class  
'biothings.web.options.openapi._ChildContext'>, '_HasDescription': <class  
'biothings.web.options.openapi._HasDescription'>, '_HasExternalDocs': <class  
'biothings.web.options.openapi._HasExternalDocs'>, '_HasParameters': <class  
'biothings.web.options.openapi._HasParameters'>, '_HasSummary': <class  
'biothings.web.options.openapi._HasSummary'>, '_HasTags': <class  
'biothings.web.options.openapi._HasTags'>}  
  
type(typ: str, **kwargs)  
  
class biothings.web.options.openapi.OpenAPIPathItemContext(parent)  
Bases: _ChildContext, _HasSummary, _HasDescription, _HasParameters  
  
CHILD_CONTEXTS = {'delete': ('OpenAPIOperation', 'delete'), 'get':  
('OpenAPIOperation', 'get'), 'head': ('OpenAPIOperation', 'head'), 'options':  
('OpenAPIOperation', 'options'), 'patch': ('OpenAPIOperation', 'patch'), 'post':  
('OpenAPIOperation', 'post'), 'put': ('OpenAPIOperation', 'put'), 'trace':  
('OpenAPIOperation', 'trace')}  
  
EXTENSION = True  
  
delete(**kwargs)  
    Set delete Create OpenAPIOperation and set delete  
get(**kwargs)  
    Set get Create OpenAPIOperation and set get  
head(**kwargs)  
    Set head Create OpenAPIOperation and set head  
http_method = 'trace'  
options(**kwargs)  
    Set options Create OpenAPIOperation and set options  
patch(**kwargs)  
    Set patch Create OpenAPIOperation and set patch  
post(**kwargs)  
    Set post Create OpenAPIOperation and set post  
put(**kwargs)  
    Set put Create OpenAPIOperation and set put
```

```

subclasses: MutableMapping[str, Type[_ChildContext]] = {'OpenAPIContactContext':
<class 'biothings.web.options.openapi.OpenAPIContactContext'>, 'OpenAPIContext':
<class 'biothings.web.options.openapi.OpenAPIContext'>,
'OpenAPIExternalDocsContext': <class
'biothings.web.options.openapi.OpenAPIExternalDocsContext'>, 'OpenAPIInfoContext':
<class 'biothings.web.options.openapi.OpenAPIInfoContext'>, 'OpenAPILicenseContext':
<class 'biothings.web.options.openapi.OpenAPILicenseContext'>, 'OpenAPIOperation':
<class 'biothings.web.options.openapi.OpenAPIOperation'>, 'OpenAPIParameterContext':
<class 'biothings.web.options.openapi.OpenAPIParameterContext'>,
'OpenAPIPathItemContext': <class
'biothings.web.options.openapi.OpenAPIPathItemContext'>, '_BaseContext': <class
'biothings.web.options.openapi._BaseContext'>, '_ChildContext': <class
'biothings.web.options.openapi._ChildContext'>, '_HasDescription': <class
'biothings.web.options.openapi._HasDescription'>, '_HasExternalDocs': <class
'biothings.web.options.openapi._HasExternalDocs'>, '_HasParameters': <class
'biothings.web.options.openapi._HasParameters'>, '_HasSummary': <class
'biothings.web.options.openapi._HasSummary'>, '_HasTags': <class
'biothings.web.options.openapi._HasTags'>}

trace(**kwargs)
    Set trace Create OpenAPIOperation and set trace

```

biothings.web.query**biothings.web.query.builder**

Biothings Query Builder

Turn the biothings query language to that of the database. The interface contains a query term (q) and query options.

Depending on the underlying database choice, the data type of the query term and query options vary. At a minimum, a query builder should support:

q: str, a query term,

when not provided, always perform a match all query. when provided as an empty string, always match none.

options: dotdict, optional query options.

scopes: list[str], the fields to look for the query term.

the meaning of scopes being an empty list or a None object/not provided is controlled by specific class implementations or not defined.

source: list[str], fields to return in the result. size: int, maximum number of hits to return. **from**: int, starting index of result to return. sort: str, customized sort keys for result list

aggs: str, customized aggregation string. post_filter: str, when provided, the search hits are filtered after the aggregations are calculated. facet_size: int, maximum number of agg results.

```
class biothings.web.query.builder.ESQueryBuilder(user_query=None, scopes_regexs=(),
                                                 scopes_default=('_id',), allow_random_query=True,
                                                 allow_nested_query=False, metadata=None)
```

Bases: object

Build an Elasticsearch query with elasticsearch-dsl.

apply_extras(*search, options*)

Process non-query options and customize their behaviors. Customized aggregation syntax string is translated here.

build(*q=None, **options*)

Build a query according to q and options. This is the public method called by API handlers.

Regarding scopes:

scopes: [str] nonempty, match query. scopes: NoneType, or [], no scope, so query string query.

Additionally support these options:

explain: include es scoring information userquery: customized function to interpret q

- **additional keywords are passed through as es keywords**

for example: ‘explain’, ‘version’ ...

- **multi-search is supported when q is a list. all queries**

are built individually and then sent in one request.

default_match_query(*q, scopes, options*)

Override this to customize default match query. By default it implements a multi_match query.

default_string_query(*q, options*)

Override this to customize default string query. By default it implements a query string query.

class biothings.web.query.builder.ESScrollID(*seq: object*)

Bases: UserString

class biothings.web.query.builder.ESUserQuery(*path*)

Bases: object

get_filter(*named_query*)

get_query(*named_query, **kwargs*)

has_filter(*named_query*)

has_query(*named_query*)

property logger

class biothings.web.query.builder.Group(*term, scopes*)

Bases: tuple

Create new instance of Group(term, scopes)

scopes

Alias for field number 1

term

Alias for field number 0

class biothings.web.query.builder.MongoQueryBuilder(*default_scopes=('_id',)*)

Bases: object

build(*q, **options*)

```

class biothings.web.query.builder.QStringParser(default_scopes=('id',),
                                                patterns=((('(?P<scope>\w+):(?P<term>[^:]+)'),
                                                ()),), gpnames=('term', 'scope'))

Bases: object

parse(q)

class biothings.web.query.builder.Query(term, scopes)
Bases: tuple
Create new instance of Query(term, scopes)

scopes
    Alias for field number 1

term
    Alias for field number 0

exception biothings.web.query.builder.RawQueryInterrupt(data)
Bases: Exception

class biothings.web.query.builder.SQLQueryBuilder(tables, default_scopes=('id',), default_limit=10)
Bases: object

build(q, **options)

```

biothings.web.query.engine

Search Execution Engine

Take the output of the query builder and feed to the corresponding database engine. This stage typically resolves the db destination from a biothing_type and applies presentation and/or networking parameters.

Example:

```
>>> from biothings.web.query import ESQueryBackend
>>> from elasticsearch import Elasticsearch
>>> from elasticsearch_dsl import Search
```

```
>>> backend = ESQueryBackend(Elasticsearch())
>>> backend.execute(Search().query("match", _id="1017"))
```

```
>>> _["hits"]["hits"][0]["_source"].keys()
dict_keys(['taxid', 'symbol', 'name', ... ])
```

```

class biothings.web.query.engine.AsyncESQueryBackend(client, indices=None, scroll_time='1m',
                                                       scroll_size=1000, multisearch_concurrency=5,
                                                       total_hits_as_int=True)

```

Bases: *ESQueryBackend*

Execute an Elasticsearch query

```
async execute(query, **options)
```

Execute the corresponding query. Must return an awaitable. May override to add more. Handle uncaught exceptions.

Options:

fetch_all: also return a scroll_id for this query (default: false) biotesting_type: which type's corresponding indices to query (default in config.py)

```
class biothings.web.query.engine.ESQueryBackend(client, indices=None)
```

Bases: object

```
adjust_index(original_index, query, **options)
```

Override to get specific ES index.

```
execute(query, **options)
```

```
exception biothings.web.query.engine.EndScrollInterrupt
```

Bases: ResultInterrupt

```
class biothings.web.query.engine.MongoQueryBackend(client, collections)
```

Bases: object

```
execute(query, **options)
```

```
exception biothings.web.query.engine.RawResultInterrupt(data)
```

Bases: ResultInterrupt

```
exception biothings.web.query.engine.ResultInterrupt(data)
```

Bases: Exception

```
class biothings.web.query.engine.SQLQueryBackend(client)
```

Bases: object

```
execute(query, **options)
```

biothings.web.query.formatter

Search Result Formatter

Transform the raw query result into consumption-friendly structures by possibly removing from, adding to, and/or flattening the raw response from the database engine for one or more individual queries.

```
class biothings.web.query.formatter.Doc(dict=None, /, **kwargs)
```

Bases: FormatterDict

```
{  
    "_id": ... , "_score": ... , ...  
}
```

```
class biothings.web.query.formatter.ESResultFormatter(licenses=None, license_transform=None,  
                                                    field_notes=None, excluded_keys=())
```

Bases: ResultFormatter

Class to transform the results of the Elasticsearch query generated prior in the pipeline. This contains the functions to extract the final document from the elasticsearch query result in **Elasticsearch Query**. This also contains the code to flatten a document etc.

```
transform(response, **options)
```

Transform the query response to a user-friendly structure. Mainly deconstruct the elasticsearch response structure and hand over to transform_doc to apply the options below.

Options:

generic transformations for dictionaries # _____ dotfield: flatten a dictionary using dotfield notation _sorted: sort keys alphabetically in ascending order always_list: ensure the fields specified are lists or wrapped in a list allow_null: ensure the fields specified are present in the result,

the fields may be provided as type None or [].

additional multisearch result transformations # _____ template: base dict for every result, for example: {“success”: true} templates: a different base for every result, replaces the setting above template_hit: a dict to update every positive hit result, default: {“found”: true} template_miss: a dict to update every query with no hit, default: {“found”: false}

document format and content management # _____ biothing_type: result document type to apply customized transformation.

for example, add license field basing on document type’s metadata.

one: return the individual document if there’s only one hit. ignore this setting

if there are multiple hits. return None if there is no hit. this option is not effective when aggregation results are also returned in the same query.

native: bool, if the returned result is in python primitive types. version: bool, if _version field is kept. score: bool, if _score field is kept. with_total: bool, if True, the response will include max_total documents,

and a message to tell how many query terms return greater than the max_size of hits. The default is False. An example when with_total is True: {

```
‘max_total’: 100, ‘msg’: ‘12 query terms return > 1000 hits, using from=1000 to retrieve
the remaining hits’, ‘hits’: [...]
```

}

transform_aggs(res)

Transform the aggregations field and make it more presentable. For example, these are the fields of a two level nested aggregations:

```
aggregations.<term>.doc_count_error_upper_bound aggregations.<term>.sum_other_doc_count
aggregations.<term>.buckets.key    aggregations.<term>.buckets.key_as_string    aggregations.<term>.buckets.doc_count aggregations.<term>.buckets.<nested_term>.* (recursive)
```

After the transformation, we’ll have:

```
facets.<term>._type    facets.<term>.total    facets.<term>.missing    facets.<term>.other
facets.<term>.terms.count    facets.<term>.terms.term    facets.<term>.terms.<nested_term>.*
(recursive)
```

Note the first level key change doesn’t happen here.

transform_hit(path, doc, options)

Transform an individual search hit result. By default add licenses for the configured fields.

If a source has a license url in its metadata, Add “_license” key to the corresponding fields. Support dot field representation field alias.

If we have the following settings in web_config.py

```
LICENSE_TRANSFORM = {
    “exac_nontcga”: “exac”, “snpeff.ann”: “snpeff”}
```

},

Then GET /v1/variant/chr6:g.38906659G>A should look like: {

```
“exac”: {  
    “_license”: “http://bit.ly/2H9c4hg”, “af”: 0.00002471},  
  
“exac_nontcga”: {  
    “_license”: “http://bit.ly/2H9c4hg”, ← “af”: 0.00001883}, ...
```

} And GET /v1/variant/chr14:g.35731936G>C could look like: {

```
“snpeff”: {  
    “_license”: “http://bit.ly/2suyRKt”, “ann”: [{“_license”: “http://bit.ly/2suyRKt”, ←  
        “effect”: “intron_variant”, “feature_id”: “NM_014672.3”, ...}, {“_license”: “http://bit.  
ly/2suyRKt”, ← “effect”: “intron_variant”, “feature_id”: “NM_001256678.1”, ...},  
    ...]  
}, ...  
}
```

The arrow marked fields would not exist without the setting lines.

transform_mapping(mapping, prefix=None, search=None)

Transform Elasticsearch mapping definition to user-friendly field definitions metadata results.

trasform_jmespath(path: str, doc, options) → None

Transform any target field in doc using jmespath query syntax. The jmespath query parameter value should have the pattern of “<target_list_fieldname>|<jmespath_query_expression>” <target_list_fieldname> can be any sub-field of the input doc using dot notation, e.g. “aaa.bbb”.

If empty or “.”, it will be the root field.

The flexible jmespath syntax allows to filter/transform any nested objects in the input doc on the fly. The output of the jmespath transformation will then be used to replace the original target field value. ... rubric:: Examples

- **filtering an array sub-field**

```
jmespath=tags[?name==`Metadata`] # filter tags array by name field jmes-  
path=aaa.bbb[?(sub_a==`val_a`||sub_a==`val_aa`)%26%26sub_b==`val_b`] # use %26%26 for  
&&
```

static traverse(obj, leaf_node=False)

Output path-dictionary pairs. For example, input: {

```
‘exac_nontcga’: {‘af’: 0.00001883}, ‘gnomad_exome’: {‘af’: {‘af’: 0.0000119429, ‘af_afr’:  
0.000123077}}, ‘snpeff’: {‘ann’: [{‘effect’: ‘intron_variant’,  
‘feature_id’: ‘NM_014672.3’}, {‘effect’: ‘intron_variant’, ‘feature_id’:  
‘NM_001256678.1’}]}
```

} will be translated to a generator: (

```
(“exac_nontcga”, {“af”: 0.00001883}), (“gnomad_exome.af”, {“af”: 0.0000119429, “af_afr”:  
0.000123077}), (“gnomad_exome”, {“af”: {“af”: 0.0000119429, “af_afr”: 0.000123077}}),  
 (“snpeff.ann”, {“effect”: “intron_variant”, “feature_id”: “NM_014672.3”}), (“snpeff.ann”, {“ef-  
fect”: “intron_variant”, “feature_id”: “NM_001256678.1”}), (“snpeff.ann”, [{...}, {...}]),  
 (“snpeff”, {“ann”: [{...}, {...}]})}, (‘, {‘exac_nontcga’: {...}, ‘gnomad_exome’: {...},  
 ‘snpeff’: {...}}))
```

) or when traversing leaf nodes: (

```

        ('exac_nontcga.af', 0.00001883), ('gnomad_exome.af.af', 0.0000119429), ('gno-
        mad_exome.af.af_afr', 0.000123077), ('snpeff.ann.effect', 'intron_variant'),
        ('snpeff.ann.feature_id', 'NM_014672.3'), ('snpeff.ann.effect', 'intron_variant'),
        ('snpeff.ann.feature_id', 'NM_001256678.1')
    )

class biothings.web.query.formatter.FormatterDict(dict=None, /, **kwargs)
    Bases: UserDict
    collapse(key)
    exclude(keys)
    include(keys)
    wrap(key, cls)

class biothings.web.query.formatter.Hits(dict=None, /, **kwargs)
    Bases: FormatterDict
    {
        "total": ... , "hits": [
            { ... }, { ... }, ...
        ]
    }

class biothings.web.query.formatter.MongoResultFormatter
    Bases: ResultFormatter
    transform(result, **options)

class biothings.web.query.formatter.ResultFormatter
    Bases: object
    transform(response)
    transform_mapping(mapping, prefix=None, search=None)

exception biothings.web.query.formatter.ResultFormatterException
    Bases: Exception

class biothings.web.query.formatter.SQLResultFormatter
    Bases: ResultFormatter
    transform(result, **options)

```

biothings.web.query.pipeline

```

class biothings.web.query.pipeline.AsyncESQueryPipeline(builder, backend, formatter, **settings)
    Bases: QueryPipeline
    async fetch(**kwargs)
    async search(**kwargs)

```

```
class biothings.web.query.pipeline.ESQueryPipeline(builder=None, backend=None, formatter=None,
                                                 *args, **kwargs)

    Bases: QueryPipeline

    fetch(id, **options)

    search(q, **options)

class biothings.web.query.pipeline.MongoQueryPipeline(builder, backend, formatter, **settings)

    Bases: QueryPipeline

class biothings.web.query.pipeline.QueryPipeline(builder, backend, formatter, **settings)

    Bases: object

    fetch(id, **options)

    search(q, **options)

exception biothings.web.query.pipeline.QueryPipelineException(code: int = 500, summary: str = '',
                                                               details: object = None)

    Bases: Exception

    code: int = 500

    details: object = None

    summary: str = ''

exception biothings.web.query.pipeline.QueryPipelineInterrupt(data)

    Bases: QueryPipelineException

class biothings.web.query.pipeline.SQLQueryPipeline(builder, backend, formatter, **settings)

    Bases: QueryPipeline

biothings.web.query.pipeline.capturesESExceptions(func)
```

biothings.web.settings

biothings.web.settings.configs

```
class biothings.web.settings.configs.ConfigModule(config=None, parent=None, validators=(),
                                                **kwargs)
```

Bases: object

A wrapper for the settings that configure the web API.

- Environment variables can override settings of the same names.
- Default values are defined in biothings.web.settings.default.

```
class biothings.web.settings.configs.ConfigPackage(root, modules)
```

Bases: NamedTuple

Create new instance of ConfigPackage(root, modules)

modules: Collection

Alias for field number 1

```
root: object
    Alias for field number 0

biothings.web.settings.configs.load(config='config')

biothings.web.settings.configs.load_module(config, default=None)
    Load a config module.

config:
    1. a module object
    2. a fully qualified module name
    3. a file path to a module
```

biothings.web.settings.default

Biothings Web Settings Default

biothings.web.settings.validators

```
class biothings.web.settings.validators.DBParamValidator
    Bases: object
    validate(config)

class biothings.web.settings.validators.MongoParamValidator
    Bases: object
    validate(config)

class biothings.web.settings.validators.SubmoduleValidator
    Bases: object
    validate(config)

class biothings.web.settings.validators.WebAPIValidator
    Bases: object
    validate(config)
```

biothings.web.templates

The “templates” folder stores HTML templates for native biothings web handlers and is structured as a dummy module to facilitate location resolution. Typically, `biothings.web.templates.__path__[0]` returns a string representing the location of this folder on the file system. This folder is intended to be used internally by the SDK developer.

6.8 biothings.tests

6.8.1 biothings.tests.hub

```
class biothings.tests.hub.DatabaseCollectionTesting(db_url, db, collection)
    Bases: object

    Constructor that takes in three items db_url - string - the mongoDB url to connect to db - string - name of DB to
    use collection - string - name of collection in db

    test_database_index()
    test_document_name()
    test_documents_taxid(taxid)
    test_field_does_not_exist(_id)
    test_field_taxid(taxid)
    test_field_unique_id(_id)
    test_total_document_count(expected_count)
```

6.8.2 biothings.tests.web

Biothings Test Helpers

There are two types of test classes that provide utilities to three types of test cases, developed in the standalone apps.

The two types of test classes are:

BiothingsWebTest, which targets a running web server. BiothingsWebAppTest, which targets a web server config file.

To further illustrate, for any biothings web applications, it typically conforms to the following architectures:

Layer 3: A web server that implements the behaviors defined below. Layer 2: A config file that defines how to serve data from ES. Layer 1: An Elasticsearch server with data.

And for the two types of test classes, to explain their differences in the context of the layered design described above:

BiothingsWebTest targets an existing Layer 3 endpoint. BiothingsWebAppTest targets layer 2 and runs its own layer 3. Note no utility is provided to directly talk to layer 1.

The above discussed the python structures provided as programming utilities, on the other hand, there are three types of use cases, or testing objectives:

L3 Data test, which is aimed to test the data integrity of an API.

It subclasses BiothingsWebTest and ensures all layers working. The data has to reside in elasticsearch already.

L3 Feature test, which is aimed to test the API implementation.

It makes sure the settings in config file is reflected. These tests work on production data and require constant updates to keep the test cases in sync with the actual data. These test cases subclass BiothingsWebTest as well and asl require existing production data in elasticsearch.

L2 Feature test, doing basically the same things as above but uses

a small set of data that it ingests into elasticsearch. This is a lightweight test for development and automated testings for each new commit. It comes with data it will ingest in ES and does not require any existing data setup to run.

To illustrate the differences in a chart:

Objectives	Class	Test Target	ES Has Data	Automated Testing Trigger	L3 Data Test	BiothingsWebTest
A Running API	Yes	Data Release				
		L3 Feature T.	BiothingsWebTest	A Running API	Yes	Data Release & New Commit
			bAppTest	A config module	No*	New Commit
						* For L2 Feature Test, data is defined in the test cases and will be automatically ingested into

Elasticsearch at the start of the testing and get deleted after testing finishes. The other two types of testing require existing production data on the corresponding ES servers.

In development, it is certainly possible for a particular test case to fall under multiple test types, then the developer can use proper inheritance structures to avoid repeating the specific test case.

In terms of naming conventions, sometimes the L3 tests are grouped together and called remote tests, as they mostly target remote servers. And the L2 tests are called local tests, as they starts a local server.

L3 Envs:

`TEST_SCHEME TEST_PREFIX TEST_HOST TEST_CONF`

L2 Envs:

`TEST_KEEPDATA <Config Module Override>`

`biothings.tests.web.BiothingsDataTest`

alias of `BiothingsWebTest`

`biothings.tests.web.BiothingsTestCase`

alias of `BiothingsWebAppTest`

`class biothings.tests.web.BiothingsWebAppTest(methodName: str = 'runTest')`

Bases: `BiothingsWebTestCase`, `AsyncHTTPTestCase`

Starts the tornado application to run tests locally. Need a config.py under the test class folder.

`TEST_DATA_DIR_NAME: str | None = None`

`property config`

`get_app()`

Should be overridden by subclasses to return a `tornado.web.Application` or other `.HTTPServer` callback.

`get_new_ioloop()`

Returns the `.IOLoop` to use for this test.

By default, a new `.IOLoop` is created for each test. Subclasses may override this method to return `.IOLoop.current()` if it is not appropriate to use a new `.IOLoop` in each tests (for example, if there are global singletons using the default `.IOLoop`) or if a per-test event loop is being provided by another system (such as `pytest-asyncio`).

Deprecated since version 6.3: This method will be removed in Tornado 7.0.

`get_url(path)`

Try best effort to get a full url to make a request. Return an absolute url when class var ‘host’ is defined. If not, return a path relative to the host root.

```
request(path, method='GET', expect=200, **kwargs)
```

Use requests library to make an HTTP request. Ensure path is translated to an absolute path. Conveniently check if status code is as expected.

```
class biothings.tests.web.BiothingsWebTest
```

Bases: *BiothingsWebTestBase*

```
classmethod setup_class()
```

this is the setup method when pytest run tests from this class

```
class biothings.tests.web.BiothingsWebTestBase
```

Bases: *object*

```
get_url(path)
```

Try best effort to get a full url to make a request. Return an absolute url when class var ‘host’ is defined. If not, return a path relative to the host root.

```
host = ''
```

```
static msgpack_ok(packed_bytes)
```

Load msgpack into a dict

```
prefix = 'v1'
```

```
query(method='GET', endpoint='query', hits=True, data=None, json=None, **kwargs)
```

Make a query and assert positive hits by default. Assert zero hit when hits is set to False.

```
request(path, method='GET', expect=200, **kwargs)
```

Use requests library to make an HTTP request. Ensure path is translated to an absolute path. Conveniently check if status code is as expected.

```
scheme = 'http'
```

```
tearDown()
```

By default, a new “IOLoop” is constructed for each test and is available as “self.io_loop”. To maintain the desired test behavior, it is necessary to clear the current IOLoop at the end of each test function. See class AsyncTestCase in the reference: Reference: https://www.tornadoweb.org/en/branch6.4/_modules/tornado/testing.html

```
static value_in_result(value, result: dict | list, key: str, case_insensitive: bool = False) → bool
```

Check if value is in result at specific key

Elasticsearch does not care if a field has one or more values (arrays), so you may get a search with multiple values in one field. You were expecting a result of type T but now you have a List[T] which is bad. In testing, usually any one element in the list eq. to the value you’re looking for, you don’t really care which. This helper function checks if the value is at a key, regardless of the details of nesting, so you can just do this:

```
assert self.value_in_result(value, result, 'where.it.should.be')
```

Caveats: case_insensitive only calls .lower() and does not care about locale/ unicode/anything

Parameters

- **value** – value to look for
- **result** – dict or list of input, most likely from the APIs
- **key** – dot delimited key notation
- **case_insensitive** – for str comparisons, invoke .lower() first

Returns

boolean indicating whether the value is found at the key

Raises

TypeError – when case_insensitive set to true on unsupported types

6.9 biothings.utils

6.9.1 biothings.utils.aws

```
biothings.utils.aws.create_bucket(name, region=None, aws_key=None, aws_secret=None, acl=None,
ignore_already_exists=False)
```

Create a S3 bucket “name” in optional “region”. If aws_key and aws_secret are set, S3 client will these, otherwise it’ll use default system-wide setting. “acl” defines permissions on the bucket: “private” (default), “public-read”, “public-read-write” and “authenticated-read”

```
biothings.utils.aws.download_s3_file(s3key, localfile=None, aws_key=None, aws_secret=None,
s3_bucket=None, overwrite=False)
```

```
biothings.utils.aws.get_s3_file(s3key, localfile=None, return_what=False, aws_key=None,
aws_secret=None, s3_bucket=None)
```

```
biothings.utils.aws.get_s3_file_contents(s3key, aws_key=None, aws_secret=None, s3_bucket=None) →
bytes
```

```
biothings.utils.aws.get_s3_folder(s3folder, basedir=None, aws_key=None, aws_secret=None,
s3_bucket=None)
```

```
biothings.utils.aws.get_s3_static_website_url(s3key, aws_key=None, aws_secret=None,
s3_bucket=None)
```

```
biothings.utils.aws.get_s3_url(s3key, aws_key=None, aws_secret=None, s3_bucket=None)
```

```
biothings.utils.aws.key_exists(bucket, s3key, aws_key=None, aws_secret=None)
```

```
biothings.utils.aws.send_s3_big_file(localfile, s3key, overwrite=False, acl=None, aws_key=None,
aws_secret=None, s3_bucket=None, storage_class=None)
```

Multipart upload for file bigger than 5GiB

```
biothings.utils.aws.send_s3_file(localfile, s3key, overwrite=False, permissions=None, metadata=None,
content=None, content_type=None, aws_key=None, aws_secret=None,
s3_bucket=None, redirect=None)
```

save a localfile to s3 bucket with the given key. bucket is set via S3_BUCKET it also save localfile’s lastmodified time in s3 file’s metadata

Parameters

redirect (str) – if not None, set the redirect property of the object so it produces a 301 when accessed

```
biothings.utils.aws.send_s3_folder(folder, s3basedir=None, acl=None, overwrite=False, aws_key=None,
aws_secret=None, s3_bucket=None)
```

```
biothings.utils.aws.set_static_website(name, aws_key=None, aws_secret=None, index='index.html',
error='error.html')
```

6.9.2 biothings.utils.backend

Backend access class.

```
class biothings.utils.backend.DocBackendBase
```

Bases: object

```
drop()
```

```
finalize()
```

if needed, for example for bulk updates, perform flush at the end of updating. Final optimization or compacting can be done here as well.

```
get_from_id(id)
```

```
get_id_list()
```

```
insert(doc_li)
```

```
name = 'Undefined'
```

```
prepare()
```

if needed, add extra preparation steps here.

```
property target_name
```

```
update(id, extra_doc)
```

update only, no upsert.

```
property version
```

```
class biothings.utils.backend.DocBackendOptions(cls, es_index=None, es_host=None,  
                                                es_doc_type=None, mongo_target_db=None,  
                                                mongo_target_collection=None)
```

Bases: object

```
class biothings.utils.backend.DocESBackend(esidxer=None)
```

Bases: *DocBackendBase*

esidxer is an instance of utils.es.ESIndexer class.

```
count()
```

```
classmethod create_from_options(options)
```

Function that recreates itself from a DocBackendOptions class. Probably a needless rewrite of `__init__...`

```
drop()
```

```
finalize()
```

if needed, for example for bulk updates, perform flush at the end of updating. Final optimization or compacting can be done here as well.

```
get_from_id(id)
```

```
get_id_list(step=None)
```

```
insert(doc_li)
```

```
mget_from_ids(ids, step=100000, only_source=True, asiter=True, **kwargs)
    ids is an id list. always return a generator

name = 'es'

prepare(update_mapping=True)
    if needed, add extra preparation steps here.

query(query=None, verbose=False, step=10000, scroll='10m', only_source=True, **kwargs)
    Function that takes a query and returns an iterator to query results.

remove_from_ids(ids, step=10000)

property target_alias

property target_esidxer

property target_name

update(id, extra_doc)
    update only, no upsert.

property version

class biothings.utils.backend.DocMemoryBackend(target_name=None)
    Bases: DocBackendBase
    target_dict is None or a dict.

drop()

finalize()
    dump target_dict into a file.

get_from_id(id)

get_id_list()

insert(doc_li)

name = 'memory'

property target_name

update(id, extra_doc)
    update only, no upsert.

class biothings.utils.backend.DocMongoBackend(target_db, target_collection=None)
    Bases: DocBackendBase
    target_collection is a pymongo collection object.

count()

count_from_ids(ids, step=100000)
    return the count of docs matching with input ids normally, it does not need to query in batches, but MongoDB has a BSON size limit of 16M bytes, so too many ids will raise a pymongo.errors.DocumentTooLarge error.

drop()
```

```
finalize()
    flush all pending writes.

get_from_id(id)

get_id_list()

insert(docs)

mget_from_ids(ids, asiter=False)
    ids is an id list. returned doc list should be in the same order of the
    input ids. non-existing ids are ignored.

name = 'mongo'

remove_from_ids(ids, step=10000)

property target_db

property target_name

update(docs, upsert=False)
    if id does not exist in the target_collection, the update will be ignored except if upsert is True

update_diff(diff, extra=None)
    update a doc based on the diff returned from diff.diff_doc “extra” can be passed (as a dictionary) to add
    common fields to the updated doc, e.g. a timestamp.

property version

biothings.utils.backend.DocMongoDBBackend
alias of DocMongoBackend
```

6.9.3 biothings.utils.common

This module contains util functions may be shared by both BioThings data-hub and web components. In general, do not include utils depending on any third-party modules.

```
class biothings.utils.common.BiothingsJSONEncoder(*, skipkeys=False, ensure_ascii=True,
                                                check_circular=True, allow_nan=True,
                                                sort_keys=False, indent=None, separators=None,
                                                default=None)
```

Bases: `JSONEncoder`

A class to dump Python Datetime object. `json.dumps(data, cls=DateTimeJSONEncoder, indent=indent)`

Constructor for `JSONEncoder`, with sensible defaults.

If `skipkeys` is false, then it is a `TypeError` to attempt encoding of keys that are not str, int, float or None. If `skipkeys` is True, such items are simply skipped.

If `ensure_ascii` is true, the output is guaranteed to be str objects with all incoming non-ASCII characters escaped. If `ensure_ascii` is false, the output can contain non-ASCII characters.

If `check_circular` is true, then lists, dicts, and custom encoded objects will be checked for circular references during encoding to prevent an infinite recursion (which would cause an `RecursionError`). Otherwise, no such check takes place.

If allow_nan is true, then NaN, Infinity, and -Infinity will be encoded as such. This behavior is not JSON specification compliant, but is consistent with most JavaScript based encoders and decoders. Otherwise, it will be a ValueError to encode such floats.

If sort_keys is true, then the output of dictionaries will be sorted by key; this is useful for regression tests to ensure that JSON serializations can be compared on a day-to-day basis.

If indent is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. None is the most compact representation.

If specified, separators should be an (item_separator, key_separator) tuple. The default is (', ', ': ') if *indent* is None and (',', ':') otherwise. To get the most compact JSON representation, you should specify (',', ':') to eliminate whitespace.

If specified, default is a function that gets called for objects that can't otherwise be serialized. It should return a JSON encodable version of the object or raise a `TypeError`.

`default(o)`

Implement this method in a subclass such that it returns a serializable object for *o*, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement `default` like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

`class biothings.utils.common.DummyConfig(name, doc=None)`

Bases: `module`

This class allows “import config” or “from biothings import config” to work without actually creating a config.py file:

```
import sys
from biothings.utils.common import DummyConfig
sys.modules[“config”] = DummyConfig(‘config’)
sys.modules[“biothings.config”] = DummyConfig(‘config’)
```

`class biothings.utils.common.LogPrint(log_f, log=1, timestamp=0)`

Bases: `object`

If this class is set to `sys.stdout`, it will output both `log_f` and `__stdout__`. `log_f` is a file handler.

`close()`

`fileno()`

`flush()`

`pause()`

`resume()`

`start()`

`write(text)`

`biothings.utils.common.SubStr(input_string, start_string='', end_string='', include=0)`

Return the substring between start_string and end_string. If start_string is “”, cut string from the beginning of input_string. If end_string is “”, cut string to the end of input_string. If either start_string or end_string can not be found from input_string, return “”. The end_pos is the first position of end_string after start_string. If multi-occurrence, cut at the first position. include=0(default), does not include start/end_string; include=1: include start/end_string.

`biothings.utils.common.addsuffix(filename, suffix, noext=False)`

Add suffix in front of “.extension”, so keeping the same extension. if noext is True, remove extension from the filename.

`async biothings.utils.common.aiogunzipall(folder, pattern, job_manager, pinfo)`

Gunzip all files in folder matching pattern. job_manager is used for parallelisation, and pinfo is a pre-filled dict used by job_manager to report jobs in the hub (see bt.utils.manager.JobManager)

`biothings.utils.common.anyfile(infile, mode='r')`

return a file handler with the support for gzip/zip compressed files. if infile is a two value tuple, then first one is the compressed file; the second one is the actual filename in the compressed file. e.g., ('a.zip', 'aa.txt')

`biothings.utils.common.ask(prompt, options='YN')`

Prompt Yes or No, return the upper case ‘Y’ or ‘N’.

`class biothings.utils.common.dotdict`

Bases: dict

`biothings.utils.common.dump(obj, filename, protocol=4, compress='gzip')`

Saves a compressed object to disk protocol version 4 is the default for py3.8, supported since py3.4

`biothings.utils.common.dump2gridfs(obj, filename, db, protocol=2)`

Save a compressed (support gzip only) object to MongoDB gridfs.

`biothings.utils.common.file_newer(source, target)`

return True if source file is newer than target file.

`biothings.utils.common.filter_dict(d, keys)`

Remove keys from dict “d”. “keys” is a list of string, dotfield notation can be used to express nested keys. If key to remove doesn’t exist, silently ignore it

`biothings.utils.common.find_classes_subclassing(mods, baseclass)`

Given a module or a list of modules, inspect and find all classes which are a subclass of the given baseclass, inside those modules

`biothings.utils.common.find_doc(k, keys)`

Used by jsonld insertion in www.api.es._insert_jsonld

`biothings.utils.common.find_value_in_doc(dotfield, value, doc)`

Explore mixed dictionary using dotfield notation and return value. Stringify before search Support wildcard searching The comparison is case-sensitive Example:

```
X = {"a": {"b": "1"}, "x": [{"y": "3", "z": "4"}, "5"]} Y = [{"a": {"b": "1"}, {"x": [{"y": "3", "z": "4"}, "5"]}], Z = [{"a": {"b": "1"}, {"x": [{"y": "34567", "z": "4"}, "5"]}]} assert find_value_in_doc("a.b", "1", X) assert find_value_in_doc("x.y", "3", X) assert find_value_in_doc("x.y", "3*7", Z) assert find_value_in_doc("x.y", "34567", Z) assert find_value_in_doc("x", "5", Y) assert find_value_in_doc("a.b", "c", X) is False assert find_value_in_doc("a", "c", X) is False
```

`biothings.utils.common.get_class_from_classpath(class_path)`

`biothings.utils.common.get_compressed_outfile(filename, compress='gzip')`

Get a output file handler with given compress method. currently support gzip/bz2/lzma, lzma only available in py3

`biothings.utils.common.get_dotfield_value(dotfield, d)`

Explore dictionary d using dotfield notation and return value. Example:

```
d = {"a":{"b":1}}.
get_dotfield_value("a.b",d) => 1
```

`biothings.utils.common.get_loop()`

Since Python 3.10, a Deprecation warning is emitted if there is no running event loop. In future Python releases, a RuntimeError will be raised instead.

Ref: https://docs.python.org/3/library/asyncio-eventloop.html#asyncio.get_event_loop

`biothings.utils.common.get_loop_with_max_workers(max_workers=None)`

`biothings.utils.common.get_plugin_name_from_local_manifest(path)`

`biothings.utils.common.get_plugin_name_from_remote_manifest(url)`

`biothings.utils.common.get_random_string()`

`biothings.utils.common.get_timestamp()`

`biothings.utils.common.gunzip(f, pattern='*.gz')`

`biothings.utils.common.gunzipall(folder, pattern='*.gz')`

gunzip all *.gz files in “folder”

`class biothings.utils.common.inf`

Bases: object

Represents Inf type, but not as a float

`biothings.utils.common.is_filehandle(fh)`

return True/False if fh is a file-like object

`biothings.utils.common.is_float(f)`

return True if input is a float.

`biothings.utils.common.is_int(s)`

return True or False if input string is integer or not.

`biothings.utils.common.is_scalar(f)`

`biothings.utils.common.is_seq(li)`

return True if input is either a list or a tuple.

`biothings.utils.common.is_str(s)`

return True or False if input is a string or not. python3 compatible.

`biothings.utils.common.iter_n(iterable, n, with_cnt=False)`

Iterate an iterator by chunks (of n) if with_cnt is True, return (chunk, cnt) each time ref <http://stackoverflow.com/questions/8991506/iterate-an-iterator-by-chunks-of-n-in-python>

`biothings.utils.common.json_encode(obj)`

Tornado-aimed json encoder, it does the same job as `tornado.escape.json_encode` but also deals with datetime encoding

`biothings.utils.common.json_serial(obj)`

JSON serializer for objects not serializable by default json code

`biothings.utils.common.list2dict(a_list, keyitem, alwayslist=False)`

Return a dictionary with specified keyitem as key, others as values. keyitem can be an index or a sequence of indexes. For example:

```
li = [['A', 'a', 1],  
      ['B', 'a', 2],  
      ['A', 'b', 3]]  
list2dict(li, 0)---> {'A': [('a', 1), ('b', 3)],  
                        'B': ('a', 2)}
```

If `alwayslist` is True, values are always a list even there is only one item in it.

```
list2dict(li, 0, True)---> {'A': [('a', 1), ('b', 3)],  
                             'B': [('a', 2)]}
```

`biothings.utils.common.loadobj(filename, mode='file')`

Loads a compressed object from disk file (or file-like handler) or MongoDB gridfs file (mode='gridfs')

```
obj = loadobj('data.pyobj')  
obj = loadobj(('data.pyobj', mongo_db), mode='gridfs')
```

`biothings.utils.common.md5sum(fname)`

`biothings.utils.common.merge(x, dx)`

Merge dictionary `dx` (`x`) into dictionary `x`. If `__REPLACE__` key is present in any level `z` in `dx`, `z` in `x` is replaced, instead of merged, with `z` in `dx`.

`class biothings.utils.common.nan`

Bases: `object`

Represents NaN type, but not as a float

`biothings.utils.common.newer(t0, t1, fint='%Y%m%d')`

`t0` and `t1` are string of timestamps matching “format” pattern. Return True if `t1` is newer than `t0`.

`biothings.utils.common.open_anyfile(infile, mode='r')`

a context manager can be used in “with” stmt. accepts a filehandle or anything accepted by `anyfile` function.

```
with open_anyfile('test.txt') as in_f:  
    do_something()
```

`biothings.utils.common.open_compressed_file(filename)`

Get a read-only file-handler for compressed file, currently support gzip/bz2/lzma, lzma only available in py3

`biothings.utils.common.parse_folder_name_from_url(url)`

`biothings.utils.common.rmdashfr(top)`

Recursively delete dirs and files from “top” directory, then delete “top” dir

```
biothings.utils.common.run_once()
should_run_task_1 = run_once() print(should_run_task_1()) -> True print(should_run_task_1()) -> False
print(should_run_task_1()) -> False print(should_run_task_1()) -> False

should_run_task_2 = run_once() print(should_run_task_2('2a'))-> True print(should_run_task_2('2b'))-> True
print(should_run_task_2('2a')) -> False print(should_run_task_2('2b')) -> False ...

biothings.utils.common.safe_unicode(s, mask='#')
replace non-decodable char into "#".

biothings.utils.common.safewfile(filename, prompt=True, default='C', mode='w')
return a file handle in 'w' mode,use alternative name if same name exist. if prompt == 1, ask for overwriting,appending or changing name, else, changing to available name automatically.

biothings.utils.common.sanitize_tarfile(tar_object, directory)
Prevent user-assisted remote attackers to overwrite arbitrary files via a .. (dot dot) sequence in filenames in a TAR archive, a related issue to CVE-2007-4559

biothings.utils.common.setup_logfile(logfile)

biothings.utils.common.sizeof_fmt(num, suffix='B')

biothings.utils.common.split_ids(q)
```

split input query string into list of ids. any of ````

|,```` as the separator,

but perserving a phrase if quoted (either single or double quoted) more detailed rules see: <http://docs.python.org/2/library/shlex.html#parsing-rules>

e.g.:

```
>>> split_ids('CDK2 CDK3')
['CDK2', 'CDK3']
>>> split_ids('"CDK2 CDK3"')
```

CDk4')

['CDK2 CDK3', 'CDK4']

```
class biothings.utils.common.splitstr
```

Bases: str

Type representing strings with space in it

```
biothings.utils.common.timessofar(t0, clock=0, t1=None)
```

return the string(eg.'3m3.42s') for the passed real time/CPU time so far from given t0 (return from t0=time.time() for real time/ t0=time.clock() for CPU time).

```
biothings.utils.common.traverse(obj, leaf_node=False)
```

Output path-dictionary pairs. For example, input: {

```
'exac_nontcga': {'af': 0.00001883}, 'gnomad_exome': {'af': {'af': 0.0000119429, 'af_afr': 0.000123077}}, 'snpeff': {'ann': [{{'effect': 'intron_variant', 'feature_id': 'NM_014672.3'}, {'effect': 'intron_variant', 'feature_id': 'NM_001256678.1'}}]}
```

} will be translated to a generator: (

```
(“exac_nontcga”, {"af": 0.00001883}), (“gnomad_exome.af”, {"af": 0.0000119429, “af_afr”: 0.000123077}), (“gnomad_exome”, {"af": {"af": 0.0000119429, “af_afr”: 0.000123077}}), (“snpeff.ann”, {"effect": “intron_variant”, “feature_id”: “NM_014672.3”}), (“snpeff.ann”, {"effect": “intron_variant”, “feature_id”: “NM_001256678.1”}), (“snpeff.ann”, [{ ... }, { ... }]), (“snpeff”, {"ann": [{ ... }, { ... }]})}, (‘, {'exac_nontcga': {...}, ‘gnomad_exome’: {...}, ‘snpeff’: {...}}))
```

) or when traversing leaf nodes:

```
(‘exac_nontcga.af’, 0.00001883), (‘gnomad_exome.af.af’, 0.0000119429), (‘gnomad_exome.af.af_afr’, 0.000123077), (‘snpeff.ann.effect’, ‘intron_variant’), (‘snpeff.ann.feature_id’, ‘NM_014672.3’), (‘snpeff.ann.effect’, ‘intron_variant’), (‘snpeff.ann.feature_id’, ‘NM_001256678.1’)
```

)

`biothings.utils.common.uncompressall(folder)`

Try to uncompress any known archive files in folder

`biothings.utils.common.untarall(folder, pattern='*.tar')`

untar all *.tar files in “folder”

`biothings.utils.common.untargzall(folder, pattern='*.tar.gz')`

gunzip and untar all *.tar.gz files in “folder”

`biothings.utils.common.unxzall(folder, pattern='*.xz')`

unxz all xz files in “folder”, in “folder”

`biothings.utils.common.unzipall(folder, pattern='*.zip')`

unzip all zip files in “folder”, in “folder”

6.9.4 `biothings.utils.configuration`

```
class biothings.utils.configuration.ConfigAttrMeta(confmod: biothings.utils.configuration.MetaField
= <factory>, section:
    biothings.utils.configuration.Text = <factory>,
    description:
        biothings.utils.configuration.Paragraph =
            <factory>, readonly:
                biothings.utils.configuration.Flag = <factory>,
                hidden: biothings.utils.configuration.Flag =
                    <factory>, invisible:
                        biothings.utils.configuration.Flag = <factory>)
```

Bases: `object`

`asdict()`

`commit()`

`confmod: MetaField`

`description: Paragraph`

`feed(field, value)`

`hidden: Flag`

`invisible: Flag`

```

readonly: Flag
reset()
section: Text
update(meta)

class biothings.utils.configuration.ConfigLine(seq)
Bases: UserString

PATTERNS = (('hidden', re.compile('^\#\s?-\\s*hide\\s*-\\s?#\s*$')), <function ConfigLine.<lambda>>), ('invisible', re.compile('^\#\s?-\\s*invisible\\s*-\\s?#\s*$')), <function ConfigLine.<lambda>>), ('readonly', re.compile('^\#\s?-\\s*readonly\\s*-\\s?#\s*$')), <function ConfigLine.<lambda>>), ('section', re.compile('^\#\s?\\*\s*(.*))\\s*\\*\s?#\s*$')), <function ConfigLine.<lambda>>), ('description', re.compile('.*\s?#\s+(.*$)'), <function ConfigLine.<lambda>>))

match()

class biothings.utils.configuration.ConfigLines(initlist=None)
Bases: UserList

parse(attrs=())

class biothings.utils.configuration.ConfigurationDefault(default, desc)
Bases: object

exception biothings.utils.configuration.ConfigurationError
Bases: Exception

class biothings.utils.configuration.ConfigurationValue(code)
Bases: object

type to wrap default value when it's code and needs to be interpreted later code is passed to eval() in the context of the whole "config" dict (so for instance, paths declared before in the configuration file can be used in the code passed to eval) code will also be executed through exec() if eval() raised a syntax error. This would happen when code contains statements, not just expression. In that case, a variable should be created in these statements (named the same as the original config variable) so the proper value can be through ConfigurationManager.

get_value(name, conf)
Return value by eval'ing code in self.code, in the context of given configuration dict (namespace), for given config parameter name.

class biothings.utils.configuration.ConfigurationWrapper(default_config, conf)
Bases: object

Wraps and manages configuration access and edit. A singleton instance is available throughout all hub apps using biothings.config or biothings.hub.config after calling import biothings.hub. In addition to providing config value access, either from config files or database, config manager can supersede attributes of a class with values coming from the database, allowing dynamic configuration of hub's elements.

When constructing a ConfigurationWrapper instance, variables will be defined with default values coming from default_config, then they can be overridden by conf's values, or new variables will be added if not defined in default_conf. Only metadata come from default_config will be used.

get_value_from_db(name)

```

```
get_value_from_file(name)
property modified
property readonly
reset(name=None)
show()
store_value_to_db(name, value)
supersede(klass)
    supersede class variable with db values
class biothings.utils.configuration.Flag(value=None)
    Bases: MetaField
    default
        alias of bool
    feed(value)

class biothings.utils.configuration.MetaField(value=None)
    Bases: object
    clear()
    default
        alias of None
    feed(value)

    property value

class biothings.utils.configuration.Paragraph(value=None)
    Bases: MetaField
    default
        alias of list
    feed(value)

    property value

class biothings.utils.configuration.Text(value=None)
    Bases: MetaField
    feed(value)

biothings.utils.configuration.is_jsonable(x)
biothings.utils.configuration.set_default_folder(data_archive_root, sub_folder)
    set default sub folder based on data_archive_root
```

6.9.5 biothings.utils.dataloader

Utility functions for parsing flatfiles, mapping to JSON, cleaning.

class biothings.utils.dataloader.MinType

Bases: object

biothings.utils.dataloader.alwayslist(value)

If input value is not a list/tuple type, return it as a single value list.

biothings.utils.dataloader.boolean_convert(d, convert_keys=None, level=0)

Convert values specified by *convert_keys* in document *d* to boolean. Dotfield notation can be used to specify inner keys.

Note that *None* values are converted to *False* in Python. Use *dict_sweep()* before calling this function if such *False* values are not expected. See <https://github.com/biothings/biothings.api/issues/274> for details.

biothings.utils.dataloader.dict_apply(d, key, value, sort=True)

add value to d[key], append it if key exists

```
>>> d = {'a': 1}
>>> dict_apply(d, 'a', 2)
{'a': [1, 2]}
>>> dict_apply(d, 'a', 3)
{'a': [1, 2, 3]}
>>> dict_apply(d, 'b', 2)
{'a': 1, 'b': 2}
```

biothings.utils.dataloader.dict_attrmerge(dict_li, removedup=True, sort=True, special_fns=None)

```
dict_attrmerge([{'a': 1, 'b':[2,3]}, 
               {'a': [1,2], 'b':[3,5], 'c'=4}])
```

should return

```
{'a': [1,2], 'b':[2,3,5], 'c'=4}
```

special_fns is a dictionary of {attr: merge_fn} used for some special attr, which need special merge_fn e.g., {‘uniprot’: _merge_uniprot}

biothings.utils.dataloader.dict_convert(_dict, keyfn=None, valuefn=None)

Return a new dict with each key converted by keyfn (if not None), and each value converted by valuefn (if not None).

biothings.utils.dataloader.dict_nodup(_dict, sort=True)

biothings.utils.dataloader.dict_sweep(d, vals=None, remove_invalid_list=False)

Remove keys whose values are “.”, “-”, “”, “NA”, “none”, “”; and remove empty dictionaries

Parameters

- **d (dict)** – a dictionary
- **vals (str or list)** – a string or list of strings to sweep, or None to use the default values
- **remove_invalid_list (boolean)** – when true, will remove key for which list has only one value, which is part of “vals”. Ex:

```
test_dict = {'gene': [None, None], 'site': ["Intron", None], 'snp_
↪build' : 136}
```

with `remove_invalid_list == False`:

```
{'gene': [None], 'site': ['Intron'], 'snp_build': 136}
```

with `remove_invalid_list == True`:

```
{'site': ['Intron'], 'snp_build': 136}
```

`biothings.utils.dataloader.dict_to_list(gene_d)`

return a list of genedoc from genedoc dictionary and make sure the “_id” field exists.

`biothings.utils.dataloader.dict_traverse(d, func, traverse_list=False)`

Recursively traverse dictionary d, calling `func(k,v)` for each key/value found. func must return a tuple(new_key,new_value)

`biothings.utils.dataloader.dict_walk(dictionary, key_func)`

Recursively apply key_func to dict's keys

`biothings.utils.dataloader.dupline_seperator(dupline, dup_sep, dup_idx=None, strip=False)`

for a line like this:

```
a b1,b2 c1,c2
```

return a generator of this list (breaking out of the duplicates in each field):

```
[(a,b1,c1),  
 (a,b2,c1),  
 (a,b1,c2),  
 (a,b2,c2)]
```

Example:

```
dupline_seperator(dupline=['a', 'b1,b2', 'c1,c2'],  
                  dup_idx=[1,2],  
                  dup_sep=',')
```

if `dup_idx` is None, try to split on every field. if `strip` is True, also trim out of extra spaces.

`biothings.utils.dataloader.file_merge(infiles, outfile=None, header=1, verbose=1)`

Merge a list of input files with the same format. If `header` is n then the top n lines will be discarded since reading the 2nd file in the list.

`biothings.utils.dataloader.float_convert(d, include_keys=None, exclude_keys=None)`

Convert elements in a document to floats.

By default, traverse all keys. If `include_keys` is specified, only convert the list from `include_keys` a.b, a.b.c. If `exclude_keys` is specified, only exclude the list from `exclude_keys`.

Parameters

- `d` – a dictionary to traverse keys on
- `include_keys` – only convert these keys (optional)
- `exclude_keys` – exclude all other keys except these keys (optional)

Returns

generate key, value pairs

`biothings.utils.dataloader.id_strip(id_list)`

`biothings.utils.dataloader.int_convert(d, include_keys=None, exclude_keys=None)`

Convert elements in a document to integers.

By default, traverse all keys If `include_keys` is specified, only convert the list from `include_keys` a.b, a.b.c If `exclude_keys` is specified, only exclude the list from `exclude_keys`

Parameters

- `d` – a dictionary to traverse keys on
- `include_keys` – only convert these keys (optional)
- `exclude_keys` – exclude all other keys except these keys (optional)

Returns

generate key, value pairs

`biothings.utils.dataloader.list2dict(a_list, keyitem, alwayslist=False)`

Return a dictionary with specified `keyitem` as key, others as values. `keyitem` can be an index or a sequence of indexes. For example:

```
li=[[ 'A' , 'a' , 1] ,
 [ 'B' , 'a' , 2] ,
 [ 'A' , 'b' , 3]]
list2dict(li,0)---> { 'A' :[( 'a' , 1) , ( 'b' , 3)] ,
 'B' :( 'a' , 2)}
```

If `alwayslist` is True, values are always a list even there is only one item in it:

```
list2dict(li,0,True)---> { 'A' :[( 'a' , 1) , ( 'b' , 3)] ,
 'B' :[( 'a' , 2)]}
```

`biothings.utils.dataloader.list_itemcnt(a_list)`

Return number of occurrence for each item in the list.

`biothings.utils.dataloader.list_split(d, sep)`

Split fields by `sep` into comma separated lists, strip.

`biothings.utils.dataloader.listitems(a_list, *idx)`

Return multiple items from list by given indexes.

`biothings.utils.dataloader.listsort(a_list, by, reverse=False, cmp=None, key=None)`

Given `a_list` is a list of sub(list/tuple.), return a new list sorted by the `i`th (given from “`by`” item) item of each sublist.

`biothings.utils.dataloader.llist(li, sep='\n')`

Nicely output the list with each item a line.

`biothings.utils.dataloader.merge_dict(dict_li, attr_li, missingvalue=None)`

Merging multiple dictionaries into a new one. Example:

```
In [136]: d1 = {'id1': 100, 'id2': 200}
In [137]: d2 = {'id1': 'aaa', 'id2': 'bbb', 'id3': 'ccc'}
In [138]: merge_dict([d1,d2], ['number', 'string'])
Out[138]:
{'id1': {'number': 100, 'string': 'aaa'},
```

(continues on next page)

(continued from previous page)

```
'id2': {'number': 200, 'string': 'bbb'},
'id3': {'string': 'ccc'}}}
In [139]: merge_dict([d1,d2], ['number', 'string'], missingvalue='NA')
Out[139]:
{'id1': {'number': 100, 'string': 'aaa'},
'id2': {'number': 200, 'string': 'bbb'},
'id3': {'number': 'NA', 'string': 'ccc'}}
```

biothings.utils.dataloader.merge_duplicate_rows(rows, db)

@param rows: rows to be grouped by @param db: database name, string

biothings.utils.dataloader.merge_root_keys(doc1, doc2, exclude=None)

Ex: d1 = {"_id":1,"a":"a","b":{"k":"b"}}
d2 = {"_id":1,"a":"A","b":{"k":"B"},"c":123}

Both documents have the same _id, and 2 root keys, “a” and “b”. Using this storage, the resulting document will be:

```
{'_id': 1, 'a': ['A', 'a'], 'b': [{'k': 'B'}, {'k': 'b'}], "c":123}
```

biothings.utils.dataloader.merge_struct(v1, v2, aslistofdict=None, include=None, exclude=None)

merge two structures, v1 and v2, into one. :param aslistofdict: a string indicating the key name that should be treated as a list of dict :param include: when given a list of strings, only merge these keys (optional) :param exclude: when given a list of strings, exclude these keys from merging (optional)

biothings.utils.dataloader.normalized_value(value, sort=True)

Return a “normalized” value: 1. if a list, remove duplicate and sort it 2. if a list with one item, convert to that single item only 3. if a list, remove empty values 4. otherwise, return value as it is.

biothings.utils.dataloader.rec_handler(infile, block_end='\n', skip=0, include_block_end=False, as_list=False)

A generator to return a record (block of text) at once from the *infile*. The record is separated by one or more empty lines by default. *skip* can be used to skip top n-th lines if *include_block_end* is True, the line matching *block_end* will also be returned. If *as_list* is True, return a list of lines in one record.

biothings.utils.dataloader.safe_type(f, val)

Convert an input string to int/float/... using passed function. If the conversion fails then None is returned. If value of a type other than a string then the original value is returned.

biothings.utils.dataloader.tab2dict(datafile, cols, key, alwayslist=False, **kwargs)**biothings.utils.dataloader.tab2dict_iter(datafile, cols, key, alwayslist=False, **kwargs)**

Parameters

- **cols (array of int)** – an array of indices (of a list) indicating which element(s) are kept in bulk
- **key (int)** – an index (of a list) indicating which element is treated as a bulk key

Iterate *datafile* by row, subset each row (as a list of strings) by *cols*. Adjacent rows sharing the same value at the *key* index are put into one bulk. Each bulk is then transformed to a dict with the value at the *key* index as the dict key.

E.g. given the following datafile, cols=[0,1,2], and key=1, two bulks are generated:

key

```

a1 b1 c1 ----- a2 b1 c2 # bulk_1 => {b1: [(a1, c1), (a2, c2), (a3, c3)]} #
a3 b1 c3 ----- a4 b2 c4 ----- a5 b2 c5
# bulk_2 => {b2: [(a4, c4), (a5, c5), (a6, c6)]} # a6 b2 c6 -----
```

`biothings.utils.dataloader.tab2list(datafile, cols, **kwargs)`

`biothings.utils.dataloader.tabfile_feeder(datafile, header=1, sep='\t', includefn=None, coerce_unicode=True, assert_column_no=None)`

a generator for each row in the file.

`biothings.utils.dataloader.tabfile_tester(datafile, header=1, sep='\t')`

`biothings.utils.dataloader.to_boolean(val, true_str=None, false_str=None)`

Normalize str value to boolean value

`biothings.utils.dataloader.to_float(val)`

convert an input string to int

`biothings.utils.dataloader.to_int(val)`

convert an input string to float

`biothings.utils.dataloader.to_number(val)`

convert an input string to int/float.

`biothings.utils.dataloader.traverse_keys(d, include_keys=None, exclude_keys=None)`

Return all key, value pairs for a document.

By default, traverse all keys If include_keys is specified, only traverse the list from include_keys a.b, a.b.c If exclude_keys is specified, only exclude the list from exclude_keys

if a key in include_keys/exclude_keys is not found in d, it's skipped quietly.

Parameters

- **d** – a dictionary to traverse keys on
- **include_keys** – only traverse these keys (optional)
- **exclude_keys** – exclude all other keys except these keys (optional)

Returns

generate key, value pairs

`biothings.utils.dataloader.unique_ids(src_module)`

`biothings.utils.dataloader.unlist(d)`

`biothings.utils.dataloader.unlist_incl(d, include_keys=None, exclude_keys=None)`

Unlist elements in a document.

If there is 1 value in the list, set the element to that value. Otherwise, leave the list unchanged.

By default, traverse all keys If include_keys is specified, only traverse the list from include_keys a.b, a.b.c If exclude_keys is specified, only exclude the list from exclude_keys

Parameters

- **d** – a dictionary to unlist
- **include_keys** – only unlist these keys (optional)
- **exclude_keys** – exclude all other keys except these keys (optional)

Returns

generate key, value pairs

`biothings.utils.dataloader.update_dict_recur(d, u)`

Update dict `d` with dict `u`'s values, recursively (so existing values in `d` but not in `u` are kept even if nested)

`biothings.utils.dataloader.updated_dict(_dict, attrs)`

Same as `dict.update`, but return the updated dictionary.

`biothings.utils.dataloader.value_convert(_dict, fn, traverse_list=True)`

For each value in `_dict`, apply `fn` and then update `_dict` with return the value. If `traverse_list` is True and a value is a list, apply `fn` to each item of the list.

`biothings.utils.dataloader.value_convert_incl(d, fn, include_keys=None, exclude_keys=None)`

Convert elements in a document using a function `fn`.

By default, traverse all keys. If `include_keys` is specified, only convert the list from `include_keys` a.b, a.b.c. If `exclude_keys` is specified, only exclude the list from `exclude_keys`

Parameters

- `d` – a dictionary to traverse keys on
- `fn` – function to convert elements with
- `include_keys` – only convert these keys (optional)
- `exclude_keys` – exclude all other keys except these keys (optional)

Returns

generate key, value pairs

`biothings.utils.dataloader.value_convert_to_number(d, skipped_keys=None)`

Convert string numbers into integers or floats; skip converting certain keys in `skipped_keys` list.

6.9.6 biothings.utils.diff

Utils to compare two list of gene documents, requires to setup Biothing Hub.

`biothings.utils.diff.diff_collections(b1, b2, use_parallel=True, step=10000)`

`b1, b2` are one of supported backend class in `databuild.backend`. e.g.:

```
b1 = DocMongoDBBackend(c1)
b2 = DocMongoDBBackend(c2)
```

`biothings.utils.diff.diff_collections_batches(b1, b2, result_dir, step=10000)`

`b2` is new collection, `b1` is old collection

`biothings.utils.diff.diff_docs_jsonpatch(b1, b2, ids, fastdiff=False, excludeAttrs=None)`

if `fastdiff` is True, only compare the whole doc, do not traverse into each attributes.

`biothings.utils.diff.get_backend(uri, db, col, bk_type)`

`biothings.utils.diff.get_mongodb_uri(backend)`

`biothings.utils.diff.two_docs_iterator(b1, b2, id_list, step=10000, verbose=False)`

6.9.7 biothings.utils.doc_traversal

Some utility functions that do document traversal

class biothings.utils.doc_traversal.Queue

Bases: object

isempty()

pop()

get next obj from queue

push(*obj*)

put obj on queue

exception biothings.utils.doc_traversal.QueueEmptyError

Bases: Exception

class biothings.utils.doc_traversal.Stack

Bases: object

isempty()

pop()

push(*obj*)

put obj on stack

exception biothings.utils.doc_traversal.StackEmptyError

Bases: Exception

biothings.utils.doc_traversal.breadth_first_recursive_traversal(*doc*, *path=None*)

doesn't exactly implement breadth first ordering it seems, not sure why...

biothings.utils.doc_traversal.breadth_first_traversal(*doc*)

Yield a 2 element tuple for every k, v pair in document items (nodes are visited in breadth first order k is itself a tuple of keys annotating the path for this node (v) to root v is the node value)

biothings.utils.doc_traversal.depth_first_recursive_traversal(*doc*, *path=None*)

biothings.utils.doc_traversal.depth_first_traversal(*doc*)

Yield a 2 element tuple for every k, v pair in document items (nodes are visited in depth first order k is itself a tuple of keys annotating the path for this node (v) to root v is the node value)

6.9.8 biothings.utils.docs

biothings.utils.docs.exists_or_null(*doc*, *field*, *val=None*)

biothings.utils.docs.flatten_doc(*doc*, *outfield_sep='.'*, *sort=True*)

This function will flatten an elasticsearch document (really any json object). outfield_sep is the separator between the fields in the return object. sort specifies whether the output object should be sorted alphabetically before returning

(otherwise output will remain in traversal order)

biothings.utils.docs.flatten_doc_2(*doc*, *outfield_sep='.'*, *sort=True*)

6.9.9 biothings.utils.dotfield

`biothings.utils.dotfield.compose_dot_fields_by_fields(doc, fields)`

Reverse function of `parse_dot_fields`

`biothings.utils.dotfield.make_object(attr, value)`

Create dictionary following the input dot notation and the value.

Example:

```
make_object('a.b.c', 100) --> {a:{b:{c:100}}}
make_object(['a', 'b', 'c'], 100) --> {a:{b:{c:100}}}
```

`biothings.utils.dotfield.merge_object(obj1, obj2)`

`biothings.utils.dotfield.parse_dot_fields(doc)`

Example: `parse_dot_fields({‘a’: 1, ‘b.c’: 2, ‘b.a.c’: 3})` should return `{‘a’: 1, ‘b’: {‘a’: {‘c’: 3}, ‘c’: 2}}`

6.9.10 biothings.utils.dotstring

`biothings.utils.dotstring.key_value(dictionary, key)`

Return a generator for all values in a dictionary specific by a dotstring (key)

if key is not found from the dictionary, None is returned.

Parameters

- **dictionary** – a dictionary to return values from
- **key** – key that specifies a value in the dictionary

Returns

generator for values that match the given key

`biothings.utils.dotstring.last_element(d, key_list)`

Return the last element and key for a document d given a docstring.

A document d is passed with a list of keys key_list. A generator is then returned for all elements that match all keys. Note that there may be a 1-to-many relationship between keys and elements due to lists in the document.

Parameters

- **d** – document d to return elements from
- **key_list** – list of keys that specify elements in the document d

Returns

generator for elements that match all keys

`biothings.utils.dotstring.list_length(d, field)`

Return the length of a list specified by field.

If field represents a list in the document, then return its length. Otherwise return 0.

Parameters

- **d** – a dictionary
- **field** – the dotstring field specifying a list

`biothings.utils.dotstring.remove_key(dictionary, key)`

Remove field specified by the docstring key

Parameters

- **dictionary** – a dictionary to remove the value from
- **key** – key that specifies an element in the dictionary

Returns

dictionary after changes have been made

`biothings.utils.dotstring.set_key_value(dictionary, key, value)`

Set values all values in dictionary matching a dotstring key to a specified value.

if key is not found in dictionary, it just skip quietly.

Parameters

- **dictionary** – a dictionary to set values in
- **key** – key that specifies an element in the dictionary

Returns

dictionary after changes have been made

6.9.11 biothings.utils.es

```
class biothings.utils.es.Collection(colname, db)
    Bases: object
    count()
    find(filter=None, projection=None, *args, **kwargs)
    find_one(*args, **kwargs)
    insert_one(document, *args, **kwargs)
    remove(query)
    replace_one(filter, replacement, upsert=False, *args, **kwargs)
    save(doc, *args, **kwargs)
    update(*args, **kwargs)
    update_many(filter, update, upsert=False, *args, **kwargs)
    update_one(filter, update, upsert=False, *args, **kwargs)
```

`class biothings.utils.es.Database`

Bases: `IDatabase`

`CONFIG = None`

property address

Returns sufficient information so a connection to a database can be created. Information can be a dictionary, object, etc... and depends on the actual backend

create_collection(*colname*)

Create a table/collecton named colname. If backend is using a schema-based database (ie. SQL), backend should enforce the schema with at least field “_id” as the primary key (as a string).

class biothings.utils.es.ESIndex(*client, index_name*)

Bases: object

An Elasticsearch Index Wrapping A Client. Counterpart for pymongo.collection.Collection

property doc_type**class biothings.utils.es.ESIndexer(*index, doc_type=_doc, es_host='localhost:9200', step=500, step_size=10, number_of_shards=1, number_of_replicas=0, check_index=True, **kwargs*)**

Bases: object

build_index(kwargs)****check_index()**

Check if index is an alias, and update self._index to point to actual index

TODO: the overall design of ESIndexer is not great. If we are exposing ES

implementation details (such as the abilities to create and delete indices, create and update aliases, etc.) to the user of this Class, then this method doesn't seem that out of place.

clean_field(*field, dryrun=True, step=5000*)

remove a top-level field from ES index, if the field is the only field of the doc, remove the doc as well. step is the size of bulk update on ES try first with dryrun turned on, and then perform the actual updates with dryrun off.

close()**count(**kwargs)****count_src(**kwargs)****create_index(**kwargs)****create_repository(*repo_name, settings*)****delete_doc(*id*)**

delete a doc from the index based on passed id.

delete_docs(*ids, step=None*)

delete a list of docs in bulk.

delete_index(*index=None*)**doc_feeder(**kwargs)****doc_feeder_using_helper(**kwargs)****exists(**kwargs)**

return True/False if a biotesting id exists or not.

exists_index(*index: str | None = None*)**find_biggest_doc(*fields_li, min=5, return_doc=False*)**

return the doc with the max number of fields from fields_li.

flush_and_refresh()**get_alias**(*index: str | None = None, alias_name: str | None = None*) → List[str]

Get indices with alias associated with given index name or alias name

Parameters

- **index** – name of index
- **alias_name** – name of alias

Returns

Mapping of index names with their aliases

get_biothing(kwargs)****get_docs(**kwargs)**

Return matching docs for given ids iterable, if not found return None. A generator is returned to the matched docs. If only_source is False, the entire document is returned, otherwise only the source is returned.

get_id_list(kwargs)****get_indice_names_by_settings**(*index: str | None = None, sort_by_creation_date=False, reverse=False*) → List[str]

Get list of indices names associated with given index name, using indices' settings

Parameters

- **index** – name of index
- **sort_by_creation_date** – sort the result by indice's creation_date
- **reverse** – control the direction of the sorting

Returns

list of index names (str)

get_indices_from_snapshots(*repo_name, snapshot_name*)**get_internal_number_of_replicas()****get_mapping()**

return the current index mapping

get_mapping_meta()

return the current _meta field.

get_repository(*repo_name*)**get_restore_status**(*index_name=None*)**get_settings**(*index: str | None = None*) → Mapping[str, Mapping]

Get indices with settings associated with given index name

Parameters**index** – name of index**Returns**

Mapping of index names with their settings

get_snapshot_status(*repo, snapshot*)

get_snapshots(repo_name, snapshot_name)

index(doc, id=None, action='index')

add a doc to the index. If id is not None, the existing doc will be updated.

index_bulk(docs, step=None, action='index')

mexists(kwargs)**

open()

optimize(kwargs)**

optimize the default index.

reindex(src_index, is_remote=False, **kwargs)

In order to reindex from remote, - src es_host must be set to an IP which the current ES host can connect to.

It means that if 2 indices locate in same host, the es_host can be set to localhost, but if they are in different hosts, an IP must be used instead.

- If src host uses SSL, https must be included in es_host. Eg: <https://192.168.1.10:9200>

- **src host must be whitelisted in the current ES host.**

Ref: <https://www.elastic.co/guide/en/elasticsearch/reference/7.17/reindex-upgrade-remote.html>

restore(repo_name, snapshot_name, index_name=None, purge=False)

sanitize_settings(settings)

Clean up settings dictionary to remove those static fields cannot be updated.

like: "uuid", "provided_name", "creation_date", "version",

settings will be updated both in-place and returned as well.

set_internal_number_of_replicas(number_of_replicas=None)

snapshot(repo, snapshot, mode=None, **params)

update(id, extra_doc, upsert=True)

update an existing doc with extra_doc. allow to set upsert=True, to insert new docs.

update_alias(alias_name: str, index: str | None = None)

Create or update an ES alias pointing to an index

Creates or updates an alias in Elasticsearch, associated with the given index name or the underlying index of the ESIndexer instance.

When the alias name does not exist, it will be created. If an existing alias already exists, it will be updated to only associate with the index.

When the alias name already exists, an exception will be raised, UNLESS the alias name is the same as index name that the ESIndexer is initialized with. In this case, the existing index with the name collision will be deleted, and the alias will be created in its place. This feature is intended for seamless migration from an index to an alias associated with an index for zero-downtime installs.

Parameters

- **alias_name** – name of the alias
- **index** – name of the index to associate with alias. If None, the index of the ESIndexer instance is used.

Raises

[**IndexerException**](#) –

update_docs(*partial_docs*, *upsert=True*, *step=None*, ***kwargs*)

update a list of partial_docs in bulk. allow to set upsert=True, to insert new docs.

update_mapping(*m*)

update_mapping_meta(*meta*)

update_settings(*settings*, *close=False*, ***params*)

Parameters

- **settings** (-) – should be valid ES index's settings.
- **close** (-) – In order to update static settings, the index must be closed first.

Ref: <https://www.elastic.co/guide/en/elasticsearch/reference/7.17/index-modules.html#index-modules-settings>

exception biothings.utils.es.**IndexerException**

Bases: `Exception`

exception biothings.utils.es.**MappingError**

Bases: `Exception`

`biothings.utils.es.generate_es_mapping(inspect_doc, init=True, level=0)`

Generate an ES mapping according to "inspect_doc", which is produced by `biothings.utils.inspect` module

`biothings.utils.es.get_api()`

`biothings.utils.es.get_cmd()`

`biothings.utils.es.get_data_plugin()`

`biothings.utils.es.get_doc_type(es_client, index_name)`

`biothings.utils.es.get_es(es_host, timeout=120, max_retries=3, retry_on_timeout=False)`

`biothings.utils.es.get_event()`

`biothings.utils.es.get_hub_config()`

`biothings.utils.es.get_hub_db_conn()`

`biothings.utils.es.get_last_command()`

`biothings.utils.es.get_source_fullname(col_name)`

`biothings.utils.es.get_src_build()`

`biothings.utils.es.get_src_build_config()`

`biothings.utils.es.get_src_conn()`

`biothings.utils.es.get_src_db(conn=None)`

`biothings.utils.es.get_src_dump()`

`biothings.utils.es.get_src_master()`

```
biothings.utils.es.verify_ids(doc_iter, es_host, index, doc_type=None, step=100000)
```

verify how many docs from input interator/list overlapping with existing docs.

```
biothings.utils.es.wrapper(func)
```

this wrapper allows passing index and doc_type from wrapped method.

6.9.12 biothings.utils.exclude_ids

```
class biothings.utils.exclude_ids.ExcludeFieldsById(exclusion_ids, field_lst, min_list_size=1000)
```

Bases: object

This class provides a framework to exclude fields for certain identifiers. Up to three arguments are passed to this class, an identifier list, a list of fields to remove, and minimum list size. The identifier list is a list of document identifiers to act on. The list of fields are fields that will be removed; they are specified using a dotstring notation. The minimum list size is the minimum number of elements that should be in a list in order for it to be removed. The ‘drugbank’, ‘chebi’, and ‘ndc’ data sources were manually tested with this class.

Fields to truncate are specified by field_lst. The dot-notation is accepted.

6.9.13 biothings.utils.hub

```
exception biothings.utils.hub.AlreadyRunningException
```

Bases: Exception

```
class biothings.utils.hub.BaseHubReloader(paths, reload_func, wait=5.0)
```

Bases: object

Monitor sources’ code and reload hub accordingly to update running code

Monitor given paths for directory deletion/creation and for file deletion/creation. Poll for events every ‘wait’ seconds.

```
poll()
```

Start monitoring changes on files and/directories

```
watched_files()
```

Return a list of files/directories being watched

```
class biothings.utils.hub.CommandDefinition
```

Bases: dict

```
exception biothings.utils.hub.CommandError
```

Bases: Exception

```
class biothings.utils.hub.CommandInformation
```

Bases: dict

```
exception biothings.utils.hub.CommandNotAllowed
```

Bases: Exception

```
class biothings.utils.hub.CompositeCommand(cmd)
```

Bases: str

Defines a composite hub commands, that is, a new command made of other commands. Useful to define shortcuts when typing commands in hub console.

```
class biothings.utils.hub.HubShell(**kwargs: Any)
```

Bases: InteractiveShell

Create a configurable given a config config.

Parameters

- **config** (*Config*) – If this is empty, default values are used. If config is a Config instance, it will be used to configure the instance.
- **parent** (*Configurable instance, optional*) – The parent Configurable instance of this object.

Notes

Subclasses of Configurable must call the `__init__()` method of Configurable *before* doing anything else and using `super()`:

```
class MyConfigurable(Configurable):
    def __init__(self, config=None):
        super(MyConfigurable, self).__init__(config=config)
        # Then any other code you need to finish initialization.
```

This ensures that instances will be configured properly.

`cmd = None`

`cmd_cnt = None`

`classmethod command_info(id=None, running=None, failed=None)`

`eval(line, return_cmdinfo=False, secure=False)`

`extract_command_name(cmd)`

`help(func=None)`

Display help on given function/object or list all available commands

`launch(pfunc)`

Helper to run a command and register it pfunc is partial taking no argument. Command name is generated from partial's func and arguments

`launched_commands = []`

`pending_outputs = []`

`classmethod refresh_commands()`

`register_command(cmd, result, force=False)`

Register a command ‘cmd’ inside the shell (so we can keep track of it). ‘result’ is the original value that was returned when cmd was submitted. Depending on the type, returns a cmd number (ie. result was an asyncio task and we need to wait before getting the result) or directly the result of ‘cmd’ execution, returning, in that case, the output.

`register_managers(managers)`

`restart(force=False, stop=False)`

```
classmethod save_cmd(_id, cmd)
classmethod set_command_counter()
set_commands(basic_commands, *extra_ns)
stop(force=False)

exception biothings.utils.hub.NoSuchCommand
Bases: Exception

class biothings.utils.hub.TornadoAutoReloadHubReloader(paths, reload_func, wait=5)
Bases: BaseHubReloader
Reloader based on tornado.autoreload module
Monitor given paths for directory deletion/creation and for file deletion/creation. Poll for events every ‘wait’ seconds.

add_watch(paths)
This method recursively adds the input paths, and their children to tornado autoreload for watching them. If any file changes, the tornado will call our hook to reload the hub.

Each path will be forced to become an absolute path. If a path is matched excluding patterns, it will be ignored. Only file is added for watching. Directory will be passed to another add_watch.

monitor()

watched_files()
Return a list of files/directories being watched

biothings.utils.hub.exclude_from_reloader(path)

biothings.utils.hub.get_hub_reloader(*args, **kwargs)

biothings.utils.hub.jsonreadify(cmd)

biothings.utils.hub.publish_data_version(s3_bucket, s3_folder, version_info, update_latest=True,
aws_key=None, aws_secret=None)
```

Update remote files:

- **versions.json: add version_info to the JSON list**
or replace if arg version_info is a list
- latest.json: update redirect so it points to latest version url

“versions” is dict such as:

```
{"build_version": "...",           # version name for this release/build
 "require_version": "...",          # version required for incremental update
 "target_version": "...",           # version reached once update is applied
 "type" : "incremental|full"       # release type
 "release_date" : "...",            # ISO 8601 timestamp, release date/time
 "url": "http...."}                # url pointing to release metadata
```

biothings.utils.hub.stats(src_dump)

`biothings.utils.hub.template_out(field, confdict)`

Return field as a templated-out filed, substituting some “%(...s” part with confdict. Fields can follow dotfield notation. Fields like “\$(...)” are replaced with a timestamp following specified format (see time.strftime) Example:

```
confdict = {"a": "one"}
field = "%(a)s_two_three_$(%Y%m)"
=> "one_two_three_201908" # assuming we're in August 2019
```

6.9.14 `biothings.utils.hub_db`

hub_db module is a place-holder for internal hub database functions. Hub DB contains informations about sources, configurations variables, etc... It's for internal usage. When biothings.config_for_app() is called, this module will be “filled” with the actual implementations from the specified backend (specified in config.py, or defaulting to MongoDB).

Hub DB can be implemented over different backend, it's originally been done using MongoDB, so the dialect is very inspired by pymongo. Any hub db backend implementation must implement the functions and classes below. See biothings.utils.mongo and biothings.utils.sqlite3 for some examples.

```
class biothings.utils.hub_db.ChangeListener
    Bases: object
    read()

class biothings.utils.hub_db.ChangeWatcher
    Bases: object
    classmethod add(listener)

    col_entity = {'cmd': 'command', 'hub_config': 'config', 'src_build': 'build',
                  'src_build_config': 'build_config', 'src_dump': 'source', 'src_master': 'master'}

    do_publish = False

    event_queue = <Queue at 0x7ffb4781b760 maxsize=0>

    listeners = {}

    classmethod monitor(func, entity, op)

    classmethod publish()

    classmethod wrap(getfunc)
```

`class biothings.utils.hub_db.Collection(colname, db)`

Bases: object

Defines a minimal subset of MongoDB collection behavior. Note: Collection instances must be pickleable (if not, __getstate__ can be implemented to deal with those attributes for instance)

Init args can differ depending on the backend requirements. colname is the only one required.

`count()`

Return the number of documents in the collection

`database()`

Return the database name

find(*args, **kwargs)

Return an iterable of documents matching criterias defined in `*args[0]` (which will be a dict). Query dialect is a minimal one, inspired by MongoDB. Dict can contain the name of a key, and the value being searched for. Ex: `{"field1": "value1"}` will return all documents where `field1 == "value1"`. Nested key (`field1.subfield1`) aren't supported (no need to implement). Exact matches only are required.

If no query is passed, or if query is an empty dict, return all documents.

find_one(*args, **kwargs)

Return one document from the collection. `*args` will contain a dict with the query parameters. See also `find()`

insert_one(doc)

Insert a document in the collection. Raise an error if already inserted

property name

Return the collection/table name

remove(query)

Delete all documents matching 'query'

replace_one(query, doc)

Replace a document matching 'query' (or the first found one) with passed doc

save(doc)

Shortcut to `update_one()` or `insert_one()`. Save the document, by either inserting if it doesn't exist, or update existing one

update(query, what)

Same as `update_one()` but operate on all documents matching 'query'

update_one(query, what, upsert=False)

Update one document (or the first matching query). See `find()` for query parameter. "what" tells how to update the document. `$set/$unset/$push` operators must be implemented (refer to MongoDB documentation for more). Nested keys operation aren't necessary.

class biothings.utils.hub_db.IDatabase

Bases: `object`

This class declares an interface and partially implements some of it, mimicking `mongokit.Connection` class. It's used to keep used document model. Any internal backend should implement (derives) this interface

property address

Returns sufficient information so a connection to a database can be created. Information can be a dictionary, object, etc... and depends on the actual backend

collection_names()

Return a list of all collections (or tables) found in this database

create_collection(colname)

Create a table/collecton named `colname`. If backend is using a schema-based database (ie. SQL), backend should enforce the schema with at least field "`_id`" as the primary key (as a string).

biothings.utils.hub_db.backup(folder='.', archive=None)

Dump the whole `hub_db` database in given folder. "archive" can be pass to specify the target filename, otherwise, it's randomly generated

Note: this doesn't backup source/merge data, just the internal data used by the hub

```
biothings.utils.hub_db.get_api()
biothings.utils.hub_db.get_cmd()
biothings.utils.hub_db.get_data_plugin()
biothings.utils.hub_db.get_event()
biothings.utils.hub_db.get_hub_config()
biothings.utils.hub_db.get_src_build()
biothings.utils.hub_db.get_src_build_config()
biothings.utils.hub_db.get_src_dump()
biothings.utils.hub_db.get_src_master()
biothings.utils.hub_db.restore(archive, drop=False)
    Restore database from given archive. If drop is True, then delete existing collections
biothings.utils.hub_db.setup(config)
```

6.9.15 biothings.utils.info

```
class biothings.utils.info.DevInfo
    Bases: object
    get()

class biothings.utils.info.FieldNote(path)
    Bases: object
    get_field_notes()
        Return the cached field notes associated with this instance.
```

6.9.16 biothings.utils.inspect

This module contains util functions may be shared by both BioThings data-hub and web components. In general, do not include utils depending on any third-party modules. Note: unitests available in biothings.tests.hub

```
class biothings.utils.inspect.BaseMode
    Bases: object
    key = None

    merge(target, tomerge)
        Merge two different maps together (from tomerge into target)

    post(mapt, mode, clean)
```

```
report(struct, drep, orig_struct=None)
```

Given a data structure “struct” being inspected, report (fill) “drep” dictionary with useful values for this mode, under drep[self.key] key. Sometimes “struct” is already converted to its analytical value at this point (inspect may count number of dict and would force to pass struct as “1”, instead of the whole dict, where number of keys could be then be reported), “orig_struct” is that case contains the original structure that was to be reported, whatever the pre-conversion step did.

```
template = {}
```

```
class biothings.utils.inspect.DeepStatsMode
```

Bases: *StatsMode*

```
key = '_stats'
```

```
merge(target_stats, tomerge_stats)
```

Merge two different maps together (from tomerge into target)

```
post(mapt, mode, clean)
```

```
report(val, drep, orig_struct=None)
```

Given a data structure “struct” being inspected, report (fill) “drep” dictionary with useful values for this mode, under drep[self.key] key. Sometimes “struct” is already converted to its analytical value at this point (inspect may count number of dict and would force to pass struct as “1”, instead of the whole dict, where number of keys could be then be reported), “orig_struct” is that case contains the original structure that was to be reported, whatever the pre-conversion step did.

```
template = {'_stats': {'__vals': [], '_count': 0, '_max': -inf, '_min': inf}}
```

```
class biothings.utils.inspect.FieldInspectValidation(warnings: set() = <factory>, types: set = <factory>, has_multiple_types: bool = False)
```

Bases: *object*

```
has_multiple_types: bool = False
```

```
types: set
```

```
warnings: set()
```

```
class biothings.utils.inspect.FieldInspection(field_name: str, field_type: str, stats: dict = None, warnings: list = <factory>)
```

Bases: *object*

```
field_name: str
```

```
field_type: str
```

```
stats: dict = None
```

```
warnings: list
```

```
class biothings.utils.inspect.IdentifiersMode
```

Bases: *RegexMode*

```
ids = None
```

```
key = '_ident'
```

```
matchers = None
```

class biothings.utils.inspect.InspectionValidation

Bases: object

This class provide a mechanism to validate and flag any field which:

- contains whitespace
- contains upper cased letter or special characters

(lower-cased is recommended, in some cases the upper-case field names are acceptable, so we should raise it as a warning and let user to confirm it's necessary)

- **when the type inspection detects more than one types**

(but a mixed or single value and an array of same type of values are acceptable, or the case of mixed integer and float should be acceptable too)

Usage:

```
` result = InspectionValidation.validate(data) `
```

Adding more rules:

- add new code, and message to Warning Enum
- add a new staticmethod for validate new rule and named in format: *validate_{warning_code}*
- add new rule to docstring.

```
INVALID_CHARACTERS_PATTERN = '[^a-zA-Z0-9_.]'
```

```
NUMERIC_FIELDS = ['int', 'float']
```

```
SPACE_PATTERN = ' '
```

class Warning(value)

Bases: Enum

An enumeration.

```
W001 = 'field name contains whitespace.'
```

```
W002 = 'field name contains uppercase.'
```

```
W003 = 'field name contains special character. Only alphanumeric, dot, or underscore are valid.'
```

```
W004 = 'field name has more than one type.'
```

```
to_dict()
```

```
static validate(data: List[FieldInspection]) → Dict[str, FieldInspectValidation]
```

```
static validate_W001(field_inspection: FieldInspection, field_validation: FieldInspectValidation) → bool
```

```
static validate_W002(field_inspection: FieldInspection, field_validation: FieldInspectValidation) → bool
```

```
static validate_W003(field_inspection: FieldInspection, field_validation: FieldInspectValidation) → bool
```

```
static validate_W004(field_inspection: FieldInspection, field_validation: FieldInspectValidation) → bool
```

class biothings.utils.inspect.RegexMode

Bases: BaseMode

```
matchers = []
```

merge(target, tomerge)

Merge two different maps together (from tomerge into target)

```
report(val, drep, orig_struct=None)
```

Given a data structure “struct” being inspected, report (fill) “drep” dictionary with useful values for this mode, under drep[self.key] key. Sometimes “struct” is already converted to its analytical value at this point (inspect may count number of dict and would force to pass struct as “1”, instead of the whole dict, where number of keys could be then be reported), “orig_struct” is that case contains the original structure that was to be reported, whatever the pre-conversion step did.

```
class biothings.utils.inspect.StatsMode
```

Bases: *BaseMode*

```
flatten_stats(stats)
```

```
key = '_stats'
```

```
maxminiflist(val, func)
```

```
merge(target_stats, tomerge_stats)
```

Merge two different maps together (from tomerge into target)

```
report(struct, drep, orig_struct=None)
```

Given a data structure “struct” being inspected, report (fill) “drep” dictionary with useful values for this mode, under drep[self.key] key. Sometimes “struct” is already converted to its analytical value at this point (inspect may count number of dict and would force to pass struct as “1”, instead of the whole dict, where number of keys could be then be reported), “orig_struct” is that case contains the original structure that was to be reported, whatever the pre-conversion step did.

```
sumiflist(val)
```

```
template = {'_stats': {'_count': 0, '_max': -inf, '_min': inf, '_none': 0}}
```

```
biothings.utils.inspect.compute_metadata(mapt, mode)
```

```
biothings.utils.inspect.flatten_and_validate(data, do_validate=True)
```

```
biothings.utils.inspect.flatten_inspection_data(data: Dict[str, Any], current_deep: int = 0,  
                                                parent_name: str | None = None, parent_type: str |  
                                                None = None) → List[FieldInspection]
```

This function will convert the multiple depth nested inspection data into a flatten list Nested key will be appended with the parent key and seperate with a dot.

```
biothings.utils.inspect.get_converters(modes, logger=<module 'logging' from  
                                         '/home/docs/.asdf/install/python/3.10.13/lib/python3.10/logging/__init__.py'>)
```

```
biothings.utils.inspect.get_mode_layer(mode)
```

```
biothings.utils.inspect.inspect(struct, key=None, mapt=None, mode='type', level=0, logger=<module  
                                         'logging' from  
                                         '/home/docs/.asdf/install/python/3.10.13/lib/python3.10/logging/__init__.py'>)
```

Explore struct and report types contained in it.

Parameters

- **struct** – is the data structure to explore
- **mapt** – if not None, will complete that type map with passed struct. This is useful when iterating over a dataset of similar data, trying to find a good type summary contained in that dataset.
- **level** – is for internal purposes, mostly debugging

- **mode** – see inspect_docs() documentation

```
biothings.utils.inspect.inspect_docs(docs, mode='type', clean=True, merge=False, logger=<module
    'logging' from
    '/home/docs/.asdf/install/python/3.10.13/lib/python3.10/logging/__init__.py'>,
    pre_mapping=False, limit=None, sample=None, metadata=True,
    auto_convert=True)
```

Inspect docs and return a summary of its structure:

Parameters

- **mode** – possible values are:
 - “type”: (default) explore documents and report strict data structure
 - **“mapping”**: same as type but also perform test on data so guess best mapping
(eg. check if a string is splitable, etc...). Implies merge=True
 - “stats”: explore documents and compute basic stats (count,min,max,sum)
 - **“deepstats”**: same as stats but record values and also compute mean,stdev,median
(memory intensive...)
 - “jsonschema”, same as “type” but returned a json-schema formatted result
- **mode** can also be a list of modes, eg. [“type”, “mapping”]. There’s little overhead computing multiple types as most time is spent on actually getting the data.
- **clean** – don’t delete recorded values or temporary results
- **merge** – merge scalar into list when both exist (eg. {“val”...} and [{“val”...}])
- **limit** – can limit the inspection to the x first docs (None = no limit, inspects all)
- **sample** – in combination with limit, randomly extract a sample of ‘limit’ docs (so not necessarily the x first ones defined by limit). If random.random() is greater than sample, doc is inspected, otherwise it’s skipped
- **metadata** – compute metadata on the result
- **auto_convert** – run converters automatically (converters are used to convert one mode’s output to another mode’s output, eg. type to jsonschema)

```
biothings.utils.inspect.merge_field_inspections_validations(field_inspections:
    List[FieldInspection],
    field_validations: Dict[str,
    FieldInspectValidation])
```

Adding any warnings from field_validations to field_inspections with corresponding field name

```
biothings.utils.inspect.merge_record(target, tomerge, mode)
```

```
biothings.utils.inspect.merge_scalar_list(mapt, mode)
```

```
biothings.utils.inspect.run_converters(_map, converters, logger=<module 'logging' from
    '/home/docs/.asdf/install/python/3.10.13/lib/python3.10/logging/__init__.py'>)
```

```
biothings.utils.inspect.simplify_inspection_data(field_inspections: List[FieldInspection]) →
    List[Dict[str, Any]]
```

```
biothings.utils.inspect.stringify_inspect_doc(dmap)
```

```
biothings.utils.inspect.typify_inspect_doc(dmap)
```

dmap is an inspect which was converted to be stored in a database, namely actual python types were stringify to be storables. This function does the oposite and restore back python types within the inspect doc

6.9.17 biothings.utils.jsondiff

The MIT License (MIT)

Copyright (c) 2014 Ilya Volkov

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

```
biothings.utils.jsondiff.make(src, dst, **kwargs)
```

6.9.18 biothings.utils.jsonpatch

Apply JSON-Patches (RFC 6902)

```
class biothings.utils.jsonpatch.AddOperation(operation)
```

Bases: *PatchOperation*

Adds an object property or an array element.

```
apply(obj)
```

Abstract method that applies patch operation to specified object.

```
class biothings.utils.jsonpatch.COPYOperation(operation)
```

Bases: *PatchOperation*

Copies an object property or an array element to a new location

```
apply(obj)
```

Abstract method that applies patch operation to specified object.

```
exception biothings.utils.jsonpatch.InvalidJsonPatch
```

Bases: *JsonPatchException*

Raised if an invalid JSON Patch is created

```
class biothings.utils.jsonpatch.JsonPatch(patch)
```

Bases: *object*

A JSON Patch is a list of Patch Operations.

```
>>> patch = JsonPatch([
...     {'op': 'add', 'path': '/foo', 'value': 'bar'},
...     {'op': 'add', 'path': '/baz', 'value': [1, 2, 3]},
...     {'op': 'remove', 'path': '/baz/1'},
...     {'op': 'test', 'path': '/baz', 'value': [1, 3]},
...     {'op': 'replace', 'path': '/baz/0', 'value': 42},
...     {'op': 'remove', 'path': '/baz/1'},
... ])
>>> doc = {}
>>> result = patch.apply(doc)
>>> expected = {'foo': 'bar', 'baz': [42]}
>>> result == expected
True
```

JsonPatch object is iterable, so you could easily access to each patch statement in loop:

```
>>> lpatch = list(patch)
>>> expected = {'op': 'add', 'path': '/foo', 'value': 'bar'}
>>> lpatch[0] == expected
True
>>> lpatch == patch.patch
True
```

Also JsonPatch could be converted directly to bool if it contains any operation statements:

```
>>> bool(patch)
True
>>> bool(JsonPatch([]))
False
```

This behavior is very handy with `make_patch()` to write more readable code:

```
>>> old = {'foo': 'bar', 'numbers': [1, 3, 4, 8]}
>>> new = {'baz': 'qux', 'numbers': [1, 4, 7]}
>>> patch = make_patch(old, new)
>>> if patch:
...     # document have changed, do something useful
...     patch.apply(old)
{...}
```

apply(*orig_obj*, *in_place=False*, *ignore_conflicts=False*, *verify=False*)

Applies the patch to given object.

Parameters

- **obj** (*dict*) – Document object.
- **in_place** (*bool*) – Tweaks way how patch would be applied - directly to specified *obj* or to his copy.

Returns

Modified *obj*.

classmethod from_diff(*src*, *dst*)

Creates JsonPatch instance based on comparing of two document objects. Json patch would be created for *src* argument against *dst* one.

Parameters

- **src** (*dict*) – Data source document object.
- **dst** (*dict*) – Data source document object.

Returns

JsonPatch instance.

```
>>> src = {'foo': 'bar', 'numbers': [1, 3, 4, 8]}
>>> dst = {'baz': 'qux', 'numbers': [1, 4, 7]}
>>> patch = JsonPatch.from_diff(src, dst)
>>> new = patch.apply(src)
>>> new == dst
True
```

classmethod from_string(*patch_str*)

Creates *JsonPatch* instance from string source.

Parameters

- patch_str** (*str*) – JSON patch as raw string.

Returns

JsonPatch instance.

to_string()

Returns patch set as JSON string.

exception biothings.utils.jsonpatch.JsonPatchConflict

Bases: *JsonPatchException*

Raised if patch could not be applied due to conflict situation such as: - attempt to add object key then it already exists; - attempt to operate with nonexistence object key; - attempt to insert value to array at position beyond of its size; - etc.

exception biothings.utils.jsonpatch.JsonPatchException

Bases: *Exception*

Base Json Patch exception

exception biothings.utils.jsonpatch.JsonPatchTestFailed

Bases: *JsonPatchException*, *AssertionError*

A Test operation failed

class biothings.utils.jsonpatch.MoveOperation(*operation*)

Bases: *PatchOperation*

Moves an object property or an array element to new location.

apply(*obj*)

Abstract method that applies patch operation to specified object.

class biothings.utils.jsonpatch.PatchOperation(*operation*)

Bases: *object*

A single operation inside a JSON Patch.

apply(*obj*)

Abstract method that applies patch operation to specified object.

```
class biothings.utils.jsonpatch.RemoveOperation(operation)
```

Bases: *PatchOperation*

Removes an object property or an array element.

```
apply(obj)
```

Abstract method that applies patch operation to specified object.

```
class biothings.utils.jsonpatch.ReplaceOperation(operation)
```

Bases: *PatchOperation*

Replaces an object property or an array element by new value.

```
apply(obj)
```

Abstract method that applies patch operation to specified object.

```
class biothings.utils.jsonpatch.TestOperation(operation)
```

Bases: *PatchOperation*

Test value by specified location.

```
apply(obj)
```

Abstract method that applies patch operation to specified object.

```
biothings.utils.jsonpatch.apply_patch(doc, patch, in_place=False, ignore_conflicts=False, verify=False)
```

Apply list of patches to specified json document.

Parameters

- **doc** (*dict*) – Document object.
- **patch** (*list or str*) – JSON patch as list of dicts or raw JSON-encoded string.
- **in_place** (*bool*) – While True patch will modify target document. By default patch will be applied to document copy.
- **ignore_conflicts** (*bool*) – Ignore JsonConflicts errors
- **verify** (*bool*) – works with *ignore_conflicts* = True, if errors and *verify* is True (recommended), make sure the resulting objects is the same as the original one. *ignore_conflicts* and *verify* are used to run patches multiple times and get rid of errors when operations can't be performed multiple times because the object has already been patched. This will force *in_place* to False in order the comparison to occur.

Returns

Patched document object.

Return type

dict

```
>>> doc = {'foo': 'bar'}
>>> patch = [{"op": "add", "path": "/baz", "value": "qux"}]
>>> other = apply_patch(doc, patch)
>>> doc is not other
True
>>> other == {'foo': 'bar', 'baz': 'qux'}
True
>>> patch = [{"op": "add", "path": "/baz", "value": "qux"}]
>>> apply_patch(doc, patch, in_place=True) == {'foo': 'bar', 'baz': 'qux'}
True
```

(continues on next page)

(continued from previous page)

```
>>> doc == other  
True
```

`biothings.utils.jsonpatch.get_loadjson()`

adds the `object_pairs_hook` parameter to `json.load` when possible

The “`object_pairs_hook`” parameter is used to handle duplicate keys when loading a JSON object. This parameter does not exist in Python 2.6. This methods returns an unmodified `json.load` for Python 2.6 and a partial function with `object_pairs_hook` set to `multidict` for Python versions that support the parameter.

`biothings.utils.jsonpatch.make_patch(src, dst)`

Generates patch by comparing of two document objects. Actually is a proxy to `JsonPatch.from_diff()` method.

Parameters

- `src (dict)` – Data source document object.
- `dst (dict)` – Data source document object.

```
>>> src = {'foo': 'bar', 'numbers': [1, 3, 4, 8]}  
>>> dst = {'baz': 'qux', 'numbers': [1, 4, 7]}  
>>> patch = make_patch(src, dst)  
>>> new = patch.apply(src)  
>>> new == dst  
True
```

`biothings.utils.jsonpatch.multidict(ordered_pairs)`

Convert duplicate keys values to lists.

`biothings.utils.jsonpatch.reapply_patch(doc, patch)`

Apply or (safely) re-apply patch to doc

6.9.19 `biothings.utils.jsonschema`

`biothings.utils.jsonschema.generate_json_schema(dmap)`

`biothings.utils.jsonschema.test()`

6.9.20 `biothings.utils.loggers`

`class biothings.utils.loggers.Colors(value)`

Bases: `Enum`

An enumeration.

`CRITICAL = '#7b0099'`

`DEBUG = '#a1a1a1'`

`ERROR = 'danger'`

`INFO = 'good'`

```

NOTSET = '#d6d2d2'

WARNING = 'warning'

class biothings.utils.loggers.EventRecorder(*args, **kwargs)
Bases: StreamHandler
Initialize the handler.
If stream is not specified, sys.stderr is used.

emit(record)
Emit a record.

If a formatter is specified, it is used to format the record. The record is then written to the stream with a trailing newline. If exception information is present, it is formatted using traceback.print_exception and appended to the stream. If the stream has an ‘encoding’ attribute, it is used to determine how to do the output to the stream.

class biothings.utils.loggers.GZipRotator
Bases: object

class biothings.utils.loggers.LookUpList(initlist)
Bases: UserList

find(val)
find_index(val)

class biothings.utils.loggers.Range(start: int | float = 0, end: int | float = inf)
Bases: object

end: int | float = inf
start: int | float = 0

class biothings.utils.loggers.ReceiverGroup(initlist=None)
Bases: UserList

class biothings.utils.loggers.Record(range, value)
Bases: NamedTuple
Create new instance of Record(range, value)

range: Range
Alias for field number 0

value: Enum
Alias for field number 1

class biothings.utils.loggers.ShellLogger(*args, **kwargs)
Bases: Logger
Custom “levels” for input going to the shell and output coming from it (just for naming)
Initialize the logger with a name and an optional level.

INPUT = 1001
OUTPUT = 1000

```

```
input(msg, *args, **kwargs)
output(msg, *args, **kwargs)

class biothings.utils.loggers.SlackHandler(webhook, mentions)
Bases: StreamHandler
Initialize the handler.
If stream is not specified, sys.stderr is used.

emit(record)
Emit a record.
If a formatter is specified, it is used to format the record. The record is then written to the stream with a trailing newline. If exception information is present, it is formatted using traceback.print_exception and appended to the stream. If the stream has an 'encoding' attribute, it is used to determine how to do the output to the stream.

static send(webhook, message, level, mentions=())
class biothings.utils.loggers.SlackMentionPolicy(policy)
Bases: object
mentions(level)

class biothings.utils.loggers.SlackMessage
Bases: object
build()
markdown(text, prefixes=(), suffixes=())
plaintext(text, color)

class biothings.utils.loggers.Squares(value)
Bases: Enum
An enumeration.
CRITICAL = ':large_purple_square:'
DEBUG = ':white_large_square:'
ERROR = ':large_red_square:'
INFO = ':large_blue_square:'
NOTSET = ''
WARNING = ':large_orange_square:'

class biothings.utils.loggers.WSLogHandler(listener)
Bases: StreamHandler
when listener is a bt.hub.api.handlers.ws.LogListener instance, log statements are propagated through existing websocket
Initialize the handler.
If stream is not specified, sys.stderr is used.
```

emit(record)

Emit a record.

If a formatter is specified, it is used to format the record. The record is then written to the stream with a trailing newline. If exception information is present, it is formatted using traceback.print_exception and appended to the stream. If the stream has an ‘encoding’ attribute, it is used to determine how to do the output to the stream.

payload(record)**class biothings.utils.loggers.WSShellHandler(listener)**

Bases: [WSLogHandler](#)

when listener is a bt.hub.api.handlers.ws.LogListener instance, log statements are propagated through existing websocket

Initialize the handler.

If stream is not specified, sys.stderr is used.

payload(record)**biothings.utils.loggers.configure_file_handler(logger, logfile, formater=None, force=False)****biothings.utils.loggers.create_logger(log_folder, logger_name, level=10)**

Create and return a file logger if log_folder is provided. If log_folder is None, no file handler will be created.

biothings.utils.loggers.get_logger(logger_name, log_folder=None, handlers=('console', 'file', 'slack'), timestamp=None, force=False)

Configure a logger object from logger_name and return (logger, logfile)

biothings.utils.loggers.setup_default_log(default_logger_name, log_folder, level=10)

6.9.21 biothings.utils.manager

class biothings.utils.manager.BaseManager(job_manager, poll_schedule=None)

Bases: object

clean_stale_status()

During startup, search for action in progress which would have been interrupted and change the state to “canceled”. Ex: some downloading processes could have been interrupted, at startup, “downloading” status should be changed to “canceled” so to reflect actual state on these datasources. This must be overridden in subclass.

poll(state, func, col)

Search for source in collection ‘col’ with a pending flag list containing ‘state’ and call ‘func’ for each document found (with doc as only param)

class biothings.utils.manager.BaseSourceManager(job_manager, datasource_path='dataload.sources', *args, **kwargs)

Bases: [BaseManager](#)

Base class to provide source management: discovery, registration Actual launch of tasks must be defined in subclasses

SOURCE_CLASS = None

filter_class(klass)

Gives opportunity for subclass to check given class and decide to keep it or not in the discovery process. Returning None means “skip it”.

find_classes(src_module, fail_on_notfound=True)

Given a python module, return a list of classes in this module, matching SOURCE_CLASS (must inherit from)

register_classes(klasses)

Register each class in self.register dict. Key will be used to retrieve the source class, create an instance and run method from it. It must be implemented in subclass as each manager may need to access its sources differently,based on different keys.

register_source(src, fail_on_notfound=True)

Register a new data source. src can be a module where some classes are defined. It can also be a module path as a string, or just a source name in which case it will try to find information from default path.

register_sources(sources)**class biothings.utils.manager.BaseStatusRegisterer**

Bases: object

property collection

Return collection object used to fetch doc in which we store status

load_doc(key_name, stage)

Find document using key_name and stage, stage being a key within the document matching a specific process name: Ex: {“_id”:”123”,”snapshot”:”abc”}

load_doc(“abc”,”snapshot”)

will return the document. Note key_name is first used to find the doc by its _id. Ex: with another doc {“_id” : “abc”, “snapshot” : “somethingelse”}

load_doc{“abc”,”snapshot”}

will return doc with _id=”abc”, not “123”

register_status(doc, stage, status, transient=False, init=False, **extra)**class biothings.utils.manager.CLIJobManager(loop=None)**

Bases: object

This is the minimal JobManager used in CLI mode to run async jobs, with the compatible methods as JobManager. It won’t use a dedicated ProcessPool or ThreadPool, and will just run async job directly in the asyncio loop (which runs jobs in threads by default).

async defer_to_process(pinfo=None, func=None, *args, **kwargs)

keep the same signature as JobManager.defer_to_process. The passed pinfo is ignored. defer_to_process will still run func in the thread using defer_to_thread method.

async defer_to_thread(pinfo=None, func=None, *args)

keep the same signature as JobManager.defer_to_thread. The passed pinfo is ignored

class biothings.utils.manager.JobManager(loop, process_queue=None, thread_queue=None, max_memory_usage=None, num_workers=None, num_threads=None, auto_recycle=True)

Bases: object


```
show_pendings(running=None)
stop(force=False, recycling=False, wait=1)
submit(pfunc, schedule=None)
    Helper to submit and run tasks. Tasks will run async'ly. pfunc is a functools.partial schedule is a string
    representing a cron schedule, task will then be scheduled accordingly.
top(action='summary')

exception biothings.utils.manager.ManagerError
    Bases: Exception

exception biothings.utils.manager.ResourceError
    Bases: Exception

exception biothings.utils.manager.ResourceNotFound
    Bases: Exception

exception biothings.utils.manager.UnknownResource
    Bases: Exception

biothings.utils.manager.do_work(job_id, ptype, pinfo=None, func=None, *args, **kwargs)
biothings.utils.manager.find_process(pid)
biothings.utils.manager.norm(value, maxlen)
    just a helper to clean/prepare job's values printing
biothings.utils.manager.track(func)
```

6.9.22 biothings.utils.mongo

```
class biothings.utils.mongo.Collection(*args, **kwargs)
    Bases: HandleAutoReconnectMixin, Collection
    Get / create a Mongo collection.

    Raises TypeError if name is not an instance of str. Raises InvalidName if name is not a valid collection
    name. Any additional keyword arguments will be used as options passed to the create command. See
    create_collection() for valid options.

    If create is True, collation is specified, or any additional keyword arguments are present, a create command
    will be sent, using session if specified. Otherwise, a create command will not be sent and the collection will
    be created implicitly on first use. The optional session argument is only used for the create command, it is
    not associated with the collection afterward.
```

Parameters

- *database*: the database to get a collection from
- *name*: the name of the collection to get
- *create* (optional): if True, force collection creation even without options being set
- *codec_options* (optional): An instance of CodecOptions. If None (the default) database.codec_options is used.
- *read_preference* (optional): The read preference to use. If None (the default) database.read_preference is used.

- *write_concern* (optional): An instance of `WriteConcern`. If `None` (the default) `database.write_concern` is used.
- *read_concern* (optional): An instance of `ReadConcern`. If `None` (the default) `database.read_concern` is used.
- *collation* (optional): An instance of `Collation`. If a collation is provided, it will be passed to the create collection command.
- *session* (optional): a `ClientSession` that is used with the create collection command
- ***kwargs* (optional): additional keyword arguments will be passed as options for the create collection command

Changed in version 4.2: Added the `clusteredIndex` and `encryptedFields` parameters.

Changed in version 4.0: Removed the `reindex`, `map_reduce`, `inline_map_reduce`, `parallel_scan`, `initialize_unordered_bulk_op`, `initialize_ordered_bulk_op`, `group`, `count`, `insert`, `save`, `update`, `remove`, `find_and_modify`, and `ensure_index` methods. See the [pymongo4-migration-guide](#).

Changed in version 3.6: Added `session` parameter.

Changed in version 3.4: Support the `collation` option.

Changed in version 3.2: Added the `read_concern` option.

Changed in version 3.0: Added the `codec_options`, `read_preference`, and `write_concern` options. Removed the `uuid_subtype` attribute. Collection no longer returns an instance of `Collection` for attribute names with leading underscores. You must use dict-style lookups instead::

```
collection['__my_collection__']
```

Not:

```
collection.__my_collection__
```

See also:

The MongoDB documentation on [collections](#).

```
count(filter=None, **kwargs)
```

```
insert(doc_or_docs, *args, **kwargs)
```

```
remove(spec_or_id=None, **kwargs)
```

```
save(doc, *args, **kwargs)
```

```
update(spec, doc, *args, **kwargs)
```

```
class biothings.utils.mongo.Database(*args, **kwargs)
```

Bases: `HandleAutoReconnectMixin`, `Database`

Get a database by client and name.

Raises `TypeError` if *name* is not an instance of `str`. Raises `InvalidName` if *name* is not a valid database name.

Parameters

- *client*: A `MongoClient` instance.
- *name*: The database name.
- *codec_options* (optional): An instance of `CodecOptions`. If `None` (the default) `client.codec_options` is used.

- *read_preference* (optional): The read preference to use. If `None` (the default) `client.read_preference` is used.
- *write_concern* (optional): An instance of `WriteConcern`. If `None` (the default) `client.write_concern` is used.
- *read_concern* (optional): An instance of `ReadConcern`. If `None` (the default) `client.read_concern` is used.

See also:

The MongoDB documentation on [databases](#).

Changed in version 4.0: Removed the `eval`, `system_js`, `error`, `last_status`, `previous_error`, `reset_error_history`, `authenticate`, `logout`, `collection_names`, `current_op`, `add_user`, `remove_user`, `profiling_level`, `set_profiling_level`, and `profiling_info` methods. See the [pymongo4-migration-guide](#).

Changed in version 3.2: Added the `read_concern` option.

Changed in version 3.0: Added the `codec_options`, `read_preference`, and `write_concern` options. `Database` no longer returns an instance of `Collection` for attribute names with leading underscores. You must use dict-style lookups instead::

```
db['__my_collection__']
```

Not:

```
db.__my_collection__
```

```
collection_names(include_system_collections=True, session=None)
```

```
class biothings.utils.mongo.DatabaseClient(*args, **kwargs)
```

Bases: `HandleAutoReconnectMixin`, `MongoClient`, `IDatabase`

Client for a MongoDB instance, a replica set, or a set of mongoses.

Warning: Starting in PyMongo 4.0, `directConnection` now has a default value of `False` instead of `None`. For more details, see the relevant section of the PyMongo 4.x migration guide: [pymongo4-migration-direct-connection](#).

The client object is thread-safe and has connection-pooling built in. If an operation fails because of a network error, `ConnectionFailure` is raised and the client reconnects in the background. Application code should handle this exception (recognizing that the operation failed) and then continue to execute.

The `host` parameter can be a full [mongodb URI](#), in addition to a simple hostname. It can also be a list of hostnames but no more than one URI. Any port specified in the host string(s) will override the `port` parameter. For username and passwords reserved characters like ‘:’, ‘/’, ‘+’ and ‘@’ must be percent encoded following RFC 2396:

```
from urllib.parse import quote_plus

uri = "mongodb://$s:$s@$s" % (
    quote_plus(user), quote_plus(password), host)
client = MongoClient(uri)
```

Unix domain sockets are also supported. The socket path must be percent encoded in the URI:

```
uri = "mongodb://$s:$s@$s" % (
    quote_plus(user), quote_plus(password), quote_plus(socket_path))
client = MongoClient(uri)
```

But not when passed as a simple hostname:

```
client = MongoClient('/tmp/mongodb-27017.sock')
```

Starting with version 3.6, PyMongo supports `mongodb+srv://` URIs. The URI must include one, and only one, hostname. The hostname will be resolved to one or more DNS SRV records which will be used as the seed list for connecting to the MongoDB deployment. When using SRV URIs, the `authSource` and `replicaSet` configuration options can be specified using TXT records. See the [Initial DNS Seedlist Discovery spec](#) for more details. Note that the use of SRV URIs implicitly enables TLS support. Pass `tls=false` in the URI to override.

Note: MongoClient creation will block waiting for answers from DNS when `mongodb+srv://` URIs are used.

Note: Starting with version 3.0 the MongoClient constructor no longer blocks while connecting to the server or servers, and it no longer raises `ConnectionFailure` if they are unavailable, nor `ConfigurationError` if the user's credentials are wrong. Instead, the constructor returns immediately and launches the connection process on background threads. You can check if the server is available like this:

```
from pymongo.errors import ConnectionFailure
client = MongoClient()
try:
    # The ping command is cheap and does not require auth.
    client.admin.command('ping')
except ConnectionFailure:
    print("Server not available")
```

Warning: When using PyMongo in a multiprocessing context, please read [multiprocessing](#) first.

Note: Many of the following options can be passed using a MongoDB URI or keyword parameters. If the same option is passed in a URI and as a keyword parameter the keyword parameter takes precedence.

Parameters

- `host` (optional): hostname or IP address or Unix domain socket path of a single mongod or mongos instance to connect to, or a mongodb URI, or a list of hostnames (but no more than one mongodb URI). If `host` is an IPv6 literal it must be enclosed in '[' and ']' characters following the RFC2732 URL syntax (e.g. '[::1]' for localhost). Multihomed and round robin DNS addresses are **not** supported.
- `port` (optional): port number on which to connect
- `document_class` (optional): default class to use for documents returned from queries on this client
- `tz_aware` (optional): if `True`, `datetime` instances returned as values in a document by this MongoClient will be timezone aware (otherwise they will be naive)
- `connect` (optional): if `True` (the default), immediately begin connecting to MongoDB in the background. Otherwise connect on the first operation.
- `type_registry` (optional): instance of `TypeRegistry` to enable encoding and decoding of custom types.

- *datetime_conversion*: Specifies how UTC datetimes should be decoded within BSON. Valid options include ‘datetime_ms’ to return as a DatetimeMS, ‘datetime’ to return as a datetime.datetime and raising a ValueError for out-of-range values, ‘datetime_auto’ to return DatetimeMS objects when the underlying datetime is out-of-range and ‘datetime_clamp’ to clamp to the minimum and maximum possible datetimes. Defaults to ‘datetime’. See handling-out-of-range-datetime for details.

Other optional parameters can be passed as keyword arguments:

- *directConnection* (optional): if **True**, forces this client to connect directly to the specified MongoDB host as a standalone. If **false**, the client connects to the entire replica set of which the given MongoDB host(s) is a part. If this is **True** and a mongodb+srv:// URI or a URI containing multiple seeds is provided, an exception will be raised.
- *maxPoolSize* (optional): The maximum allowable number of concurrent connections to each connected server. Requests to a server will block if there are *maxPoolSize* outstanding connections to the requested server. Defaults to 100. Can be either 0 or None, in which case there is no limit on the number of concurrent connections.
- *minPoolSize* (optional): The minimum required number of concurrent connections that the pool will maintain to each connected server. Default is 0.
- *maxIdleTimeMS* (optional): The maximum number of milliseconds that a connection can remain idle in the pool before being removed and replaced. Defaults to *None* (no limit).
- *maxConnecting* (optional): The maximum number of connections that each pool can establish concurrently. Defaults to 2.
- *timeoutMS*: (integer or None) Controls how long (in milliseconds) the driver will wait when executing an operation (including retry attempts) before raising a timeout error. **0** or **None** means no timeout.
- *socketTimeoutMS*: (integer or None) Controls how long (in milliseconds) the driver will wait for a response after sending an ordinary (non-monitoring) database operation before concluding that a network error has occurred. **0** or **None** means no timeout. Defaults to *None* (no timeout).
- *connectTimeoutMS*: (integer or None) Controls how long (in milliseconds) the driver will wait during server monitoring when connecting a new socket to a server before concluding the server is unavailable. **0** or **None** means no timeout. Defaults to **20000** (20 seconds).
- *server_selector*: (callable or None) Optional, user-provided function that augments server selection rules. The function should accept as an argument a list of `ServerDescription` objects and return a list of server descriptions that should be considered suitable for the desired operation.
- *serverSelectionTimeoutMS*: (integer) Controls how long (in milliseconds) the driver will wait to find an available, appropriate server to carry out a database operation; while it is waiting, multiple server monitoring operations may be carried out, each controlled by *connectTimeoutMS*. Defaults to **30000** (30 seconds).
- *waitForQueueTimeoutMS*: (integer or None) How long (in milliseconds) a thread will wait for a socket from the pool if the pool has no free sockets. Defaults to *None* (no timeout).
- *heartbeatFrequencyMS*: (optional) The number of milliseconds between periodic server checks, or *None* to accept the default frequency of 10 seconds.

- *serverMonitoringMode*: (optional) The server monitoring mode to use. Valid values are the strings: “auto”, “stream”, “poll”. Defaults to “auto”.
- *appname*: (string or None) The name of the application that created this MongoClient instance. The server will log this value upon establishing each connection. It is also recorded in the slow query log and profile collections.
- *driver*: (pair or None) A driver implemented on top of PyMongo can pass a `DriverInfo` to add its name, version, and platform to the message printed in the server log when establishing a connection.
- *event_listeners*: a list or tuple of event listeners. See `monitoring` for details.
- *retryWrites*: (boolean) Whether supported write operations executed within this MongoClient will be retried once after a network error. Defaults to True. The supported write operations are:
 - `bulk_write()`, as long as `UpdateMany` or `DeleteMany` are not included.
 - `delete_one()`
 - `insert_one()`
 - `insert_many()`
 - `replace_one()`
 - `update_one()`
 - `find_one_and_delete()`
 - `find_one_and_replace()`
 - `find_one_and_update()`

Unsupported write operations include, but are not limited to, `aggregate()` using the `$out` pipeline operator and any operation with an unacknowledged write concern (e.g. `{w: 0}`). See <https://github.com/mongodb/specifications/blob/master/source/retryable-writes/retryable-writes.rst>

- *retryReads*: (boolean) Whether supported read operations executed within this MongoClient will be retried once after a network error. Defaults to True. The supported read operations are: `find()`, `find_one()`, `aggregate()` without `$out`, `distinct()`, `count()`, `estimated_document_count()`, `count_documents()`, `pymongo.collection.Collection.watch()`, `list_indexes()`, `pymongo.database.Database.watch()`, `list_collections()`, `pymongo.mongo_client.MongoClient.watch()`, and `list_databases()`.

Unsupported read operations include, but are not limited to `command()` and any `getMore` operation on a cursor.

Enabling retryable reads makes applications more resilient to transient errors such as network failures, database upgrades, and replica set failovers. For an exact definition of which errors trigger a retry, see the [retryable reads specification](#).

- *compressors*: Comma separated list of compressors for wire protocol compression. The list is used to negotiate a compressor with the server. Currently supported options are “snappy”, “zlib” and “zstd”. Support for snappy requires the `python-snappy` package. zlib support requires the Python standard library `zlib` module. zstd requires the `zstandard` package. By default no compression is used. Compression support must also be enabled on the server. MongoDB 3.6+ supports snappy and zlib compression. MongoDB 4.2+ adds support for zstd. See `network-compression-example` for details.

- *zlibCompressionLevel*: (int) The zlib compression level to use when zlib is used as the wire protocol compressor. Supported values are -1 through 9. -1 tells the zlib library to use its default compression level (usually 6). 0 means no compression. 1 is best speed. 9 is best compression. Defaults to -1.
- *uuidRepresentation*: The BSON representation to use when encoding from and decoding to instances of UUID. Valid values are the strings: “standard”, “pythonLegacy”, “javaLegacy”, “csharpLegacy”, and “unspecified” (the default). New applications should consider setting this to “standard” for cross language compatibility. See handling-uuid-data-example for details.
- *unicode_decode_error_handler*: The error handler to apply when a Unicode-related error occurs during BSON decoding that would otherwise raise `UnicodeDecodeError`. Valid options include ‘strict’, ‘replace’, ‘backslashreplace’, ‘surrogateescape’, and ‘ignore’. Defaults to ‘strict’.
- *srvServiceName*: (string) The SRV service name to use for “`mongodb+srv://`” URIs. Defaults to “`mongodb`”. Use it like so:

```
MongoClient("mongodb+srv://example.com/?srvServiceName=customname")
```

- *srvMaxHosts*: (int) limits the number of mongos-like hosts a client will connect to. More specifically, when a “`mongodb+srv://`” connection string resolves to more than *srvMaxHosts* number of hosts, the client will randomly choose an *srvMaxHosts* sized subset of hosts.

Write Concern options:

(Only set if passed. No default values.)

- *w*: (integer or string) If this is a replica set, write operations will block until they have been replicated to the specified number or tagged set of servers. *w=<int>* always includes the replica set primary (e.g. *w=3* means write to the primary and wait until replicated to **two** secondaries). Passing *w=0* **disables write acknowledgement** and all other write concern options.
- *wTimeoutMS*: (integer) Used in conjunction with *w*. Specify a value in milliseconds to control how long to wait for write propagation to complete. If replication does not complete in the given timeframe, a timeout exception is raised. Passing *wTimeoutMS=0* will cause **write operations to wait indefinitely**.
- *journal*: If True block until write operations have been committed to the journal. Cannot be used in combination with *fsync*. Write operations will fail with an exception if this option is used when the server is running without journaling.
- *fsync*: If True and the server is running without journaling, blocks until the server has synced all data files to disk. If the server is running with journaling, this acts the same as the *j* option, blocking until write operations have been committed to the journal. Cannot be used in combination with *j*.

Replica set keyword arguments for connecting with a replica set - either directly or via a mongos:

- *replicaSet*: (string or None) The name of the replica set to connect to. The driver will verify that all servers it connects to match this name. Implies that the hosts specified are a seed list and the driver should attempt to find all members of the set. Defaults to None.

Read Preference:

- *readPreference*: The replica set read preference for this client. One of `primary`, `primaryPreferred`, `secondary`, `secondaryPreferred`, or `nearest`. Defaults to `primary`.
- *readPreferenceTags*: Specifies a tag set as a comma-separated list of colon-separated key-value pairs. For example `dc:ny,rack:1`. Defaults to `None`.
- *maxStalenessSeconds*: (integer) The maximum estimated length of time a replica set secondary can fall behind the primary in replication before it will no longer be selected for operations. Defaults to `-1`, meaning no maximum. If `maxStalenessSeconds` is set, it must be a positive integer greater than or equal to 90 seconds.

See also:

[/examples/server_selection](#)

Authentication:

- *username*: A string.
- *password*: A string.

Although `username` and `password` must be percent-escaped in a MongoDB URI, they must not be percent-escaped when passed as parameters. In this example, both the space and slash special characters are passed as-is:

```
MongoClient(username="user name", password="pass/word")
```

- *authSource*: The database to authenticate on. Defaults to the database specified in the URI, if provided, or to “`admin`”.
- *authMechanism*: See [MECHANISMS](#) for options. If no mechanism is specified, PyMongo automatically SCRAM-SHA-1 when connected to MongoDB 3.6 and negotiates the mechanism to use (SCRAM-SHA-1 or SCRAM-SHA-256) when connected to MongoDB 4.0+.
- *authMechanismProperties*: Used to specify authentication mechanism specific options. To specify the service name for GSSAPI authentication pass `authMechanismProperties='SERVICE_NAME:<service name>'`. To specify the session token for MONGODB-AWS authentication pass `authMechanismProperties='AWS_SESSION_TOKEN:<session token>'`.

See also:

[/examples/authentication](#)

TLS/SSL configuration:

- *tls*: (boolean) If `True`, create the connection to the server using transport layer security. Defaults to `False`.
- *tlsInsecure*: (boolean) Specify whether TLS constraints should be relaxed as much as possible. Setting `tlsInsecure=True` implies `tlsAllowInvalidCertificates=True` and

`tlsAllowInvalidHostnames=True`. Defaults to `False`. Think very carefully before setting this to `True` as it dramatically reduces the security of TLS.

- `tlsAllowInvalidCertificates`: (boolean) If `True`, continues the TLS handshake regardless of the outcome of the certificate verification process. If this is `False`, and a value is not provided for `tlsCAFile`, PyMongo will attempt to load system provided CA certificates. If the python version in use does not support loading system CA certificates then the `tlsCAFile` parameter must point to a file of CA certificates. `tlsAllowInvalidCertificates=False` implies `tls=True`. Defaults to `False`. Think very carefully before setting this to `True` as that could make your application vulnerable to on-path attackers.
- `tlsAllowInvalidHostnames`: (boolean) If `True`, disables TLS hostname verification. `tlsAllowInvalidHostnames=False` implies `tls=True`. Defaults to `False`. Think very carefully before setting this to `True` as that could make your application vulnerable to on-path attackers.
- `tlsCAFile`: A file containing a single or a bundle of “certification authority” certificates, which are used to validate certificates passed from the other end of the connection. Implies `tls=True`. Defaults to `None`.
- `tlsCertificateKeyFile`: A file containing the client certificate and private key. Implies `tls=True`. Defaults to `None`.
- `tlsCRLFile`: A file containing a PEM or DER formatted certificate revocation list. Implies `tls=True`. Defaults to `None`.
- `tlsCertificateKeyFilePassword`: The password or passphrase for decrypting the private key in `tlsCertificateKeyFile`. Only necessary if the private key is encrypted. Defaults to `None`.
- `tlsDisableOCSPEndpointCheck`: (boolean) If `True`, disables certificate revocation status checking via the OCSP responder specified on the server certificate. `tlsDisableOCSPEndpointCheck=False` implies `tls=True`. Defaults to `False`.
- `ssl`: (boolean) Alias for `tls`.

Read Concern options:

(If not set explicitly, this will use the server default)

- `readConcernLevel`: (string) The read concern level specifies the level of isolation for read operations. For example, a read operation using a read concern level of `majority` will only return data that has been written to a majority of nodes. If the level is left unspecified, the server default will be used.

Client side encryption options:

(If not set explicitly, client side encryption will not be enabled.)

- `auto_encryption_opts`: A `AutoEncryptionOpts` which configures this client to automatically encrypt collection commands and automatically decrypt results. See automatic-client-side-encryption for an example. If a `MongoClient` is configured with `auto_encryption_opts` and a non-`None` `maxPoolSize`, a separate internal `MongoClient` is created if any of the following are true:
 - A `key_vault_client` is not passed to `AutoEncryptionOpts`
 - `bypass_auto_encryption=False` is passed to `AutoEncryptionOpts`

Stable API options:

(If not set explicitly, Stable API will not be enabled.)

- `server_api`: A `ServerApi` which configures this client to use Stable API. See `versioned-api-ref` for details.

See also:

The MongoDB documentation on [connections](#).

Changed in version 4.5: Added the `serverMonitoringMode` keyword argument.

Changed in version 4.2: Added the `timeoutMS` keyword argument.

Changed in version 4.0:

- Removed the `fsync`, `unlock`, `is_locked`, `database_names`, and `close_cursor` methods. See the `pymongo4-migration-guide`.
- Removed the `waitQueueMultiple` and `socketKeepAlive` keyword arguments.
- The default for `uuidRepresentation` was changed from `pythonLegacy` to `unspecified`.
- Added the `srvServiceName`, `maxConnecting`, and `srvMaxHosts` URI and keyword arguments.

Changed in version 3.12: Added the `server_api` keyword argument. The following keyword arguments were deprecated:

- `ssl_certfile` and `ssl_keyfile` were deprecated in favor of `tlsCertificateKeyFile`.

Changed in version 3.11: Added the following keyword arguments and URI options:

- `tlsDisableOCSPEndpointCheck`
- `directConnection`

Changed in version 3.9: Added the `retryReads` keyword argument and URI option. Added the `tlsInsecure` keyword argument and URI option. The following keyword arguments and URI options were deprecated:

- `wTimeout` was deprecated in favor of `wTimeoutMS`.
- `j` was deprecated in favor of `journal`.
- `ssl_cert_reqs` was deprecated in favor of `tlsAllowInvalidCertificates`.
- `ssl_match_hostname` was deprecated in favor of `tlsAllowInvalidHostnames`.
- `ssl_ca_certs` was deprecated in favor of `tlsCAFile`.
- `ssl_certfile` was deprecated in favor of `tlsCertificateKeyFile`.
- `ssl_crlfile` was deprecated in favor of `tlsCRLFile`.
- `ssl_pem_passphrase` was deprecated in favor of `tlsCertificateKeyFilePassword`.

Changed in version 3.9: `retryWrites` now defaults to `True`.

Changed in version 3.8: Added the `server_selector` keyword argument. Added the `type_registry` keyword argument.

Changed in version 3.7: Added the `driver` keyword argument.

Changed in version 3.6: Added support for `mongodb+srv://` URIs. Added the `retryWrites` keyword argument and URI option.

Changed in version 3.5: Add `username` and `password` options. Document the `authSource`, `authMechanism`, and `authMechanismProperties` options. Deprecated the `socketKeepAlive` keyword argument and `URI` option. `socketKeepAlive` now defaults to `True`.

Changed in version 3.0: `MongoClient` is now the one and only client class for a standalone server, mongos, or replica set. It includes the functionality that had been split into `MongoReplicaSetClient`: it can connect to a replica set, discover all its members, and monitor the set for stepdowns, elections, and reconfigs.

The `MongoClient` constructor no longer blocks while connecting to the server or servers, and it no longer raises `ConnectionFailure` if they are unavailable, nor `ConfigurationError` if the user's credentials are wrong. Instead, the constructor returns immediately and launches the connection process on background threads.

Therefore the `alive` method is removed since it no longer provides meaningful information; even if the client is disconnected, it may discover a server in time to fulfill the next operation.

In PyMongo 2.x, `MongoClient` accepted a list of standalone MongoDB servers and used the first it could connect to:

```
MongoClient(['host1.com:27017', 'host2.com:27017'])
```

A list of multiple standalones is no longer supported; if multiple servers are listed they must be members of the same replica set, or mongoses in the same sharded cluster.

The behavior for a list of mongoses is changed from “high availability” to “load balancing”. Before, the client connected to the lowest-latency mongos in the list, and used it until a network error prompted it to re-evaluate all mongoses’ latencies and reconnect to one of them. In PyMongo 3, the client monitors its network latency to all the mongoses continuously, and distributes operations evenly among those with the lowest latency. See `mongos-load-balancing` for more information.

The `connect` option is added.

The `start_request`, `in_request`, and `end_request` methods are removed, as well as the `auto_start_request` option.

The `copy_database` method is removed, see the `copy_database` examples for alternatives.

The `MongoClient.disconnect()` method is removed; it was a synonym for `close()`.

`MongoClient` no longer returns an instance of `Database` for attribute names with leading underscores. You must use dict-style lookups instead:

```
client['__my_database__']
```

Not:

```
client.__my_database__
```

```
class biothings.utils.mongo.DummyCollection
```

Bases: `dotdict`

`count()`

`drop()`

```
class biothings.utils.mongo.DummyDatabase
```

Bases: `dotdict`

`collection_names()`

```
class biothings.utils.mongo.HandleAutoReconnectMixin(*args, **kwargs)
```

Bases: object

This mixin will decor any non-hidden method with handle_autoreconnect decorator

```
exception biothings.utils.mongo.MaxRetryAutoReconnectException(message: str = "", errors:
```

Mapping[str, Any] | Sequence[Any] | None = None

Bases: AutoReconnect

Raised when we reach maximum retry to connect to Mongo server

```
biothings.utils.mongo.check_document_size(doc)
```

Return True if doc isn't too large for mongo DB

```
biothings.utils.mongo.doc_feeder(collection, step=1000, s=None, e=None, inbatch=False, query=None,
                                 batch_callback=None, fields=None, logger=<module 'logging' from
                                 '/home/docs/asdf/install/python3.10.13/lib/python3.10/logging/__init__.py'>,
                                 session_refresh_interval=5)
```

An iterator returning docs in a collection, with batch query.

Additional filter query can be passed via *query*, e.g., *doc_feeder(collection, query={'taxid': {'\$in': [9606, 10090, 10116]}})* *batch_callback* is a callback function as *fn(index, t)*, called after every batch. *fields* is an optional parameter to restrict the fields to return.

session_refresh_interval is 5 minutes by default. We call refreshSessions command every 5 minutes to keep a session alive, otherwise the session

and all cursors attached (explicitly or implicitly) to the session will time out after idling for 30 minutes, even if we have *no_cursor_timeout* set True for a cursor. See <https://www.mongodb.com/docs/manual/reference/command/refreshSessions/> and <https://www.mongodb.com/docs/manual/reference/method/cursor.noCursorTimeout/#session-idle-timeout-overrides-nocursortimeout>

```
biothings.utils.mongo.get_api(conn=None)
```

```
biothings.utils.mongo.get_cache_filename(col_name)
```

```
biothings.utils.mongo.get_cmd(conn=None)
```

```
biothings.utils.mongo.get_conn(server, port)
```

```
biothings.utils.mongo.get_data_plugin(conn=None)
```

```
biothings.utils.mongo.get_event(conn=None)
```

```
biothings.utils.mongo.get_hub_config(conn=None)
```

```
biothings.utils.mongo.get_hub_db_conn()
```

```
biothings.utils.mongo.get_last_command(conn=None)
```

```
biothings.utils.mongo.get_previous_collection(new_id)
```

Given 'new_id', an _id from src_build, as the "new" collection, automatically select an "old" collection. By default, src_build's documents will be sorted according to their name (_id) and old collection is the one just before new_id.

Note: because there can be more than one build config used, the actual build config name is first determined using new_id collection name,

then the find().sort() is done on collections containing that build config name.

```
biothings.utils.mongo.get_source_fullname(col_name)
Assuming col_name is a collection created from an upload process, find the main source & sub_source associated.
biothings.utils.mongo.get_source_fullnames(col_names)

biothings.utils.mongo.get_src_build(conn=None)

biothings.utils.mongo.get_src_build_config(conn=None)

biothings.utils.mongo.get_src_conn()

biothings.utils.mongo.get_src_db(conn=None)

biothings.utils.mongo.get_src_dump(conn=None)

biothings.utils.mongo.get_src_master(conn=None)

biothings.utils.mongo.get_target_conn()

biothings.utils.mongo.get_target_db(conn=None)

biothings.utils.mongo.get_target_master(conn=None)

biothings.utils.mongo.handle_autoreconnect(cls_instance, func)
```

After upgrading the pymongo package from 3.12 to 4.x, the “AutoReconnect: connection pool paused” problem appears quite often. It is not clear that the problem happens with our codebase, maybe a pymongo’s problem.

This function is an attempt to handle the AutoReconnect exception, without modifying our codebase. When the exception is raised, we just wait for some time, then retry. If the error still happens after MAX_RETRY, it must be a connection-related problem. We should stop retrying and raise error.

Ref: <https://github.com/newgene/biothings.api/pull/40#issuecomment-1185334545>

```
biothings.utils.mongo.id_feeder(col, batch_size=1000, build_cache=True, logger=<module 'logging' from '/home/docs/asdf/install/python/3.10.13/lib/python3.10/logging/__init__.py'>, force_use=False, force_build=False, validate_only=False)
```

Return an iterator for all _ids in collection “col”.

Search for a valid cache file if available, if not, return a doc_feeder for that collection. Valid cache is a cache file that is newer than the collection.

“db” can be “target” or “src”. “build_cache” True will build a cache file as _ids are fetched, if no cache file was found. “force_use” True will use any existing cache file and won’t check whether it’s valid or not. “force_build” True will build a new cache even if current one exists and is valid. “validate_only” will directly return [] if the cache is valid (convenient way to check if the cache is valid).

```
biothings.utils.mongo.invalidate_cache(col_name, col_type='src')
```

```
biothings.utils.mongo.requires_config(func)
```

6.9.23 biothings.utils.parallel

Utils for running parallel jobs.

```
biothings.utils.parallel.collection_partition(src_collection_list, step=100000)
```

This function is deprecated, not used anywhere

```
biothings.utils.parallel.run_jobs_on_ipythoncluster(worker, task_list,
                                                    shutdown_ipengines_after_done=False)
```

```
biothings.utils.parallel.run_jobs_on_parallel(worker, task_list, executor_args=None)
```

This method will run multiple workers to handle the task_list, in a process pool, which is an easy way to run and manage processes.

Parameters: - worker: a callable, which will be apply for an item of the task_list - task_list: a iterable, which contains task data should be processed. - executor_args: should be valid parameters for initializing a ProcessPoolExecutor.

6.9.24 biothings.utils.parallel_mp

```
class biothings.utils.parallel_mp.ErrorHandler(errpath, chunk_num)
```

Bases: object

```
handle(exception)
```

```
class biothings.utils.parallel_mp.ParallelResult(agg_function, agg_function_init)
```

Bases: object

```
aggregate(curr)
```

```
biothings.utils.parallel_mp.agg_by_append(prev, curr)
```

```
biothings.utils.parallel_mp.agg_by_sum(prev, curr)
```

```
biothings.utils.parallel_mp.run_parallel_on_ids_dir(fun, ids_dir, backend_options=None,
                                                    agg_function=<function agg_by_append>,
                                                    agg_function_init=[], outpath=None,
                                                    num_workers=2, mget_chunk_size=10000,
                                                    ignore_None=True, error_path=None,
                                                    **query_kwargs)
```

This function will run function fun on chunks defined by the files in ids_dir.

All parameters are fed to run_parallel_on_iterable, except:

Params ids_dir

Directory containing only files with ids, one per line. The number of files defines the number of chunks.

```
biothings.utils.parallel_mp.run_parallel_on_ids_file(fun, ids_file, backend_options=None,
                                                    agg_function=<function agg_by_append>,
                                                    agg_function_init=[], chunk_size=1000000,
                                                    num_workers=2, outpath=None,
                                                    mget_chunk_size=10000, ignore_None=True,
                                                    error_path=None, **query_kwargs)
```

Implementation of run_parallel_on_iterable, where iterable comes from the lines of a file.

All parameters are fed to run_on_ids_iterable, except:

Parameters

ids_file – Path to file with ids, one per line.

```
biothings.utils.parallel_mp.run_parallel_on_iterable(fun, iterable, backend_options=None,
                                                    agg_function=<function agg_by_append>,
                                                    agg_function_init=None,
                                                    chunk_size=1000000, num_workers=2,
                                                    outpath=None, mget_chunk_size=10000,
                                                    ignore_None=True, error_path=None,
                                                    **query_kwargs)
```

This function will run a user function on all documents in a backend database in parallel using multiprocessing.Pool. The overview of the process looks like this:

Chunk (into chunks of size “chunk_size”) items in iterable, and run the following script on each chunk using a multiprocessing.Pool object with “num_workers” processes:

For each document in list of ids in this chunk (documents retrieved in chunks of “mget_chunk_size”):

Run function “fun” with parameters (doc, chunk_num, f <file handle only passed if “outpath” is not None>), and aggregate the result with the current results using function “agg_function”.

Parameters

- **fun** – The function to run on all documents. If outpath is NOT specified, fun must accept two parameters: (doc, chunk_num), where doc is the backend document, and chunk_num is essentially a unique process id. If outpath IS specified, an additional open file handle (correctly tagged with the current chunk’s chunk_num) will also be passed to fun, and thus it must accept three parameters: (doc, chunk_num, f)
- **iterable** – Iterable of ids.
- **backend_options** – An instance of biothings.utils.backend.DocBackendOptions. This contains the options necessary to instantiate the correct backend class (ES, mongo, etc).
- **agg_function** – This function aggregates the return value of each run of function fun. It should take 2 parameters: (prev, curr), where prev is the previous aggregated result, and curr is the output of the current function run. It should return some value that represents the aggregation of the previous aggregated results with the output of the current function.
- **agg_function_init** – Initialization value for the aggregated result.
- **chunk_size** – Length of the ids list sent to each chunk.
- **num_workers** – Number of processes that consume chunks in parallel. <https://docs.python.org/2/library/multiprocessing.html#multiprocessing.pool.multiprocessing.Pool>
- **outpath** – Base path for output files. Because function fun can be run many times in parallel, each chunk is sequentially numbered, and the output file name for any chunk is outpath_{chunk_num}, e.g., if outpath is out, all output files will be of the form: /path/to/cwd/out_1, /path/to/cwd/out_2, etc.
- **error_path** – Base path for error files. If included, exceptions inside each chunk thread will be printed to these files.
- **mget_chunk_size** – The size of each mget chunk inside each chunk thread. In each thread, the ids list is consumed by passing chunks to a mget_by_ids function. This parameter controls the size of each mget.
- **ignore_None** – If set, then falsy values will not be aggregated (0, [], None, etc) in the aggregation step. Default True.

All other parameters are fed to the backend query.

```
biothings.utils.parallel_mp.run_parallel_on_query(fun, backend_options=None, query=None,
                                                agg_function=<function agg_by_append>,
                                                agg_function_init=[], chunk_size=1000000,
                                                num_workers=2, outpath=None,
                                                mget_chunk_size=10000, ignore_None=True,
                                                error_path=None, full_doc=False,
                                                **query_kwargs)
```

Implementation of run_parallel_on_ids_iterable, where the ids iterable comes from the result of a query on the specified backend.

All parameters are fed to run_parallel_on_ids_iterable, except:

Parameters

- **query** – ids come from results of this query run on backend, default: “match_all”
- **full_doc** – If True, a list of documents is passed to each subprocess, rather than ids that are looked up later. Should be faster? Unknown how this works with very large query sets...

6.9.25 biothings.utils.parsers

```
biothings.utils.parsers.docker_source_info_parser(url)
```

Parameters

url – file url include docker connection string format:
 docker://CONNECTION_NAME?image=DOCKER_IMAGE&tag=TAG&dump_command=”python
 run.py”&path=/path/to/file the CONNECTION_NAME must be defined in the biothings Hub
 config. example: docker://CONNECTION_NAME?image=docker_image&tag=docker_tag&dump_command=”python
 run.py”&path=/path/to/file docker://CONNECTION_NAME?image=docker_image&tag=docker_tag&dump_command
 run.py”&path=/path/to/file docker://CONNECTION_NAME?image=docker_image&tag=docker_tag&dump_command
 run.py”&path=/path/to/file docker”//CONNECTION_NAME?image=docker_image&tag=docker_tag&dump_command
 run.py”&path=/path/to/file

Returns

```
biothings.utils.parsers.json_array_parser(patterns: Iterable[str] | None = None) → Callable[[str],
                                         Generator[dict, None, None]]
```

Create JSON Array Parser given filename patterns

For use with manifest.json based plugins. The data comes in a JSON that is an JSON array, containing multiple documents.

Parameters

patterns – glob-compatible patterns for filenames, like `.json`, `data.json`

Returns

`parser_func`

```
biothings.utils.parsers.ndjson_parser(patterns: Iterable[str] | None = None) → Callable[[str],
                                         Generator[dict, None, None]]
```

Create NDJSON Parser given filename patterns

For use with manifest.json based plugins. Caveat: Only handles valid NDJSON (no extra newlines, UTF8, etc.)

Parameters

patterns – glob-compatible patterns for filenames, like `.ndjson`, `data.ndjson`

Returns

Generator that takes in a `data_folder` and returns documents from NDJSON files that matches the filename patterns

Return type

`parser_func`

6.9.26 `biothings.utils.redis`

```
class biothings.utils.redis.RedisClient(connection_params)
    Bases: object
    check()
    client = None
    classmethod get_client(params)
    get_db(db_name=None)
        Return a redict client instance from a database name or database number (if db_name is an integer)
    initialize(deep=False)
        Careful: this may delete data. Prepare Redis instance to work with biothings hub: - database 0: this db is used to store a mapping between
            database index and database name (so a database can be accessed by name). This method will flush this db and prepare it.
        • any other databases will be flushed if deep is True, making the redis server fully dedicated to
    property mapdb
    pick_db()
        Return a database number, preferably not used (db doesn't exist). If no database available (all are used), will be one and flush it...
exception biothings.utils.redis.RedisClientError
    Bases: Exception
```

6.9.27 `biothings.utils.serializer`

```
class biothings.utils.serializer.URL(seq)
    Bases: UserString
    remove(param='format')

biothings.utils.serializer.json_dumps(data, indent=False, sort_keys=False)
biothings.utils.serializer.json_loads(json_str: bytes | str) → Any
    Load a JSON string or bytes using orjson
biothings.utils.serializer.load_json(json_str: bytes | str) → Any
    Load a JSON string or bytes using orjson
```

```

biothings.utils.serializer.orjson_default(o)
    The default function passed to orjson to serialize non-serializable objects
biothings.utils.serializer.to_json(data, indent=False, sort_keys=False)

biothings.utils.serializer.to_json_0(data)
    deprecated

biothings.utils.serializer.to_json_file(data, fobj, indent=False, sort_keys=False)

biothings.utils.serializer.to_msgpack(data)

biothings.utils.serializer.to_yaml(data, stream=None, Dumper=<class 'yaml.dumper.SafeDumper'>,
                                    default_flow_style=False)

```

6.9.28 biothings.utils.shelve

6.9.29 biothings.utils.sqlite3

```
class biothings.utils.sqlite3.Collection(colname, db)
```

Bases: object

```
bulk_write(docs, *args, **kwargs)
```

```
count()
```

```
property database
```

```
drop()
```

```
find(*args, **kwargs)
```

```
find_one(*args, **kwargs)
```

```
findv2(*args, **kwargs)
```

This is a new version of find() that uses json feature of sqlite3, will replace find in the future

```
get_conn()
```

```
insert(docs, *args, **kwargs)
```

```
insert_one(doc)
```

```
property name
```

```
remove(query)
```

```
rename(new_name, dropTarget=False)
```

```
replace_one(query, doc, upsert=False)
```

```
save(doc)
```

```
update(query, what, upsert=False)
```

```
update_one(query, what, upsert=False)
```

```
class biothings.utils.sqlite3.Cursor(inserted_count)
    Bases: object

class biothings.utils.sqlite3.Database(db_folder, name=None)
    Bases: IDatabase

    CONFIG = <ConfigurationWrapper over <module 'config' from
        '/home/docs/checkouts/readthedocs.org/user_builds/biothingsapi/checkouts/latest/
        biothings/hub/default_config.py'>>

    property address
        Returns sufficient information so a connection to a database can be created. Information can be a dictionary,
        object, etc... and depends on the actual backend

    collection_names()
        Return a list of all collections (or tables) found in this database

    create_collection(colname)
        Create a table/colleciton named colname. If backend is using a schema-based database (ie. SQL), backend
        should enforce the schema with at least field “_id” as the primary key (as a string).

    create_if_needed(table)

    get_conn()

class biothings.utils.sqlite3.DatabaseClient
    Bases: IDatabase

    biothings.utils.sqlite3.get_api()
    biothings.utils.sqlite3.get_cmd()
    biothings.utils.sqlite3.get_data_plugin()
    biothings.utils.sqlite3.get_event()
    biothings.utils.sqlite3.get_hub_config()
    biothings.utils.sqlite3.get_hub_db_conn()
    biothings.utils.sqlite3.get_last_command()
    biothings.utils.sqlite3.get_source_fullname(col_name)
        Assuming col_name is a collection created from an upload process, find the main source & sub_source associated.

    biothings.utils.sqlite3.get_src_build()
    biothings.utils.sqlite3.get_src_build_config()
    biothings.utils.sqlite3.get_src_conn()
    biothings.utils.sqlite3.get_src_db()
    biothings.utils.sqlite3.get_src_dump()
    biothings.utils.sqlite3.get_src_master()
    biothings.utils.sqlite3.requires_config(func)
```

6.9.30 biothings.utils.version

Functions to return versions of things.

`biothings.utils.version.check_new_version(folder, max_commits=10)`

Given a folder pointing to a Git repo, return a dict containing info about remote commits not applied yet to the repo, or empty dict if nothing new.

`biothings.utils.version.get_biothings_commit()`

Gets the biothings commit information.

`biothings.utils.version.get_python_exec_version()`

return Python version

`biothings.utils.version.get_python_version()`

Get a list of python packages installed and their versions.

`biothings.utils.version.get_repository_information(app_dir=None)`

Get the repository information for the local repository, if it exists.

`biothings.utils.version.get_software_info(app_dir=None)`

return current application info

`biothings.utils.version.get_source_code_info(src_file)`

Given a path to a source code, try to find information about repository, revision, URL pointing to that file, etc...
Return None if nothing can be determined. Tricky cases:

- `src_file` could refer to another repo, within current repo (namely a remote data plugin, cloned within the api's plugins folder)
- `src_file` could point to a folder, when for instance a datapluging is analized. This is because we can't point to an uploader file since it's dynamically generated

`biothings.utils.version.get_version(folder)`

return revision of a git folder

`biothings.utils.version.set_versions(config, app_folder)`

Propagate versions (git branch name) in config module. Also set app and biothings folder paths (though not exposed as a config param since they are lower-cased, see `biothings.__init__.py`, regex `PARAM_PAT`)

6.10 biothings.hub

```
class biothings.hub.HubCommands
```

Bases: `OrderedDict`

```
class biothings.hub.HubSSHSERVER
```

Bases: `SSHServer`

`PASSWORDS = []`

`SHELL = None`

begin_auth(*username*)

Authentication has been requested by the client

This method will be called when authentication is attempted for the specified user. Applications should use this method to prepare whatever state they need to complete the authentication, such as loading in the set of authorized keys for that user. If no authentication is required for this user, this method should return *False* to cause the authentication to immediately succeed. Otherwise, it should return *True* to indicate that authentication should proceed.

If blocking operations need to be performed to prepare the state needed to complete the authentication, this method may be defined as a coroutine.

Parameters

username (*str*) – The name of the user being authenticated

Returns

A *bool* indicating whether authentication is required

connection_lost(*exc*)

Called when a connection is lost or closed

This method is called when a connection is closed. If the connection is shut down cleanly, *exc* will be *None*. Otherwise, it will be an exception explaining the reason for the disconnect.

connection_made(*connection*)

Called when a connection is made

This method is called when a new TCP connection is accepted. The *conn* parameter should be stored if needed for later use.

Parameters

conn (*SSHSERVERConnection*) – The connection which was successfully opened

password_auth_supported()

Return whether or not password authentication is supported

This method should return *True* if password authentication is supported. Applications wishing to support it must have this method return *True* and implement [validate_password\(\)](#) to return whether or not the password provided by the client is valid for the user being authenticated.

By default, this method returns *False* indicating that password authentication is not supported.

Returns

A *bool* indicating if password authentication is supported or not

session_requested()

Handle an incoming session request

This method is called when a session open request is received from the client, indicating it wishes to open a channel to be used for running a shell, executing a command, or connecting to a subsystem. If the application wishes to accept the session, it must override this method to return either an *SSHSERVERSession* object to use to process the data received on the channel or a tuple consisting of an *SSHSERVERChannel* object created with [create_server_channel](#) and an *SSHSERVERSession*, if the application wishes to pass non-default arguments when creating the channel.

If blocking operations need to be performed before the session can be created, a coroutine which returns an *SSHSERVERSession* object can be returned instead of the session itself. This can be either returned directly or as a part of a tuple with an *SSHSERVERChannel* object.

To reject this request, this method should return *False* to send back a “Session refused” response or raise a [ChannelOpenError](#) exception with the reason for the failure.

The details of what type of session the client wants to start will be delivered to methods on the `SSHServerSession` object which is returned, along with other information such as environment variables, terminal type, size, and modes.

By default, all session requests are rejected.

Returns

One of the following:

- An `SSHServerSession` object or a coroutine which returns an `SSHServerSession`
- A tuple consisting of an `SSHServerChannel` and the above
- A *callable* or coroutine handler function which takes `AsyncSSH` stream objects for `stdin`, `stdout`, and `stderr` as arguments
- A tuple consisting of an `SSHServerChannel` and the above
- `False` to refuse the request

Raises

`ChannelOpenError` if the session shouldn't be accepted

`validate_password(username, password)`

Return whether password is valid for this user

This method should return `True` if the specified password is a valid password for the user being authenticated. It must be overridden by applications wishing to support password authentication.

If the password provided is valid but expired, this method may raise `PasswordChangeRequired` to request that the client provide a new password before authentication is allowed to complete. In this case, the application must override `change_password()` to handle the password change request.

This method may be called multiple times with different passwords provided by the client. Applications may wish to limit the number of attempts which are allowed. This can be done by having `password_auth_supported()` begin returning `False` after the maximum number of attempts is exceeded.

If blocking operations need to be performed to determine the validity of the password, this method may be defined as a coroutine.

By default, this method returns `False` for all passwords.

Parameters

- `username` (`str`) – The user being authenticated
- `password` (`str`) – The password sent by the client

Returns

A `bool` indicating if the specified password is valid for the user being authenticated

Raises

`PasswordChangeRequired` if the password provided is expired and needs to be changed

`class biothings.hub.HubSSHServerSession(name, shell)`

Bases: `SSHServerSession`

`break_received(sec)`

The client has sent a break

This method is called when the client requests that the server perform a break operation on the terminal. If the break is performed, this method should return `True`. Otherwise, it should return `False`.

By default, this method returns `False` indicating that no break was performed.

Parameters

msec (*int*) – The duration of the break in milliseconds

Returns

A *bool* to indicate if the break operation was performed or not

connection_made(chan)

Called when a channel is opened successfully

This method is called when a channel is opened successfully. The channel parameter should be stored if needed for later use.

Parameters

chan (`SSHSERVERCHANNEL`) – The channel which was successfully opened.

data_received(data, datatype)

Called when data is received on the channel

This method is called when data is received on the channel. If an encoding was specified when the channel was created, the data will be delivered as a string after decoding with the requested encoding. Otherwise, the data will be delivered as bytes.

Parameters

- **data** (*str* or *bytes*) – The data received on the channel
- **datatype** – The extended data type of the data, from extended data types

eof_received()

Called when EOF is received on the channel

This method is called when an end-of-file indication is received on the channel, after which no more data will be received. If this method returns *True*, the channel remains half open and data may still be sent. Otherwise, the channel is automatically closed after this method returns. This is the default behavior for classes derived directly from `SSHSession`, but not when using the higher-level streams API. Because input is buffered in that case, streaming sessions enable half-open channels to allow applications to respond to input read after an end-of-file indication is received.

eval_lines(lines)

exec_requested(command)

The client has requested to execute a command

This method should be implemented by the application to perform whatever processing is required when a client makes a request to execute a command. It should return *True* to accept the request, or *False* to reject it.

If the application returns *True*, the `session_started()` method will be called once the channel is fully open. No output should be sent until this method is called.

By default this method returns *False* to reject all requests.

Parameters

command (*str*) – The command the client has requested to execute

Returns

A *bool* indicating if the exec request was allowed or not

session_started()

Called when the session is started

This method is called when a session has started up. For client and server sessions, this will be called once a shell, exec, or subsystem request has been successfully completed. For TCP and UNIX domain socket sessions, it will be called immediately after the connection is opened.

shell_requested()

The client has requested a shell

This method should be implemented by the application to perform whatever processing is required when a client makes a request to open an interactive shell. It should return *True* to accept the request, or *False* to reject it.

If the application returns *True*, the `session_started()` method will be called once the channel is fully open. No output should be sent until this method is called.

By default this method returns *False* to reject all requests.

Returns

A *bool* indicating if the shell request was allowed or not

soft_eof_received()

The client has sent a soft EOF

This method is called by the line editor when the client send a soft EOF (Ctrl-D on an empty input line).

By default, soft EOF will trigger an EOF to an outstanding read call but still allow additional input to be received from the client after that.

```
class biothings.hub.HubServer(source_list, features=None, name='BioThings Hub',
                               managers_custom_args=None, api_config=None, reloader_config=None,
                               dataupload_config=None, websocket_config=None, autohub_config=None)
```

Bases: `object`

Helper to setup and instantiate common managers usually used in a hub (eg. dumper manager, uploader manager, etc...) “source_list” is either:

- a list of string corresponding to paths to datasources modules
- a package containing sub-folders with datasources modules

Specific managers can be retrieved adjusting “features” parameter, where each feature corresponds to one or more managers. Parameter defaults to all possible available. Managers are configured/init in the same order as the list, so if a manager (eg. job_manager) is required by all others, it must be the first in the list. “managers_custom_args” is an optional dict used to pass specific arguments while init managers:

```
managers_custom_args={"upload": {"poll_schedule": "*/5 * * * *"}}
```

will set poll schedule to check upload every 5min (instead of default 10s) “reloader_config”, “dataupload_config”, “autohub_config” and “websocket_config” can be used to customize reloader, dataupload and websocket. If None, default config is used. If explicitly False, feature is deactivated.

```
DEFAULT_API_CONFIG = {}
```

```
DEFAULT_AUTOHUB_CONFIG = {'es_host': None, 'indexer_factory': None,
                          'validator_class': None, 'version_urls': []}
```

```
DEFAULT_DATAUPLOAD_CONFIG = {'upload_root': '.biothings_hub/archive/dataupload'}
```

```
DEFAULT_FEATURES = ['config', 'job', 'dump', 'upload', 'datapluging', 'source',
                    'build', 'auto_archive', 'diff', 'index', 'snapshot', 'auto_snapshot_cleaner',
                    'release', 'inspect', 'sync', 'api', 'terminal', 'reloader', 'dataupload', 'ws',
                    'readonly', 'upgrade', 'autohub', 'hooks']
```

```
DEFAULT_MANAGERS_ARGS = {'upload': {'poll_schedule': '* * * * */10'}}  
DEFAULT_RELOADER_CONFIG = {'folders': None, 'managers': ['source_manager', 'assistant_manager'], 'reload_func': None}  
  
DEFAULT_WEBSOCKET_CONFIG = {}  
  
add_api_endpoint(endpoint_name, command_name, method, **kwargs)  
    Add an API endpoint to expose command named “command_name” using HTTP method “method”.  
    **kwargs are used to specify more arguments for EndpointDefinition  
  
before_configure()  
    Hook triggered before configure(), used eg. to adjust features list  
  
before_start()  
  
clean_features(features)  
    Sanitize (ie. remove duplicates) features  
  
configure()  
  
configure_api_endpoints()  
  
configure_api_manager()  
  
configure_auto_archive_manager()  
  
configure_auto_snapshot_cleaner_manager()  
  
configure_autohub_feature()  
    See bt.hub.standalone.AutoHubFeature  
  
configure_build_manager()  
  
configure_commands()  
    Configure hub commands according to available managers  
  
configure_config_feature()  
  
configure_dataplugin_manager()  
  
configure_dataupload_feature()  
  
configure_diff_manager()  
  
configure_dump_manager()  
  
configure_extra_commands()  
    Same as configure_commands() but commands are not exposed publicly in the shell (they are shortcuts or commands for API endpoints, supporting commands, etc...)  
  
configure_hooks_feature()  
    Ingest user-defined commands into hub namespace, giving access to all pre-defined commands (commands, extra_commands). This method prepare the hooks but the ingestion is done later when all commands are defined  
  
configure_index_manager()  
  
configure_inspect_manager()
```

```

configure_ioloop()
configure_job_manager()
configure_managers()
configure_READONLY_API_endpoints()

Assuming read-write API endpoints have previously been defined (self.api_endpoints set) extract commands and their endpoint definitions only when method is GET. That is, for any given API definition honoring REST principle for HTTP verbs, generate endpoints only for which actions are read-only actions.

configure_READONLY_feature()
    Define then expose read-only Hub API endpoints so Hub can be accessed without any risk of modifying data

configure_release_manager()
configure_reloader_feature()
configure_remaining_features()
configure_snapshot_manager()
configure_source_manager()
configure_sync_manager()
configure_terminal_feature()
configure_upgrade_feature()
    Allows a Hub to check for new versions (new commits to apply on running branch) and apply them on current code base

configure_upload_manager()
configure_ws_feature()
export_command_documents(filepath)
get_websocket_urls()
ingest_hooks()
mixargs(feat, params=None)
process_hook_file(hook_file)
quick_index(datasource_name, doc_type, indexer_env, subsource=None, index_name=None, **kwargs)
    Intention for datasource developers to quickly create an index to test their datasources. Automatically create temporary build config, build collection Then call the index method with the temporary build collection's name

start()

class biothings.hub.JobRenderer
    Bases: object
    cron_and_strdelta_info(job)

```

```
render(job)
render_cron(c)
render_func(f)
render_lambda(l)
render_method(m)
render_partial(p)
render_strdelta(job)

biothings.hub.get_schedule(loop)
    try to render job in a human-readable way...
async biothings.hub.start_ssh_server(loop, name, passwords, keys=['bin/ssh_host_key'], shell=None,
                                         host='', port=8022)

biothings.hub.status(managers)
    Return a global hub status (number of sources, documents, etc...) according to available managers
```

6.10.1 Modules

`biothings.hub.api`

```
class biothings.hub.api.EndpointDefinition
    Bases: dict

biothings.hub.api.create_handlers(shell, command_defs)
biothings.hub.api.generate_api_routes(shell, commands)
biothings.hub.api.generate_endpoint_for_callable(name, command, method, force_bodyargs)
biothings.hub.api.generate_endpoint_for_composite_command(name, command, method)
biothings.hub.api.generate_endpoint_for_display(name, command, method)
biothings.hub.api.generate_handler(shell, name, command_defs)
biothings.hub.api.start_api(app, port, check=True, wait=5, retry=5, settings=None)
```

`biothings.hub.api.managers`

```
class biothings.hub.api.manager.APIManager(log_folder=None, *args, **kwargs)
    Bases: BaseManager

    create_api(api_id, es_host, index, doc_type, port, description=None, **kwargs)
    delete_api(api_id)
    get/apis()
    register_status(api_id, status, **extra)
```

```
restore_running_apis()
    If some APIs were running but the hub stopped, re-start APIs as hub restarts

setup()
setup_log()
start_api(api_id)
stop_api(api_id)

exception biothings.hub.api.manager.APIManagerException
    Bases: Exception
```

biothings.hub.api.handlers.base

```
class biothings.hub.api.handlers.base.BaseHandler(application: Application, request: HTTPServerRequest, **kwargs: Any)
    Bases: DefaultHandler

initialize(managers, **kwargs)

class biothings.hub.api.handlers.base.DefaultHandler(application: Application, request: HTTPServerRequest, **kwargs: Any)
    Bases: RequestHandler

options(*args, **kwargs)
set_default_headers()
    Override this to set HTTP headers at the beginning of the request.
    For example, this is the place to set a custom Server header. Note that setting such headers in the normal flow of request processing may not do what you want, since headers may be reset during error handling.

write(result)
    Writes the given chunk to the output buffer.
    To write the output to the network, use the flush() method below.
    If the given chunk is a dictionary, we write it as JSON and set the Content-Type of the response to be application/json. (if you want to send JSON as a different Content-Type, call set_header after calling write()).
    Note that lists are not converted to JSON because of a potential cross-site security vulnerability. All JSON output should be wrapped in a dictionary. More details at http://haacked.com/archive/2009/06/25/json-hijacking.aspx/ and https://github.com/facebook/tornado/issues/1009

write_error(status_code, **kwargs)
    Override to implement custom error pages.
    write_error may call write, render, set_header, etc to produce output as usual.
    If this error was caused by an uncaught exception (including HTTPError), an exc_info triple will be available as kwargs["exc_info"]. Note that this exception may not be the “current” exception for purposes of methods like sys.exc_info() or traceback.format_exc.
```

```
class biothings.hub.api.handlers.base.GenericHandler(application: Application, request:  
                                                    HTTPServerRequest, **kwargs: Any)
```

Bases: *DefaultHandler*

```
delete(*args, **kwargs)
```

```
get(*args, **kwargs)
```

```
head(*args, **kwargs)
```

```
initialize(shell, **kwargs)
```

```
post(*args, **kwargs)
```

```
put(*args, **kwargs)
```

```
class biothings.hub.api.handlers.base.RootHandler(application: Application, request:  
                                                    HTTPServerRequest, **kwargs: Any)
```

Bases: *DefaultHandler*

```
async get()
```

```
initialize(features, hub_name=None, **kwargs)
```

biothings.hub.api.handlers.log

```
class biothings.hub.api.handlers.log.DefaultCORSHeaderMixin
```

Bases: *object*

```
set_default_headers()
```

```
class biothings.hub.api.handlers.log.HubLogDirHandler(application: Application, request:  
                                                    HTTPServerRequest, **kwargs: Any)
```

Bases: *DefaultCORSHeaderMixin*, *RequestHandler*

```
get(filename)
```

```
initialize(path)
```

```
class biothings.hub.api.handlers.log.HubLogFileHandler(application: Application, request:  
                                                    HTTPServerRequest, **kwargs: Any)
```

Bases: *DefaultCORSHeaderMixin*, *StaticFileHandler*

```
async get(path: str, include_body: bool = True) → None
```

If request path is a gz file, we will uncompress it first, then return get with the uncompress file path

```
options(*args, **kwargs)
```

```
biothings.hub.api.handlers.log.get_log_content(file_path, **kwargs)
```

biothings.hub.api.handlers.shell

```
class biothings.hub.api.handlers.shell.ShellHandler(application: Application, request: HTTPServerRequest, **kwargs: Any)
    Bases: GenericHandler
        initialize(shell, shellog, **kwargs)
            put()
```

biothings.hub.api.handlers.upload

```
class biothings.hub.api.handlers.upload.UploadHandler(application: Application, request: HTTPServerRequest, **kwargs: Any)
    Bases: GenericHandler
        data_received(chunk)
            Implement this method to handle streamed request data.
            Requires the .stream_request_body decorator.
            May be a coroutine for flow control.
        initialize(upload_root, **kwargs)
        parse_head()
        post(src_name)
        prepare()
            Called at the beginning of a request before get/post/etc.
            Override this method to perform common initialization regardless of the request method.
            Asynchronous support: Use async def or decorate this method with .gen.coroutine to make it asynchronous. If this method returns an Awaitable execution will not proceed until the Awaitable is done.
            New in version 3.1: Asynchronous support.
```

biothings.hub.api.handlers.ws

```
class biothings.hub.api.handlers.ws.HubDBListener
    Bases: ChangeListener
    Get events from Hub DB and propagate them through the websocket instance
        read(event)
class biothings.hub.api.handlers.ws.LogListener(*args, **kwargs)
    Bases: ChangeListener
        read(event)
class biothings.hub.api.handlers.ws.ShellListener(*args, **kwargs)
    Bases: LogListener
```

```
class biothings.hub.api.handlers.ws.WebSocketConnection(session, listeners)
Bases: SockJSConnection

Listen to Hub DB through a listener object, and publish events to any client connected

SockJSConnection.__init__() takes only a session as argument, and there's no way to pass custom settings. In
order to use that class, we need to use partial to partially init the instance with 'listeners' and let the rest use the
'session'

parameter:
    pconn      =      partial(WebSocketConnection, listeners=listeners)      ws_router      =
    sockjs.tornado.SockJSRouter(pconn, "/path")

clients = {}

on_close()
    Default on_close handler.

on_message(message)
    Default on_message handler. Must be overridden in your application

on_open(info)
    Default on_open() handler.

    Override when you need to do some initialization or request validation. If you return False, connection will
    be rejected.

    You can also throw Tornado HTTPError to close connection.

request
    ConnectionInfo object which contains caller IP address, query string parameters and cookies asso-
    ciated with this request (if any).

publish(message)
```

biothings.hub.autoupdate

biothings.hub.autoupdate.dumper

```
class biothings.hub.autoupdate.dumper(*args, **kwargs)
Bases: HTTPDumper

This dumper is used to maintain a BioThings API up-to-date. BioThings data is available as either as an Elastic-
Search snapshot when full update, and a collection of diff files for incremental updates. It will either download
incremental updates and apply diff, or trigger an ElasticSearch restore if the latest version is a full update. This
dumper can also be configured with precedence rules: when a full and a incremental update is available, rules
can set so full is preferably used over incremental (size can also be considered when selecting the preferred way).
```

```
AUTO_UPLOAD = False
AWS_ACCESS_KEY_ID = None
AWS_SECRET_ACCESS_KEY = None
SRC_NAME = None
SRC_ROOT_FOLDER = None
```

```

TARGET_BACKEND = None
VERSION_URL = None

anonymous_download(remoteurl, localfile, headers=None)
auth_download(bucket_name, key, localfile, headers=None)

property base_url
check_compat(build_meta)
choose_best_version(versions)
    Out of all compatible versions, choose the best: 1. choose incremental vs. full according to preferences 2. version must be the highest (most up-to-date)
compare_remote_local(remote_version, local_version, orig_remote_version, orig_local_version)
create_todump_list(force=False, version='latest', url=None)
    Fill self.to_dump list with dict("remote":remote_path,"local":local_path) elements. This is the todo list for the dumper. It's a good place to check whether needs to be downloaded. If 'force' is True though, all files will be considered for download
download(remoteurl, localfile, headers=None)
    Download "remotefile" to local location defined by 'localfile' Return relevant information about remotefile (depends on the actual client)
find_update_path(version, backend_version=None)
    Explore available versions and find the path to update the hub up to "version", starting from given backend_version (typically current version found in ES index). If backend_version is None (typically no index yet), a complete path will be returned, from the last compatible "full" release up-to the latest "diff" update. Returned is a list of dict, where each dict is a build metadata element containing information about each update (see versions.json), the order of the list describes the order the updates should be performed.
async get_target_backend()
    Example: [{

        'host': 'es6.mygene.info:9200', 'index': 'mygene_allspecies_20200823_ufkwdv79', 'index_alias': 'mygene_allspecies', 'version': '20200906', 'count': 38729977
    }]
async info(version='latest')
    Display version information (release note, etc...) for given version {
        "info": ... "release_note": ...
    }
load_remote_json(url)
post_dump(*args, **kwargs)
    Placeholder to add a custom process once the whole resource has been dumped. Optional.
prepare_client()
    Depending on presence of credentials, inject authentication in client.get()

```

```
remote_is_better(remotefile, localfile)
    Determine if remote is better
    Override if necessary.

async reset_target_backend()

property target_backend

async versions()
    Display all available versions. Example: [{  
        'build_version': '20171003', 'url': 'https://biotools-releases.s3.amazonaws.com:443/mygene.info/20171003.json', 'release_date': '2017-10-06T11:58:39.749357', 'require_version': None,  
        'target_version': '20171003', 'type': 'full'  
}, ...]
```

biothings.hub.autoupdate.uploader

```
class biothings.hub.autoupdate.uploader.BiothingsUploader(*args, **kwargs)
    Bases: BaseSourceUploader
    db_conn_info is a database connection info tuple (host,port) to fetch/store information about the datasource's state.

    AUTO_PURGE_INDEX = False
    SYNCER_FUNC = None
    TARGET_BACKEND = None
    async apply_diff(build_meta, job_manager, **kwargs)
    clean_archived_collections()
    get_snapshot_repository_config(build_meta)
        Return (name,config) tuple from build_meta, where name is the repo name, and config is the repo config
    async load(*args, **kwargs)
        Main resource load process, reads data from doc_c using chunk sized as batch_size. steps defines the different processes used to laod the resource: - "data" : will store actual data into single collections - "post" : will perform post data load operations - "master" : will register the master document in src_master
    name = None
    async restore_snapshot(build_meta, job_manager, **kwargs)
    property syncer_func
    property target_backend
    async update_data(batch_size, job_manager, **kwargs)
        Look in data_folder and either restore a snapshot to ES or apply diff to current ES index
```

biothings.hub.databuild**biothings.hub.databuild.backend**

Backend for storing merged genedoc after building. Support MongoDB, ES, CouchDB

class biothings.hub.databuild.backend.LinkTargetDocMongoBackend(*args, **kwargs)

Bases: *TargetDocBackend*

This backend type act as a dummy target backend, the data is actually stored in source database. It means only one datasource can be linked to that target backend, as a consequence, when this backend is used in a merge, there's no actual data merge. This is useful when "merging/indexing" only one datasource, where the merge step is just a duplication of datasource data.

drop()

get_backend_url()

Return backend URL (see `create_backend()` for formats)

name = 'link'

property target_collection

class biothings.hub.databuild.backend.ShardedTargetDocMongoBackend(*args, **kwargs)

Bases: *TargetDocMongoBackend*

`target_collection` is a pymongo collection object.

prepare()

if needed, add extra preparation steps here.

class biothings.hub.databuild.backend.SourceDocBackendBase(build_config, build, master, dump, sources)

Bases: *DocBackendBase*

get_build_configuration(build_name)

get_src_master_docs()

get_src_metadata()

validate_sources(sources=None)

class biothings.hub.databuild.backend.SourceDocMongoBackend(build_config, build, master, dump, sources)

Bases: *SourceDocBackendBase*

get_build_configuration(build_name)

get_src_master_docs()

get_src_metadata()

Return source versions which have been previously accessed wit this backend object or all source versions if none were accessed. Accessing means going through `__getitem__` (the usual way) and allows to auto-keep track of sources of interest, thus returning versions only for those.

validate_sources(sources=None)

```
class biothings.hub.databuild.backend.TargetDocBackend(*args, **kwargs)
Bases: DocBackendBase

generate_target_name(build_config_name)

get_backend_url()
    Return backend URL (see create_backend() for formats)

post_merge()

set_target_name(target_name, build_name=None)
    Create/prepare a target backend, either strictly named “target_name” or named derived from “build_name”
    (for temporary backends)

property target_name

class biothings.hub.databuild.backend.TargetDocMongoBackend(*args, **kwargs)
Bases: TargetDocBackend, DocMongoBackend

target_collection is a pymongo collection object.

set_target_name(target_name=None, build_name=None)
    Create/prepare a target backend, either strictly named “target_name” or named derived from “build_name”
    (for temporary backends)

biothings.hub.databuild.backend.create_backend(db_col_names, name_only=False, follow_ref=False,
                                              **kwargs)
    Guess what's inside ‘db_col_names’ and return the corresponding backend. - It could be a string (will first check
    for an src_build doc to check
        a backend_url field, if nothing there, will lookup a mongo collection in target database)
    • or a tuple(“target|src”, “col_name”)
    • or a (“mongodb://user:pass@host”, “db”, “col_name”) URI.
    • or a (“es_host:port”, “index_name”, “doc_type”)

If name_only is true, just return the name uniquely identifying the collection or index URI connection.

biothings.hub.databuild.backend.generate_folder(root_folder, old_db_col_names, new_db_col_names)

biothings.hub.databuild.backend.merge_src_build_metadata(build_docs)
    Merge metadata from src_build documents. A list of docs should be passed, the order is important: the 1st
    element has the less precedence, the last the most. It means that, when needed, some values from documents on
    the “left” of the list may be overridden by one on the right. Ex: build_version field Ideally, build docs shouldn't
    have any sources in common to prevent any unexpected conflicts...
```

biothings.hub.databuild.buildconfig

A build config contains autobuild configs and other information.

TODO: not all features already supported in the code

For example: {

```
    “_id”: “mynews”, “name”: “mynews”, “doc_type”: “news”, “sources”: [“mynews”], “root”: [“mynews”],
    “builder_class”: “biothings.hub.databuild.builder.DataBuilder”, “autobuild”: { ... }, “autopublish”: { ... },
    }, “build_version”: “%Y%m%d%H%M”
```

```

}

Autobuild: - build - diff/snapshot
Autopublish: - release note - publish
Autorelease: - release

class biothings.hub.databuild.buildconfig.AutoBuildConfig(confdict)
    Bases: object

    Parse automation configurations after each steps following ‘build’.
    Example: {

        “autobuild”: {
            “schedule”: “0 8 * * 7”, // Make a build every 08:00 on Sunday. “type”: “diff”, // Auto create a
            // “diff” w/previous version.

            // The other option is “snapshot”.

            “env”: “local”, // ES env to create an index and snapshot,
            // not required when type above is diff. // Setting the env also implies auto snapshot. // It
            could be in addition to auto diff. // Also accept (indexer_env, snapshot_env).

        }, “autopublish”: {

            “type”: “snapshot”, // Auto publish new snapshots for new builds.
            // The type field can also be ‘diff’.

            “env”: “prod”, // The release environment to publish snapshot.
            // Or the release environment to publish diff. // This field is required for either type.

            “note”: True // If we should publish with a release note
            // TODO not implemented yet

        }, “autorelease”: {

            “schedule”: “0 0 * * 1”, // Make a release every Monday at midnight
            // (if there’s a new published version.)

            “type”: “full”, // Only auto install full releases.
            // The release type can also be ‘incremental’.

        }
    }
}
```

} The terms below are used interchangeably.

```

BUILD_TYPES = ('diff', 'snapshot')

RELEASE_TO_BUILD = {'full': 'snapshot', 'incremental': 'diff'}

RELEASE_TYPES = ('incremental', 'full')

export()

should_diff_new_build()

should_install_new_diff()

should_install_new_release()

```

Install the latest version regardless of update type/path.

```
should_install_new_snapshot()
should_publish_new_diff()
should_publish_new_snapshot()
should_snapshot_new_build()

exception biothings.hub.databuild.buildconfig.AutoBuildConfigError
    Bases: Exception
biothings.hub.databuild.buildconfig.test()
```

biothings.hub.databuild.builder

```
exception biothings.hub.databuild.builder.BuilderException
    Bases: Exception
class biothings.hub.databuild.builder.BuilderManager(source_backend_factory=None,
                                                       target_backend_factory=None,
                                                       builder_class=None, poll_schedule=None,
                                                       *args, **kwargs)
```

Bases: *BaseManager*

BuilderManager deals with the different builders used to merge datasources. It is connected to src_build() via sync(), where it grabs build information and register builder classes, ready to be instantiate when triggering builds. source_backend_factory can be a optional factory function (like a partial) that builder can call without any argument to generate a SourceBackend. Same for target_backend_factory for the TargetBackend. builder_class if given will be used as the actual Builder class used for the merge and will be passed same arguments as the base DataBuilder. It can also be a list of classes, in which case the default used one is the first, when it's necessary to define multiple builders.

archive_merge(merge_name)

Delete merged collections and associated metadata

build_config_info()

build_info(id=None, conf_name=None, fields=None, only_archived=False, status=None)

Return build information given an build _id, or all builds if _id is None. “fields” can be passed to select which fields to return or not (mongo notation for projections), if None return everything except:

- “mapping” (too long)

If id is None, more are filtered:

- “sources” and some of “build_config”

only_archived=True will return archived merges only status: will return only successful/failed builds. Can be “success” or “failed”

clean_stale_status()

During startup, search for action in progress which would have been interrupted and change the state to “canceled”. Ex: some donwloading processes could have been interrupted, at startup, “downloading” status should be changed to “canceled” so to reflect actual state on these datasources. This must be overriden in subclass.

clean_temp_collections(build_name, date=None, prefix="")

Delete all target collections created from builder named “build_name” at given date (or any date is none given – carefull...). Date is a string (YYYYMMDD or regex) Common collection name prefix can also be specified if needed.

configure()

Sync with src_build_config and register all build config

create_build_configuration(name, doc_type, sources, roots=None, builder_class=None, params=None, archived=False)

delete_build_configuration(name)

delete_merge(merge_name)

Delete merged collections and associated metadata

delete_merged_data(merge_name)

find_builder_classes()

Find all available build class:

1. classes passed during manager init (build_class) (that includes the default builder)
2. all subclassing DataBuilder in:
 - a. biothings.hub.databuilder.*
 - b. hub.databuilder.* (app-specific)

get_builder(col_name)

get_builder_class(build_config_name)

builder class can be specified different way (in order): 1. within the build_config document (so, per configuration) 2. or defined in the builder manager (so, per manager) 3. or default to DataBuilder

get_query_for_list_merge(only_archived, status=None)

list_merge(build_config=None, only_archived=False)

list_sources(build_name)

List all registered sources used to trigger a build named ‘build_name’

merge(build_name, sources=None, target_name=None, **kwargs)

Trigger a merge for build named ‘build_name’. Optional list of sources can be passed (one single or a list). target_name is the target collection name used to store to merge data. If none, each call will generate a unique target_name.

poll()

Check “whatsnew()” to identify builds which could be automatically built, if {“autobuild” : {...}} is part of the build configuration. “autobuild” contains a dict with “schedule” (aiocron/crontab format), so each build configuration can have a different polling schedule.

register_builder(build_name)

resolve_builder_class(klass)

Resolve class/partial definition to (obj,”type”,”mod.class”) where names (class name, module, docstring, etc...) can directly be accessed whether it’s a standard class or not

```
save_mapping(name, mapping=None, dest='build', mode='mapping')

setup_log()

property source_backend

property target_backend

trigger_merge(doc)

update_build_configuration(name, doc_type, sources, roots=None, builder_class=None, params=None,
                           archived=False)

upsert_build_conf(name, doc_type, sources, roots, builder_class, params, archived)

whatsnew(build_name=None, old=None)
    Return datasources which have changed since last time (last time is datasource information from metadata,
    either from given old src_build doc name, or the latest found if old=None)

class biothings.hub.databuild.builder.DataBuilder(build_name, source_backend, target_backend,
                                                    log_folder, doc_root_key='root', mappers=None,
                                                    default_mapper_class=<class 'bioth-
                                                    ings.hub.databuild.mapper.TransparentMapper'>,
                                                    sources=None, target_name=None, **kwargs)

Bases: object

Generic data builder.

property build_config

check_ready(force=False)

clean_old_collections()

document_cleaner(src_name, *args, **kwargs)
    Return a function taking a document as argument, cleaning the doc as needed, and returning that doc. If
    no function is needed, None. Note: the returned function must be pickleable, careful with lambdas and
    closures.

generate_document_query(src_name)

get_build_version()
    Generate an arbitrary major build version. Default is using a timestamp (YYMMDD) ‘.’ char isn’t allowed
    in build version as it’s reserved for minor versions

get_custom_metadata(sources, job_manager)
    If more metadata is required, this method can be overridden and should return a dict. Existing metadata
    dict will be update with that one before storage.

get_mapper_for_source(src_name, init=True)

get_mapping(sources)
    Merge mappings from src_master

get_pinfo()
    Return dict containing information about the current process (used to report in the hub)
```

get_predicates()

Return a list of predicates (functions returning true/false, as in math logic) which instructs/dictates if job manager should start a job (process/thread)

get_root_document_sources()**get_stats(sources, job_manager)**

Return a dictionnary of metadata for this build. It's usually app-specific and this method may be overridden as needed. By default though, the total number of documents in the merged collection is stored (key "total")

Return dictionary will be merged with any existing metadata in src_build collection. This behavior can be changed by setting a special key within metadata dict: {"__REPLACE__": True} will... replace existing metadata with the one returned here.

"job_manager" is passed in case parallelization is needed. Be aware that this method is already running in a dedicated thread, in order to use job_manager, the following code must be used at the very beginning of its implementation: asyncio.set_event_loop(job_manager.loop)

get_target_name()**init_mapper(mapper_name)****init_state()****keep_archive = 10****property logger****merge(sources=None, target_name=None, force=False, ids=None, steps=('merge', 'post', 'metadata'), job_manager=None, *args, **kwargs)**

Merge given sources into a collection named target_name. If sources argument is omitted, all sources defined for this merger will be merged together, according to what is defined insrc_build_config. If target_name is not defined, a unique name will be generated.

Optional parameters:

- force=True will bypass any safety check
- ids: list of _ids to merge, specifically. If None, all documents are merged.
- steps:
 - merge: actual merge step, create merged documents and store them
 - post: once merge, run optional post-merge process
 - **metadata: generate and store metadata (depends on merger, usually specifies the amount of merged data, source versions, etc...)**

merge_order(other_sources)

Optionally we can override this method to customize the order in which sources should be merged. Default as sorted by name.

async merge_source(src_name, batch_size=100000, ids=None, job_manager=None)**async merge_sources(source_names, steps=('merge', 'post'), batch_size=100000, ids=None, job_manager=None)**

Merge resources from given source_names or from build config. Identify root document sources from the list to first process them. ids can be a list of documents to be merged in particular.

```
post_merge(source_names, batch_size, job_manager)
prepare(state=None)
register_status(status, transient=False, init=False, **extra)
```

Register current build status. A build status is a record in src_build. The key used in this dict is target_name. Then, any operation acting on this target_name is registered in a “jobs” list.

```
resolve_sources(sources)
```

Source can be a string that may contain regex chars. It’s useful when you have plenty of sub-collections prefixed with a source name. For instance, given a source named “blah” stored in as many collections as chromosomes, instead of passing each name as “blah_1”, “blah_2”, etc... “**blah_***” can be specified in build_config. This method resolves potential regexed source name into real, existing collection names

```
setup(sources=None, target_name=None)
```

```
setup_log()
```

```
property source_backend
```

```
store_metadata(res, sources, job_manager)
```

```
property target_backend
```

```
unprepare()
```

reset anything that’s not pickleable (so self can be pickled) return what’s been reset as a dict, so self can be restored once pickled

```
update_src_meta_stats()
```

```
class biothings.hub.databuild.builder.LinkDataBuilder(build_name, source_backend, target_backend,
                                                       *args, **kwargs)
```

Bases: *DataBuilder*

LinkDataBuilder creates a link to the original datasource to be merged, without actually copying the data (merged collection remains empty). This builder is only valid when using only one datasource (thus no real merge) is declared in the list of sources to be merged, and is useful to prevent data duplication between the datasource itself and the resulting merged collection.

```
async merge_source(src_name, *args, **kwargs)
```

```
exception biothings.hub.databuild.builder.ResumeException
```

Bases: *Exception*

```
biothings.hub.databuild.builder.fix_batch_duplicates(docs, fail_if_struct_is_different=False)
```

Remove duplicates from docs based on _id. If _id’s the same but structure is different (not real “duplicates”, but different documents with the same _ids), merge docs all together (dict.update) or raise an error if fail_if_struct_is_different.

```
biothings.hub.databuild.builder.merger_worker(col_name, dest_name, ids, mapper, cleaner, upsert,
                                               merger, batch_num, merger_kwargs=None)
```

```
biothings.hub.databuild.builder.pending(build_name, action_name)
```

```
biothings.hub.databuild.builder.set_pending_to_build(conf_name=None)
```

biothings.hub.databuild.differ

```
class biothings.hub.databuild.differ.BaseDiffer(diff_func, job_manager, log_folder)
    Bases: object

diff(old_db_col_names, new_db_col_names, batch_size=100000, steps=('content', 'mapping', 'reduce',
    'post'), mode=None, exclude=None)
        wrapper over diff_cols() coroutine, return a task

async diff_cols(old_db_col_names, new_db_col_names, batch_size, steps, mode=None, exclude=None)
        Compare new with old collections and produce diff files. Root keys can be excluded from comparison with
        "exclude" parameter

    *_db_col_names can be:
        1. a collection name (as a string) assuming they are in the target database.
        2. tuple with 2 elements, the first one is then either "source" or "target" to respectively specify src
           or target database, and the second element is the collection name.
        3. tuple with 3 elements (URI, db, collection), looking like: ("mongodb://user:pass@host", "dbname", "collection"), allowing to specify any connection on
           any server

    steps: - 'content' will perform diff on actual content.
        • 'mapping' will perform diff on ES mappings (if target collection involved)
        • 'reduce' will merge diff files, trying to avoid having many small files
        • 'post' is a hook to do stuff once everything is merged (override method post_diff_cols)

    mode: 'purge' will remove any existing files for this comparison while 'resume' will happily ignore
        existing data and to whatever it's requested (like running steps="post" on existing diff folder...)

diff_type = None

get_metadata()

get_pinfo()
    Return dict containing information about the current process (used to report in the hub)

get_predicates()

post_diff_cols(old_db_col_names, new_db_col_names, batch_size, steps, mode=None, exclude=None)
    Post diff process hook. This coroutine will run in a dedicated thread

register_status(status, transient=False, init=False, **extra)

setup_log(old=None, new=None)

class biothings.hub.databuild.differ.ColdHotDiffer(diff_func, job_manager, log_folder)
    Bases: BaseDiffer

async diff_cols(old_db_col_names, new_db_col_names, *args, **kwargs)
    Compare new with old collections and produce diff files. Root keys can be excluded from comparison with
    "exclude" parameter

    *_db_col_names can be:
        1. a collection name (as a string) assuming they are in the target database.
```

2. tuple with 2 elements, the first one is then either “source” or “target” to respectively specify src or target database, and the second element is the collection name.
3. tuple with 3 elements (URI, db, collection), looking like: (“mongodbs://user:pass@host”, “dbname”, “collection”), allowing to specify any connection on any server

steps: - ‘content’ will perform diff on actual content.

- ‘mapping’ will perform diff on ES mappings (if target collection involved)
- ‘reduce’ will merge diff files, trying to avoid having many small files
- ‘post’ is a hook to do stuff once everything is merged (override method post_diff_cols)

mode: ‘purge’ will remove any existing files for this comparison while ‘resume’ will happily ignore existing data and to whatever it’s requested (like running steps=“post” on existing diff folder...)

get_metadata()

```
class biothings.hub.databuild.differ.ColdHotJsonDiffer(diff_func=<function diff_docs_jsonpatch>,
                                                       *args, **kwargs)
```

Bases: *ColdHotJsonDifferBase, JsonDiffer*

diff_type = ‘coldhot-jsondiff’

```
class biothings.hub.databuild.differ.ColdHotJsonDifferBase(diff_func, job_manager, log_folder)
```

Bases: *ColdHotDiffer*

post_diff_cols(*old_db_col_names, new_db_col_names, batch_size, steps, mode=None, exclude=None*)

Post-process the diff files by adjusting some jsondiff operation. Here’s the process. For updated documents, some operations might illegal in the context of cold/hot merged collections. Case #1: “remove” op in an update

from a cold/premerge collection, we have that doc:

```
coldd = {“_id”:1, “A”:”123”, “B”:”456”, “C”:True}
```

from the previous hot merge we have this doc:

```
prevd = {“_id”:1, “D”:”789”, “C”:True, “E”:”abc”}
```

At that point, the final document, fully merged and indexed is:

```
finald = {“_id”:1, “A”:”123”, “B”:”456”, “C”:True, “D”:”789”, “E”:”abc”}
```

We can notice field “C” is common to coldd and prevd.

from the new hot merge, we have:

```
newd = {“_id”:1, “E”:”abc”} # C and D don’t exist anymore
```

Diffing prevd vs. newd will give jsondiff operations:

```
[{‘op’: ‘remove’, ‘path’: ‘/C’}, {‘op’: ‘remove’, ‘path’: ‘/D’}]
```

The problem here is ‘C’ is removed while it was already in cold merge, it should stay because it has come with some resource involved in the premerge (dependent keys, eg. myvariant, “observed” key comes with certain sources) => the jsondiff operation on “C” must be discarded.

Note: If operation involved a root key (not ‘/a/c’ for instance) and if that key is found in the premerge, then

then remove the operation. (note we just consider root keys, if the deletion occurs deeper in the document, it’s just a legal operation updating inner content)

For deleted documents, the same kind of logic applies Case #2: “delete”

from a cold/premerge collection, we have that doc:
`coldd = {"_id":1, "A":"123", "B":"456", "C":True}`

from the previous hot merge we have this doc:
`prevd = {"_id":1, "D":"789", "C":True}`

fully merged doc:
`finald = {"_id":1, "A":"123", "B":"456", "C":True, "D":"789"}`

from the new hot merge, we have:
`newd = {} # document doesn't exist anymore`

Diffing prevd vs. newd will mark document with `_id == 1` to be deleted. The problem is we have data for `_id=1` on the premerge collection, if we delete the whole document we'd lose too much information. => the deletion must be converted into specific "remove" jsondiff operations, for the root keys found in prevd on not in coldd

(in that case: `[{"op":'remove', "path":'/D'}]`, and not "C" as C is in premerge)

```
class biothings.hub.databuild.differ.ColdHotSelfContainedJsonDiffer(diff_func=<function
diff_docs_jsonpatch>,
*args, **kwargs)
```

Bases: `ColdHotJsonDifferBase, SelfContainedJsonDiffer`

diff_type = 'coldhot-jsondiff-selfcontained'

```
class biothings.hub.databuild.differ.DiffReportRendererBase(max_reported_ids=None,
max_randomly_picked=None,
detailed=False)
```

Bases: `object`

save(report, filename)

Save report output (rendered) into filename

```
class biothings.hub.databuild.differ.DiffReportTxt(max_reported_ids=None,
max_randomly_picked=None, detailed=False)
```

Bases: `DiffReportRendererBase`

save(report, filename='report.txt')

Save report output (rendered) into filename

```
exception biothings.hub.databuild.differ.DifferException
```

Bases: `Exception`

```
class biothings.hub.databuild.differ.DifferManager(poll_schedule=None, *args, **kwargs)
```

Bases: `BaseManager`

DifferManager deals with the different differ objects used to create and analyze diff between datasources.

build_diff_report(diff_folder, detailed=True, max_reported_ids=None)

Analyze diff files in diff_folder and give a summary of changes. max_reported_ids is the number of IDs contained in the report for each part. detailed will trigger a deeper analysis, takes more time.

clean_stale_status()

During startup, search for action in progress which would have been interrupted and change the state to "canceled". Ex: some downloading processes could have been interrupted, at startup, "downloading" status should be changed to "canceled" so to reflect actual state on these datasources. This must be overridden in subclass.

```
configure(partial_differs=(<class 'biothings.hub.databuild.differ.JsonDiffer'>, <class  
    'biothings.hub.databuild.differ.SelfContainedJsonDiffer'>))  
  
diff(diff_type, old, new, batch_size=100000, steps=('content', 'mapping', 'reduce', 'post'), mode=None,  
    exclude=('_timestamp',))  
Run a diff to compare old vs. new collections. using differ algorithm diff_type. Results are stored in a  
diff folder. Steps can be passed to choose what to do: - count: will count root keys in new collections and  
stores them as statistics. - content: will diff the content between old and new. Results (diff files) format  
depends on diff_type  
  
diff_info()  
  
diff_report(old_db_col_names, new_db_col_names, report_filename='report.txt', format='txt',  
    detailed=True, max_reported_ids=None, max_randomly_picked=None, mode=None)  
  
get_pinfo()  
Return dict containing information about the current process (used to report in the hub)  
  
get_predicates()  
  
poll(state, func)  
Search for source in collection 'col' with a pending flag list containing 'state' and call 'func' for each  
document found (with doc as only param)  
  
rebuild_diff_file_list(diff_folder)  
  
register_differ(klass)  
  
setup_log()  
  
trigger_diff(diff_type, doc, **kwargs)  
Launch a diff given a src_build document. In order to know the first collection to diff against,  
get_previous_collection() method is used.  
  
class biothings.hub.databuild.differ.JsonDiffer(diff_func=<function diff_docs_jsonpatch>, *args,  
    **kwargs)  
Bases: BaseDiffer  
diff_type = 'jsondiff'  
  
class biothings.hub.databuild.differ.SelfContainedJsonDiffer(diff_func=<function  
    diff_docs_jsonpatch>, *args,  
    **kwargs)  
Bases: JsonDiffer  
diff_type = 'jsondiff-selfcontained'  
  
biothings.hub.databuild.differ.diff_worker_count(id_list, db_col_names, batch_num)  
  
biothings.hub.databuild.differ.diff_worker_new_vs_old(id_list_new, old_db_col_names,  
    new_db_col_names, batch_num, diff_folder,  
    diff_func, exclude=None,  
    selfcontained=False)  
  
biothings.hub.databuild.differ.diff_worker_old_vs_new(id_list_old, new_db_col_names, batch_num,  
    diff_folder)
```

`biothings.hub.databuild.differ.reduce_diffs(diffs, num, diff_folder, done_folder)`
`biothings.hub.databuild.differ.set_pending_to_diff(col_name)`

biothings.hub.databuild.mapper

class biothings.hub.databuild.mapper.BaseMapper(name=None, *args, **kwargs)

Bases: `object`

Basic mapper used to convert documents. if mapper's name matches source's metadata's mapper, mapper.convert(docs) call will be used to process/convert/whatever passed documents

load()

Do whatever is required to fill mapper with mapping data Can be called multiple time, the first time only will load data

process(docs)

Convert given docs into other docs.

class biothings.hub.databuild.mapper.IDBaseMapper(name=None, convert_func=None, *args, **kwargs)

Bases: `BaseMapper`

Provide mapping between different sources

‘name’ may match a “mapper” metatdata field (see uploaders). If None, mapper will be applied to any document from a resource without “mapper” argument

need_load()

process(docs, key_to_convert='_id', transparent=True)

Process ‘key_to_convert’ document key using mapping. If transparent and no match, original key will be used (so there’s no change). Else, if no match, document will be discarded (default). Warning: key to be translated must not be None (it’s considered a non-match)

translate(_id, transparent=False)

Return _id translated through mapper, or _id itself if not part of mapper If ‘transparent’ and no match, original _id will be returned

class biothings.hub.databuild.mapper.TransparentMapper(name=None, *args, **kwargs)

Bases: `BaseMapper`

load(*args, **kwargs)

Do whatever is required to fill mapper with mapping data Can be called multiple time, the first time only will load data

process(docs, *args, **kwargs)

Convert given docs into other docs.

biothings.hub.databuild.prebuilder

```
class biothings.hub.databuild.prebuilder.BasePreCompiledDataProvider(name)
    Bases: object
        'name' is a way to identify this provider (usually linked to a database name behind the scene)
    get_all()
        Iterate over all register _ids, return a list of collection names where they can be found
    register(_id, col_name)
        Tell provider that _id can be found in collection named 'col_name'
class biothings.hub.databuild.prebuilder.MongoDBPreCompiledDataProvider(db_name, name,
                                                                      connection_params)
    Bases: BasePreCompiledDataProvider
        'name' is a way to identify this provider (usually linked to a database name behind the scene)
    get_all(batch_size=100000)
        Iterate over all register _ids, return a list of collection names where they can be found
    register(_id, col_name)
        Tell provider that _id can be found in collection named 'col_name'
class biothings.hub.databuild.prebuilder.RedisPreCompiledDataProvider(name,
                                                                      connection_params)
    Bases: BasePreCompiledDataProvider
        'name' is a way to identify this provider (usually linked to a database name behind the scene)
    get_all()
        Iterate over all register _ids, return a list of collection names where they can be found
    register(_id, col_name)
        Tell provider that _id can be found in collection named 'col_name'
```

biothings.hub.databuild.syncer

```
class biothings.hub.databuild.syncer.BaseSyncer(job_manager, log_folder)
    Bases: object
    diff_type = None
    get_pinfo()
    get_predicates()
    get_target_backend()
    load_metadata(diff_folder)
    post_sync_cols(diff_folder, batch_size, mode, force, target_backend, steps)
        Post-sync hook, can be implemented in sub-class
    register_status(status, transient=False, init=False, **extra)
```

```

setup_log(build_name=None)

sync(diff_folder=None, batch_size=10000, mode=None, target_backend=None, steps=('mapping', 'content',
    'meta', 'post'), debug=False)
    wrapper over sync_cols() coroutine, return a task

async sync_cols(diff_folder, batch_size=10000, mode=None, force=False, target_backend=None,
    steps=('mapping', 'content', 'meta', 'post'), debug=False)
    Sync a collection with diff files located in diff_folder. This folder contains a metadata.json file which
    describes the different involved collection: "old" is the collection/index to be synced, "new" is the collec-
    tion that should be obtained once all diff files are applied (not used, just informative). If target_backend
    (bt.databuild.backend.create_backend() notation), then it will replace "old" (that is, the one being synced)

target_backend_type = None

class biothings.hub.databuild.syncer.ESColdHotJsonDiffSelfContainedSyncer(job_manager,
    log_folder)
    Bases: BaseSyncer
    diff_type = 'coldhot-jsondiff-selfcontained'
    target_backend_type = 'es'

class biothings.hub.databuild.syncer.ESColdHotJsonDiffSyncer(job_manager, log_folder)
    Bases: BaseSyncer
    diff_type = 'coldhot-jsondiff'
    target_backend_type = 'es'

class biothings.hub.databuild.syncer.ESJsonDiffSelfContainedSyncer(job_manager, log_folder)
    Bases: BaseSyncer
    diff_type = 'jsondiff-selfcontained'
    target_backend_type = 'es'

class biothings.hub.databuild.syncer.ESJsonDiffSyncer(job_manager, log_folder)
    Bases: BaseSyncer
    diff_type = 'jsondiff'
    target_backend_type = 'es'

class biothings.hub.databuild.syncer.MongoJsonDiffSelfContainedSyncer(job_manager,
    log_folder)
    Bases: BaseSyncer
    diff_type = 'jsondiff-selfcontained'
    target_backend_type = 'mongo'

class biothings.hub.databuild.syncer.MongoJsonDiffSyncer(job_manager, log_folder)
    Bases: BaseSyncer
    diff_type = 'jsondiff'
    target_backend_type = 'mongo'

```

```
exception biothings.hub.databuild.syncer.SyncerException
```

Bases: Exception

```
class biothings.hub.databuild.syncer.SyncerManager(*args, **kwargs)
```

Bases: *BaseManager*

SyncerManager deals with the different syncer objects used to synchronize different collections or indices using diff files

```
clean_stale_status()
```

During startup, search for action in progress which would have been interrupted and change the state to “canceled”. Ex: some downloading processes could have been interrupted, at startup, “downloading” status should be changed to “canceled” so to reflect actual state on these datasources. This must be overridden in subclass.

```
configure(klasses=None)
```

Register default syncers (if klasses is None) or given klasses. klasses is a list of class, or a list of partially initialized classes.

```
register_syncer(klass)
```

```
setup_log()
```

```
sync(backend_type, old_db_col_names, new_db_col_names, diff_folder=None, batch_size=10000,  
mode=None, target_backend=None, steps=('mapping', 'content', 'meta', 'post'), debug=False)
```

```
class biothings.hub.databuild.syncer.ThrottledESColdHotJsonDiffSelfContainedSyncer(max_sync_workers,  
*args,  
**kwargs)
```

Bases: *ThrottlerSyncer*, *ESColdHotJsonDiffSelfContainedSyncer*

```
class biothings.hub.databuild.syncer.ThrottledESColdHotJsonDiffSyncer(max_sync_workers,  
*args, **kwargs)
```

Bases: *ThrottlerSyncer*, *ESColdHotJsonDiffSyncer*

```
class biothings.hub.databuild.syncer.ThrottledESJsonDiffSelfContainedSyncer(max_sync_workers,  
*args,  
**kwargs)
```

Bases: *ThrottlerSyncer*, *ESJsonDiffSelfContainedSyncer*

```
class biothings.hub.databuild.syncer.ThrottledESJsonDiffSyncer(max_sync_workers, *args,  
**kwargs)
```

Bases: *ThrottlerSyncer*, *ESJsonDiffSyncer*

```
class biothings.hub.databuild.syncer.ThrottlerSyncer(max_sync_workers, *args, **kwargs)
```

Bases: *BaseSyncer*

```
get_predicates()
```

```
biothings.hub.databuild.syncer.sync_es_coldhot_jsondiff_worker(diff_file, es_config,  
new_db_col_names, batch_size,  
cnt, force=False,  
selfcontained=False,  
metadata=None, debug=False)
```

```
biothings.hub.databuild.syncer.sync_es_for_update(diff_file, indexer, diffupdates, batch_size, res,  
debug)
```

```
biothings.hub.databuild.syncer.sync_es_jsondiff_worker(diff_file, es_config, new_db_col_names,
batch_size, cnt, force=False,
selfcontained=False, metadata=None,
debug=False)
```

Worker to sync data between a new mongo collection and an elasticsearch index

```
biothings.hub.databuild.syncer.sync_mongo_jsondiff_worker(diff_file, old_db_col_names,
new_db_col_names, batch_size, cnt,
force=False, selfcontained=False,
metadata=None, debug=False)
```

Worker to sync data between a new and an old mongo collection

biothings.hub.dataexport

biothings.hub.dataexport.ids

```
biothings.hub.dataexport.ids.export_ids(col_name)
```

Export all _ids from collection named col_name. If col_name refers to a build where a cold_collection is defined, will also extract _ids and sort/uniq them to have the full list of _ids of the actual merged (cold+hot) collection
Output file is stored in DATA_EXPORT_FOLDER/ids, defaulting to <DATA_ARCHIVE_ROOT>/export/ids.
Output filename is returned as the end, if successful.

```
biothings.hub.dataexport.ids.upload_ids(ids_file, redirect_from, s3_bucket, aws_key, aws_secret)
```

Upload file ids_file into s3_bucket and modify redirect_from key's metadata so redirect_from link will now point to ids_file redirect_from s3 key must exist.

biothings.hub.dataindex

biothings.hub.dataindex.idcache

```
class biothings.hub.dataindex.idcache.IDCache
```

Bases: *object*

```
load(name, id_provider, flush=True)
```

name is the cache name id_provider returns batch of ids, ie. list(_ids) flush to delete existing cache

```
mark_done(_ids)
```

```
class biothings.hub.dataindex.idcache.RedisIDCache(name, connection_params)
```

Bases: *IDCache*

```
load(id_provider, flush=True)
```

name is the cache name id_provider returns batch of ids, ie. list(_ids) flush to delete existing cache

```
mark_done(_ids)
```

biothings.hub.dataindex.indexer_cleanup

```
class biothings.hub.dataindex.indexer_cleanup.CleanUpResult(iterable=(), /)
    Bases: list

class biothings.hub.dataindex.indexer_cleanup.Cleaner(collection, indexers, logger=None)
    Bases: object

    async clean(cleanups)

    find(env=None, keep=3, **filters)

    static plain_text(cleanups)

biothings.hub.dataindex.indexer_cleanup.test_clean()
biothings.hub.dataindex.indexer_cleanup.test_find()
biothings.hub.dataindex.indexer_cleanup.test_str()
```

biothings.hub.dataindex.indexer_payload

```
class biothings.hub.dataindex.indexer_payload.IndexMappings(dict=None, /, **kwargs)
    Bases: _IndexPayload

    async finalize(client)
        Generate the ES payload format of the corresponding entities originally in Hub representation. May require querying the ES client for certain metadata to determine the compatible data format.

class biothings.hub.dataindex.indexer_payload.IndexSettings(dict=None, /, **kwargs)
    Bases: _IndexPayload

    async finalize(client)
        Generate the ES payload format of the corresponding entities originally in Hub representation. May require querying the ES client for certain metadata to determine the compatible data format.
```

biothings.hub.dataindex.indexer

```
class biothings.hub.dataindex.indexer.ColdHotIndexer(build_doc, indexer_env, index_name)
    Bases: object

    MongoDB to Elasticsearch 2-pass Indexer. (
        1st pass: <MongoDB Cold Collection>, # static data
        2nd pass: <MongoDB Hot Collection> # changing data
    ) =>
        <Elasticsearch Index>

INDEXER
    alias of Indexer

    async index(job_manager, batch_size=10000, steps=('pre', 'index', 'post'), ids=None, mode=None, **kwargs)
```

```
class biothings.hub.dataindex.indexer.DynamicIndexerFactory(urls, es_host, suffix='_current')
```

Bases: object

In the context of autohub/standalone instances, create indexer with parameters taken from versions.json URL. A list of URLs is provided so the factory knows how to create these indexers for each URLs. There's no way to "guess" an ES host from a URL, so this parameter must be specified as well, common to all URLs "suffix" param is added at the end of index names.

create(name)

```
class biothings.hub.dataindex.indexer.IndexManager(*args, **kwargs)
```

Bases: *BaseManager*

An example of config dict for this module. {

```
    "indexer_select": {
        None: "hub.dataindex.indexer.DrugIndexer", # default "build_config.cold_collection" :
        "mv.ColdHotVariantIndexer",
    },
    "env": {
        "prod": {
            "host": "localhost:9200", "indexer": {
                "args": {
                    "timeout": 300, "retry_on_timeout": True, "max_retries": 10,
                },
                "bulk": {
                    "chunk_size": 50, "raise_on_exception": False
                },
                "concurrency": 3
            },
            "index": [
                # for information only, only used in index_info {"index": "my-drugs_current", "doc_type": "drug"}, {"index": "mygene_current", "doc_type": "gene"}
            ],
        },
        "dev": { ... }
    }
}
```

DEFAULT_INDEXER

alias of *Indexer*

clean_stale_status()

During startup, search for action in progress which would have been interrupted and change the state to "canceled". Ex: some downloading processes could have been interrupted, at startup, "downloading" status should be changed to "canceled" so to reflect actual state on these datasources. This must be overridden in subclass.

cleanup(env=None, keep=3, dryrun=True, **filters)

Delete old indices except for the most recent ones.

Examples

```
>>> index_cleanup()  
>>> index_cleanup("production")  
>>> index_cleanup("local", build_config="demo")  
>>> index_cleanup("local", keep=0)  
>>> index_cleanup(_id="")
```

configure(conf)

get_indexes_by_name(index_name=None, env_name=None, limit=10)

Accept an index_name and return a list of indexes get from all elasticsearch environments or from specific elasticsearch environment.

If index_name is blank, it will be return all indexes. limit can be used to specify how many indexes should be return.

The list of indexes will be like this: [

```
{  
    "index_name": "...", "build_version": "...", "count": 1000, "creation_date":  
    1653468868933, "environment": {  
        "name": "env name", "host": "localhost:9200",  
    }  
},  
]
```

get_pinfo()

Return dict containing information about the current process (used to report in the hub)

get_predicates()

index(indexer_env, build_name, index_name=None, ids=None, **kwargs)

Trigger an index creation to index the collection build_name and create an index named index_name (or build_name if None). Optional list of IDs can be passed to index specific documents.

index_info(remote=False)

Show index manager config with enhanced index information.

update_metadata(indexer_env, index_name, build_name=None, _meta=None)

Update _meta field of the index mappings, basing on

1. the _meta value provided, including {}.
2. the _meta value of the build_name in src_build.
3. the _meta value of the build with the same name as the index.

Examples

```

update_metadata("local", "mynews_201228_vsdevjd") update_metadata("local", "mynews_201228_current",
"mynews_201228_vsdevjd", _meta={ }) update_metadata("local", "mynews_201228_vsdevjd")
_update_metadata("author": "b") update_metadata("local", "mynews_201228_vsdevjd")

validate_mapping(mapping, env)

class biothings.hub.dataindex.Indexer(build_doc, indexer_env, index_name)
    Bases: object
    MongoDB -> Elasticsearch Indexer.

    async do_index(job_manager, batch_size, ids, mode, **kwargs)
    async index(job_manager, **kwargs)
        Build an Elasticsearch index (self.es_index_name) with data from MongoDB collection
        (self.mongo_collection_name).
        "ids" can be passed to selectively index documents.

    "mode" can have the following values:
        • 'purge': will delete an index if it exists.
        • 'resume': will use an existing index and add missing documents.
        • 'merge': will merge data to an existing index.
        • 'index' (default): will create a new index.

    async post_index(*args, **kwargs)
    async pre_index(*args, mode, **kwargs)
    setup_log()

class biothings.hub.dataindex.IndexerCumulativeResult(dict=None, /, **kwargs)
    Bases: _IndexerResult

exception biothings.hub.dataindex.IndexerException
    Bases: Exception

class biothings.hub.dataindex.IndexerStepResult(dict=None, /, **kwargs)
    Bases: _IndexerResult

class biothings.hub.dataindex.MainIndexStep(indexer)
    Bases: Step
    method: property(abc.abstractmethod(lambda _: ...)) = 'do_index'
    name: property(abc.abstractmethod(lambda _: ...)) = 'index'
    state
        alias of MainIndexJSR

class biothings.hub.dataindex.PostIndexStep(indexer)
    Bases: Step
    method: property(abc.abstractmethod(lambda _: ...)) = 'post_index'

```

```
name: property(abc.abstractmethod(lambda _: ...)) = 'post'

state
    alias of PostIndexJSR

class biothings.hub.dataindex.indexer.PreIndexStep(indexer)
    Bases: Step
        method: property(abc.abstractmethod(lambda _: ...)) = 'pre_index'
        name: property(abc.abstractmethod(lambda _: ...)) = 'pre'

        state
            alias of PreIndexJSR

class biothings.hub.dataindex.indexer.ProcessInfo(indexer, concurrency)
    Bases: object
        get_pinfo(step='', description='')
            Return dict containing information about the current process (used to report in the hub)
        get_predicates()

class biothings.hub.dataindex.indexer.Step(indexer)
    Bases: ABC
        catalog = {'index': <class 'biothings.hub.dataindex.indexer.MainIndexStep'>,
                   'post': <class 'biothings.hub.dataindex.indexer.PostIndexStep'>, 'pre': <class 'biothings.hub.dataindex.indexer.PreIndexStep'>}

        classmethod dispatch(name)
        async execute(*args, **kwargs)

        method: <property object at 0x7f0c16b009a0>
        name: <property object at 0x7f0c16b00900>
        static order(steps)
        state: <property object at 0x7f0c16b00950>

biothings.hub.dataindex.indexer_registrar

class biothings.hub.dataindex.indexer_registrar.IndexJobStateRegistrar(collection, build_name,
                           index_name,
                           **context)
    Bases: object
    failed(error)
    static prune(collection)
    started(step='index')
    succeed(result)
```

```
class biothings.hub.dataindex.indexer_registrar.MainIndexJSR(collection, build_name, index_name,
**context)
    Bases: IndexJobStateRegistrar
    started()

class biothings.hub.dataindex.indexer_registrar.PostIndexJSR(collection, build_name, index_name,
**context)
    Bases: IndexJobStateRegistrar
    started()

class biothings.hub.dataindex.indexer_registrar.PreIndexJSR(collection, build_name, index_name,
**context)
    Bases: IndexJobStateRegistrar
    started()
    succeed(result)

class biothings.hub.dataindex.indexer_registrar.Stage(value)
    Bases: Enum
    An enumeration.
    DONE = 2
    READY = 0
    STARTED = 1
    at(stage)

biothings.hub.dataindex.indexer_registrar.test_registrar()
```

biothings.hub.dataindex.indexer_schedule

```
class biothings.hub.dataindex.indexer_schedule.Schedule(total, batch_size)
    Bases: object
    completed()
    suffix(string)

biothings.hub.dataindex.indexer_schedule.test_01()
biothings.hub.dataindex.indexer_schedule.test_02()
biothings.hub.dataindex.indexer_schedule.test_03()
biothings.hub.dataindex.indexer_schedule.test_04()
```

biothings.hub.dataindex.indexer_task

```
class biothings.hub.dataindex.indexer_task.ESIndex(client, index_name, **bulk_index_args)
```

Bases: *ESIndex*

mexists(*ids*)

Return a list of tuples like [

 (_id_0, True), (_id_1, False), (_id_2, True),

]

mget(*ids*)

Return a list of documents like [

 { “_id”: “0”, “a”: “b” }, { “_id”: “1”, “c”: “d” }, # 404s are skipped

]

mindex(*docs*)

Index and return the number of docs indexed.

```
class biothings.hub.dataindex.indexer_task.IndexingTask(es, mongo, ids, mode=None, logger=None, name='task')
```

Bases: *object*

Index one batch of documents from MongoDB to Elasticsearch. The documents to index are specified by their ids.

dispatch()

index()

merge()

resume()

```
class biothings.hub.dataindex.indexer_task.Mode(value)
```

Bases: *Enum*

An enumeration.

INDEX = ‘index’

MERGE = ‘merge’

PURGE = ‘purge’

RESUME = ‘resume’

```
biothings.hub.dataindex.indexer_task.dispatch(mg_client_args, mg_dbs_name, mg_col_name, es_client_args, es_blk_args, es_idx_name, ids, mode, name)
```

```
biothings.hub.dataindex.indexer_task.test0()
```

```
biothings.hub.dataindex.indexer_task.test1()
```

```
biothings.hub.dataindex.indexer_task.test_00()
```

```
biothings.hub.dataindex.indexer_task.test_clients()
```

biothings.hub.dataindex.snapshooter

```

class biothings.hub.dataindex.snapshooter.Bucket(client, bucket, region=None)
    Bases: object
        create(acl='private')
        exists()

class biothings.hub.dataindex.snapshooter.CloudStorage(type: str, access_key: str, secret_key: str,
                                                       region: str = 'us-west-2')
    Bases: object
        access_key: str
        get()
        region: str = 'us-west-2'
        secret_key: str
        type: str

class biothings.hub.dataindex.snapshooter.CumulativeResult(dict=None, /, **kwargs)
    Bases: _SnapshotResult

class biothings.hub.dataindex.snapshooter.ProcessInfo(env)
    Bases: object
    JobManager Process Info. Reported in Biothings Studio.
        get_pinfo(step, snapshot, description="")
        get_predicates()

class biothings.hub.dataindex.snapshooter.RenderedStr(seq)
    Bases: _UserString

class biothings.hub.dataindex.snapshooter.RepositoryConfig(dict=None, /, **kwargs)
    Bases: UserDict
    {
        “type”: “s3”, “name”: “s3-$(Y)”, “settings”: {
            “bucket”: “<SNAPSHOT_BUCKET_NAME>”, “base_path”: “mynews.info/${Y}”, # per year
        }
    }
    property bucket
    format(doc=None)
        Template special values in this config.
        For example: {
            “bucket”: “backup-$(Y)”, “base_path” : “snapshots/%(_meta.build_version)s”
        } where “_meta.build_version” value is taken from doc in dot field notation, and the current year replaces
            “$(Y)”.

```

```
property region
property repo

class biothings.hub.dataindex.snapshooter.SnapshotEnv(job_manager, cloud, repository, indexer,
                                                       **kwargs)
Bases: object
post_snapshot(cfg, index, snapshot, **kwargs)
pre_snapshot(cfg, index, snapshot, **kwargs)
setup_log(index)
snapshot(index, snapshot=None, recreate_repo=False)

class biothings.hub.dataindex.snapshooter.SnapshotManager(index_manager, *args, **kwargs)
Bases: BaseManager
Hub ES Snapshot Management
Config Ex:
# env.<name>: {
    "cloud": {
        "type": "aws", # default, only one supported. "access_key": <----->, "secret_key": <----->, "region": "us-west-2"
    }, "repository": {
        "name": "s3-$(Y)", "type": "s3", "settings": {
            "bucket": "<SNAPSHOT_BUCKET_NAME>", "base_path": "my-
gene.info/$(Y)", # year
        }, "acl": "private",
    }, "indexer": {
        "name": "local", "args": {
            "timeout": 100, "max_retries": 5
        }
    }, "monitor_delay": 15,
}
clean_stale_status()

During startup, search for action in progress which would have been interrupted and change the state to "canceled". Ex: some downloading processes could have been interrupted, at startup, "downloading" status should be changed to "canceled" so to reflect actual state on these datasources. This must be overridden in subclass.

cleanup(env=None, keep=3, group_by='build_config', dryrun=True, **filters)
Delete past snapshots and keep only the most recent ones.
```

Examples

```
>>> snapshot_cleanup()
>>> snapshot_cleanup("s3_outbreak")
>>> snapshot_cleanup("s3_outbreak", keep=0)
```

configure(conf)

delete_snapshots(snapshots_data)

list_snapshots(env=None, **filters)

static pending_snapshot(build_name)

poll(state, func)

Search for source in collection ‘col’ with a pending flag list containing ‘state’ and call ‘func’ for each document found (with doc as only param)

snapshot(snapshot_env, index, snapshot=None, recreate_repo=False)

Create a snapshot named “snapshot” (or, by default, same name as the index) from “index” according to environment definition (repository, etc...) “env”.

snapshot_a_build(build_doc)

Create a snapshot basing on the autobuild settings in the build config. If the build config associated with this build has: {

“autobuild”: {

“type”: “snapshot”, // implied when env is set. env must be set. “env”: “local” // which es env to make the snapshot.

} Attempt to make a snapshot for this build on the specified es env “local”.

snapshot_info(env=None, remote=False)

class biothings.hub.dataindex.snapshooter.StepResult(dict=None, /, **kwargs)

Bases: _SnapshotResult

class biothings.hub.dataindex.snapshooter.TemplateStr(seq)

Bases: _UserString

biothings.hub.dataindex.snapshot_cleanup

biothings.hub.dataindex.snapshot_cleanup.delete(collection, element, envs)

biothings.hub.dataindex.snapshot_cleanup.find(collection, env=None, keep=3, group_by=None, return_db_cols=False, **filters)

biothings.hub.dataindex.snapshot_cleanup/plain_text(element)

biothings.hub.dataindex.snapshot_cleanup/test_find()

biothings.hub.dataindex.snapshot_cleanup/test_print()

biothings.hub.dataindex.snapshot_registrar

```
class biothings.hub.dataindex.snapshot_registrar.MainSnapshotState(col, _id)
    Bases: _TaskState
    func = '_snapshot'
    name = 'snapshot'
    regex = True
    step = 'snapshot'

class biothings.hub.dataindex.snapshot_registrar.PostSnapshotState(col, _id)
    Bases: _TaskState
    func = 'post_snapshot'
    name = 'post'
    regex = True
    step = 'post-snapshot'

class biothings.hub.dataindex.snapshot_registrar.PreSnapshotState(col, _id)
    Bases: _TaskState
    func = 'pre_snapshot'
    name = 'pre'
    step = 'pre-snapshot'

biothings.hub.dataindex.snapshot_registrar.audit(src_build, logger=None)
biothings.hub.dataindex.snapshot_registrar.dispatch(step)
biothings.hub.dataindex.snapshot_registrar.test()
```

biothings.hub.dataindex.snapshot_repo

```
class biothings.hub.dataindex.snapshot_repo.Repository(client, repository)
    Bases: object
    create(**body)
    delete()
    exists()
    verify(config)
```

A repository is consider properly setup and working, when: - passes verification of ElasticSearch - it's settings must match with the snapshot's config.

```
biothings.hub.dataindex.snapshot_repo.test_01()
biothings.hub.dataindex.snapshot_repo.test_02()
```

biothings.hub.dataindex.snapshot_task

```
class biothings.hub.dataindex.snapshot_task.Snapshot(client, repository, snapshot)
    Bases: object
    create(indices)
    delete()
    exists()
    state()

biothings.hub.dataindex.snapshot_task.test_01()
biothings.hub.dataindex.snapshot_task.test_02()
```

biothings.hub.datainspect**biothings.hub.datainspect.inspector**

```
exception biothings.hub.datainspect.inspector InspectorError
    Bases: Exception
class biothings.hub.datainspect.inspector InspectorManager(upload_manager, build_manager,
    *args, **kwargs)
```

Bases: *BaseManager*

clean_stale_status()

During startup, search for action in progress which would have been interrupted and change the state to “canceled”. Ex: some downloading processes could have been interrupted, at startup, “downloading” status should be changed to “canceled” so to reflect actual state on these datasources. This must be overridden in subclass.

flatten(*data_provider, mode=*(‘type’, ‘stats’), *do_validate=True*)

get_backend_provider_info(*data_provider*)

inspect(*data_provider, mode=*‘type’, *batch_size=10000, limit=None, sample=None, **kwargs*)

Inspect given data provider: - backend definition, see bt.hub.dababuild.create_backend for supported format), eg “merged_collection” or (“src”, “clinvar”)

- or callable yielding documents

Mode: - “type”: will inspect and report type map found in data (internal/non-standard format) - “mapping”: will inspect and return a map compatible for later

ElasticSearch mapping generation (see bt.utils.es.generate_es_mapping)

- “stats”: will inspect and report types + different counts found in data, giving a detailed overview of the volumetry of each fields and sub-fields
- “jsonschema”, same as “type” but result is formatted as json-schema standard
- limit: when set to an integer, will inspect only x documents.

- sample: combined with limit, for each document, if random.random() <= sample (float), the document is inspected. This option allows to inspect only a sample of data.

setup_log()

Setup and return a logger instance

```
biothings.hub.datainspect.inspector.inspect_data(backend_provider, ids, mode, pre_mapping,  
                                                **kwargs)
```

biothings.hub.dataloader**biothings.hub.dataloader.dumper**

```
class biothings.hub.dataloader.dumper.APIDumper(src_name=None, src_root_folder=None,  
                                               log_folder=None, archive=None)
```

Bases: *BaseDumper*

Dump data from APIs

This will run API calls in a clean process and write its results in one or more NDJSON documents.

Populate the static methods `get_document` and `get_release` in your subclass, along with other necessary bits common to all dumper.

For details on specific parts, read the docstring for individual methods.

An example subclass implementation can be found in the unii data source for MyGene.info.

property client

```
create_todump_list(force=False, **kwargs)
```

This gets called by method `dump`, to populate `self.to_dump`

```
download(remote_file, localfile)
```

Runs helper function in new process to download data

This is run in a new process by the `do_dump` coroutine of the parent class. Then this will spawn another process that actually does all the work. This method is mostly for setting up the environment, setting up the the process pool executor to correctly use spawn and using concurrent.futures to simply run tasks in the new process, and periodically check the status of the task.

Explanation: because this is actually running inside a process that forked from a threaded process, the internal state is more or less corrupt/broken, see *man 2 fork* for details. More discussions are in Slack from some time in 2021 on why it has to be forked and why it is broken.

Caveats: the existing job manager will not know how much memory the actual worker process is using.

```
static get_document() → Generator[Tuple[str, Any], None, None]
```

Get document from API source

Populate this method to yield documents to be stored on disk. Every time you want to save something to disk, do this: `>>> yield 'name_of_file.ndjson', {'stuff': 'you want saved'}` While the type definition says Any is accepted, it has to be JSON serializable, so basically Python dictionaries/lists with strings and numbers as the most basic elements.

Later on in your uploader, you can treat the files as NDJSON documents, i.e. one JSON document per line.

It is recommended that you only do the minimal necessary processing in this step.

A default HTTP client is not provided so you get the flexibility of choosing your most favorite tool.

This MUST be a static method or it cannot be properly serialized to run in a separate process.

This method is expected to be blocking (synchronous). However, be sure to properly SET TIMEOUTS. You open the resources here in this function so you have to deal with properly checking/closing them. If the invoker forcefully stops this method, it will leave a mess behind, therefore we do not do that.

You can do a 5 second timeout using the popular requests package by doing something like this: >>> import requests >>> r = requests.get('https://example.org', timeout=5.0) You can catch the exception or setup retries. If you cannot handle the situation, just raise exceptions or not catch them. APIDumper will handle it properly: documents are only saved when the entire method completes successfully.

static get_release() → str

Get the string for the release information.

This is run in the main process and thread so it must return quickly. This must be populated

Returns

string representing the release.

need_prepare()

check whether some prepare step should be executed before running dump

prepare_client()

do initialization to make the client ready to dump files

release_client()

Do whatever necessary (like closing network connection) to “release” the client

remote_is_better(*remotefile*, *localfile*)

If there is a simple method to check whether remote is better

```
class biothings.hub.dataloader.dumper.BaseDumper(src_name=None, src_root_folder=None,
log_folder=None, archive=None)
```

Bases: object

ARCHIVE = True

AUTO_UPLOAD = True

MAX_PARALLEL_DUMP = None

SCHEDULE = None

SLEEP_BETWEEN_DOWNLOAD = 0.0

SRC_NAME = None

SRC_ROOT_FOLDER = None

SUFFIX_ATTR = 'release'

property client

create_todump_list(*force=False*, **kwargs)

Fill self.to_dump list with dict(“remote”:remote_path,”local”:local_path) elements. This is the todo list for the dumper. It’s a good place to check whether needs to be downloaded. If ‘force’ is True though, all files will be considered for download

property current_data_folder

```
property current_release
async do_dump(job_manager=None)
download(remotefile, localfile)
    Download “remotefile” to local location defined by ‘localfile’ Return relevant information about remotefile
    (depends on the actual client)
async dump(steps=None, force=False, job_manager=None, check_only=False, **kwargs)
    Dump (ie. download) resource as needed this should be called after instance creation ‘force’ argument
    will force dump, passing this to create_todump_list() method.
get_pinfo()
    Return dict containing information about the current process (used to report in the hub)
get_predicates()
    Return a list of predicates (functions returning true/false, as in math logic) which instructs/dictates if job
    manager should start a job (process/thread)
init_state()
property logger
mark_success(dry_run=True)
    Mark the datasource as successful dumped. It’s useful in case the datasource is unstable, and need to be
    manually downloaded.
need_prepare()
    check whether some prepare step should be executed before running dump
property new_data_folder
    Generate a new data folder path using src_root_folder and specified suffix attribute. Also sync current
    (aka previous) data folder previously registered in database. This method typically has to be called in
    create_todump_list() when the dumper actually knows some information about the resource, like the actual
    release.
post_download(remotefile, localfile)
    Placeholder to add a custom process once a file is downloaded. This is a good place to check file’s integrity.
    Optional
post_dump(*args, **kwargs)
    Placeholder to add a custom process once the whole resource has been dumped. Optional.
post_dump_delete_files()
    Delete files after dump
    Invoke this method in post_dump to synchronously delete the list of paths stored in self.to_delete, in order.
    Non-recursive. If directories need to be removed, build the list such that files residing in the directory are
    removed first and then the directory. (Hint: see os.walk(dir, topdown=False))
prepare(state={})
prepare_client()
    do initialization to make the client ready to dump files
prepare_local_folders(localfile)
prepare_src_dump()
```

```
register_status(status, transient=False, dry_run=False, **extra)
release_client()
    Do whatever necessary (like closing network connection) to “release” the client
remote_is_better(remotefile, localfile)
    Compared to local file, check if remote file is worth downloading. (like either bigger or newer for instance)
setup_log()
property src_doc
property src_dump
to_delete: List[str | bytes | PathLike]
    Populate with list of relative path of files to delete
unprepare()
    reset anything that's not pickleable (so self can be pickled) return what's been reset as a dict, so self can be restored once pickled
class biothings.hub.dataloader.DockerContainerDumper(*args, **kwargs)
Bases: BaseDumper
Start a docker container (typically runs on a different server) to prepare the data file on the remote container, and then download this file to the local data source folder. This dumper will do the following steps: - Booting up a container from provided parameters: image, tag, container_name. - The container entrypoint will be override by this long running command: “tail -f /dev/null” - When the container_name and image is provided together, the dumper will try to run the container_name.

If the container with container_name does not exist, the dumper will start a new container from image param, and set its name as container_name.



- Run the dump_command inside this container. This command MUST block the dumper until the data file is completely prepare.
  - It will guarantees that the remote file is ready for downloading.
- Run the get_version_cmd inside this container - if it provided. Set this command output as self.release.
- Download the remote file via Docker API, extract the downloaded .tar file.
- When the downloading is complete:
  - if keep_container=false: Remove the above container after.
  - if keep_container=true: leave this container running.
- If there are any exception when dump data, the remote container won't be removed, it will help us address the problem.

```

These are supported connection types from the Hub server to the remote Docker host server:

- ssh: Prerequisite: the SSH Key-Based Authentication is configured
- unix: Local connection
- http: Use an insecure HTTP connection over a TCP socket
- **https: Use a secured HTTPS connection using TLS. Prerequisite:**
 - The Docker API on the remote server MUST BE secured with TLS

- A TLS key pair is generated on the Hub server and placed inside the same data plugin folder or the data source folder

All info about Docker client connection MUST BE defined in the *config.py* file, under the DOCKER_CONFIG key, Ex Optional DOCKER_HOST can be used to override the docker connections in any docker dumper regardless of the value of the src_url For example, you can set DOCKER_HOST="localhost" for local testing:

```
DOCKER_CONFIG = {  
    "CONNECTION_NAME_1": {  
        "tls_cert_path": "/path/to/cert.pem", "tls_key_path": "/path/to/key.pem", "client_url":  
        "https://remote-docker-host:port"  
    }, "CONNECTION_NAME_2": {  
        "client_url": "ssh://user@remote-docker-host"  
    }, "localhost": {  
        "client_url": "unix://var/run/docker.sock"  
    }  
}  
DOCKER_HOST = "localhost"
```

The data_url should match the following format:

```
docker://CONNECTION_NAME?image=DOCKER_IMAGE&tag=TAG&path=/path/to/remote_file&dump_command="the  
is custom command"&container_name=CONTAINER_NAME&keep_container=true&get_version_cmd="cmd"  
# NOQA
```

Supported params:

- image: (Optional) the Docker image name
- tag: (Optional) the image tag
- container_name: (Optional) If this param is provided, the *image* param will be discard when dumper run.
- path: (Required) path to the remote file inside the Docker container.
- dump_command: (Required) This command will be run inside the Docker container in order to create the remote file.
- keep_container: (Optional) accepted values: true/false, default: false.
 - If keep_container=true, the remote container will be persisted.
 - If keep_container=false, the remote container will be removed in the end of dump step.
- get_version_cmd: (Optional) The custom command for checking release version of local and remote file. Note that:
 - This command must run-able in both local Hub (for checking local file) and remote container (for checking remote file).
 - “{}” MUST exists in the command, it will be replace by the data file path when dumper runs,
ex: get_version_cmd="md5sum {} | awk '{ print \$1 }'" will be run as: md5sum /path/to/remote_file | awk '{ print \$1 }' and /path/to/local_file

Ex:

- docker://CONNECTION_NAME?image=IMAGE_NAME&tag=IMAGE_TAG&path=/path/to/remote_file(inside the container)&dump_command="run something with output is written to -O /path/to/remote_file (inside the container)" # NOQA
- docker://CONNECTION_NAME?container_name=CONTAINER_NAME&path=/path/to/remote_file(inside the container)&dump_command="run something with output is written to -O /path/to/remote_file (inside the container)&keep_container=true&get_version_cmd="md5sum {} | awk '{ print \$1 }'" # NOQA
- docker://localhost?image=dockstore_dumper&path=/data/dockstore_crawled/data.ndjson&dump_command="/home/dockstore.sh"&keep_container=1
- docker://localhost?image=dockstore_dumper&tag=latest&path=/data/dockstore_crawled/data.ndjson&dump_command=dockstore.sh"&keep_container=True # NOQA
- docker://localhost?image=praqla/network-multitool&tag=latest&path=/tmp/annotations.zip&dump_command="/usr/bin/wget https://s3.pgkb.org/data/annotations.zip -O /tmp/annotations.zip"&keep_container=false&get_version_cmd="md5sum {} | awk '{ print \$1 }'" # NOQA
- docker://localhost?container_name=<YOUR CONTAINER NAME>&path=/tmp/annotations.zip&dump_command="https://s3.pgkb.org/data/annotations.zip -O /tmp/annotations.zip"&keep_container=true&get_version_cmd="md5sum {} | awk '{ print \$1 }'" # NOQA

Container metadata: - All above params in the data_url can be pre-config in the Dockerfile by adding LABELs. This config will be used as the fallback of the data_url params:

The dumper will find those params from both data_url and container metadata. If a param does not exist in data_url, dumper will use its value from container metadata (of course if it exists).

For example:

```
... Dockerfile LABEL "path"="/tmp/annotations.zip" LABEL "dump_command"="/usr/bin/wget https://s3.pgkb.org/data/annotations.zip -O /tmp/annotations.zip" LABEL keep_container="true" LABEL desc=test LABEL container_name=mydocker
```

COUNTAINER_NAME = None

DATA_PATH = None

DOCKER_CLIENT_URL = None

DOCKER_IMAGE = None

DUMP_COMMAND = None

GET_VERSION_CMD = None

KEEP_CONTAINER = False

MAX_PARALLEL_DUMP = 1

ORIGINAL_CONTAINER_STATUS = None

TIMEOUT = 300

async create_todump_list(force=False, job_manager=None, **kwargs)

Create the list of files to dump, called in dump method. This method will execute dump_command to generate the remote file in docker container, so we define this method as async to make it non-blocking.

delete_or_restore_container()

Delete the container if it's created by the dumper, or restore it to its original status if it's pre-existing.

download(*remote_file*, *local_file*)

Download “remotefile” to local location defined by ‘localfile’ Return relevant information about remotefile (depends on the actual client)

generate_remote_file()

Execute dump_command to generate the remote file, called in create_todump_list method

get_remote_file()

return the remote file path within the container. In most of cases, dump_command should either generate this file or check if it's ready if there is another automated pipeline generates this file.

get_remote_lastmodified(*remote_file*)

get the last modified time of the remote file within the container using stat command

need_prepare()

check whether some prepare step should be executed before running dump

post_dump(*args, **kwargs)

Delete container or restore the container status if necessary, called in the dump method after the dump is done (during the “post” step)

prepare_client()

do initialization to make the client ready to dump files

prepare_dumper_params()

Read all docker dumper parameters from either the data plugin manifest or the Docker image or container metadata. Of course, at least one of docker_image or container_name parameters must be defined in the data plugin manifest first. If the parameter is not defined in the data plugin manifest, we will try to read it from the Docker image metadata.

prepare_local_folders(*localfile*)

prepare the local folder for the localfile, called in download method

prepare_remote_container()

prepare the remote container and set self.container, called in create_todump_list method

release_client()

Release the docker client connection, called in dump method

remote_is_better(*remote_file*, *local_file*)

Compared to local file, check if remote file is worth downloading. (like either bigger or newer for instance)

set_data_path()**set_dump_command()****set_get_version_cmd()****set_keep_container()****set_release()**

call the get_version_cmd to get the releases, called in get_todump_list method. if get_version_cmd is not defined, use timestamp as the release.

This is currently a blocking method, assuming get_version_cmd is a quick command. But if necessary, we can make it async in the future.

```

property source_config

exception biothings.hub.dataloader.DockerContainerException
    Bases: Exception

class biothings.hub.dataloader.dumper.DummyDumper(*args, **kwargs)
    Bases: BaseDumper
    DummyDumper will do nothing... (useful for datasources that can't be downloaded anymore but still need to be integrated, ie. fill src_dump, etc...)
    async dump(force=False, job_manager=None, *args, **kwargs)
        Dump (ie. download) resource as needed this should be called after instance creation 'force' argument will force dump, passing this to create_todump_list() method.

    prepare_client()
        do initialization to make the client ready to dump files

exception biothings.hub.dataloader.DumperException
    Bases: Exception

class biothings.hub.dataloader.dumper.DumperManager(job_manager,
                                                       datasource_path='dataload.sources', *args,
                                                       **kwargs)
    Bases: BaseSourceManager

SOURCE_CLASS
    alias of BaseDumper

call(src, method_name, *args, **kwargs)
    Create a dumper for datasource "src" and call method "method_name" on it, with given arguments. Used to create arbitrary calls on a dumper. "method_name" within dumper definition must a coroutine.

clean_stale_status()
    During startup, search for action in progress which would have been interrupted and change the state to "canceled". Ex: some downloading processes could have been interrupted, at startup, "downloading" status should be changed to "canceled" so to reflect actual state on these datasources. This must be overridden in subclass.

async create_and_call(klass, method_name, *args, **kwargs)

async create_and_dump(klass, *args, **kwargs)

create_instance(klass)

dump_all(force=False, **kwargs)
    Run all dumper, except manual ones

dump_info()

dump_src(src, force=False, skip_manual=False, schedule=False, check_only=False, **kwargs)

get_schedule(dumper_name)
    Return the corresponding schedule for dumper_name Example result: {
        "cron": "0 9 * * *",
        "strdelta": "15h:20m:33s",
    }

```

```
get_source_ids()
    Return displayable list of registered source names (not private)

mark_success(src, dry_run=True)

register_classes(klasses)
    Register each class in self.register dict. Key will be used to retrieve the source class, create an instance and run method from it. It must be implemented in subclass as each manager may need to access its sources differently,based on different keys.

schedule_all(raise_on_error=False, **kwargs)
    Run all dumper, except manual ones

source_info(source=None)

class biothings.hub.dataload.dumper.FTPDumper(src_name=None, src_root_folder=None,
                                               log_folder=None, archive=None)
    Bases: BaseDumper

    BLOCK_SIZE: int | None = None
    CWD_DIR = ''
    FTP_HOST = ''
    FTP_PASSWD = ''
    FTP_TIMEOUT = 600.0
    FTP_USER = ''

    download(remotefile, localfile)
        Download “remotefile” to local location defined by ‘localfile’ Return relevant information about remotefile (depends on the actual client)

    need_prepare()
        check whether some prepare step should executed before running dump

    prepare_client()
        do initialization to make the client ready to dump files

    release_client()
        Do whatever necessary (like closing network connection) to “release” the client

    remote_is_better(remotefile, localfile)
        ‘remotefile’ is relative path from current working dir (CWD_DIR), ‘localfile’ is absolute path

class biothings.hub.dataload.dumper.FilesystemDumper(src_name=None, src_root_folder=None,
                                                       log_folder=None, archive=None)
    Bases: BaseDumper

    This dumpers works locally and copy (or move) files to datasource folder

    FS_OP = 'cp'

    download(remotefile, localfile)
        Download “remotefile” to local location defined by ‘localfile’ Return relevant information about remotefile (depends on the actual client)
```

need_prepare()
check whether some prepare step should be executed before running dump

prepare_client()
Check if ‘cp’ and ‘mv’ executable exists...

release_client()
Do whatever necessary (like closing network connection) to “release” the client

remote_is_better(*remotefile*, *localfile*)
Compared to local file, check if remote file is worth downloading. (like either bigger or newer for instance)

class biothings.hub.dataloader.dumper.GitDumper(*src_name=None*, *src_root_folder=None*, *log_folder=None*, *archive=None*)

Bases: *BaseDumper*

Git dumper gets data from a git repo. Repo is stored in SRC_ROOT_FOLDER (without versioning) and then versions/releases are fetched in SRC_ROOT_FOLDER/<release>

DEFAULT_BRANCH = None

GIT_REPO_URL = None

download(*remotefile*, *localfile*)
Download “*remotefile*” to local location defined by ‘*localfile*’ Return relevant information about *remotefile* (depends on the actual client)

async dump(*release='HEAD'*, *force=False*, *job_manager=None*, *kwargs*)**
Dump (ie. download) resource as needed this should be called after instance creation ‘*force*’ argument will force dump, passing this to *create_todump_list()* method.

need_prepare()
check whether some prepare step should be executed before running dump

property new_data_folder
Generate a new data folder path using *src_root_folder* and specified suffix attribute. Also sync current (aka previous) data folder previously registered in database. This method typically has to be called in *create_todump_list()* when the dumper actually knows some information about the resource, like the actual release.

prepare_client()
Check if ‘git’ executable exists

release_client()
Do whatever necessary (like closing network connection) to “release” the client

remote_is_better(*remotefile*, *localfile*)
Compared to local file, check if remote file is worth downloading. (like either bigger or newer for instance)

class biothings.hub.dataloader.dumper.GoogleDriveDumper(*src_name=None*, *src_root_folder=None*, *log_folder=None*, *archive=None*)

Bases: *HTTPDumper*

download(*remoteurl*, *localfile*)

remoteurl is a google drive link containing a document ID, such as:

- <https://drive.google.com/open?id=<1234567890ABCDEF>>
- <https://drive.google.com/file/d/<1234567890ABCDEF>/view>

It can also be just a document ID

get_document_id(url)

prepare_client()

do initialization to make the client ready to dump files

remote_is_better(remote_file, local_file)

Determine if remote is better

Override if necessary.

```
class biothings.hub.dataload.dumper.HTTPDumper(src_name=None, src_root_folder=None,  
                                              log_folder=None, archive=None)
```

Bases: *BaseDumper*

Dumper using HTTP protocol and “requests” library

IGNORE_HTTP_CODE = []

RESOLVE_FILENAME = False

VERIFY_CERT = True

download(remoteurl, localfile, headers={})

Download ‘remoteurl’ to local location defined by ‘localfile’ Return relevant information about remotefile
(depends on the actual client)

need_prepare()

check whether some prepare step should be executed before running dump

prepare_client()

do initialization to make the client ready to dump files

release_client()

Do whatever necessary (like closing network connection) to “release” the client

remote_is_better(remote_file, local_file)

Determine if remote is better

Override if necessary.

```
class biothings.hub.dataload.dumper.LastModifiedBaseDumper(src_name=None,  
                                                          src_root_folder=None,  
                                                          log_folder=None, archive=None)
```

Bases: *BaseDumper*

Use SRC_URLS as a list of URLs to download and implement create_todump_list() according to that list. Should be used in parallel with a dumper talking the actual underlying protocol

SRC_URLS = []

create_todump_list(force=False)

Fill self.to_dump list with dict(“remote”:remote_path,”local”:local_path) elements. This is the todo list for the dumper. It’s a good place to check whether needs to be downloaded. If ‘force’ is True though, all files will be considered for download

set_release()

Set self.release attribute as the last-modified datetime found in the last SRC_URLS element (so release is the datetime of the last file to download)

```
class biothings.hub.dataload.dumper.LastModifiedFTPDumper(src_name=None, src_root_folder=None, log_folder=None, archive=None)
```

Bases: *LastModifiedBaseDumper*

SRC_URLS containing a list of URLs pointing to files to download, use FTP's MDTM command to check whether files should be downloaded. The release is generated from the last file's MDTM in SRC_URLS, and formatted according to RELEASE_FORMAT. See also LastModifiedHTTPDumper, working the same way but for HTTP protocol. Note: this dumper is a wrapper over FTPDumper, one URL will give one FTPDumper instance.

```
RELEASE_FORMAT = '%Y-%m-%d'
```

```
download(urlremotefile, localfile, headers={})
```

Download “remotefile” to local location defined by ‘localfile’ Return relevant information about remotefile (depends on the actual client)

```
get_client_for_url(url)
```

```
get_remote_file(url)
```

```
prepare_client()
```

do initialization to make the client ready to dump files

```
release_client()
```

Do whatever necessary (like closing network connection) to “release” the client

```
remote_is_better(urlremotefile, localfile)
```

Compared to local file, check if remote file is worth downloading. (like either bigger or newer for instance)

```
set_release()
```

Set self.release attribute as the last-modified datetime found in the last SRC_URLS element (so release is the datetime of the last file to download)

```
class biothings.hub.dataload.dumper.LastModifiedHTTPDumper(src_name=None, src_root_folder=None, log_folder=None, archive=None)
```

Bases: *HTTPDumper*, *LastModifiedBaseDumper*

Given a list of URLs, check Last-Modified header to see whether the file should be downloaded. Sub-class should only have to declare SRC_URLS. Optionally, another field name can be used instead of Last-Modified, but date format must follow RFC 2616. If that header doesn't exist, it will always download the data (bypass) The release is generated from the last file's Last-Modified in SRC_URLS, and formatted according to RELEASE_FORMAT.

```
ETAG = 'ETag'
```

```
LAST_MODIFIED = 'Last-Modified'
```

```
RELEASE_FORMAT = '%Y-%m-%d'
```

```
RESOLVE_FILENAME = True
```

```
remote_is_better(remotefile, localfile)
```

Determine if remote is better

Override if necessary.

set_release()

Set self.release attribute as the last-modified datetime found in the last SRC_URLs element (so release is the datetime of the last file to download)

class biothings.hub.dataloader.dumper.ManualDumper(*args, **kwargs)

Bases: *BaseDumper*

This dumper will assist user to dump a resource. It will usually expect the files to be downloaded first (sometimes there's no easy way to automate this process). Once downloaded, a call to dump() will make sure everything is fine in terms of files and metadata

async dump(path, release=None, force=False, job_manager=None, **kwargs)

Dump (ie. download) resource as needed this should be called after instance creation ‘force’ argument will force dump, passing this to create_todump_list() method.

property new_data_folder

Generate a new data folder path using src_root_folder and specified suffix attribute. Also sync current (aka previous) data folder previously registered in database. This method typically has to be called in create_todump_list() when the dumper actually knows some information about the resource, like the actual release.

prepare(state={})**prepare_client()**

do initialization to make the client ready to dump files

class biothings.hub.dataloader.dumper.WgetDumper(src_name=None, src_root_folder=None, log_folder=None, archive=None)

Bases: *BaseDumper*

create_todump_list(force=False, **kwargs)

Fill self.to_dump list with dict(“remote”:remote_path,”local”:local_path) elements. This is the todo list for the dumper. It’s a good place to check whether needs to be downloaded. If ‘force’ is True though, all files will be considered for download

download(remoteurl, localfile)

Download “remotefile” to local location defined by ‘localfile’ Return relevant information about remotefile (depends on the actual client)

need_prepare()

check whether some prepare step should be executed before running dump

prepare_client()

Check if ‘wget’ executable exists

release_client()

Do whatever necessary (like closing network connection) to “release” the client

remote_is_better(remote_file, localfile)

Compared to local file, check if remote file is worth downloading. (like either bigger or newer for instance)

biothings.hub.dataloader.source

```
class biothings.hub.dataloader.source.SourceManager(source_list, dump_manager, upload_manager,
data_plugin_manager)
```

Bases: *BaseSourceManager*

Helper class to get information about a datasource, whether it has a dumper and/or uploaders associated.

find_sources(*paths*)

get_source(*name, debug=False*)

get_sources(*id=None, debug=False, detailed=False*)

reload()

reset(*name, key='upload', subkey=None*)

Reset, ie. delete, internal data (src_dump document) for given source name, key subkey. This method is useful to clean outdated information in Hub’s internal database.

Ex: key=upload, name=mysource, subkey=mysubsource, will delete entry in corresponding src_dump doc (_id=mysource), under key “upload”, for sub-source named “mysubsource”

“key” can be either ‘download’, ‘upload’ or ‘inspect’. Because there’s no such notion of subkey for dumpers (ie. ‘download’, subkey is optional).

save_mapping(*name, mapping=None, dest='master', mode='mapping'*)

set_mapping_src_meta(*subsrc, mini*)

sumup_source(*src, detailed=False*)

Return minimal info about src

biothings.hub.dataloader.storage**biothings.hub.dataloader.sync**

Deprecated. This module is not used any more.

```
class biothings.hub.dataloader.sync.MongoSync
```

Bases: *object*

add_update(*source, merge_collection, ids*)

delete(*merge_collection, field, ids*)

main(*diff_filepath, merge_collection, field*)

biothings.hub.dataloader.uploader

```
class biothings.hub.dataloader.uploader.BaseSourceUploader(db_conn_info, collection_name=None,  
log_folder=None, *args, **kwargs)
```

Bases: object

Default datasource uploader. Database storage can be done in batch or line by line. Duplicated records aren't not allowed

db_conn_info is a database connection info tuple (host,port) to fetch/store information about the datasource's state.

check_ready(force=False)

clean_archived_collections()

```
config = <ConfigurationWrapper over <module 'config' from  
'/home/docs/checkouts/readthedocs.org/user_builds/biothingsapi/checkouts/latest/  
biothings/hub/default_config.py'>>
```

classmethod create(db_conn_info, *args, **kwargs)

Factory-like method, just return an instance of this uploader (used by SourceManager, may be overridden in sub-class to generate more than one instance per class, like a true factory. This is usefull when a resource is splitted in different collection but the data structure doesn't change (it's really just data splitted accros multiple collections, usually for parallelization purposes). Instead of having actual class for each split collection, factory will generate them on-the-fly.

property fullname

generate_doc_src_master()

get_current_and_new_master()

classmethod get_mapping()

Return ES mapping

get_pinfo()

Return dict containing information about the current process (used to report in the hub)

get_predicates()

Return a list of predicates (functions returning true/false, as in math logic) which instructs/dictates if job manager should start a job (process/thread)

init_state()

keep_archive = 10

```
async load(steps=('data', 'post', 'master', 'clean'), force=False, batch_size=10000, job_manager=None,  
**kwargs)
```

Main resource load process, reads data from doc_c using chunk sized as batch_size. steps defines the different processes used to laod the resource: - “data” : will store actual data into single collections - “post” : will perform post data load operations - “master” : will register the master document in src_master

load_data(data_path)

Parse data from data_path and return structure ready to be inserted in database In general, data_path is a folder path. But in parallel mode (use parallelizer option), data_path is a file path :param data_path: It can be a folder path or a file path :return: structure ready to be inserted in database

```

main_source = None
make_temp_collection()
    Create a temp collection for dataloading, e.g., entrez_geneinfo_INEMO.

name = None
post_update_data(steps, force, batch_size, job_manager, **kwargs)
    Override as needed to perform operations after data has been uploaded

prepare(state={})
    Sync uploader information with database (or given state dict)

prepare_src_dump()
    Sync with src_dump collection, collection information (src_doc) Return src_dump collection

regex_name = None
register_status(status, subkey='upload', **extra)
    Register step status, ie. status for a sub-resource

save_doc_src_master(_doc)

setup_log()

storage_class
    alias of BasicStorage

switch_collection()
    after a successful loading, rename temp_collection to regular collection name, and renaming existing collection to a temp name for archiving purpose.

unprepare()
    reset anything that's not pickleable (so self can be pickled) return what's been reset as a dict, so self can be restored once pickled

async update_data(batch_size, job_manager)
    Iterate over load_data() to pull data and store it

update_master()

class biothings.hub.dataloader.uploader.DummySourceUploader(db_conn_info, collection_name=None, log_folder=None, *args, **kwargs)
    Bases: BaseSourceUploader

    Dummy uploader, won't upload any data, assuming data is already there but make sure every other bit of information is there for the overall process (usefull when online data isn't available anymore)

    db_conn_info is a database connection info tuple (host,port) to fetch/store information about the datasource's state.

check_ready(force=False)

prepare_src_dump()
    Sync with src_dump collection, collection information (src_doc) Return src_dump collection

async update_data(batch_size, job_manager=None, release=None)
    Iterate over load_data() to pull data and store it

```

```
class biothings.hub.dataload.uploader.IgnoreDuplicatedSourceUploader(db_conn_info,
                                                                      collection_name=None,
                                                                      log_folder=None, *args,
                                                                      **kwargs)
```

Bases: *BaseSourceUploader*

Same as default uploader, but will store records and ignore if any duplicated error occurring (use with caution...). Storage is done using batch and unordered bulk operations.

db_conn_info is a database connection info tuple (host,port) to fetch/store information about the datasource's state.

storage_class

alias of `IgnoreDuplicatedStorage`

```
class biothings.hub.dataload.uploader.MergerSourceUploader(db_conn_info, collection_name=None,
                                                               log_folder=None, *args, **kwargs)
```

Bases: *BaseSourceUploader*

db_conn_info is a database connection info tuple (host,port) to fetch/store information about the datasource's state.

storage_class

alias of `MergerStorage`

```
class biothings.hub.dataload.uploader.NoBatchIgnoreDuplicatedSourceUploader(db_conn_info,
                                                                           collection_name=None,
                                                                           log_folder=None,
                                                                           *args,
                                                                           **kwargs)
```

Bases: *BaseSourceUploader*

Same as default uploader, but will store records and ignore if any duplicated error occurring (use with caution...). Storage is done line by line (slow, not using a batch) but preserve order of data in input file.

db_conn_info is a database connection info tuple (host,port) to fetch/store information about the datasource's state.

storage_class

alias of `NoBatchIgnoreDuplicatedStorage`

```
class biothings.hub.dataload.uploader.NoDataSourceUploader(db_conn_info, collection_name=None,
                                                               log_folder=None, *args, **kwargs)
```

Bases: *BaseSourceUploader*

This uploader won't upload any data and won't even assume there's actual data (different from `DummySourceUploader` on this point). It's useful for instance when mapping need to be stored (`get_mapping()`) but data doesn't come from an actual upload (ie. generated)

db_conn_info is a database connection info tuple (host,port) to fetch/store information about the datasource's state.

storage_class

alias of `NoStorage`

async update_data(batch_size, job_manager=None)

Iterate over `load_data()` to pull data and store it

```
class biothings.hub.dataloader.uploader.ParallelizedSourceUploader(db_conn_info,  
                                collection_name=None,  
                                log_folder=None, *args,  
                                **kwargs)
```

Bases: *BaseSourceUploader*

db_conn_info is a database connection info tuple (host,port) to fetch/store information about the datasource's state.

jobs()

Return list of (**arguments*) passed to self.load_data, in order. for each parallelized jobs. Ex:
[(x,1),(y,2),(z,3)] If only one argument is required, it still must be passed as a 1-element tuple

async update_data(*batch_size*, *job_manager=None*)

Iterate over load_data() to pull data and store it

exception biothings.hub.dataloader.uploader.ResourceError

Bases: Exception

exception biothings.hub.dataloader.uploader.ResourceNotReady

Bases: Exception

class biothings.hub.dataloader.UploaderManager(*poll_schedule=None*, **args*, ***kwargs*)

Bases: *BaseSourceManager*

After registering datasources, manager will orchestrate source uploading.

SOURCE_CLASS

alias of *BaseSourceUploader*

clean_stale_status()

During startup, search for action in progress which would have been interrupted and change the state to “canceled”. Ex: some downloading processes could have been interrupted, at startup, “downloading” status should be changed to “canceled” so to reflect actual state on these datasources. This must be overridden in subclass.

async create_and_load(*klass*, **args*, *kwargs*)**

async create_and_update_master(*klass*, *dry=False*)

create_instance(*klass*)

filter_class(*klass*)

Gives opportunity for subclass to check given class and decide to keep it or not in the discovery process.
Returning None means “skip it”.

get_source_ids()

Return displayable list of registered source names (not private)

poll(*state*, *func*)

Search for source in collection ‘col’ with a pending flag list containing ‘state’ and and call ‘func’ for each document found (with doc as only param)

register_classes(*klasses*)

Register each class in self.register dict. Key will be used to retrieve the source class, create an instance and run method from it. It must be implemented in subclass as each manager may need to access its sources differently,based on different keys.

```
source_info(source=None)
update_source_meta(src, dry=False)
    Trigger update for registered resource named 'src'.
upload_all(raise_on_error=False, **kwargs)
    Trigger upload processes for all registered resources. **kwargs are passed to upload_src() method
upload_info()
upload_src(src, *args, **kwargs)
    Trigger upload for registered resource named 'src'. Other args are passed to uploader's load() method
biothings.hub.dataloader.uploader.set_pending_to_upload(src_name)
```

biothings.hub.dataloader.validator

biothings.hub.dataplugin

biothings.hub.dataplugin.assistant

```
class biothings.hub.dataplugin.assistant.AdvancedPluginLoader(plugin_name)
    Bases: BasePluginLoader
    can_load_plugin()
        Return True if loader is able to load plugin (check data folder content)
    load_plugin()
        Load plugin and register its components
    loader_type = 'advanced'

exception biothings.hub.dataplugin.assistant.AssistantException
    Bases: Exception

class biothings.hub.dataplugin.assistant.AssistantManager(data_plugin_manager,
                                                          dumper_manager, uploader_manager,
                                                          keylookup=None, de-
                                                          fault_export_folder='hub/dataloader/sources',
                                                          *args, **kwargs)
    Bases: BaseSourceManager
    configure(klasses=[<class 'biothings.hub.dataplugin.assistant.GithubAssistant'>, <class
                      'biothings.hub.dataplugin.assistant.LocalAssistant'>])
    create_instance(klass, url)
    export(plugin_name, folder=None, what=['dumper', 'uploader', 'mapping'], purge=False)
        Export generated code for a given plugin name, in given folder (or use DEFAULT_EXPORT_FOLDER
        if None). Exported information can be: - dumper: dumper class generated from the manifest - uploader:
        uploader class generated from the manifest - mapping: mapping generated from inspection or from the
        manifest If "purge" is true, any existing folder/code will be deleted first, otherwise, will raise an error if
        some folder/files already exist.
    export_dumper(plugin_name, folder)
```

```

export_mapping(plugin_name, folder)
export_uploader(plugin_name, folder)
load(autodiscover=True)
    Load plugins registered in internal Hub database and generate/register dumpers & uploaders accordingly. If autodiscover is True, also search DATA_PLUGIN_FOLDER for existing plugin directories not registered yet in the database, and register them automatically.
load_plugin(plugin)
register_classes(klasses)
    Register each class in self.register dict. Key will be used to retrieve the source class, create an instance and run method from it. It must be implemented in subclass as each manager may need to access its sources differently,based on different keys.
register_url(url)
setup_log()
    Setup and return a logger instance
submit(url)
unregister_url(url=None, name=None)
update_plugin_name(plugin, new_name)

class biothings.hub.dataplugin.assistant.AssistedDumper
    Bases: object
    DATA_PLUGIN_FOLDER = None

class biothings.hub.dataplugin.assistant.AssistedUploader
    Bases: object
    DATA_PLUGIN_FOLDER = None

class biothings.hub.dataplugin.assistant.BaseAssistant(url)
    Bases: object
    can_handle()
        Return true if assistant can handle the code
    data_plugin_manager = None
    dumper_manager = None
    handle()
        Access self.url and do whatever is necessary to bring code to life within the hub... (hint: that may involve creating a dumper on-the-fly and register that dumper to a manager...)
    keylookup = None
    load_plugin()
        Load plugin and register its components
    property loader
        Return loader object able to interpret plugin's folder content

```

```
loaders = {'advanced': <class
'biothings.hub.dataplugin.assistant.AdvancedPluginLoader'>, 'manifest': <class
'biothings.hub.dataplugin.assistant.ManifestBasedPluginLoader'>}

property plugin_name
    Return plugin name, parsed from self.url and set self._src_folder as path to folder containing dataplug-in
    source code

plugin_type = None

register_loader()

setup_log()
    Setup and return a logger instance

uploader_manager = None

class biothings.hub.dataplugin.assistant.BasePluginLoader(plugin_name)
    Bases: object

    can_load_plugin()
        Return True if loader is able to load plugin (check data folder content)

    get_plugin_obj()

    invalidate_plugin(error)

    load_plugin()
        Load plugin and register its components

    loader_type = None

    setup_log()
        Setup and return a logger instance

class biothings.hub.dataplugin.assistant.GithubAssistant(url)
    Bases: BaseAssistant

    can_handle()
        Return true if assistant can handle the code

    get_classdef()

    handle()
        Access self.url and do whatever is necessary to bring code to life within the hub... (hint: that may involve
        creating a dumper on-the-fly and register that dumper to a manager...)

    property plugin_name
        Return plugin name, parsed from self.url and set self._src_folder as path to folder containing dataplug-in
        source code

    plugin_type = 'github'

exception biothings.hub.dataplugin.assistant.LoaderException
    Bases: Exception

class biothings.hub.dataplugin.assistant.LocalAssistant(url)
    Bases: BaseAssistant
```

```

can_handle()
    Return true if assistant can handle the code

get_classdef()

handle()
    Access self.url and do whatever is necessary to bring code to life within the hub... (hint: that may involve
    creating a dumper on-the-fly and register that dumper to a manager...)

property plugin_name
    Return plugin name, parsed from self.url and set self._src_folder as path to folder containing dataplug-in
    source code

plugin_type = 'local'

class biothings.hub.dataplug-in.assistant.ManifestBasedPluginLoader(plugin_name)
    Bases: BasePluginLoader

can_load_plugin()
    Return True if loader is able to load plugin (check data folder content)

    dumper_registry = {'docker': <class
    'biothings.hub.dataload.dumper.DockerContainerDumper', 'ftp': <class
    'biothings.hub.dataload.dumper.LastModifiedFTPDumper', 'http': <class
    'biothings.hub.dataload.dumper.LastModifiedHTTPDumper', 'https': <class
    'biothings.hub.dataload.dumper.LastModifiedHTTPDumper'>}

get_code_for_mod_name(mod_name)
    Returns string literal and name of function, given a path

    Parameters
        mod_name – string with module name and function name, separated by colon

    Returns

        containing
            • indented string literal for the function specified
            • name of the function

    Return type
        Tuple[str, str]

get_dumper_dynamic_class(dumper_section, metadata)
get_uploader_dynamic_class(uploader_section, metadata, sub_source_name="")
get_uploader_dynamic_classes(uploader_section, metadata, data_plugin_folder)
interpret_manifest(manifest, data_plugin_folder)

load_plugin()
    Load plugin and register its components

loader_type = 'manifest'

```

biothings.hub.dataplugin.manager

```
class biothings.hub.dataplugin.manager.DataPluginManager(job_manager,
                                                       datasource_path='dataload.sources',
                                                       *args, **kwargs)

Bases: DumperManager

load(plugin_name, *args, **kwargs)

class biothings.hub.dataplugin.manager.GitDataPlugin(src_name=None, src_root_folder=None,
                                                    log_folder=None, archive=None)

Bases: GitDumper

prepare_src_dump()

class biothings.hub.dataplugin.manager.ManualDataPlugin(*args, **kwargs)

Bases: ManualDumper

async dump(*args, **kwargs)
    Dump (ie. download) resource as needed this should be called after instance creation 'force' argument
    will force dump, passing this to create_todump_list() method.

prepare_src_dump()
```

biothings.hub.datarelease

```
biothings.hub.datarelease.set_pending_to_publish(col_name)

biothings.hub.datarelease.set_pending_to_release_note(col_name)
```

biothings.hub.datarelease.publisher

```
class biothings.hub.datarelease.publisher.BasePublisher(envconf, log_folder, es_backups_folder,
                                                       *args, **kwargs)
```

Bases: BaseManager, BaseStatusRegisterer

property category

clean_stale_status()

During startup, search for action in progress which would have been interrupted and change the state to “canceled”. Ex: some downloading processes could have been interrupted, at startup, “downloading” status should be changed to “canceled” so to reflect actual state on these datasources. This must be overridden in subclass.

property collection

Return collection object used to fetch doc in which we store status

create_bucket(bucket_conf, credentials)

get_pinfo()

Return dict containing information about the current process (used to report in the hub)

```

get_pre_post_previous_result(build_doc, key_value)
    In order to start a pre- or post- pipeline, a first previous result, fed all along the pipeline to the next step, has to be defined, and depends on the type of publisher.

get_predicates()

get_release_note_filename(build_version)

load_build(key_name, stage=None)

publish_release_notes(release_folder, build_version, s3_release_folder, s3_release_bucket, aws_key, aws_secret, prefix='release_')

register_status(bdoc, status, transient=False, init=False, **extra)

run_pre_post(key, stage, key_value, repo_conf, build_doc)
    Run pre- and post- publish steps (stage) for given key (eg. "snapshot", "diff"). key_value is the value of the key inside "key" dict (such as a snapshot name or a build name) These steps are defined in config file.

setup()

setup_log(build_name=None)

step_archive(step_conf, build_doc, previous)

step_upload(step_conf, build_doc, previous)

step_upload_s3(step_conf, build_doc, previous)

template_out_conf(build_doc)

trigger_release_note(doc, **kwargs)
    Launch a release note generation given a src_build document. In order to know the first collection to compare with, get_previous_collection() method is used. release_note() method will get **kwargs for more optional parameters.

class biothings.hub.datarelease.publisher.DiffPublisher(diff_manager, *args, **kwargs)
    Bases: BasePublisher

get_pre_post_previous_result(build_doc, key_value)
    In order to start a pre- or post- pipeline, a first previous result, fed all along the pipeline to the next step, has to be defined, and depends on the type of publisher.

get_release_note_filename(build_version)
    Post-publish hook, running steps declared in config, but also whatever would be defined in a sub-class

post_publish(build_name, repo_conf, build_doc)
    Pre-publish hook, running steps declared in config, but also whatever would be defined in a sub-class

pre_publish(previous_build_name, repo_conf, build_doc)
    Publish diff files and metadata about the diff files, release note, etc... on s3. Using build_name, a src_build document is fetched, and a diff release is searched. If more than one diff release is found, "previous_build" must be specified to pick the correct one. - steps:
        • pre/post: optional steps processed as first and last steps.
        • reset: highly recommended, reset synced flag in diff files so they won't get skipped when used...

```

- upload: upload diff_folder content to S3
- meta: publish/register the version as available for auto-updating hubs

reset_synced(diff_folder, backend=None)

Remove “synced” flag from any pyobj file in diff_folder

run_post_publish_diff(build_name, repo_conf, build_doc)

run_pre_publish_diff(previous_build_name, repo_conf, build_doc)

exception biothings.hub.datarelease.publisher.PublisherException

Bases: Exception

class biothings.hub.datarelease.publisher.ReleaseManager(*diff_manager, snapshot_manager, poll_schedule=None, *args, **kwargs*)

Bases: *BaseManager, BaseStatusRegisterer*

DEFAULT_DIFF_PUBLISHER_CLASS

alias of *DiffPublisher*

DEFAULT_SNAPSHOT_PUBLISHER_CLASS

alias of *SnapshotPublisher*

build_release_note(old_colname, new_colname, note=None) → ReleaseNoteSource

Build a release note containing most significant changes between build names “old_colname” and “new_colname”. An optional end note can be added to bring more specific information about the release.

Return a dictionary containing significant changes.

clean_stale_status()

During startup, search for action in progress which would have been interrupted and change the state to “canceled”. Ex: some downloading processes could have been interrupted, at startup, “downloading” status should be changed to “canceled” so to reflect actual state on these datasources. This must be overridden in subclass.

property collection

Return collection object used to fetch doc in which we store status

configure(release_confdict)

Configure manager with release “confdict”. See config_hub.py in API for the format.

create_release_note(old, new, filename=None, note=None, format='txt')

Generate release note files, in TXT and JSON format, containing significant changes summary between target collections old and new. Output files are stored in a diff folder using generate_folder(old,new).

‘filename’ can optionally be specified, though it’s not recommended as the publishing pipeline, using these files, expects a filenames convention.

‘note’ is an optional free text that can be added to the release note, at the end.

txt ‘format’ is the only one supported for now.

create_release_note_from_build(build_doc)

get_pinfo()

Return dict containing information about the current process (used to report in the hub)

get_predicates()

```

get_release_note(old, new, format='txt', prefix='release_*')

load_build(key_name, stage=None)

poll(state, func)
    Search for source in collection ‘col’ with a pending flag list containing ‘state’ and and call ‘func’ for each document found (with doc as only param)

publish(publisher_env, snapshot_or_build_name, *args, **kwargs)

publish_build(build_doc)

publish_diff(publisher_env, build_name, previous_build=None, steps=('pre', 'reset', 'upload', 'meta', 'post'))

publish_snapshot(publisher_env, snapshot, build_name=None, previous_build=None, steps=('pre', 'meta', 'post'))

register_status(bdoc, stage, status, transient=False, init=False, **extra)

release_info(env=None, remote=False)

reset_synced(old, new)
    Reset sync flags for diff files produced between “old” and “new” build. Once a diff has been applied, diff files are flagged as synced so subsequent diff won’t be applied twice (for optimization reasons, not to avoid data corruption since diff files can be safely applied multiple times). In any needs to apply the diff another time, diff files needs to reset.

setup()

setup_log(build_name=None)

class biothings.hub.datarelease.publisher.SnapshotPublisher(snapshot_manager, *args, **kwargs)
Bases: BasePublisher

get_pre_post_previous_result(build_doc, key_value)
    In order to start a pre- or post- pipeline, a first previous result, fed all along the pipeline to the next step, has to be defined, and depends on the type of publisher.

post_publish(snapshot_name, repo_conf, build_doc)
    Post-publish hook, running steps declared in config, but also whatever would be defined in a sub-class

pre_publish(snapshot_name, repo_conf, build_doc)
    Pre-publish hook, running steps declared in config, but also whatever would be defined in a sub-class

publish(snapshot, build_name=None, previous_build=None, steps=('pre', 'meta', 'post'))
    Publish snapshot metadata to S3. If snapshot repository is of type “s3”, data isn’t actually uploaded/published since it’s already there on s3. If type “fs”, some “pre” steps can be added to the RELEASE_CONFIG parameter to archive and upload it to s3. Metadata about the snapshot, release note, etc... is then uploaded in correct buckets as defined in config, and “post” steps can be run afterward.

    Though snapshots don’t need any previous version to be applied on, a release note with significant changes between current snapshot and a previous version could have been generated. By default, snapshot name is used to pick one single build document and from the document, get the release note information.

run_post_publish_snapshot(snapshot_name, repo_conf, build_doc)

run_pre_publish_snapshot(snapshot_name, repo_conf, build_doc)

```

biothings.hub.datarelease.releasenote

```
class biothings.hub.datarelease.releasenote.ReleaseNoteSource(old_src_build_reader:
                                                               ReleaseNoteSrcBuildReader,
                                                               new_src_build_reader:
                                                               ReleaseNoteSrcBuildReader,
                                                               diff_stats_from_metadata_file: dict,
                                                               addon_note: str)  
Bases: object  
diff_build_stats() → dict  
diff_datasource_info() → dict  
diff_datasource_mapping() → dict  
to_dict() → dict  
  
class biothings.hub.datarelease.releasenote.ReleaseNoteSrcBuildReader(src_build_doc: dict)  
Bases: object  
attach_cold_src_build_reader(other: ReleaseNoteSrcBuildReader)  
    Attach a cold src_build reader.  
    It's required that self is a hot src_builder reader and other is cold.  
property build_id: str  
property build_stats: dict  
property build_version: str  
property cold_collection_name: str  
property datasource_mapping: dict  
property datasource_stats: dict  
property datasource_versions: dict  
has_cold_collection() → bool  
  
class biothings.hub.datarelease.releasenote.ReleaseNoteSrcBuildReaderAdapter(src_build_reader:
                                                                           ReleaseNoteSrcBuildReader)  
Bases: object  
property build_stats  
property datasource_info  
  
class biothings.hub.datarelease.releasenote.ReleaseNoteTxt(source: ReleaseNoteSource)  
Bases: object  
save(filepath)
```

biothings.hub.datatransform**biothings.hub.datatransform.ciidstruct**

CIIDStruct - case insensitive id matching data structure

```
class biothings.hub.datatransform.ciidstruct.CIIDStruct(field=None, doc_lst=None)
```

Bases: *IDStruct*

CIIDStruct - id structure for use with the DataTransform classes. The basic idea is to provide a structure that provides a list of (original_id, current_id) pairs.

This is a case-insensitive version of IDStruct.

Initialize the structure :param field: field for documents to use as an initial id (optional) :param doc_lst: list of documents to use when building an initial list (optional)

```
add(left, right)
```

add a (original_id, current_id) pair to the list, All string values are typecast to lowercase

```
find(where, ids)
```

Case insensitive lookup of ids

biothings.hub.api.datatransform.datatransform_api

DataTransforAPI - classes around API based key lookup.

```
class biothings.hub.datatransform.datatransform_api.BiothingsAPIEdge(lookup, fields, weight=1, label=None, url=None)
```

Bases: *DataTransformEdge*

APIEdge - IDLookupEdge object for API calls

Initialize the class :param label: A label can be used for debugging purposes.

```
property client
```

property getter for client

```
client_name = None
```

```
edge_lookup(keylookup_obj, id_strct, debug=False)
```

Follow an edge given a key.

This method uses the data in the edge_object to find one key to another key using an api. :param edge: :param key: :return:

```
init_state()
```

initialize state - pickleable member variables

```
prepare_client()
```

Load the biothings_client for the class :return:

```
class biothings.hub.datatransform.datatransform_api.DataTransformAPI(input_types, output_types, *args, **kwargs)
```

Bases: *DataTransform*

Perform key lookup or key conversion from one key type to another using an API endpoint as a data source.

This class uses biothings apis to conversion from one key type to another. Base classes are used with the decorator syntax shown below:

```
@IDLookupMyChemInfo(input_types, output_types)
def load_document(doc_lst):
    for d in doc_lst:
        yield d
```

Lookup fields are configured in the ‘lookup_fields’ object, examples of which can be found in ‘IDLookupMyGeneInfo’ and ‘IDLookupMyChemInfo’.

Required Options:

- **input_types**
 - ‘type’
 - (‘type’, ‘nested_source_field’)
 - [(‘type1’, ‘nested.source_field1’), (‘type2’, ‘nested.source_field2’), …]
- **output_types:**
 - ‘type’
 - [‘type1’, ‘type2’]

Additional Options: see DataTransform class

Initialize the IDLookupAPI object.

```
batch_size = 10
default_source = '_id'
key_lookup_batch(batchiter)
```

Look up all keys for ids given in the batch iterator (1 block) :param batchiter: 1 lock of records to look up keys for :return:

```
lookup_fields = {}
```

```
class biothings.hub.datatransform.datatransform_api.DataTransformMyChemInfo(input_types, output_types=None,
skip_on_failure=False,
skip_w_regex=None)
```

Bases: *DataTransformAPI*

Single key lookup for MyChemInfo

Initialize the class by setting up the client object.

```
lookup_fields = {'chebi': 'chebi.chebi_id', 'chembl': 'chembl.molecule_chembl_id',
'drugbank': 'drugbank.drugbank_id', 'drugname': ['drugbank.name',
'unii.preferred_term', 'chebi.chebi_name', 'chembl.pref_name'], 'inchi':
['drugbank.inchi', 'chembl.inchi', 'pubchem.inchi'], 'inchikey':
['drugbank.inchi_key', 'chembl.inchi_key', 'pubchem.inchi_key'], 'pubchem':
'pubchem.cid', 'rxnorm': ['unii.rxcui'], 'unii': 'unii.unii'}

output_types = ['inchikey', 'unii', 'rxnorm', 'drugbank', 'chebi', 'chembl',
'pubchem', 'drugname']
```

```
class biothings.hub.datatransform.datatransform_api.DataTransformMyGeneInfo(input_types, output_types=None,
skip_on_failure=False,
skip_w_regex=None)
```

Bases: *DataTransformAPI*

deprecated

Initialize the class by setting up the client object.

```
lookup_fields = {'ensembl': 'ensembl.gene', 'entrezgene': 'entrezgene', 'symbol': 'symbol', 'uniprot': 'uniprot.Swiss-Prot'}
```

```
class biothings.hub.datatransform.datatransform_api.MyChemInfoEdge(lookup, field, weight=1,
label=None, url=None)
```

Bases: *BiothingsAPIEdge*

The MyChemInfoEdge uses the MyChem.info API to convert identifiers.

Parameters

- **lookup** (*str*) – The field in the API to search with the input identifier.
- **field** (*str*) – The field in the API to convert to.
- **weight** (*int*) – Weights are used to prefer one path over another. The path with the lowest weight is preferred. The default weight is 1.

```
client_name = 'drug'
```

```
class biothings.hub.datatransform.datatransform_api.MyGeneInfoEdge(lookup, field, weight=1,
label=None, url=None)
```

Bases: *BiothingsAPIEdge*

The MyGeneInfoEdge uses the MyGene.info API to convert identifiers.

Parameters

- **lookup** (*str*) – The field in the API to search with the input identifier.
- **field** (*str*) – The field in the API to convert to.
- **weight** (*int*) – Weights are used to prefer one path over another. The path with the lowest weight is preferred. The default weight is 1.

```
client_name = 'gene'
```

biothings.hub.datatransform.datatransform_mdb

DataTransform MDB module - class for performing key lookup using conversions described in a networkx graph.

```
class biothings.hub.datatransform.datatransform_mdb.CIMongoDBEdge(collection_name, lookup,
field, weight=1, label=None)
```

Bases: *MongoDBEdge*

Case-insensitive MongoDBEdge

Parameters

- **collection_name** (*str*) – The name of the MongoDB collection.
- **lookup** (*str*) – The field that will match the input identifier in the collection.
- **field** (*str*) – The output identifier field that will be read out of matching documents.

- **weight** (*int*) – Weights are used to prefer one path over another. The path with the lowest weight is preferred. The default weight is 1.

collection_find(*id_lst, lookup, field*)

Abstract out (as one line) the call to collection.find and use a case-insensitive collation

class biothings.hub.datatransform.datatransform_mdb.**DataTransformMDB**(*graph, *args, **kwargs*)

Bases: *DataTransform*

Convert document identifiers from one type to another.

The DataTransformNetworkX module was written as a decorator class which should be applied to the load_data function of a Biothings Uploader. The load_data function yields documents, which are then post processed by call and the ‘id’ key conversion is performed.

Parameters

- **graph** – nx.DiGraph (networkx 2.1) configuration graph
- **input_types** – A list of input types for the form (identifier, field) where identifier matches a node and field is an optional dotstring field for where the identifier should be read from (the default is ‘_id’).
- **output_types** (*list(str)*) – A priority list of identifiers to convert to. These identifiers should match nodes in the graph.
- **id_priority_list** (*list(str)*) – A priority list of identifiers to sort input and output types by.
- **skip_on_failure** (*bool*) – If True, documents where identifier conversion fails will be skipped in the final document list.
- **skip_w_regex** (*bool*) – Do not perform conversion if the identifier matches the regular expression provided to this argument. By default, this option is disabled.
- **skip_on_success** (*bool*) – If True, documents where identifier conversion succeeds will be skipped in the final document list.
- **idstruct_class** (*class*) – Override an internal data structure used by the this module (advanced usage)
- **copy_from_doc** (*bool*) – If true then an identifier is copied from the input source document regardless as to weather it matches an edge or not. (advanced usage)

batch_size = 1000

default_source = ‘_id’

key_lookup_batch(*batchiter*)

Look up all keys for ids given in the batch iterator (1 block) :param batchiter: 1 lock of records to look up keys for :return:

travel(*input_type, target, doc_lst*)

Traverse a graph from a start key type to a target key type using precomputed paths.

Parameters

- **start** – key type to start from
- **target** – key type to end at
- **key** – key value of type ‘start’

Returns

```
class biothings.hub.datatransform.datatransform_mdb.MongoDBEdge(collection_name, lookup, field,
                                                               weight=1, label=None,
                                                               check_index=True)
```

Bases: *DataTransformEdge*

The MongoDBEdge uses data within a MongoDB collection to convert one identifier to another. The input identifier is used to search a collection. The output identifier values are read out of that collection:

Parameters

- **collection_name** (*str*) – The name of the MongoDB collection.
- **lookup** (*str*) – The field that will match the input identifier in the collection.
- **field** (*str*) – The output identifier field that will be read out of matching documents.
- **weight** (*int*) – Weights are used to prefer one path over another. The path with the lowest weight is preferred. The default weight is 1.

property collection

getting for collection member variable

collection_find(*id_lst, lookup, field*)

Abstract out (as one line) the call to collection.find

edge_lookup(*keylookup_obj, id_struct, debug=False*)

Follow an edge given a key.

An edge represents a document and this method uses the data in the edge_object to find one key to another key using exactly one mongodb lookup. :param keylookup_obj: :param id_struct: :return:

init_state()

initialize the state of pickleable objects

prepare_collection()

Load the mongodb collection specified by collection_name. :return:

biothings.hub.datatransform.datatransform

DataTransform Module - IDStruct - DataTransform (superclass)

```
class biothings.hub.datatransform.datatransform.DataTransform(input_types, output_types,
                                                               id_priority_list=None,
                                                               skip_on_failure=False,
                                                               skip_w_regex=None,
                                                               skip_on_success=False,
                                                               idstruct_class=<class 'biothings.hub.datatransform.datatransform.IDStruct'>,
                                                               copy_from_doc=False,
                                                               debug=False)
```

Bases: *object*

DataTransform class. This class is the public interface for the DataTransform module. Much of the core logic is in the subclass.

Initialize the keylookup object and precompute paths from the start key to all target keys.

The decorator is intended to be applied to the load_data function of an uploader. The load_data function yields documents, which are then post processed by call and the ‘id’ key conversion is performed.

Parameters

- **G** – nx.DiGraph (networkx 2.1) configuration graph
- **collections** – list of mongodb collection names
- **input_type** – key type to start key lookup from
- **output_types** – list of all output types to convert to
- **id_priority_list (list(str))** – A priority list of identifiers to sort input and output types by.
- **id_struct_class** – IDStruct used to manager/fetch IDs from docs
- **copy_from_doc** – if transform failed using the graph, try to get value from the document itself when output_type == input_type. No check is performed, it's a straight copy. If checks are needed (eg. check that an ID referenced in the doc actually exists in another collection, nodes with self-loops can be used, so ID resolution will be forced to go through these loops to ensure data exists)

DEFAULT_WEIGHT = 1

batch_size = 1000

debug = False

default_source = '_id'

property id_priority_list

Property method for getting id_priority_list

key_lookup_batch(batchiter)

Core method for looking up all keys in batch (iterator) :param batchiter: :return:

lookup_one(doc)

KeyLookup on document. This method is called as a function call instead of a decorator on a document iterator.

sort_input_by_priority_list(input_types)

Reorder the given input_types to follow a priority list. Inputs not in the priority list should remain in their given order at the end of the list.

sort_output_by_priority_list(output_types)

Reorder the given output_types to follow a priority list. Outputs not in the priority list should remain in their given order at the end of the list.

class biothings.hub.datatransform.datatransform.DataTransformEdge(label=None)

Bases: object

DataTransformEdge. This class contains information needed to transform one key to another.

Initialize the class :param label: A label can be used for debugging purposes.

edge_lookup(keylookup_obj, id_strct, debug=False)

virtual method for edge lookup. Each edge class is responsible for its own lookup procedures given a keylookup_obj and an id_strct :param keylookup_obj: :param id_strct: - list of tuples (orig_id, current_id) :return:

init_state()

initialize the state of pickleable objects

property logger
getter for the logger property

prepare(state=None)
Prepare class state objects (pickleable objects)

setup_log()
setup the logger member variable

unprepare()
reset anything that's not pickleable (so self can be pickled) return what's been reset as a dict, so self can be restored once pickled

class biothings.hub.datatransform.datatransform.IDStruct(field=None, doc_lst=None)
Bases: object

IDStruct - id structure for use with the DataTransform classes. The basic idea is to provide a structure that provides a list of (original_id, current_id) pairs.

Initialize the structure :param field: field for documents to use as an initial id (optional) :param doc_lst: list of documents to use when building an initial list (optional)

add(left, right)
add a (original_id, current_id) pair to the list

static find(where, ids)
Find all ids in dictionary where

find_left(ids)
Find left values given a list of ids

find_right(ids)
Find the first id founding by searching the (_, right) identifiers

get_debug(key)
Get debug information for a given key

property id_lst
Build up a list of current ids

import_debug(other)
import debug information the entire IDStruct object

left(key)
Determine if the id (left, _) is registered

lookup(left, right)
Find if a (left, right) pair is already in the list

right(key)
Determine if the id (_, right) is registered

set_debug(left, label, right)
Set debug (left, right) debug values for the structure

static side(_id, where)
Find if an _id is a key in where

`transfer_debug(key, other)`

transfer debug information for one key in the IDStruct object

`class biothings.hub.datatransform.datatransform.RegExEdge(from_regex, to_regex, weight=1, label=None)`

Bases: `DataTransformEdge`

The RegExEdge allows an identifier to be transformed using a regular expression. POSIX regular expressions are supported.

Parameters

- `from_regex (str)` – The first parameter of the regular expression substitution.
- `to_regex (str)` – The second parameter of the regular expression substitution.
- `weight (int)` – Weights are used to prefer one path over another. The path with the lowest weight is preferred. The default weight is 1.

`edge_lookup(keylookup_obj, id_strct, debug=False)`

Transform identifiers using a regular expression substitution.

`biothings.hub.datatransform.datatransform.nested_lookup(doc, field)`

Performs a nested lookup of doc using a period(.) delimited list of fields. This is a nested dictionary lookup.
:param doc: document to perform lookup on :param field: period delimited list of fields :return:

`biothings.hub.datatransform.histogram`

DataTransform Histogram class - track keylookup statistics

`class biothings.hub.datatransform.histogram.Histogram`

Bases: `object`

Histogram - track keylookup statistics

`update_edge(vert1, vert2, size)`

Update the edge histogram

`update_io(input_type, output_type, size)`

Update the edge histogram

`biothings.hub.standalone`

This standalone module is originally located at “biothings/standalone” repo. It’s used for Standalone/Autohub instance.

`class biothings.hub.standalone.AutoHubFeature(managers, version_urls, indexer_factory=None, validator_class=None, *args, **kwargs)`

Bases: `object`

`version_urls` is a list of URLs pointing to versions.json file. The name of the data release is taken from the URL (http://...s3.amazonaws.com/<the_name>/versions.json) unless specified as a dict: {“name” : “custom_name”, “url” : “<http://...>”}

If `indexer_factory` is passed, it’ll be used to create indexer used to dump/check versions currently installed on ES, restore snapshot, index, etc... A `indexer_factory` is typically used to generate indexer dynamically (ES host, index name, etc...) according to URLs for instance. See `standalone.hub.DynamicIndexerFactory` class for an

example. It is typically used when lots of data releases are being managed by the Hub (so no need to manually update STANDALONE_CONFIG parameter).

If indexer_factory is None, a config param named STANDALONE_CONFIG is used, format is the following:

```
{"_default"
 [{"es_host": "...", "index": "...", "doc_type": "..."}, {"the_name": {"es_host": "...", "index": "...", "doc_type": "..."}]}
```

When a data release named (from URL) matches an entry, it's used to configured which ES backend to target, otherwise the default one is used.

If validator_class is passed, it'll be used to provide validation methods for installing step. If validator_class is None, the AutoHubValidator will be used as fallback.

DEFAULT_DUMPER_CLASS

alias of *BiothingsDumper*

DEFAULT_UPLOADER_CLASS

alias of *BiothingsUploader*

DEFAULT_VALIDATOR_CLASS

alias of AutoHubValidator

configure()

Either configure autohub from static definition (STANDALONE_CONFIG) where different hard-coded names of indexes can be managed on different ES server, or use a indexer factory where index names are taken from version_urls but only one ES host is used.

configure_auto_release(config)

extract(urls)

get_class_name(folder)

Return class-compliant name from a folder name

get_folder_name(url)

install(src_name, version='latest', dry=False, force=False, use_no_downtime_method=True)

Update hub's data up to the given version (default is latest available), using full and incremental updates to get up to that given version (if possible).

list_biothings()

Example: [{‘name’: ‘mygene.info’, ‘url’: ‘<https://biothings-releases.s3-us-west-2.amazonaws.com/mygene.info/versions.json>’}]

```
class biothings.hub.standalone.AutoHubServer(source_list, features=None, name='BioThings Hub',
                                              managers_custom_args=None, api_config=None,
                                              reloader_config=None, dataupload_config=None,
                                              websocket_config=None, autohub_config=None)
```

Bases: *HubServer*

Helper to setup and instantiate common managers usually used in a hub (eg. dumper manager, uploader manager, etc...) “source_list” is either:

- a list of string corresponding to paths to datasources modules
- a package containing sub-folders with datasources modules

Specific managers can be retrieved adjusting “features” parameter, where each feature corresponds to one or more managers. Parameter defaults to all possible available. Managers are configured/init in the same order as the list, so if a manager (eg. job_manager) is required by all others, it must be the first in the list. “managers_custom_args” is an optional dict used to pass specific arguments while init managers:

```
managers_custom_args={"upload": {"poll_schedule": "*/5 * * * *"}}
```

will set poll schedule to check upload every 5min (instead of default 10s) “reloader_config”, “dataupload_config”, “autohub_config” and “websocket_config” can be used to customize reloader, dataupload and websocket. If None, default config is used. If explicitly False, feature is deactivated.

```
DEFAULT_FEATURES = ['job', 'autohub', 'terminal', 'config', 'ws']
```

6.10.2 Commands

biothings.hub.commands

This document will show you all available commands that can be used when you access the Hub shell, and their usages.

am

This is a instance of type: <class ‘biothings.hub.dataplugin.assistant.AssistantManager’>

api

This is a instance of type: <class ‘biothings.hub.api.manager.APIManager’>

apply(src, *args, **kwargs)

method upload_src in module biothings.hub.dataload.uploader

upload_src(src, *args, **kwargs) method of biothings.hub.dataload.uploader.UploaderManager instance

Trigger upload for registered resource named ‘src’. Other args are passed to uploader’s load() method

archive(merge_name)

method archive_merge in module biothings.hub.databuild.builder

archive_merge(merge_name) method of biothings.hub.databuild.builder.BuilderManager instance

Delete merged collections and associated metadata

backend(src, method_name, *args, **kwargs)

method call in module biothings.hub.dataload.dumper

call(src, method_name, *args, **kwargs) method of biothings.hub.dataload.dumper.DumperManager instance

Create a dumper for datasource “src” and call method “method_name” on it, with given arguments. Used to create arbitrary calls on a dumper. “method_name” within dumper definition must a coroutine.

backup(folder='.', archive=None)

function backup in module biothings.utils.hub_db

backup(folder='.', archive=None)

Dump the whole hub_db database in given folder. “archive” can be pass to specify the target filename, otherwise, it’s randomly generated

Note: this doesn’t backup source/merge data, just the internal data used by the hub

bm

This is a instance of type: <class ‘biothings.hub.databuild.builder.BuilderManager’>

build(*id*)

function <lambda> in module biothings.hub

<lambda> lambda id

build_config_info()

method build_config_info in module biothings.hub.databuild.builder

build_config_info() method of biothings.hub.databuild.builder.BuilderManager instance

build_save_mapping(*name*, *mapping=None*, *dest='build'*, *mode='mapping'*)

method save_mapping in module biothings.hub.databuild.builder

save_mapping(name, mapping=None, dest='build', mode='mapping') method of biothings.hub.databuild.builder.BuilderManager instance

builds(*id=None*, *conf_name=None*, *fields=None*, *only_archived=False*)

method build_info in module biothings.hub.databuild.builder

build_info(*id=None*, *conf_name=None*, *fields=None*, *only_archived=False*) method of biothings.hub.databuild.builder.BuilderManager instance

Return build information given an build _id, or all builds if _id is None. “fields” can be passed to select which fields to return or not (mongo notation for projections), if None return everything except:

- “mapping” (too long)

If id is None, more are filtered:

- “sources” and some of “build_config”

only_archived=True will return archived merges only

check(*src*, *force=False*, *skip_manual=False*, *schedule=False*, *check_only=False*, *kwargs*)**

method dump_src in module biothings.hub.dataload.dumper

dump_src(src, force=False, skip_manual=False, schedule=False, check_only=False, **kwargs) method of biothings.hub.dataload.dumper.DumperManager instance

command(*id*, **args*, *kwargs*)**

function <lambda> in module biothings.utils.hub

<lambda> lambda id, *args, **kwargs

commands(*id=None*, *running=None*, *failed=None*)

method command_info in module biothings.utils.hub

command_info(id=None, running=None, failed=None) method of traitlets.traitlets.MetaHasTraits instance

config()

method show in module biothings.utils.configuration

show() method of biothings.utils.configuration.ConfigurationWrapper instance

create_api(api_id, es_host, index, doc_type, port, description=None, **kwargs)

method create_api in module biothings.hub.api.manager

create_api(api_id, es_host, index, doc_type, port, description=None, **kwargs) method of biothings.hub.api.manager.APIManager instance

create_build_conf(name, doc_type, sources, roots=[], builder_class=None, params={}, archived=False)

method create_build_configuration in module biothings.hub.databuild.builder

create_build_configuration(name, doc_type, sources, roots=[], builder_class=None, params={}, archived=False) method of biothings.hub.databuild.builder.BuilderManager instance

create_release_note(old, new, filename=None, note=None, format='txt')

method create_release_note in module biothings.hub.datarelease.publisher

**create_release_note(old, new, filename=None, note=None, format='txt') method of
biothings.hub.datarelease.publisher.ReleaseManager instance**

Generate release note files, in TXT and JSON format, containing significant changes summary between target collections old and new. Output files are stored in a diff folder using generate_folder(old,new).

‘filename’ can optionally be specified, though it’s not recommended as the publishing pipeline, using these files, expects a filenames convention.

‘note’ is an optional free text that can be added to the release note, at the end.

txt ‘format’ is the only one supported for now.

delete_api(api_id)

method delete_api in module biothings.hub.api.manager

delete_api(api_id) method of biothings.hub.api.manager.APIManager instance

delete_build_conf(name)

method delete_build_configuration in module biothings.hub.databuild.builder

delete_build_configuration(name) method of biothings.hub.databuild.builder.BuilderManager instance

**diff(diff_type, old, new, batch_size=100000, steps=['content', 'mapping', 'reduce', 'post'], mode=None,
exclude=['_timestamp'])**

method diff in module biothings.hub.databuild.differ

**diff(diff_type, old, new, batch_size=100000, steps=['content', 'mapping', 'reduce', 'post'], mode=None,
exclude=['_timestamp']) method of biothings.hub.databuild.differ.DifferManager instance**

Run a diff to compare old vs. new collections. using differ algorithm diff_type. Results are stored in a diff folder.

Steps can be passed to choose what to do: - count: will count root keys in new collections and stores them as statistics. - content: will diff the content between old and new. Results (diff files) format depends on diff_type

diff_info()

method diff_info in module biothings.hub.databuild.differ

diff_info() method of biothings.hub.databuild.differ.DifferManager instance

dim

This is a instance of type: <class ‘biothings.hub.databuild.differ.DifferManager’>

dm

This is a instance of type: <class ‘biothings.hub.dataload.dumper.DumperManager’>

download(src, force=False, skip_manual=False, schedule=False, check_only=False, **kwargs)

method dump_src in module biothings.hub.dataload.dumper

dump_src(src, force=False, skip_manual=False, schedule=False, check_only=False, **kwargs) method of biothings.hub.dataload.dumper.DumperManager instance

dpm

This is a instance of type: <class ‘biothings.hub.dataplugin.manager.DataPluginManager’>

dump(src, force=False, skip_manual=False, schedule=False, check_only=False, **kwargs)

method dump_src in module biothings.hub.dataload.dumper

dump_src(src, force=False, skip_manual=False, schedule=False, check_only=False, **kwargs) method of biothings.hub.dataload.dumper.DumperManager instance

dump_all(force=False, **kwargs)

method dump_all in module biothings.hub.dataload.dumper

dump_all(force=False, **kwargs) method of biothings.hub.dataload.dumper.DumperManager instance

Run all dumpers, except manual ones

dump_info()

method dump_info in module biothings.hub.dataload.dumper

dump_info() method of biothings.hub.dataload.dumper.DumperManager instance

dump_plugin(src, force=False, skip_manual=False, schedule=False, check_only=False, **kwargs)

method dump_src in module biothings.hub.dataload.dumper

dump_src(src, force=False, skip_manual=False, schedule=False, check_only=False, **kwargs) method of biothings.hub.dataplugin.manager.DataPluginManager instance

export_command_documents(filepath)

method export_command_documents in module biothings.hub

export_command_documents(filepath) method of biothings.hub.HubServer instance

export_plugin(plugin_name, folder=None, what=['dumper', 'uploader', 'mapping'], purge=False)

method export in module biothings.hub.dataplugin.assistant

export(plugin_name, folder=None, what=['dumper', 'uploader', 'mapping'], purge=False) method of biothings.hub.dataplugin.assistant.AssistantManager instance

Export generated code for a given plugin name, in given folder (or use DEFAULT_EXPORT_FOLDER if None).

Exported information can be: - dumper: dumper class generated from the manifest - uploader: uploader class generated from the manifest - mapping: mapping generated from inspection or from the manifest If “purge” is true, any existing folder/code will be deleted first, otherwise, will raise an error if some folder/files already exist.

expose(endpoint_name, command_name, method, **kwargs)

method add_api_endpoint in module biothings.hub

add_api_endpoint(endpoint_name, command_name, method, **kwargs) method of biothings.hub.HubServer instance

Add an API endpoint to expose command named “command_name” using HTTP method “method”. **kwargs are used to specify more arguments for EndpointDefinition

g

This is a instance of type: <class ‘dict’>

get_apis()

method get_apis in module biothings.hub.api.manager

get_apis() method of biothings.hub.api.manager.APIManager instance

get_release_note(old, new, format='txt', prefix='release_*)'

method get_release_note in module biothings.hub.datarelease.publisher

get_release_note(old, new, format='txt', prefix='release_*)' method of biothings.hub.datarelease.publisher.ReleaseManager instance

help(func=None)

method help in module biothings.utils.hub

help(func=None) method of biothings.utils.hub.HubShell instance

Display help on given function/object or list all available commands

im

This is a instance of type: <class ‘biothings.hub.dataindex.indexer.IndexManager’>

index(indexer_env, build_name, index_name=None, ids=None, **kwargs)

method index in module biothings.hub.dataindex.indexer

index(indexer_env, build_name, index_name=None, ids=None, **kwargs) method of biothings.hub.dataindex.indexer.IndexManager instance

Trigger an index creation to index the collection build_name and create an index named index_name (or build_name if None). Optional list of IDs can be passed to index specific documents.

index_cleanup(env=None, keep=3, dryrun=True, **filters)

method cleanup in module biothings.hub.dataindex.indexer

cleanup(env=None, keep=3, dryrun=True, **filters) method of biothings.hub.dataindex.indexer.IndexManager instance

Delete old indices except for the most recent ones.

Examples:

```
>>> index_cleanup()
>>> index_cleanup("production")
>>> index_cleanup("local", build_config="demo")
>>> index_cleanup("local", keep=0)
>>> index_cleanup(_id="<elasticsearch_index>")
```

index_config

index_info(remote=False)

method index_info in module biothings.hub.dataindex.indexer

index_info(remote=False) method of biothings.hub.dataindex.indexer.IndexManager instance

Show index manager config with enhanced index information.

indexes_by_name(index_name=None, limit=10)

method get_indexes_by_name in module biothings.hub.dataindex.indexer

get_indexes_by_name(index_name=None, limit=10) method of biothings.hub.dataindex.indexer.IndexManager instance

Accept an index_name and return a list of indexes get from all elasticsearch environments

If index_name is blank, it will be return all indexes. limit can be used to specify how many indexes should be return.

The list of indexes will be like this: [

```
{
    "index_name": "...", "build_version": "...", "count": 1000, "creation_date": 1653468868933, "environment": {
        "name": "env name", "host": "localhost:9200",
    },
},]
```

info(src, method_name, *args, **kwargs)

method call in module biothings.hub.dataload.dumper

call(src, method_name, *args, **kwargs) method of biothings.hub.dataload.dumper.DumperManager instance

Create a dumper for datasource “src” and call method “method_name” on it, with given arguments. Used to create arbitrary calls on a dumper. “method_name” within dumper definition must a coroutine.

inspect(data_provider, mode='type', batch_size=10000, limit=None, sample=None, **kwargs)

method inspect in module biothings.hub.datainspect.inspector

inspect(data_provider, mode='type', batch_size=10000, limit=None, sample=None, **kwargs) method of biothings.hub.datainspect.inspector InspectorManager instance

Inspect given data provider: - backend definition, see bt.hub.dababuild.create_backend for

supported format), eg “merged_collection” or (“src”, “clinvar”)

- or callable yielding documents

Mode: - “type”: will inspect and report type map found in data (internal/non-standard format) - “mapping”: will inspect and return a map compatible for later

ElasticSearch mapping generation (see bt.utils.es.generate_es_mapping)

- “stats”: will inspect and report types + different counts found in data, giving a detailed overview of the volumetry of each fields and sub-fields
- “jsonschema”, same as “type” but result is formatted as json-schema standard
- limit: when set to an integer, will inspect only x documents.
- sample: combined with limit, for each document, if random.random() <= sample (float), the document is inspected. This option allows to inspect only a sample of data.

install(*src_name*, *version='latest'*, *dry=False*, *force=False*, *use_no_downtime_method=True*)

method install in module biothings.hub.standalone

**install(*src_name*, *version='latest'*, *dry=False*, *force=False*, *use_no_downtime_method=True*) method of
biothings.hub.standalone.AutoHubFeature instance**

Update hub's data up to the given version (default is latest available), using full and incremental updates to get up to that given version (if possible).

ism

This is a instance of type: <class 'biothings.hub.datainspect.inspector.InspectorManager'>

jm

This is a instance of type: <class 'biothings.utils.manager.JobManager'>

job_info()

method job_info in module biothings.utils.manager

job_info() method of biothings.utils.manager.JobManager instance

jsondiff(*src*, *dst*, *kwargs*)**

function make in module biothings.utils.jsondiff

make(*src*, *dst*, ***kwargs*)

list()

method list_biothings in module biothings.hub.standalone

list_biothings() method of biothings.hub.standalone.AutoHubFeature instance

Example: [{‘name’: ‘mygene.info’, ‘url’: ‘<https://biothings-releases.s3-us-west-2.amazonaws.com/mygene.info/versions.json>’}]

loop

This is a instance of type: <class 'asyncio.unix_events._UnixSelectorEventLoop'>

lsmERGE(*build_config=None*, *only_archived=False*)

method list_merge in module biothings.hub.databuild.builder

list_merge(*build_config=None*, *only_archived=False*) method of biothings.hub.databuild.builder.BuilderManager instance

merge(*build_name*, *sources=None*, *target_name=None*, *kwargs*)**

method merge in module biothings.hub.databuild.builder

merge(*build_name*, *sources=None*, *target_name=None*, *kwargs) method of
biothings.hub.databuild.builder.BuilderManager instance***

Trigger a merge for build named ‘*build_name*’. Optional list of sources can be passed (one single or a list). *target_name* is the target collection name used to store to merge data. If none, each call will generate a unique *target_name*.

pending

This is a instance of type: <class ‘str’>

pqueue

This is a instance of type: <class ‘concurrent.futures.process.ProcessPoolExecutor’>

publish(publisher_env, snapshot_or_build_name, *args, **kwargs)

method publish in module biothings.hub.datarelease.publisher

publish(publisher_env, snapshot_or_build_name, *args, **kwargs) method of biothings.hub.datarelease.publisher.ReleaseManager instance

publish_diff(publisher_env, build_name, previous_build=None, steps=['pre', 'reset', 'upload', 'meta', 'post'])

method publish_diff in module biothings.hub.datarelease.publisher

publish_diff(publisher_env, build_name, previous_build=None, steps=['pre', 'reset', 'upload', 'meta', 'post']) method of biothings.hub.datarelease.publisher.ReleaseManager instance

publish_snapshot(publisher_env, snapshot, build_name=None, previous_build=None, steps=['pre', 'meta', 'post'])

method publish_snapshot in module biothings.hub.datarelease.publisher

publish_snapshot(publisher_env, snapshot, build_name=None, previous_build=None, steps=['pre', 'meta', 'post']) method of biothings.hub.datarelease.publisher.ReleaseManager instance

quick_index(datasource_name, doc_type, indexer_env, subsource=None, index_name=None, **kwargs)

method quick_index in module biothings.hub

quick_index(datasource_name, doc_type, indexer_env, subsource=None, index_name=None, **kwargs)

method of biothings.hub.HubServer instance

Intention for datasource developers to quickly create an index to test their datasources. Automatically create temporary build config, build collection Then call the index method with the temporary build collection's name

register_url(url)

method register_url in module biothings.hub.dataplugin.assistant

register_url(url) method of biothings.hub.dataplugin.assistant.AssistantManager instance

release_info(env=None, remote=False)

method release_info in module biothings.hub.datarelease.publisher

release_info(env=None, remote=False) method of biothings.hub.datarelease.publisher.ReleaseManager instance

report(old_db_col_names, new_db_col_names, report_filename='report.txt', format='txt', detailed=True, max_reported_ids=None, max_randomly_picked=None, mode=None)

method diff_report in module biothings.hub.databuild.differ

diff_report(old_db_col_names, new_db_col_names, report_filename='report.txt', format='txt', detailed=True, max_reported_ids=None, max_randomly_picked=None, mode=None) method of biothings.hub.databuild.differ.DifferManager instance

reset_backend(src, method_name, *args, **kwargs)

method call in module biothings.hub.dataload.dumper

call(src, method_name, *args, **kwargs) method of biothings.hub.dataload.dumper.DumperManager instance

Create a dumper for datasource “src” and call method “method_name” on it, with given arguments. Used to create arbitrary calls on a dumper. “method_name” within dumper definition must a coroutine.

reset_synced(old, new)

method reset_synced in module biothings.hub.datarelease.publisher

reset_synced(old, new) method of biothings.hub.datarelease.publisher.ReleaseManager instance

Reset sync flags for diff files produced between “old” and “new” build. Once a diff has been applied, diff files are flagged as synced so subsequent diff won’t be applied twice (for optimization reasons, not to avoid data corruption since diff files can be safely applied multiple times). In any needs to apply the diff another time, diff files needs to reset.

resetconf(name=None)

method reset in module biothings.utils.configuration

reset(name=None) method of biothings.utils.configuration.ConfigurationWrapper instance

restart(force=False, stop=False)

method restart in module biothings.utils.hub

restart(force=False, stop=False) method of biothings.utils.hub.HubShell instance

restore(archive, drop=False)

function restore in module biothings.utils.hub_db

restore(archive, drop=False)

Restore database from given archive. If drop is True, then delete existing collections

rm

This is a instance of type: <class ‘biothings.hub.datarelease.publisher.ReleaseManager’>

rmmerge(merge_name)

method delete_merge in module biothings.hub.databuild.builder

delete_merge(merge_name) method of biothings.hub.databuild.builder.BuilderManager instance

Delete merged collections and associated metadata

sch(loop)

function get_schedule in module biothings.hub

get_schedule(loop)

try to render job in a human-readable way...

schedule(crontab, func, *args, **kwargs)

method schedule in module biothings.utils.manager

schedule(crontab, func, *args, **kwargs) method of biothings.utils.manager.JobManager instance

Helper to create a cron job from a callable “func”. *argd, and **kwargs are passed to func. “crontab” follows aicron notation.

setconf(name, value)

method store_value_to_db in module biothings.utils.configuration

store_value_to_db(name, value) method of biothings.utils.configuration.ConfigurationWrapper instance

sm

This is a instance of type: <class ‘biothings.hub.dataload.source.SourceManager’>

snapshot(snapshot_env, index, snapshot=None)

method snapshot in module biothings.hub.dataindex.snapshooter

**snapshot(snapshot_env, index, snapshot=None) method of
biothings.hub.dataindex.snapshooter.SnapshotManager instance**

Create a snapshot named “snapshot” (or, by default, same name as the index) from “index” according to environment definition (repository, etc...) “env”.

snapshot_cleanup(env=None, keep=3, group_by='build_config', dryrun=True, **filters)

method cleanup in module biothings.hub.dataindex.snapshooter

**cleanup(env=None, keep=3, group_by='build_config', dryrun=True, **filters) method of
biothings.hub.dataindex.snapshooter.SnapshotManager instance**

Delete past snapshots and keep only the most recent ones.

Examples:

```
>>> snapshot_cleanup()
>>> snapshot_cleanup("s3_outbreak")
>>> snapshot_cleanup("s3_outbreak", keep=0)
```

snapshot_config**snapshot_info(env=None, remote=False)**

method snapshot_info in module biothings.hub.dataindex.snapshooter

snapshot_info(env=None, remote=False) method of biothings.hub.dataindex.snapshooter.SnapshotManager instance

source_info(name, debug=False)

method get_source in module biothings.hub.dataload.source

get_source(name, debug=False) method of biothings.hub.dataload.source.SourceManager instance

source_reset(name, key='upload', subkey=None)

method reset in module biothings.hub.dataload.source

reset(name, key='upload', subkey=None) method of biothings.hub.dataload.source.SourceManager instance

Reset, ie. delete, internal data (src_dump document) for given source name, key subkey. This method is useful to clean outdated information in Hub’s internal database.

Ex: key=upload, name=mysource, subkey=mysubsource, will delete entry in corresponding

src_dump doc (_id=mysource), under key “upload”, for sub-source named “mysubsource”

“key” can be either ‘download’, ‘upload’ or ‘inspect’. Because there’s no such notion of subkey for dumper (ie. ‘download’, subkey is optional).

source_save_mapping(name, mapping=None, dest='master', mode='mapping')

method save_mapping in module biothings.hub.dataload.source

save_mapping(name, mapping=None, dest='master', mode='mapping') method of biothings.hub.dataload.source.SourceManager instance

sources(id=None, debug=False, detailed=False)

method get_sources in module biothings.hub.dataload.source

get_sources(id=None, debug=False, detailed=False) method of biothings.hub.dataload.source.SourceManager instance

ssm

This is a instance of type: <class ‘biothings.hub.dataindex.snapshooter.SnapshotManager’>

start_api(api_id)

method start_api in module biothings.hub.api.manager

start_api(api_id) method of biothings.hub.api.manager.APIManager instance

status(managers)

function status in module biothings.hub

status(managers)

Return a global hub status (number of sources, documents, etc...) according to available managers

stop(force=False)

method stop in module biothings.utils.hub

stop(force=False) method of biothings.utils.hub.HubShell instance

stop_api(api_id)

method stop_api in module biothings.hub.api.manager

stop_api(api_id) method of biothings.hub.api.manager.APIManager instance

sym

This is a instance of type: <class ‘biothings.hub.databuild.syncer.SyncerManager’>

sync(backend_type, old_db_col_names, new_db_col_names, diff_folder=None, batch_size=10000, mode=None, target_backend=None, steps=['mapping', 'content', 'meta', 'post'], debug=False)

method sync in module biothings.hub.databuild.syncer

sync(backend_type, old_db_col_names, new_db_col_names, diff_folder=None, batch_size=10000, mode=None, target_backend=None, steps=['mapping', 'content', 'meta', 'post'], debug=False) method of biothings.hub.databuild.syncer.SyncerManager instance

top(action='summary')

method top in module biothings.utils.manager

top(action='summary') method of biothings.utils.manager.JobManager instance

tqueue

This is a instance of type: <class ‘concurrent.futures.thread.ThreadPoolExecutor’>

um

This is a instance of type: <class ‘biothings.hub.dataload.uploader.UploaderManager’>

unregister_url(url=None, name=None)

method unregister_url in module biothings.hub.dataplugin.assistant

unregister_url(url=None, name=None) method of biothings.hub.dataplugin.assistant.AssistantManager instance

update_build_conf(name, doc_type, sources, roots=[], builder_class=None, params={}, archived=False)
method update_build_configuration in module biothings.hub.databuild.builder
update_build_configuration(name, doc_type, sources, roots=[], builder_class=None, params={}, archived=False)
method of biothings.hub.databuild.builder.BuilderManager instance

update_metadata(indexer_env, index_name, build_name=None, _meta=None)
method update_metadata in module biothings.hub.dataindex.indexer
**update_metadata(indexer_env, index_name, build_name=None, _meta=None) method of
biothings.hub.dataindex.IndexManager instance**

Update _meta field of the index mappings, basing on

1. the _meta value provided, including {}.
2. the _meta value of the build_name in src_build.
3. the _meta value of the build with the same name as the index.

Examples:

```
update_metadata("local", "mynews_201228_vsdevjd")           update_metadata("local", "mynews_201228_vsdevjd", "mynews_201228_current",
"mynews_201228_vsdevjd", _meta={})   update_metadata("local", "mynews_201228_vsdevjd", "mynews_201228_current",
                                         _meta={"author": "b"})      update_metadata("local", "mynews_201228_vsdevjd")
```

update_source_meta(src, dry=False)

method update_source_meta in module biothings.hub.dataload.uploader

update_source_meta(src, dry=False) method of biothings.hub.dataload.uploader.UploaderManager instance
Trigger update for registered resource named ‘src’.

upgrade(code_base)

function upgrade in module biothings.hub

upgrade(code_base)

Upgrade (git pull) repository for given code base name (“biothings_sdk” or “application”)

upload(src, *args, **kwargs)

method upload_src in module biothings.hub.dataload.uploader

upload_src(src, *args, **kwargs) method of biothings.hub.dataload.uploader.UploaderManager instance
Trigger upload for registered resource named ‘src’. Other args are passed to uploader’s load() method

upload_all(raise_on_error=False, **kwargs)

method upload_all in module biothings.hub.dataload.uploader

**upload_all(raise_on_error=False, **kwargs) method of biothings.hub.dataload.uploader.UploaderManager
instance**

Trigger upload processes for all registered resources. **kwargs are passed to upload_src() method

upload_info()

method upload_info in module biothings.hub.dataload.uploader

upload_info() method of biothings.hub.dataload.uploader.UploaderManager instance

validate_mapping(mapping, env)
method validate_mapping in module biothings.hub.dataindex.indexer
validate_mapping(mapping, env) method of biothings.hub.dataindex.indexer.IndexManager instance

versions(src, method_name, *args, **kwargs)
method call in module biothings.hub.dataload.dumper

call(src, method_name, *args, **kwargs) method of biothings.hub.dataload.dumper.DumperManager instance
Create a dumper for datasource “src” and call method “method_name” on it, with given arguments. Used to create arbitrary calls on a dumper. “method_name” within dumper definition must a coroutine.

whatsnew(build_name=None, old=None)
method whatsnew in module biothings.hub.databuild.builder

whatsnew(build_name=None, old=None) method of biothings.hub.databuild.builder.BuilderManager instance
Return datasources which have changed since last time (last time is datasource information from metadata, either from given old src_build doc name, or the latest found if old=None)

6.11 biothings.cli

biothings.cli.main()

The main entry point for running the BioThings CLI to test your local data plugins.

biothings.cli.setup_config()

Setup a config module necessary to launch the CLI

6.11.1 biothings.cli.dataplugin

biothings.cli.dataplugin.clean_data(dump: bool | None = False, upload: bool | None = False, clean_all: bool | None = False)

clean command for deleting all dumped files and/or drop uploaded sources tables

biothings.cli.dataplugin.create_data_plugin(name: str = "", multi_uploaders: bool | None = False, parallelizer: bool | None = False)

create command for creating a new data plugin from the template

biothings.cli.dataplugin.dump_and_upload()

dump_and_upload command for downloading source data files to local, then converting them into JSON documents and uploading them to the source database. Two steps in one command.

biothings.cli.dataplugin.dump_data()

dump command for downloading source data files to local

biothings.cli.dataplugin.inspect_source(sub_source_name: str | None = "", mode: str | None = 'type,stats', limit: int | None = None, output: str | None = None)

inspect command for giving detailed information about the structure of documents coming from the parser after the upload step

biothings.cli.dataplugin.listing(dump: bool | None = False, upload: bool | None = False, hubdb: bool | None = False)

list command for listing dumped files and/or uploaded sources

`biothings.cli.dataplugint.serve(host: str | None = 'localhost', port: int | None = 9999)`

serve command runs a simple API server for serving documents from the source database.

For example, after run ‘dump_and_upload’, we have a source_name = “test” with a document structure like this:

```
doc = {"_id": "123", "key": {"a": {"b": "1"}, "x": [{"y": "3", "z": "4"}, "5"]}}.
```

An API server will run at <http://host:port/<your source name>>, like <http://localhost:9999/test/>:

- You can see all available sources on the index page: <http://localhost:9999/>
- You can list all docs: <http://localhost:9999/test/> (default is to return the first 10 docs)
- You can paginate doc list: <http://localhost:9999/test/?start=10&limit=10>
- You can retrieve a doc by id: <http://localhost:9999/test/123>
- **You can filter out docs with one or multiple fielded terms:**
 - <http://localhost:9999/test/?q=key.a.b:1> (query by any field with dot notation like key.a.b=1)
 - <http://localhost:9999/test/?q=key.a.b:1%20AND%20key.x.y=3> (find all docs that match two fields)
 - http://localhost:9999/test/?q=key.x.z:4* (field value can contain wildcard * or ?)
 - <http://localhost:9999/test/?q=key.x:5&start=10&limit=10> (pagination also works)

`biothings.cli.dataplugint.upload_source(batch_limit: int | None = None)`

upload command for converting downloaded data from dump step into JSON documents and upload the to the source database. A local sqlite database used to store the uploaded data

6.11.2 biothings.cli.dataplugint_hub

`biothings.cli.dataplugint_hub.clean_data(plugin_name: str = "", dump: bool | None = False, upload: bool | None = False, clean_all: bool | None = False)`

clean command for deleting all dumped files and/or drop uploaded sources tables

`biothings.cli.dataplugint_hub.create_data_plugin(name: str = "", multi_uploaders: bool | None = False, parallelizer: bool | None = False)`

create command for creating a new data plugin from the template

`biothings.cli.dataplugint_hub.dump_and_upload(plugin_name: str = "")`

dump_and_upload command for downloading source data files to local, then converting them into JSON documents and uploading them to the source database. Two steps in one command.

`biothings.cli.dataplugint_hub.dump_data(plugin_name: str = "")`

dump command for downloading source data files to local

`biothings.cli.dataplugint_hub.inspect_source(plugin_name: str = "", sub_source_name: str | None = "", mode: str | None = 'type,stats', limit: int | None = None, output: str | None = None)`

inspect command for giving detailed information about the structure of documents coming from the parser after the upload step

`biothings.cli.dataplugint_hub.listing(plugin_name: str = "", dump: bool | None = False, upload: bool | None = False, hubdb: bool | None = False)`

list command for listing dumped files and/or uploaded sources

```
biothings.cli.dataplugin_hub.serve(plugin_name: str = "", host: str | None = 'localhost', port: int | None = 9999)
```

serve command runs a simple API server for serving documents from the source database.

For example, after run ‘dump_and_upload’, we have a source_name = “test” with a document structure like this:

```
doc = {"_id": "123", "key": {"a": {"b": "1"}, "x": [{"y": "3", "z": "4"}, "5"]}}
```

An API server will run at <http://host:port/<your source name>/>, like <http://localhost:9999/test/>:

- You can see all available sources on the index page: <http://localhost:9999/>
- You can list all docs: <http://localhost:9999/test/> (default is to return the first 10 docs)
- You can paginate doc list: <http://localhost:9999/test/?start=10&limit=10>
- You can retrieve a doc by id: <http://localhost:9999/test/123>
- **You can filter out docs with one or multiple fielded terms:**
 - <http://localhost:9999/test/?q=key.a.b:1> (query by any field with dot notation like key.a.b=1)
 - <http://localhost:9999/test/?q=key.a.b:1%20AND%20key.x.y=3> (find all docs that match two fields)
 - http://localhost:9999/test/?q=key.x.z:4* (field value can contain wildcard * or ?)
 - <http://localhost:9999/test/?q=key.x:5&start=10&limit=10> (pagination also works)

```
biothings.cli.dataplugin_hub.upload_source(plugin_name: str = "", batch_limit: int | None = None)
```

upload command for converting downloaded data from dump step into JSON documents and upload the to the source database. A local sqlite database used to store the uploaded data

6.11.3 biothings.cli.utils

```
biothings.cli.utils.do_clean(plugin_name=None, dump=False, upload=False, clean_all=False, logger=None)
```

Clean the dumped files, uploaded sources, or both.

```
biothings.cli.utils.do_clean_dumped_files(data_folder, plugin_name)
```

Remove all dumped files by a data plugin in the data folder.

```
biothings.cli.utils.do_clean_uploaded_sources(working_dir, plugin_name)
```

Remove all uploaded sources by a data plugin in the working directory.

```
biothings.cli.utils.do_create(name, multi_uploader=False, parallelizer=False, logger=None)
```

Create a new data plugin from the template

```
biothings.cli.utils.do_dump(plugin_name=None, show_dumped=True, logger=None)
```

Perform dump for the given plugin

```
biothings.cli.utils.do_dump_and_upload(plugin_name, logger=None)
```

Perform both dump and upload for the given plugin

```
biothings.cli.utils.do_inspect(plugin_name=None, sub_source_name=None, mode='type,stats', limit=None, merge=False, output=None, logger=None)
```

Perform inspection on a data plugin.

```
biothings.cli.utils.do_list(plugin_name=None, dump=False, upload=False, hubdb=False, logger=None)
```

List the dumped files, uploaded sources, or hubdb content.

`biothings.cli.utils.do_serve(plugin_name=None, host='localhost', port=9999, logger=None)`

`biothings.cli.utils.do_upload(plugin_name=None, show_uploaded=True, logger=None)`

 Perform upload for the given list of uploader_classes

`biothings.cli.utils.get_logger(name=None)`

 Get a logger with the given name. If name is None, return the root logger.

`biothings.cli.utils.get_manifest_content(working_dir)`

 return the manifest content of the data plugin in the working directory

`biothings.cli.utils.get_plugin_name(plugin_name=None, with_working_dir=True)`

 return a valid plugin name (the folder name contains a data plugin) When plugin_name is provided as None, it use the current working folder. when with_working_dir is True, returns (plugin_name, working_dir) tuple

`biothings.cli.utils.get_uploaded_collections(src_db, uploaders)`

 A helper function to get the uploaded collections in the source database

`biothings.cli.utils.get_uploaders(working_dir: Path)`

 A helper function to get the uploaders from the manifest file in the working directory, used in show_uploaded_sources function below

`biothings.cli.utils.is_valid_data_plugin_dir(data_plugin_dir)`

 Return True/False if the given folder is a valid data plugin folder (contains either manifest.yaml or manifest.json)

`biothings.cli.utils.load_plugin(plugin_name=None, dumper=True, uploader=True, logger=None)`

 Return a plugin object for the given plugin_name. If dumper is True, include a dumper instance in the plugin object. If uploader is True, include uploader_classes in the plugin object.

 If <plugin_name> is not valid, raise the proper error and exit.

`biothings.cli.utils.load_plugin_managers(plugin_path, plugin_name=None, data_folder=None)`

 Load a data plugin from <plugin_path>, and return a tuple of (dumper_manager, upload_manager)

`biothings.cli.utils.process_inspect(source_name, mode, limit, merge, logger, do_validate, output=None)`

 Perform inspect for the given source. It's used in do_inspect function below

`biothings.cli.utils.remove_files_in_folder(folder_path)`

 Remove all files in a folder.

`biothings.cli.utils.run_sync_or_async_job(func, *args, **kwargs)`

 When func is defined as either normal or async function/method, we will call this function properly and return the results. For an async function/method, we will use CLIJobManager to run it.

`biothings.cli.utils.serve(host, port, plugin_name, table_space)`

 Serve a simple API server to query the data plugin source.

`biothings.cli.utils.show_dumped_files(data_folder, plugin_name)`

 A helper function to show the dumped files in the data folder

`biothings.cli.utils.show_hubdb_content()`

 Output hubdb content in a pretty format.

`biothings.cli.utils.show_uploaded_sources(working_dir, plugin_name)`

 A helper function to show the uploaded sources from given plugin.

6.11.4 `biothings.cli.web_app`

```
class biothings.cli.web_app.Application(db, table_space, **settings)
```

Bases: Application

The main application class, which defines the routes and handlers.

```
class biothings.cli.web_app.BaseHandler(application: Application, request: HTTPServerRequest,  
                                         **kwargs: Any)
```

Bases: RequestHandler

```
set_default_headers()
```

Override this to set HTTP headers at the beginning of the request.

For example, this is the place to set a custom Server header. Note that setting such headers in the normal flow of request processing may not do what you want, since headers may be reset during error handling.

```
class biothings.cli.web_app.DocHandler(application: Application, request: HTTPServerRequest,  
                                         **kwargs: Any)
```

Bases: BaseHandler

The handler for the detail view of a document, e.g. /<source>/<doc_id>

```
async get(slug, item_id)
```

```
class biothings.cli.web_app.HomeHandler(application: Application, request: HTTPServerRequest,  
                                         **kwargs: Any)
```

Bases: BaseHandler

the handler for the landing page, which lists all available routes

```
async get()
```

```
exception biothings.cli.web_app.NoResultError
```

Bases: Exception

```
class biothings.cli.web_app.QueryHandler(application: Application, request: HTTPServerRequest,  
                                         **kwargs: Any)
```

Bases: BaseHandler

The handler for return a list of docs matching the query terms passed to “q” parameter e.g. /<source>/?q=<query>

```
async get(slug)
```

```
async biothings.cli.web_app.get_available_routes(db, table_space)
```

return a list available URLs/routes based on the table_space and the actual collections in the database

```
biothings.cli.web_app.get_example_queries(db, table_space)
```

Populate example queries for a given table_space

```
async biothings.cli.web_app.main(host, port, db, table_space)
```

The main function, which starts the server.

PYTHON MODULE INDEX

b

biothings.cli, 308
biothings.cli.dataplugin, 308
biothings.cli.dataplugin_hub, 309
biothings.cli.utils, 310
biothings.cli.web_app, 312
biothings.hub, 217
biothings.hub.api, 224
biothings.hub.api.handlers.base, 225
biothings.hub.api.handlers.log, 226
biothings.hub.api.handlers.shell, 227
biothings.hub.api.handlers.upload, 227
biothings.hub.api.handlers.ws, 227
biothings.hub.api.manager, 224
biothings.hub.autoupdate.dumper, 228
biothings.hub.autoupdate.uploader, 230
biothings.hub.databuild.backend, 231
biothings.hub.databuild.buildconfig, 232
biothings.hub.databuild.builder, 234
biothings.hub.databuild.differ, 239
biothings.hub.databuild.mapper, 243
biothings.hub.databuild.prebuilder, 244
biothings.hub.databuild.syncer, 244
biothings.hub.dataexport.ids, 247
biothings.hub.dataindex.idcache, 247
biothings.hub.dataindex.indexer, 248
biothings.hub.dataindex.indexer_cleanup, 248
biothings.hub.dataindex.indexer_payload, 248
biothings.hub.dataindex.indexer_registrar,
 252
biothings.hub.dataindex.indexer_schedule, 253
biothings.hub.dataindex.indexer_task, 254
biothings.hub.dataindex.snapshooter, 255
biothings.hub.dataindex.snapshot_cleanup, 257
biothings.hub.dataindex.snapshot_registrar,
 258
biothings.hub.dataindex.snapshot_repo, 258
biothings.hub.dataindex.snapshot_task, 259
biothings.hub.datainspect.inspector, 259
biothings.hub.dataload.dumper, 260
biothings.hub.dataload.source, 273
biothings.hub.dataload.storage, 273

biothings.hub.dataload.sync, 273
biothings.hub.dataload.uploader, 274
biothings.hub.dataplugin.assistant, 278
biothings.hub.dataplugin.manager, 282
biothings.hub.datarelease, 282
biothings.hub.datarelease.publisher, 282
biothings.hub.datarelease.releasenote, 286
biothings.hub.datatransform.ciidstruct, 287
biothings.hub.datatransform.datatransform,
 291
biothings.hub.datatransform.datatransform_api,
 287
biothings.hub.datatransform.datatransform_mdb,
 289
biothings.hub.datatransform.histogram, 294
biothings.hub.standalone, 294
biothings.tests, 150
biothings.tests.hub, 150
biothings.tests.web, 150
biothings.utils, 153
biothings.utils.aws, 153
biothings.utils.backend, 154
biothings.utils.common, 156
biothings.utils.configuration, 162
biothings.utils.dataload, 165
biothings.utils.diff, 170
biothings.utils.doc_traversal, 171
biothings.utils.docs, 171
biothings.utils.dotfield, 172
biothings.utils.dotstring, 172
biothings.utils.es, 173
biothings.utils.exclude_ids, 178
biothings.utils.hub, 178
biothings.utils.hub_db, 181
biothings.utils.info, 183
biothings.utils.inspect, 183
biothings.utils.jsondiff, 188
biothings.utils.jsonpatch, 188
biothings.utils.jsonschema, 192
biothings.utils.loggers, 192
biothings.utils.manager, 195
biothings.utils.mongo, 198

`biothings.utils.parallel`, 211
`biothings.utils.parallel_mp`, 211
`biothings.utils.parsers`, 213
`biothings.utils.redis`, 214
`biothings.utils.serializer`, 214
`biothings.utils.sqlite3`, 215
`biothings.utils.version`, 217
`biothings.web`, 122
`biothings.web.analytics`, 126
`biothings.web.analytics.channels`, 126
`biothings.web.analytics.events`, 127
`biothings.web.analytics.notifiers`, 128
`biothings.web.applications`, 123
`biothings.web.connections`, 126
`biothings.web.handlers`, 128
`biothings.web.handlers.base`, 128
`biothings.web.handlers.query`, 130
`biothings.web.handlers.services`, 132
`biothings.web.launcher`, 122
`biothings.web.options`, 132
`biothings.web.options.manager`, 132
`biothings.web.options.openapi`, 135
`biothings.web.query`, 141
`biothings.web.query.builder`, 141
`biothings.web.query.engine`, 143
`biothings.web.query.formatter`, 144
`biothings.web.query.pipeline`, 147
`biothings.web.services`, 124
`biothings.web.services.health`, 124
`biothings.web.services.metadata`, 124
`biothings.web.services.namespace`, 125
`biothings.web.services.query`, 124
`biothings.web.settings`, 148
`biothings.web.settings.configs`, 148
`biothings.web.settings.default`, 149
`biothings.web.settings.validators`, 149
`biothings.web.templates`, 149

INDEX

Symbols

`_ClientPool` (*class in biothings.web.connections*), 126

A

`access_key` (*biothings.hub.dataindex.snapshooter.CloudStorage*.*attribute*), 255

`add()` (*biothings.hub.datatransform.ciidstruct.CIIDStruct* *method*), 287

`add()` (*biothings.hub.datatransform.datatransform.IDStruct* *method*), 293

`add()` (*biothings.utils.hub_db.ChangeWatcher* *class method*), 181

`add()` (*biothings.web.options.manager.OptionsManager* *method*), 134

`add_api_endpoint()` (*biothings.hub.HubServer* *method*), 222

`add_update()` (*biothings.hub.dataload.sync.MongoSync* *method*), 273

`add_watch()` (*biothings.utils.hub.TornadoAutoReloadHub*.*ReloadHandler* *method*), 180

`AddOperation` (*class in biothings.utils.jsonpatch*), 188

`address` (*biothings.utils.es.Database* *property*), 173

`address` (*biothings.utils.hub_db.IDatabase* *property*), 182

`address` (*biothings.utils.sqlite3.Database* *property*), 216

`addsuffix()` (*in module biothings.utils.common*), 158

`adjust_index()` (*biothings.web.query.engine.EsQueryBackend* *method*), 144

`AdvancedPluginLoader` (*class in biothings.hub.dataplugn.assistant*), 278

`agg_by_append()` (*in module biothings.utils.parallel_mp*), 211

`agg_by_sum()` (*in module biothings.utils.parallel_mp*), 211

`aggregate()` (*biothings.utils.parallel_mp.ParallelResult* *method*), 211

`aiogunzipall()` (*in module biothings.utils.common*), 158

`allow_empty()` (*biothings.web.options.openapi.OpenAPIParameterContext* *method*), 139

`allow_reserved()` (*biothings.web.options.openapi.OpenAPIParameterContext* *method*), 139

`AlreadyRunningException`, 178

`alwayslist()` (*in module biothings.utils.dataload*), 165
`am` (*built-in variable*), 296

`AnalyticsMixin` (*class in biothings.web.analytics.notifiers*), 128

`anonymous_download()` (*biothings.hub.autoupdate.dumper.BiothingsDumper* *method*), 229

`anyfile()` (*in module biothings.utils.common*), 158

`api` (*built-in variable*), 296

`APIDumper` (*class in biothings.hub.dataload.dumper*), 260

`APIManager` (*class in biothings.hub.api.manager*), 224

`APIManagerException`, 225

`APISpecificationHandler` (*class in biothings.web.handlers.services*), 132

`Application` (*class in biothings.cli.web_app*), 312

`apply()`, 296

`apply()` (*biothings.utils.jsonpatch.AddOperation* *method*), 188

`apply()` (*biothings.utils.jsonpatch.CopyOperation* *method*), 188

`apply()` (*biothings.utils.jsonpatch.JsonPatch* *method*), 189

`apply()` (*biothings.utils.jsonpatch.MoveOperation* *method*), 190

`apply()` (*biothings.utils.jsonpatch.PatchOperation* *method*), 190

`apply()` (*biothings.utils.jsonpatch.RemoveOperation* *method*), 191

`apply()` (*biothings.utils.jsonpatch.ReplaceOperation* *method*), 191

`apply()` (*biothings.utils.jsonpatch.TestOperation* *method*), 191

`apply_diff()` (*biothings.hub.autoupdate.uploader.BiothingsUploader* *method*), 230

`apply_extras()` (*biothings.web.query.builder.EsQueryBuilder* *method*), 141

apply_patch() (in module `biothings.utils.jsonpatch`), 191
ARCHIVE (`biothings.hub.dataload.dumper.BaseDumper` attribute), 261
archive(), 296
archive_merge() (`biothings.hub.databuild.builder.BuilderManager` method), 234
asdict() (`biothings.utils.configuration.ConfigAttrMeta` method), 162
ask() (in module `biothings.utils.common`), 158
AssistantException, 278
AssistantManager (class in `biothings.hub.dataplugin.assistant`), 278
AssistedDumper (class in `biothings.hub.dataplugin.assistant`), 279
AssistedUploader (class in `biothings.hub.dataplugin.assistant`), 279
async_check() (`biothings.web.services.health.ESHealth` method), 124
AsyncESQueryBackend (class in `biothings.web.query.engine`), 143
AsyncESQueryPipeline (class in `biothings.web.query.pipeline`), 147
at() (`biothings.hub.dataindex.indexer_registrar.Stage` method), 253
attach_cold_src_build_reader() (`biothings.hub.datarelease.releasenote.ReleaseNoteSrc` method), 286
ATTRIBUTE_FIELDS (`biothings.web.options.openapi.OpenAPIContactContext` attribute), 135
ATTRIBUTE_FIELDS (`biothings.web.options.openapi.OpenAPIExternalDocsContext` attribute), 136
ATTRIBUTE_FIELDS (`biothings.web.options.openapi.OpenAPIInfoContext` attribute), 137
ATTRIBUTE_FIELDS (`biothings.web.options.openapi.OpenAPILicenseContext` attribute), 138
ATTRIBUTE_FIELDS (`biothings.web.options.openapi.OpenAPIOperation` attribute), 138
ATTRIBUTE_FIELDS (`biothings.web.options.openapi.OpenAPIParameterContext` attribute), 139
audit() (in module `biothings.hub.dataindex.snapshot_registrar`), 258
auth_download() (`biothings.hub.autoupdate.dumper.BiothingsDumper` method), 229
AUTO_PURGE_INDEX (`biothings.hub.autoupdate.uploader.BiothingsUploader` attribute), 230
AUTO_UPLOAD (`biothings.hub.autoupdate.dumper.BiothingsDumper` attribute), 228
AUTO_UPLOAD (`biothings.hub.dataload.dumper.BaseDumper` attribute), 261
AutoBuildConfig (class in `biothings.hub.databuild.buildconfig`), 233
AutoBuildConfigError, 234
AutoHubFeature (class in `biothings.hub.standalone`), 294
AutoHubServer (class in `biothings.hub.standalone`), 295
AWS_ACCESS_KEY_ID (`biothings.hub.autoupdate.dumper.BiothingsDumper` attribute), 228
AWS_SECRET_ACCESS_KEY (`biothings.hub.autoupdate.dumper.BiothingsDumper` attribute), 228

B

backend(), 296
backup(), 296
backup() (in module `biothings.utils.hub_db`), 182
base_url (`biothings.hub.autoupdate.dumper.BiothingsDumper` property), 229
BaseAPIHandler (class in `biothings.web.handlers.base`), 128
BaseRSSHandler (class in `biothings.hub.dataplugin.assistant`), 279
BaseDiffer (class in `biothings.hub.databuild.differ`), 239
BaseDumper (class in `biothings.hub.dataload.dumper`), 261
BaseHandler (class in `biothings.cli.web_app`), 312
BaseHandler (class in `biothings.hub.api.handlers.base`), 225
BaseHandler (class in `biothings.web.handlers.base`), 129
BaseHubReloader (class in `biothings.utils.hub`), 178
BaseManager (class in `biothings.utils.manager`), 195
BaseMapper (class in `biothings.hub.databuild.mapper`), 243
BaseMode (class in `biothings.utils.inspect`), 183
BasePluginLoader (class in `biothings.hub.dataplugin.assistant`), 280
BasePreCompiledDataProvider (class in `biothings.hub.databuild.prebuilder`), 244
BasePublisher (class in `biothings.hub.datarelease.publisher`), 282
BaseQueryHandler (class in `biothings.web.handlers.query`), 130
BaseSourceManager (class in `biothings.utils.manager`), 195

BaseSourceUploader (class in <i>biothings.hub.dataload.uploader</i>), 274	<i>biothings.hub.api.manager</i> module, 224
BaseStatusRegisterer (class in <i>biothings.utils.manager</i>), 196	<i>biothings.hub.autoupdate.dumper</i> module, 228
BaseSyncer (class in <i>biothings.hub.databuild.syncer</i>), 244	<i>biothings.hub.autoupdate.uploader</i> module, 230
batch_size (<i>biothings.hub.datatransform.datatransform.DbioThingshub.databuild.backend</i> attribute), 292	<i>biothings.hub.databuild.backend</i> module, 231
batch_size (<i>biothings.hub.datatransform.datatransform_dapi.DbioThingshub.databuild.buildconfig</i> attribute), 288	<i>biothings.hub.databuild.buildconfig</i> module, 232
batch_size (<i>biothings.hub.datatransform.datatransform_mdb.DbioThingshub.databuild.builder</i> attribute), 290	<i>biothings.hub.databuild.MDBBuild.builder</i> module, 234
before_configure() (<i>biothings.hub.HubServer</i> method), 222	<i>biothings.hub.databuild.differ</i> module, 239
before_start() (<i>biothings.hub.HubServer</i> method), 222	<i>biothings.hub.databuild.mapper</i> module, 243
begin_auth() (<i>biothings.hub.HubSSHSERVER</i> method), 217	<i>biothings.hub.databuild.prebuilder</i> module, 244
BiothingHandler (class in <i>biothings.web.handlers.query</i>), 130	<i>biothings.hub.databuild.syncer</i> module, 244
BiothingHubMeta (class in <i>biothings.web.services.metadata</i>), 124	<i>biothings.hub.dataexport.ids</i> module, 247
BiothingLicenses (class in <i>biothings.web.services.metadata</i>), 124	<i>biothings.hub.dataindex.idcache</i> module, 247
BiothingMappings (class in <i>biothings.web.services.metadata</i>), 124	<i>biothings.hub.dataindex.indexer</i> module, 248
BiothingMetaProp (class in <i>biothings.web.services.metadata</i>), 125	<i>biothings.hub.dataindex.indexer_cleanup</i> module, 248
biothings (<i>biothings.web.handlers.base.BaseHandler</i> property), 129	<i>biothings.hub.dataindex.indexer_payload</i> module, 248
biothings.cli module, 308	<i>biothings.hub.dataindex.indexer_registrar</i> module, 252
biothings.cli.dataplugin module, 308	<i>biothings.hub.dataindex.indexer_schedule</i> module, 253
biothings.cli.dataplugin_hub module, 309	<i>biothings.hub.dataindex.indexer_task</i> module, 254
biothings.cli.utils module, 310	<i>biothings.hub.dataindex.snapshooter</i> module, 255
biothings.cli.web_app module, 312	<i>biothings.hub.dataindex.snapshot_cleanup</i> module, 257
biothings.hub module, 217	<i>biothings.hub.dataindex.snapshot_registrar</i> module, 258
biothings.hub.api module, 224	<i>biothings.hub.dataindex.snapshot_repo</i> module, 258
biothings.hub.api.handlers.base module, 225	<i>biothings.hub.dataindex.snapshot_task</i> module, 259
biothings.hub.api.handlers.log module, 226	<i>biothings.hub.datainspect.inspector</i> module, 259
biothings.hub.api.handlers.shell module, 227	<i>biothings.hub.dataload.dumper</i> module, 260
biothings.hub.api.handlers.upload module, 227	<i>biothings.hub.dataload.source</i> module, 273
biothings.hub.api.handlers.ws module, 227	<i>biothings.hub.dataload.storage</i> module, 273

biothings.hub.dataloader.sync
 module, 273
biothings.hub.dataloader.uploader
 module, 274
biothings.hub.dataplugin.assistant
 module, 278
biothings.hub.dataplugin.manager
 module, 282
biothings.hub.datarelease
 module, 282
biothings.hub.datarelease.publisher
 module, 282
biothings.hub.datarelease.releasenote
 module, 286
biothings.hub.datatransform.ciidstruct
 module, 287
biothings.hub.datatransform.datatransform
 module, 291
biothings.hub.datatransform.datatransform_api
 module, 287
biothings.hub.datatransform.datatransform_mdb
 module, 289
biothings.hub.datatransform.histogram
 module, 294
biothings.hub.standalone
 module, 294
biothings.tests
 module, 150
biothings.tests.hub
 module, 150
biothings.tests.web
 module, 150
biothings.utils
 module, 153
biothings.utils.aws
 module, 153
biothings.utils.backend
 module, 154
biothings.utils.common
 module, 156
biothings.utils.configuration
 module, 162
biothings.utils.dataloader
 module, 165
biothings.utils.diff
 module, 170
biothings.utils.doc_traversal
 module, 171
biothings.utils.docs
 module, 171
biothings.utils.dotfield
 module, 172
biothings.utils.dotstring
 module, 172

biothings.utils.es
 module, 173
biothings.utils.exclude_ids
 module, 178
biothings.utils.hub
 module, 178
biothings.utils.hub_db
 module, 181
biothings.utils.info
 module, 183
biothings.utils.inspect
 module, 183
biothings.utils.jsondiff
 module, 188
biothings.utils.jsonpatch
 module, 188
biothings.utils.jsonschema
 module, 192
biothings.utils.loggers
 module, 192
biothings.utils.manager
 module, 195
biothings.utils.mongo
 module, 198
biothings.utils.parallel
 module, 211
biothings.utils.parallel_mp
 module, 211
biothings.utils.parsers
 module, 213
biothings.utils.redis
 module, 214
biothings.utils.serializer
 module, 214
biothings.utils.sqlite3
 module, 215
biothings.utils.version
 module, 217
biothings.web
 module, 122
biothings.web.analytics
 module, 126
biothings.web.analytics.channels
 module, 126
biothings.web.analytics.events
 module, 127
biothings.web.analytics.notifiers
 module, 128
biothings.web.applications
 module, 123
biothings.web.connections
 module, 126
biothings.web.handlers
 module, 128

biothings.web.handlers.base
 module, 128
biothings.web.handlers.query
 module, 130
biothings.web.handlers.services
 module, 132
biothings.web.launcher
 module, 122
biothings.web.options
 module, 132
biothings.web.options.manager
 module, 132
biothings.web.options.openapi
 module, 135
biothings.web.query
 module, 141
biothings.web.query.builder
 module, 141
biothings.web.query.engine
 module, 143
biothings.web.query.formatter
 module, 144
biothings.web.query.pipeline
 module, 147
biothings.web.services
 module, 124
biothings.web.services.health
 module, 124
biothings.web.services.metadata
 module, 124
biothings.web.services.namespace
 module, 125
biothings.web.services.query
 module, 124
biothings.web.settings
 module, 148
biothings.web.settings.configs
 module, 148
biothings.web.settings.default
 module, 149
biothings.web.settings.validators
 module, 149
biothings.web.templates
 module, 149
BiothingsAPI (in module *biothings.web.applications*),
 123
BiothingsAPIBaseLauncher (class in *bioth-
 ings.web.launcher*), 122
BiothingsAPIEdge (class in *bioth-
 ings.hub.datatransform.datatransform_api*),
 287
BiothingsAPILauncher (in module *bioth-
 ings.web.launcher*), 122
BiothingsDataTest (in module *biothings.tests.web*),
 151
BiothingsDBProxy (class in *bioth-
 ings.web.services.namespace*), 125
BiothingsDumper (class in *bioth-
 ings.hub.autoupdate.dumper*), 228
BiothingsESMetadata (class in *bioth-
 ings.web.services.metadata*), 125
BiothingsJSONEncoder (class in *bioth-
 ings.utils.common*), 156
BiothingsMetadata (class in *bioth-
 ings.web.services.metadata*), 125
BiothingsMongoMetadata (class in *bioth-
 ings.web.services.metadata*), 125
BiothingsNamespace (class in *bioth-
 ings.web.services.namespace*), 125
BiothingsSQLMetadata (class in *bioth-
 ings.web.services.metadata*), 125
BiothingsTestCase (in module *biothings.tests.web*),
 151
BiothingsUploader (class in *bioth-
 ings.hub.autoupdate.uploader*), 230
BiothingsWebAppTest (class in *biothings.tests.web*),
 151
BiothingsWebTest (class in *biothings.tests.web*), 152
BiothingsWebTestBase (class in *biothings.tests.web*),
 152
BLOCK_SIZE (*biothings.hub.dataload.dumper.FTPDumper*
 attribute), 268
bm (built-in variable), 296
boolean_convert() (in module *bioth-
 ings.utils.dataload*), 165
breadth_first_recursive_traversal() (in module
biothings.utils.doc_traversal), 171
breadth_first_traversal() (in module *bioth-
 ings.utils.doc_traversal*), 171
break_received() (method),
 219
broadcast() (*biothings.web.analytics.notifiers.Notifier*
 method), 128
bucket (*biothings.hub.dataindex.snapshooter.RepositoryConfig*
 property), 255
Bucket (class in *biothings.hub.dataindex.snapshooter*),
 255
build(), 297
build() (*biothings.utils.loggers.SlackMessage* method),
 194
build() (*biothings.web.query.builder.ESQueryBuilder*
 method), 142
build() (*biothings.web.query.builder.MongoQueryBuilder*
 method), 142
build() (*biothings.web.query.builder.SQLQueryBuilder*
 method), 143

build_config(*biothings.hub.databuild.builder.DataBuilder property*), 236
build_config_info(), 297
build_config_info() (*biothings.hub.databuild.builder.BuilderManager method*), 234
build_diff_report() (*biothings.hub.databuild.differ.DifferManager method*), 241
build_id(*biothings.hub.datarelease.releasenote.ReleaseNoteSrcBuilder* property), 286
build_index() (*biothings.utils.es.ESIndexer method*), 174
build_info() (*biothings.hub.databuild.builder.BuilderManager method*), 234
build_release_note() (*biothings.hub.datarelease.publisher.ReleaseManager method*), 284
build_save_mapping(), 297
build_stats(*biothings.hub.datarelease.releasenote.ReleaseNoteSrcBuilder* property), 286
build_stats(*biothings.hub.datarelease.releasenote.ReleaseNoteSrcBuilder* property), 286
BUILD_TYPES(*biothings.hub.databuild.buildconfig.AutoBuildConfigCompat attribute*), 233
build_version (*biothings.hub.datarelease.releasenote.ReleaseNoteSrcBuilder* property), 286
BuilderException, 234
BuilderManager (class in *biothings.hub.databuild.builder*), 234
builds(), 297
bulk_write() (*biothings.utils.sqlite3.Collection method*), 215

C

cache (*biothings.web.handlers.base.BaseAPIHandler attribute*), 128
cache_control_template (*biothings.web.handlers.base.BaseAPIHandler attribute*), 128
call() (*biothings.hub.dataload.dumper.DumperManager method*), 267
can_handle() (*biothings.hub.dataplug.in.assistant.BaseAssistant method*), 279
can_handle() (*biothings.hub.dataplug.in.assistant.GithubAssistant method*), 280
can_handle() (*biothings.hub.dataplug.in.assistant.LocalAssistant method*), 280
can_load_plugin() (*biothings.hub.dataplug.in.assistant.AdvancedPluginLoader method*), 278
can_load_plugin() (*biothings.hub.dataplug.in.assistant.BasePluginLoader method*), 278

method), 280
can_load_plugin() (*biothings.hub.dataplug.in.assistant.ManifestBasedPluginLoader method*), 281
capturesESEExceptions() (in module *biothings.web.query.pipeline*), 148
category(*biothings.hub.datarelease.publisher.BasePublisher property*), 282
catalog (*biothings.hub.dataindex.indexer.Step at-build*)
ChangeListener (class in *biothings.utils.hub_db*), 181
ChangeWatcher (class in *biothings.utils.hub_db*), 181
Channel (class in *biothings.web.analytics.channels*), 126
check() (*biothings.utils.redis.RedisClient method*), 214
check() (*biothings.web.services.health.DBHealth method*), 124
check() (*biothings.web.services.health.EsHealth method*), 124
check() (*biothings.web.services.health.MongoHealth method*), 124
check() (*biothings.web.services.health.SQLHealth method*), 124
check_compat() (*biothings.hub.autoupdate.dumper.BiothingsDumper method*), 229
check_constraints() (*biothings.utils.manager.JobManager method*), 197
check_document_size() (in module *biothings.utils.mongo*), 209
check_index() (*biothings.utils.es.ESIndexer method*), 174
check_new_version() (in module *biothings.utils.version*), 217
check_ready() (*biothings.hub.databuild.builder.DataBuilder method*), 236
check_ready() (*biothings.hub.dataload.uploader.BaseSourceUploader method*), 274
check_ready() (*biothings.hub.dataload.uploader.DummySourceUploader method*), 275
CHILD_CONTEXTS (*biothings.web.options.openapi.OpenAPIContext attribute*), 136
CHILD_CONTEXTS (*biothings.web.options.openapi.OpenAPIInfoContext attribute*), 137
CHILD_CONTEXTS (*biothings.web.options.openapi.OpenAPIPathItemContext attribute*), 140
choose_best_version() (*biothings.web.options.openapi.OpenAPIPathItemContext attribute*), 140

ings.hub.autoupdate.dumper.BiothingsDumper method), 229	method), 284
CIIDStruct (class in bioth- ings.hub.datatransform.ciidstruct), 287	ings.utils.manager.BaseManager method), 195
CIMongoDBEdge (class in bioth- ings.hub.datatransform.datatransform_mdb), 289	clean_staled() (biothings.utils.manager.JobManager method), 197
clean() (biothings.hub.dataindex.indexer_cleanup.Cleaner method), 248	clean_temp_collections() (bioth- ings.hub.databuild.builder.BuilderManager method), 234
clean_archived_collections() (bioth- ings.hub.autoupdate.uploader.BiothingsUploader method), 230	Cleaner (class in bioth- ings.hub.dataindex.indexer_cleanup), 248
clean_archived_collections() (bioth- ings.hub.dataload.uploader.BaseSourceUploader method), 274	cleanup() (biothings.hub.dataindex.IndexManager method), 249
clean_data() (in module biothings.cli.datapluging), 308	cleanup() (biothings.hub.dataindex.snapshotmanager.SnapshotManager method), 256
clean_data() (in module biothings.cli.datapluging_hub), 309	CleanUpResult (class in bioth- ings.hub.dataindex.indexer_cleanup), 248
clean_features() (biothings.hub.HubServer method), 222	clear() (biothings.utils.configuration.MetaField method), 164
clean_field() (biothings.utils.es.ESIndexer method), 174	client (biothings.hub.dataload.dumper.APIDumper property), 260
clean_old_collections() (bioth- ings.hub.databuild.builder.DataBuilder method), 236	client (biothings.hub.dataload.dumper.BaseDumper property), 261
clean_stale_status() (bioth- ings.hub.databuild.builder.BuilderManager method), 234	client (biothings.hub.datatransform.datatransform_api.BiothingsAPIEdge property), 287
clean_stale_status() (bioth- ings.hub.databuild.differ.DifferManager method), 241	client (biothings.utils.redis.RedisClient attribute), 214
clean_stale_status() (bioth- ings.hub.databuild.syncer.SyncerManager method), 246	client_name (biothings.hub.datatransform.datatransform_api.BiothingsA attribute), 287
clean_stale_status() (bioth- ings.hub.dataindex.indexer.IndexManager method), 249	client_name (biothings.hub.datatransform.datatransform_api.MyChemIn attribute), 289
clean_stale_status() (bioth- ings.hub.dataindex.snapshotmanager.SnapshotManager method), 256	client_name (biothings.hub.datatransform.datatransform_api.MyGeneIn attribute), 289
clean_stale_status() (bioth- ings.hub.datainspect.inspector.InspectorManager method), 259	clients (biothings.hub.api.handlers.ws.WebSocketConnection attribute), 228
clean_stale_status() (bioth- ings.hub.dataload.dumper.DumperManager method), 267	CLIJobManager (class in biothings.utils.manager), 196
clean_stale_status() (bioth- ings.hub.dataload.uploader.UploaderManager method), 277	close() (biothings.utils.common.LogPrint method), 157
clean_stale_status() (bioth- ings.hub.datarelease.publisher.BasePublisher method), 282	close() (biothings.utils.es.ESIndexer method), 174
clean_stale_status() (bioth- ings.hub.datarelease.publisher.ReleaseManager	CloudStorage (class in bioth- ings.hub.dataindex.snapshotmanager.SnapshotManager), 255
	cmd (biothings.utils.hub.HubShell attribute), 179
	cmd_cnt (biothings.utils.hub.HubShell attribute), 179
	code (biothings.web.query.pipeline.QueryPipelineException attribute), 148
	col_entity (biothings.utils.hub_db.ChangeWatcher at- tribute), 181
	cold_collection_name (bioth- ings.hub.datarelease.releasenote.ReleaseNoteSrcBuildReader property), 286
	ColdHotDiffer (class in bioth- ings.hub.databuild.differ), 239
	ColdHotIndexer (class in bioth- ings.hub.dataindex.indexer), 248
	ColdHotJsonDiffer (class in bioth- ings.hub.databuild.differ), 240

ColdHotJsonDifferBase (class in biothings.hub.databuild.differ), 240
ColdHotSelfContainedJsonDiffer (class in biothings.hub.databuild.differ), 241
collapse() (biothings.web.query.formatter.FormatterDict method), 147
collection (biothings.hub.datarelease.publisher.BasePublisher.property), 282
collection (biothings.hub.datarelease.publisher.ReleaseManager.property), 284
collection (biothings.hub.datatransform.datatransform_manager.property), 291
collection (biothings.utils.manager.BaseStatusRegisterer.property), 196
Collection (class in biothings.utils.es), 173
Collection (class in biothings.utils.hub_db), 181
Collection (class in biothings.utils.mongo), 198
Collection (class in biothings.utils.sqlite3), 215
collection_find() (biothings.hub.datatransform.datatransform_mdb.CIMongoDBEdge_file_handler().method), 290
collection_find() (biothings.hub.datatransform.datatransform_mdb.MongoDBEdge method), 291
collection_names() (biothings.utils.hub_db.IDatabase method), 182
collection_names() (biothings.utils.mongo.Database method), 200
collection_names() (biothings.utils.mongo.DummyDatabase method), 208
collection_names() (biothings.utils.sqlite3.Database method), 216
collection_partition() (in module biothings.utils.parallel), 211
Colors (class in biothings.utils.loggers), 192
COLUMNS (biothings.utils.manager.JobManager attribute), 196
command(), 297
command_info() (biothings.utils.hub.HubShell class method), 179
CommandDefinition (class in biothings.utils.hub), 178
CommandError, 178
CommandInformation (class in biothings.utils.hub), 178
CommandNotAllowed, 178
commands(), 297
commit() (biothings.utils.configuration.ConfigAttrMeta method), 162
compare_remote_local() (biothings.hub.autoupdate.dumper.BiothingsDumper method), 229
completed() (biothings.hub.dataindex.indexer_schedule.Schedule method), 253
compose_dot_fields_by_fields() (in module biothings.utils.dotfield), 172
CompositeCommand (class in biothings.utils.hub), 178
compute_metadata() (in module biothings.utils.inspect), 186
config (biothings.hub.dataload.uploader.BaseSourceUploader attribute), 274
config (biothings.tests.web.BiothingsWebAppTest property), 151
CONFIG (biothings.utils.es.Database attribute), 173
CONFIG (biothings.utils.sqlite3.Database attribute), 216
ConfigAttrMeta (class in biothings.utils.configuration), 162
ConfigLine (class in biothings.utils.configuration), 163
ConfigLines (class in biothings.utils.configuration), 163
ConfigModule (class in biothings.web.settings.configs), 148
ConfigPackage (class in biothings.web.settings.configs), 148
ConfigurationDefault (class in biothings.utils.configuration), 163
ConfigurationError, 163
ConfigurationValue (class in biothings.utils.configuration), 163
ConfigurationWrapper (class in biothings.utils.configuration), 163
configure() (biothings.hub.databuild.builder.BuilderManager method), 235
configure() (biothings.hub.databuild.differ.DifferManager method), 241
configure() (biothings.hub.databuild.syncer.SyncerManager method), 246
configure() (biothings.hub.dataindex.indexer.IndexManager method), 250
configure() (biothings.hub.dataindex.snapshooter.SnapshotManager method), 257
configure() (biothings.hub.dataplugin.assistant.AssistantManager method), 278
configure() (biothings.hub.datarelease.publisher.ReleaseManager method), 284
configure() (biothings.hub.HubServer method), 222
configure() (biothings.hub.standalone.AutoHubFeature method), 295
configure() (biothings.web.services.namespace.BiothingsDBProxy method), 125
configure_api_endpoints() (biothings.hub.HubServer method), 222
configure_api_manager() (biothings.hub.HubServer method), 222
configure_auto_archive_manager() (biothings.hub.HubServer method), 222
configure_auto_release() (biothings.hub.HubServer method), 222

<code>ings.hub.standalone.AutoHubFeature method), 295</code>	<code>configure_upload_manager()</code> (<i>biothings.hub.HubServer method</i>), 223
<code>configure_auto_snapshot_cleaner_manager()</code> (<i>biothings.hub.HubServer method</i>), 222	<code>configure_ws_feature()</code> (<i>biothings.hub.HubServer method</i>), 223
<code>configure_auhub_feature()</code> (<i>biothings.hub.HubServer method</i>), 222	<code>confmod</code> (<i>biothings.utils.configuration.ConfigAttrMeta attribute</i>), 162
<code>configure_build_manager()</code> (<i>biothings.hub.HubServer method</i>), 222	<code>connection_lost()</code> (<i>biothings.hub.HubSSHServer method</i>), 218
<code>configure_commands()</code> (<i>biothings.hub.HubServer method</i>), 222	<code>connection_made()</code> (<i>biothings.hub.HubSSHServer method</i>), 218
<code>configure_config_feature()</code> (<i>biothings.hub.HubServer method</i>), 222	<code>connection_made()</code> (<i>biothings.hub.HubSSHServerSession method</i>), 220
<code>configure_dataplug_in_manager()</code> (<i>biothings.hub.HubServer method</i>), 222	<code>contact()</code> (<i>biothings.web.options.openapi.OpenAPIInfoContext method</i>), 137
<code>configure_dataupload_feature()</code> (<i>biothings.hub.HubServer method</i>), 222	<code>COUNTAINER_NAME</code> (<i>biothings.hub.dataload.dumper.DockerContainerDumper attribute</i>), 265
<code>configure_diff_manager()</code> (<i>biothings.hub.HubServer method</i>), 222	<code>convert()</code> (<i>biothings.web.options.manager.Converter method</i>), 132
<code>configure_dump_manager()</code> (<i>biothings.hub.HubServer method</i>), 222	<code>convert_to()</code> (<i>biothings.web.options.manager.Converter method</i>), 132
<code>configure_extra_commands()</code> (<i>biothings.hub.HubServer method</i>), 222	<code>convert_to()</code> (<i>biothings.web.options.manager.FormArgCvter method</i>), 133
<code>configure_hooks_feature()</code> (<i>biothings.hub.HubServer method</i>), 222	<code>convert_to()</code> (<i>biothings.web.options.manager.JsonArgCvter method</i>), 133
<code>configure_index_manager()</code> (<i>biothings.hub.HubServer method</i>), 222	<code>Converter</code> (class in <i>biothings.web.options.manager</i>), 132
<code>configure_inspect_manager()</code> (<i>biothings.hub.HubServer method</i>), 222	<code>CopyOperation</code> (class in <i>biothings.utils.jsonpatch</i>), 188
<code>configure_ioloop()</code> (<i>biothings.hub.HubServer method</i>), 222	<code>count()</code> (<i>biothings.utils.backend.DocESBackend method</i>), 154
<code>configure_job_manager()</code> (<i>biothings.hub.HubServer method</i>), 223	<code>count()</code> (<i>biothings.utils.backend.DocMongoBackend method</i>), 155
<code>configure_managers()</code> (<i>biothings.hub.HubServer method</i>), 223	<code>count()</code> (<i>biothings.utils.es.Collection method</i>), 173
<code>configure_readonly_api_endpoints()</code> (<i>biothings.hub.HubServer method</i>), 223	<code>count()</code> (<i>biothings.utils.es.ESIndexer method</i>), 174
<code>configure_readonly_feature()</code> (<i>biothings.hub.HubServer method</i>), 223	<code>count()</code> (<i>biothings.utils.hub_db.Collection method</i>), 181
<code>configure_release_manager()</code> (<i>biothings.hub.HubServer method</i>), 223	<code>count()</code> (<i>biothings.utils.mongo.Collection method</i>), 199
<code>configure_reloader_feature()</code> (<i>biothings.hub.HubServer method</i>), 223	<code>count()</code> (<i>biothings.utils.mongo.DummyCollection method</i>), 208
<code>configure_remaining_features()</code> (<i>biothings.hub.HubServer method</i>), 223	<code>count()</code> (<i>biothings.utils.sqlite3.Collection method</i>), 215
<code>configure_snapshot_manager()</code> (<i>biothings.hub.HubServer method</i>), 223	<code>count_from_ids()</code> (<i>biothings.utils.backend.DocMongoBackend method</i>), 155
<code>configure_source_manager()</code> (<i>biothings.hub.HubServer method</i>), 223	<code>count_src()</code> (<i>biothings.utils.es.ESIndexer method</i>), 174
<code>configure_sync_manager()</code> (<i>biothings.hub.HubServer method</i>), 223	<code>create()</code> (<i>biothings.hub.dataindex.indexer.DynamicIndexerFactory method</i>), 249
<code>configure_terminal_feature()</code> (<i>biothings.hub.HubServer method</i>), 223	<code>create()</code> (<i>biothings.hub.dataindex.snapshooter.Bucket method</i>), 255
<code>configure_upgrade_feature()</code> (<i>biothings.hub.HubServer method</i>), 223	<code>create()</code> (<i>biothings.hub.dataindex.snapshot_repo.Repository method</i>), 258
	<code>create()</code> (<i>biothings.hub.dataindex.snapshot_task.Snapshot method</i>), 259
	<code>create()</code> (<i>biothings.hub.dataload.uploader.BaseSourceUploader class method</i>), 274

create_and_call()	(<i>biothings.hub.dataload.dumper.DumperManager method</i>), 267	create_release_note()	(<i>biothings.hub.datarelease.publisher.ReleaseManager method</i>), 284
create_and_dump()	(<i>biothings.hub.dataload.dumper.DumperManager method</i>), 267	create_release_note_from_build()	(<i>biothings.hub.datarelease.publisher.ReleaseManager method</i>), 284
create_and_load()	(<i>biothings.hub.dataload.uploader.UploaderManager method</i>), 277	create_repository()	(<i>biothings.utils.es.ESIndexer method</i>), 174
create_and_update_master()	(<i>biothings.hub.dataload.uploader.UploaderManager method</i>), 277	create_todump_list()	(<i>biothings.hub.autoupdate.dumper.BiothingsDumper method</i>), 229
create_api()	297	create_todump_list()	(<i>biothings.hub.dataload.dumper.APIDumper method</i>), 260
create_api()	(<i>biothings.hub.api.manager.APIManager method</i>), 224	create_todump_list()	(<i>biothings.hub.dataload.dumper.BaseDumper method</i>), 261
create_backend()	(in module <i>biothings.hub.databuild.backend</i>), 232	create_todump_list()	(<i>biothings.hub.dataload.dumper.DockerContainerDumper method</i>), 265
create_bucket()	(<i>biothings.hub.datarelease.publisher.BasePublisher method</i>), 282	create_todump_list()	(<i>biothings.hub.dataload.dumper.LastModifiedBaseDumper method</i>), 270
create_bucket()	(<i>in module biothings.utils.aws</i>), 153	create_todump_list()	(<i>biothings.hub.dataload.dumper.WgetDumper method</i>), 272
create_build_conf()	298	CRITICAL	(<i>biothings.utils.loggers.Colors attribute</i>), 192
create_build_configuration()	(<i>biothings.hub.databuild.builder.BuilderManager method</i>), 235	CRITICAL	(<i>biothings.utils.loggers.Squares attribute</i>), 194
create_collection()	(<i>biothings.utils.es.Database method</i>), 173	cron_and_strdelta_info()	(<i>biothings.hub.JobRenderer method</i>), 223
create_collection()	(<i>biothings.utils.hub_db.IDatabase method</i>), 182	CumulativeResult	(class in <i>biothings.hub.dataindex.snapshooter</i>), 255
create_collection()	(<i>biothings.utils.sqlite3.Database method</i>), 216	current_data_folder	(<i>biothings.hub.dataload.dumper.BaseDumper property</i>), 261
create_data_plugin()	(in module <i>biothings.cli.dataplugin</i>), 308	current_release	(<i>biothings.hub.dataload.dumper.BaseDumper property</i>), 261
create_data_plugin()	(in module <i>biothings.cli.dataplugin_hub</i>), 309	Cursor	(class in <i>biothings.utils.sqlite3</i>), 215
create_from_options()	(<i>biothings.utils.backend.DocESBackend method</i>), 154	CWD_DIR	(<i>biothings.hub.dataload.dumper.FTPDumper attribute</i>), 268
create_handlers()	(<i>in module biothings.hub.api</i>), 224	D	
create_if_needed()	(<i>biothings.utils.sqlite3.Database method</i>), 216	DATA_PATH	(<i>biothings.hub.dataload.dumper.DockerContainerDumper attribute</i>), 265
create_index()	(<i>biothings.utils.es.ESIndexer method</i>), 174	DATA_PLUGIN_FOLDER	(<i>biothings.hub.dataplugin.assistant.AssistedDumper attribute</i>), 279
create_instance()	(<i>biothings.hub.dataload.dumper.DumperManager method</i>), 267	DATA_PLUGIN_FOLDER	(<i>biothings.hub.dataplugin.assistant.AssistedUploader attribute</i>), 279
create_instance()	(<i>biothings.hub.dataload.uploader.UploaderManager method</i>), 277	data_plugin_manager	(<i>biothings.hub.dataplugin.assistant.BaseAssistant</i>
create_instance()	(<i>biothings.hub.dataplugin.assistant.AssistantManager method</i>), 278		
create_logger()	(<i>in module biothings.utils.loggers</i>), 195		

attribute), 279	
data_received() (biothings.hub.api.handlers.upload.UploadHandler method), 227	DBHealth (<i>class in biothings.web.services.health</i>), 124
data_received() (biothings.hub.HubSSHServerSession method), 220	DBParamValidator (<i>class in biothings.web.settings.validators</i>), 149
database (<i>biothings.utils.sqlite3.Collection</i> property), 215	debug (<i>biothings.hub.datatransform.datatransform.DataTransform</i> attribute), 292
Database (<i>class in biothings.utils.es</i>), 173	DEBUG (<i>biothings.utils.loggers.Colors</i> attribute), 192
Database (<i>class in biothings.utils.mongo</i>), 199	DEBUG (<i>biothings.utils.loggers.Squares</i> attribute), 194
Database (<i>class in biothings.utils.sqlite3</i>), 216	DeepStatsMode (<i>class in biothings.utils.inspect</i>), 184
database() (<i>biothings.utils.hub_db.Collection</i> method), 181	default (<i>biothings.utils.configuration.Flag</i> attribute), 164
DatabaseClient (<i>class in biothings.utils.mongo</i>), 200	default (<i>biothings.utils.configuration.MetaField</i> attribute), 164
DatabaseClient (<i>class in biothings.utils.sqlite3</i>), 216	default (<i>biothings.utils.configuration.Paragraph</i> attribute), 164
DatabaseCollectionTesting (<i>class in biothings.tests.hub</i>), 150	default() (<i>biothings.utils.common.BiothingsJSONEncoder</i> method), 157
DataBuilder (<i>class in biothings.hub.databuild.builder</i>), 236	DEFAULT_API_CONFIG (<i>biothings.hub.HubServer</i> attribute), 221
DATALINE (<i>biothings.utils.manager.JobManager</i> attribute), 197	DEFAULT_AUTOHUB_CONFIG (<i>biothings.hub.HubServer</i> attribute), 221
DataPluginManager (<i>class in biothings.hub.dataplugmananager</i>), 282	DEFAULT_BRANCH (<i>biothings.hub.dataload.dumper.GitDumper</i> attribute), 269
datasource_info (<i>biothings.hub.datarelease.releasenote.ReleaseNoteSrc</i> property), 286	DEFAULT_DATAUPLOAD_CONFIG (<i>biothings.hub.HubServer</i> attribute), 221
datasource_mapping (<i>biothings.hub.datarelease.releasenote.ReleaseNoteSrc</i> property), 286	DEFAULT_DATEPUBLISHER_CLASS (<i>biothings.hub.datarelease.publisher.ReleaseManager</i> attribute), 284
datasource_stats (<i>biothings.hub.datarelease.releasenote.ReleaseNoteSrc</i> property), 286	DEFAULT_DUMPCLASS (<i>biothings.hub.standalone.AutoHubFeature</i> attribute), 295
datasource_versions (<i>biothings.hub.datarelease.releasenote.ReleaseNoteSrc</i> property), 286	DEFAULTFEATURES (<i>biothings.hub.HubServer</i> attribute), 221
DataTransform (<i>class in biothings.hub.datatransform.datatransform</i>), 291	DEFAULT_FEATURES (<i>biothings.hub.standalone.AutoHubServer</i> attribute), 296
DataTransformAPI (<i>class in biothings.hub.datatransform.datatransform_api</i>), 287	DEFAULT_INDEXER (<i>biothings.hub.dataindex.indexer.IndexManager</i> attribute), 249
DataTransformEdge (<i>class in biothings.hub.datatransform.datatransform</i>), 292	DEFAULT_MANAGERS_ARGS (<i>biothings.hub.HubServer</i> attribute), 221
DataTransformMDB (<i>class in biothings.hub.datatransform</i>), 121	default_match_query() (<i>biothings.web.query.builder.ESQueryBuilder</i> method), 142
DataTransformMDB (<i>class in biothings.hub.datatransform.datatransform_mdb</i>), 290	DEFAULT_RELOADER_CONFIG (<i>biothings.hub.HubServer</i> attribute), 222
DataTransformMyChemInfo (<i>class in biothings.hub.datatransform.datatransform_api</i>), 288	DEFAULT_SNAPSHOT_PUBLISHER_CLASS (<i>biothings.hub.datarelease.publisher.ReleaseManager</i> attribute), 284
DataTransformMyGeneInfo (<i>class in biothings.hub.datatransform.datatransform_api</i>), 288	default_source (<i>biothings.hub.datatransform.datatransform.DataTransform</i> attribute), 292
	default_source (<i>biothings.hub.datatransform.datatransform_api.DataTransformAPI</i> attribute), 292

attribute), 288
default_source (biothings.hub.datatransform.datatransform_mdb.DataTransformMDB attribute), 290
default_string_query() (biothings.hub.web.query.builder.ESQueryBuilder method), 142
DEFAULT_UPLOADER_CLASS (biothings.hub.standalone.AutoHubFeature attribute), 295
DEFAULT_VALIDATOR_CLASS (biothings.hub.standalone.AutoHubFeature attribute), 295
DEFAULT_WEBSOCKET_CONFIG (biothings.hub.HubServer attribute), 222
DEFAULT_WEIGHT (biothings.hub.datatransform.datatransform.DataTransform attribute), 292
DefaultCORSHeaderMixin (class in biothings.hub.api.handlers.log), 226
DefaultHandler (class in biothings.hub.api.handlers.base), 225
DEFAULTS (biothings.web.analytics.events.Message attribute), 127
defer_to_process() (biothings.utils.manager.CLIJobManager method), 196
defer_to_process() (biothings.utils.manager.JobManager method), 197
defer_to_thread() (biothings.utils.manager.CLIJobManager method), 196
defer_to_thread() (biothings.utils.manager.JobManager method), 197
delete() (biothings.hub.api.handlers.base.GenericHandler method), 226
delete() (biothings.hub.dataindex.snapshot_repo.Repository method), 258
delete() (biothings.hub.dataindex.snapshot_task.Snapshot method), 259
delete() (biothings.hub.dataload.sync.MongoSync method), 273
delete() (biothings.web.options.openapi.OpenAPIPathItemContext method), 140
delete() (in module biothings.hub.dataindex.snapshot_cleanup), 257
delete_api(), 298
delete_api() (biothings.hub.api.manager.APIManager method), 224
delete_build_conf(), 298
delete_build_configuration() (biothings.hub.databuild.builder.BuilderManager method), 235
delete_doc() (biothings.utils.es.ESIndexer method), 174
delete_docs() (biothings.utils.es.ESIndexer method), 174
delete_index() (biothings.utils.es.ESIndexer method), 174
delete_merge() (biothings.hub.databuild.builder.BuilderManager method), 235
delete_merged_data() (biothings.hub.databuild.builder.BuilderManager method), 235
delete_or_restore_container() (biothings.hub.dataload.dumper.DockerContainerDumper method), 265
delete_snapshots() (biothings.hub.dataindex.snapshooter.SnapshotManager method), 257
deprecated() (biothings.web.options.openapi.OpenAPIParameterContext method), 139
depth_first_recursive_traversal() (in module biothings.utils.doc_traversal), 171
depth_first_traversal() (in module biothings.utils.doc_traversal), 171
description(biothings.utils.configuration.ConfigAttrMeta attribute), 162
details(biothings.web.query.pipeline.QueryPipelineException attribute), 148
DevInfo (class in biothings.utils.info), 183
dict_apply() (in module biothings.utils.dataload), 165
dict_attrmerge() (in module biothings.utils.dataload), 165
dict_convert() (in module biothings.utils.dataload), 165
dict_nodup() (in module biothings.utils.dataload), 165
dict_sweep() (in module biothings.utils.dataload), 165
dict_to_list() (in module biothings.utils.dataload), 166
dict_traverse() (in module biothings.utils.dataload), 166
dict_walk() (in module biothings.utils.dataload), 166
diff(), 298
diff() (biothings.hub.databuild.differ.BaseDiffer method), 239
diff() (biothings.hub.databuild.differ.DifferManager method), 242
diff_build_stats() (biothings.hub.datarelease.releasenote.ReleaseNoteSource method), 286
diff_collections() (in module biothings.utils.diff), 170
diff_collections_batches() (in module biothings.utils.diff), 170

diff_cols() (*biothings.hub.databuild.differ.BaseDiffer method*), 239
 diff_cols() (*biothings.hub.databuild.differ.ColdHotDiffer method*), 239
 diff_datasource_info() (*biothings.hub.datarelease.releasenote.ReleaseNoteSource method*), 286
 diff_datasource_mapping() (*biothings.hub.datarelease.releasenote.ReleaseNoteSource method*), 286
 diff_docs_jsonpatch() (*in module biothings.utils.diff*), 170
 diff_info(), 298
 diff_info() (*biothings.hub.databuild.differ.DifferManager method*), 242
 diff_report() (*biothings.hub.databuild.differ.DifferManager method*), 242
 diff_type (*biothings.hub.databuild.differ.BaseDiffer attribute*), 239
 diff_type (*biothings.hub.databuild.differ.ColdHotJsonDiff attribute*), 240
 diff_type (*biothings.hub.databuild.differ.ColdHotSelfContainedJsonDiff attribute*), 241
 diff_type (*biothings.hub.databuild.differ.JsonDiffer attribute*), 242
 diff_type (*biothings.hub.databuild.differ.SelfContainedJsonDiffer attribute*), 242
 diff_type (*biothings.hub.databuild.syncer.BaseSyncer attribute*), 244
 diff_type (*biothings.hub.databuild.syncer.ESColdHotJsonDiff attribute*), 245
 diff_type (*biothings.hub.databuild.syncer.ESColdHotJsonSyncer attribute*), 245
 diff_type (*biothings.hub.databuild.syncer.ESJsonDiff attribute*), 245
 diff_type (*biothings.hub.databuild.syncer.MongoJsonDiff attribute*), 245
 diff_worker_count() (*in module biothings.hub.databuild.differ*), 242
 diff_worker_new_vs_old() (*in module biothings.hub.databuild.differ*), 242
 diff_worker_old_vs_new() (*in module biothings.hub.databuild.differ*), 242
 DifferException, 241
 DifferManager (*class in biothings.hub.databuild.differ*), 241
 DiffPublisher (*class in biothings.hub.datarelease.publisher*), 283
 DiffReportRendererBase (*class in biothings.hub.databuild.differ*), 241
 DiffReportTxt (*class in biothings.hub.databuild.differ*), 241
 dim (*built-in variable*), 298
 dispatch() (*biothings.hub.dataindex.indexer.Step class method*), 252
 dispatch() (*biothings.hub.dataindex.indexer_task.IndexingTask method*), 254
 dispatch() (*in module biothings.hub.dataindex.indexer_task*), 254
 dispatch() (*in module biothings.hub.dataindex.snapshot_registrar*), 258
 dm (*built-in variable*), 298
 do_clean() (*in module biothings.cli.utils*), 310
 do_clean_dumped_files() (*in module biothings.cli.utils*), 310
 do_clean_uploaded_sources() (*in module biothings.cli.utils*), 310
 do_create() (*in module biothings.cli.utils*), 310
 do_dump() (*biothings.hub.dataload.dumper.BaseDumper method*), 262
 do_dump_and_upload() (*in module biothings.cli.utils*), 310
 do_index() (*biothings.hub.dataindex.indexer.Indexer method*), 251
 do_inspect() (*in module biothings.cli.utils*), 310
 do_list() (*in module biothings.cli.utils*), 310
 do_publish (*biothings.utils.hub_db.ChangeWatcher attribute*), 154
 do_serve() (*in module biothings.cli.utils*), 310
 do_upload() (*in module biothings.cli.utils*), 311
 do_work() (*in module biothings.utils.manager*), 198
 doc_feeder() (*biothings.utils.es.ESIndexer method*), 144
 doc_feeder() (*in module biothings.utils.mongo*), 209
 doc_feeder() (*in module biothings.utils.es*), 174
 doc_feeder() (*in module biothings.utils.es.ESIndexer method*), 174
 DocBackendBase (*class in biothings.utils.backend*), 154
 DocBackendOptions (*class in biothings.utils.backend*), 154
 DocESBackend (*class in biothings.utils.backend*), 154
 DocHandler (*class in biothings.cli.web_app*), 312
 DOCKER_CLIENT_URL (*biothings.hub.dataload.dumper.DockerContainerDumper attribute*), 265
 DOCKER_IMAGE (*biothings.hub.dataload.dumper.DockerContainerDumper attribute*), 265
 docker_source_info_parser() (*in module biothings.utils.parsers*), 213
 DockerContainerDumper (*class in biothings.hub.databuild.differ*), 241

ings.hub.dataload.dumper), 263

DockerContainerException, 267

DocMemoryBackend (*class in biothings.utils.backend*), 155

DocMongoBackend (*class in biothings.utils.backend*), 155

DocMongoDBBackend (*in module biothings.utils.backend*), 156

document_cleaner() (*biothings.hub.databuild.builder.DataBuilder method*), 236

DONE (*biothings.hub.dataindex.indexer_registrar.Stage attribute*), 253

dotdict (*class in biothings.utils.common*), 158

download(), 299

download() (*biothings.hub.autoupdate.dumper.BiothingsDumper method*), 229

download() (*biothings.hub.dataload.dumper.APIDumper method*), 260

download() (*biothings.hub.dataload.dumper.BaseDumper method*), 262

download() (*biothings.hub.dataload.dumper.DockerContainerDumper method*), 266

download() (*biothings.hub.dataload.dumper.FilesystemDumper method*), 268

download() (*biothings.hub.dataload.dumper.FTPDumper method*), 268

download() (*biothings.hub.dataload.dumper.GitDumper method*), 269

download() (*biothings.hub.dataload.dumper.GoogleDriveDumper method*), 269

download() (*biothings.hub.dataload.dumper.HTTPDumper method*), 270

download() (*biothings.hub.dataload.dumper.LastModifiedFTPDumper method*), 271

download() (*biothings.hub.dataload.dumper.WgetDumper method*), 272

download_s3_file() (*in module biothings.utils.aws*), 153

dpm (*built-in variable*), 299

drop() (*biothings.hub.databuild.backend.LinkTargetDocManager method*), 231

drop() (*biothings.utils.backend.DocBackendBase method*), 154

drop() (*biothings.utils.backend.DocESBackend method*), 154

drop() (*biothings.utils.backend.DocMemoryBackend method*), 155

drop() (*biothings.utils.backend.DocMongoBackend method*), 155

drop() (*biothings.utils.mongo.DummyCollection method*), 208

drop() (*biothings.utils.sqlite3.Collection method*), 215

DummyCollection (*class in biothings.utils.mongo*), 208

DummyConfig (*class in biothings.utils.common*), 157

DummyDatabase (*class in biothings.utils.mongo*), 208

DummyDumper (*class in biothings.hub.dataload.dumper*), 267

DummySourceUploader (*class in biothings.hub.dataload.uploader*), 275

dump(), 299

dump() (*biothings.hub.dataload.dumper.BaseDumper method*), 262

dump() (*biothings.hub.dataload.dumper.DummyDumper method*), 267

dump() (*biothings.hub.dataload.dumper.GitDumper method*), 269

dump() (*biothings.hub.dataload.dumper.ManualDumper method*), 272

Dump() (*biothings.hub.dataplug.in.manager.ManualDataPlugin method*), 282

dump() (*in module biothings.utils.common*), 158

dump2gridfs() (*in module biothings.utils.common*), 158

dump_all(), 299

dump_all() (*biothings.hub.dataload.dumper.DumperManager method*), 265

dump_and_upload() (*in module biothings.cli.dataplug.in*), 308

dump_and_upload() (*in module biothings.cli.dataplug.in_hub*), 309

DUMP_COMMAND (*biothings.hub.dataload.dumper.DockerContainerDumper attribute*), 265

dump_data() (*in module biothings.cli.dataplug.in*), 308

DumpedData() (*in module biothings.cli.dataplug.in_hub*), 309

dump_info(), 299

dump_info() (*biothings.hub.dataload.dumper.DumperManager method*), 267

dump_plugin(), 299

dump_src() (*biothings.hub.dataload.dumper.DumperManager method*), 267

dumper_manager (*biothings.hub.dataplug.in.assistant.BaseAssistant attribute*), 279

DumperRegistry (*biothings.hub.dataplug.in.assistant.ManifestBasedPluginLoader attribute*), 281

DumperException, 267

DumperManager (*class in biothings.hub.dataload.dumper*), 267

dupline_seperator() (*in module biothings.utils.dataload*), 166

DynamicIndexerFactory (*class in biothings.hub.dataindex.indexer*), 248

E

edge_lookup() (*biothings.hub.datatransform.datatransform.DataTransformEdge*)

method), 292
edge_lookup() *(biothings.hub.datatransform.datatransform.RegExEdge method), 294*
edge_lookup() *(biothings.hub.datatransform.datatransform_api.BiothingsAPIEdge method), 287*
edge_lookup() *(biothings.hub.datatransform.datatransform_mdb.MongoDBEdge method), 291*
email() *(biothings.web.options.openapi.OpenAPIContactContext method), 135*
emit() *(biothings.utils.loggers.EventRecorder method), 193*
emit() *(biothings.utils.loggers.SlackHandler method), 194*
emit() *(biothings.utils.loggers.WSLogHandler method), 194*
end *(biothings.utils.loggers.Range attribute), 193*
EndpointDefinition *(class in biothings.hub.api), 224*
EndScrollInterrupt, 144
eof_received() *(biothings.hub.HubSSHServerSession method), 220*
ERROR *(biothings.utils.loggers.Colors attribute), 192*
ERROR *(biothings.utils.loggers.Squares attribute), 194*
ErrorHandler *(class in biothings.utils.parallel_mp), 211*
es *(in module biothings.web.connections), 126*
ESColdHotJsonDiffSelfContainedSyncer *(class in biothings.hub.databuild.syncer), 245*
ESColdHotJsonDiffSyncer *(class in biothings.hub.databuild.syncer), 245*
ESHealth *(class in biothings.web.services.health), 124*
ESIndex *(class in biothings.hub.dataindex.indexer_task), 254*
ESIndex *(class in biothings.utils.es), 174*
ESIndexer *(class in biothings.utils.es), 174*
ESJsonDiffSelfContainedSyncer *(class in biothings.hub.databuild.syncer), 245*
ESJsonDiffSyncer *(class in biothings.hub.databuild.syncer), 245*
ESQueryBackend *(class in biothings.web.query.engine), 144*
ESQueryBuilder *(class in biothings.web.query.builder), 141*
ESQueryPipeline *(class in biothings.web.query.pipeline), 147*
ESResultFormatter *(class in biothings.web.query.formatter), 144*
ESScrollID *(class in biothings.web.query.builder), 142*
ESUserQuery *(class in biothings.web.query.builder), 142*
ETAG *(biothings.hub.dataload.dumper.LastModifiedHTTPD attribute), 271*
eval() *(biothings.utils.hub.HubShell method), 179*
eval_lines() *(biothings.hub.HubSSHServerSession method), 220*
(biothings.web.analytics.events), 127
event_queue *(biothings.utils.hub_db.ChangeWatcher attribute), 181*
(biothings.utils.loggers.EventRecorder), 193
exclude_ids() *(biothings.web.query.formatter.FormatterDict method), 147*
exclude_from_reloader() *(in module biothings.hub.ExcludeFieldsById), 180*
ExcludeFieldsById *(class in biothings.hub.HubSSHServerSession method), 220*
execute() *(biothings.hub.dataindex.indexer.Step method), 252*
execute() *(biothings.web.query.engine.AsyncESQueryBackend method), 143*
execute() *(biothings.web.query.engine.ESQueryBackend method), 144*
execute() *(biothings.web.query.engine.MongoQueryBackend method), 144*
execute() *(biothings.web.query.engine.SQLQueryBackend method), 144*
Existentialist *(class in biothings.web.options.manager), 133*
exists() *(biothings.hub.dataindex.snapshooter.Bucket method), 255*
exists() *(biothings.hub.dataindex.snapshot_repo.Repository method), 258*
exists() *(biothings.hub.dataindex.snapshot_task.Snapshot method), 259*
exists() *(biothings.utils.es.ESIndexer method), 174*
exists_index() *(biothings.utils.es.ESIndexer method), 174*
exists_or_null() *(in module biothings.utils.docs), 171*
explode() *(biothings.web.options.openapi.OpenAPIParameterContext method), 139*
export() *(biothings.hub.databuild.buildconfig.AutoBuildConfig method), 233*
export() *(biothings.hub.dataplugin.assistant.AssistantManager method), 278*
export_command_documents(), 299
export_command_documents() *(biothings.hub.HubServer method), 223*
export_dumper() *(biothings.hub.dataplugin.assistant.AssistantManager method), 278*
export_ids() *(in module biothings.hub.dataexport.ids), 247*
export_mapping() *(biothings.hub.dataplugin.assistant.AssistantManager method), 278*
export_plugin(), 299

export_uploader() (biothings.hub.dataplugintassistant.AssistantManager method), 279
expose(), 299
EXTENSION (biothings.web.options.openapi.OpenAPIContext attribute), 135
EXTENSION (biothings.web.options.openapi.OpenAPIContext attribute), 136
EXTENSION (biothings.web.options.openapi.OpenAPIExternal attribute), 136
EXTENSION (biothings.web.options.openapi.OpenAPIInfoOf attribute), 137
EXTENSION (biothings.web.options.openapi.OpenAPILicense attribute), 138
EXTENSION (biothings.web.options.openapi.OpenAPIOperat attribute), 138
EXTENSION (biothings.web.options.openapi.OpenAPIParameterContext attribute), 139
EXTENSION (biothings.web.options.openapi.OpenAPIPathItemContext attribute), 140
extract() (biothings.hub.standalone.AutoHubFeature method), 295
extract_command_name() (biothings.utils.hub.HubShell method), 179
extract_pending_info() (biothings.utils.manager.JobManager method), 197
extract_worker_info() (biothings.utils.manager.JobManager method), 197
extras() (biothings.web.handlers.query.MetadataSourceHandler method), 131

F

failed() (biothings.hub.dataindex.indexer_registrar.IndexJobStateRegistrar method), 252
FastAPIBiothingsAPI (class in biothings.web.applications), 123
FastAPILauncher (class in biothings.web.launcher), 122
feed() (biothings.utils.configuration.ConfigAttrMeta method), 162
feed() (biothings.utils.configuration.Flag method), 164
feed() (biothings.utils.configuration.MetaField method), 164
feed() (biothings.utils.configuration.Paragraph method), 164
feed() (biothings.utils.configuration.Text method), 164
fetch() (biothings.web.query.pipeline.AsyncESQueryPipeline method), 147
fetch() (biothings.web.query.pipeline.ESQueryPipeline method), 148
fetch() (biothings.web.query.pipeline.QueryPipeline method), 148

field_name (biothings.utils.inspect.FieldInspection attribute), 184
field_type (biothings.utils.inspect.FieldInspection attribute), 184
FieldInspection (class in biothings.utils.inspect), 184
FieldInspectValidation (class in biothings.utils.inspect), 184
FieldNote (class in biothings.utils.info), 183
file_deleteme() (in module biothings.utils.dataload), 166
file_newer() (in module biothings.utils.common), 158
find() (biothings.utils.common.LogPrint method), 157
FilesystemDumper (class in biothings.hub.dataload.dumper), 268
filter() (biothings.hub.dataload.uploader.UploaderManager method), 277
filter_class() (biothings.utils.filter.Filter class), 277
filter_dict() (in module biothings.utils.common), 158
finalize() (biothings.hub.dataindex.indexer_payload.IndexMappings method), 248
finalize() (biothings.hub.dataindex.indexer_payload.IndexSettings method), 248
finalize() (biothings.utils.backend.DocBackendBase method), 154
finalize() (biothings.utils.backend.DocESBackend method), 154
finalize() (biothings.utils.backend.DocMemoryBackend method), 155
finalize() (biothings.utils.backend.DocMongoBackend method), 155
find() (biothings.hub.dataindex.indexer_cleanup.Cleaner method), 248
find() (biothings.hub.datatransform.ciidstruct.CIIDStruct method), 287
find() (biothings.hub.datatransform.datatransform.IDStruct static method), 293
find() (biothings.utils.es.Collection method), 173
find() (biothings.utils.hub_db.Collection method), 181
find() (biothings.utils.loggers.LookUpList method), 193
find() (biothings.utils.sqlite3.Collection method), 215
find() (in module biothings.hub.dataindex.snapshot_cleanup), 257
find_biggest_doc() (biothings.utils.es.ESIndexer method), 174
find_builder_classes() (biothings.hub.databuild.builder.BuilderManager method), 235
find_classes() (biothings.utils.manager.BaseSourceManager method), 196
find_classes_subclassing() (in module biothings.utils.manager.BaseSourceManager), 196

ings.utils.common), 158

find_doc() (*in module biothings.utils.common*), 158

find_index() (*biothings.utils.loggers.LookUpList method*), 193

find_left() (*biothings.hub.datatransform.datatransform.IDStruct method*), 293

find_one() (*biothings.utils.es.Collection method*), 173

find_one() (*biothings.utils.hub_db.Collection method*), 182

find_one() (*biothings.utils.sqlite3.Collection method*), 215

find_process() (*in module biothings.utils.manager*), 198

find_right() (*biothings.hub.datatransform.datatransform.IDStruct method*), 293

find_sources() (*biothings.hub.dataload.source.SourceManager method*), 273

find_update_path() (*biothings.hub.autoupdate.dumper.BiothingsDumper method*), 229

find_value_in_doc() (*in module biothings.utils.common*), 158

findv2() (*biothings.utils.sqlite3.Collection method*), 215

fix_batch_duplicates() (*in module biothings.hub.databuild.builder*), 238

Flag (*class in biothings.utils.configuration*), 164

FlaskAPILauncher (*class in biothings.web.launcher*), 122

FlaskBiothingsAPI (*class in biothings.web.applications*), 123

flatten() (*biothings.hub.datainspect.inspector.InspectorManager method*), 259

flatten_and_validate() (*in module biothings.utils.inspect*), 186

flatten_doc() (*in module biothings.utils.docs*), 171

flatten_doc_2() (*in module biothings.utils.docs*), 171

flatten_inspection_data() (*in module biothings.utils.inspect*), 186

flatten_stats() (*biothings.utils.inspect.StatsMode method*), 186

float_convert() (*in module biothings.utils.dataload*), 166

flush() (*biothings.utils.common.LogPrint method*), 157

flush_and_refresh() (*biothings.utils.es.ESIndexer method*), 174

FormArgCvter (*class in biothings.web.options.manager*), 133

format (*biothings.web.handlers.base.BaseAPIHandler attribute*), 128

format() (*biothings.hub.dataindex.snapshooter.RepositoryConfig method*), 255

FormatterDict (*class in biothings*)

ings.web.query.formatter), 147

from_diff() (*biothings.utils.jsonpatch.JsonPatch class method*), 189

from_string() (*biothings.utils.jsonpatch.JsonPatch IDStruct class method*), 190

FrontPageHandler (*class in biothings.web.handlers.services*), 132

FS_OP (*biothings.hub.dataload.dumper.FilesystemDumper attribute*), 268

FTP_HOST (*biothings.hub.dataload.dumper.FTPDumper attribute*), 268

FTP_PASWD (*biothings.hub.dataload.dumper.FTPDumper attribute*), 268

FTPS_TIMEOUT (*biothings.hub.dataload.dumper.FTPDumper attribute*), 268

FTP_USER (*biothings.hub.dataload.dumper.FTPDumper attribute*), 268

FTPDumper (*class in biothings.hub.dataload.dumper*), 268

fullname (*biothings.hub.dataload.uploader.BaseSourceUploader property*), 274

func (*biothings.hub.dataindex.snapshot_registrar.MainSnapshotState attribute*), 258

func (*biothings.hub.dataindex.snapshot_registrar.PostSnapshotState attribute*), 258

func (*biothings.hub.dataindex.snapshot_registrar.PreSnapshotState attribute*), 258

G

g (*built-in variable*), 300

GA4Channel (*class in biothings.web.analytics.channels*), 126

GACChannel (*class in biothings.web.analytics.channels*), 127

GAEEvent (*class in biothings.web.analytics.events*), 127

generate_api_routes() (*in module biothings.hub.api*), 224

generate_doc_src_master() (*biothings.hub.dataload.uploader.BaseSourceUploader method*), 274

generate_document_query() (*biothings.hub.databuild.builder.DataBuilder method*), 236

generate_endpoint_for_callable() (*in module biothings.hub.api*), 224

generate_endpoint_for_composite_command() (*in module biothings.hub.api*), 224

generate_endpoint_for_display() (*in module biothings.hub.api*), 224

generate_es_mapping() (*in module biothings.utils.es*), 177

generate_folder() (*in module biothings.hub.databuild.backend*), 232

```
generate_handler() (in module biothings.hub.api), 224
generate_json_schema() (in module biothings.utils.jsonschema), 192
generate_remote_file() (biothings.hub.dataload.dumper.DockerContainerDumper method), 266
generate_target_name() (biothings.hub.databuild.backend.TargetDocBackend method), 232
GenericHandler (class in biothings.hub.api.handlers.base), 225
get() (biothings.cli.web_app.DocHandler method), 312
get() (biothings.cli.web_app.HomeHandler method), 312
get() (biothings.cli.web_app.QueryHandler method), 312
get() (biothings.hub.api.handlers.base.GenericHandler method), 226
get() (biothings.hub.api.handlers.base.RootHandler method), 226
get() (biothings.hub.api.handlers.log.HubLogDirHandler method), 226
get() (biothings.hub.api.handlers.log.HubLogFileHandler method), 226
get() (biothings.hub.dataindex.snapshooter.CloudStorage method), 255
get() (biothings.utils.info.DevInfo method), 183
get() (biothings.web.handlers.query.BiothingHandler method), 130
get() (biothings.web.handlers.query.MetadataFieldHandler method), 131
get() (biothings.web.handlers.query.MetadataSourceHandler method), 131
get() (biothings.web.handlers.query.QueryHandler method), 131
get() (biothings.web.handlers.services.APISpecificationHandler method), 132
get() (biothings.web.handlers.services.FrontPageHandler method), 132
get() (biothings.web.handlers.services.StatusHandler method), 132
get() (biothings.web.options.openapi.OpenAPIPathItemCog method), 140
get_alias() (biothings.utils.es.ESIndexer method), 175
get_all() (biothings.hub.databuild.prebuilder.BasePreCompiler method), 244
get_all() (biothings.hub.databuild.prebuilder.MongoDBPreCompiler method), 244
get_all() (biothings.hub.databuild.prebuilder.RedisPreCompiledDataBuilder method), 244
get_api() (in module biothings.utils.es), 177
get_api() (in module biothings.utils.hub_db), 183
get_api() (in module biothings.utils.mongo), 209
get_api() (in module biothings.utils.sqlite3), 216
get_apis(), 300
get_apis() (biothings.hub.api.manager.APIManager method), 224
get_app() (biothings.tests.web.BiothingsWebAppTest method), 151
get_app() (biothings.web.applications.FastAPIBiothingsAPI class method), 123
get_app() (biothings.web.applications.FlaskBiothingsAPI class method), 123
get_app() (biothings.web.applications.TornadoBiothingsAPI class method), 123
get_app() (biothings.web.launcher.BiothingsAPILauncher method), 122
get_app() (biothings.web.launcher.FastAPILauncher method), 122
get_app() (biothings.web.launcher.FlaskAPILauncher method), 122
get_app() (biothings.web.launcher.TornadoAPILauncher method), 123
get_async_client() (biothings.web.connections._ClientPool method), 126
get_available_routes() (in module biothings.cli.web_app), 312
get_backend() (in module biothings.utils.diff), 170
get_backend_provider_info() (biothings.hub.datainspect.inspector.InspectorManager method), 259
get_backend_url() (biothings.hub.databuild.backend.LinkTargetDocMongoBackend method), 231
get_backend_url() (biothings.hub.databuild.backend.TargetDocBackend method), 232
get_biothing() (biothings.utils.es.ESIndexer method), 175
get_biothings_commit() (in module biothings.utils.version), 217
get_build_configuration() (biothings.hub.databuild.backend.SourceDocBackendBase method), 231
get_build_configuration() (biothings.hub.databuild.backend.SourceDocMongoBackend method), 231
get_builder() (biothings.hub.databuild.builder.DataBuilder method), 231
get_builder_data_provider() (biothings.hub.databuild.builder.DataBuilder method), 231
get_builder_data_provider() (biothings.hub.databuild.builder.BuilderManager method), 235
get_builder_class() (biothings.hub.databuild.builder.BuilderManager method), 235
```

get_cache_filename() (in module <code>biothings.utils.mongo</code>), 209	get_document() (biothings.hub.dataload.dumper.APIDumper static method), 260
get_class_from_classpath() (in module <code>biothings.utils.common</code>), 158	get_document_id() (biothings.hub.dataload.dumper.GoogleDriveDumper method), 270
get_class_name() (biothings.hub.standalone.AutoHubFeature method), 295	get_dotfield_value() (in module <code>biothings.utils.common</code>), 159
get_classdef() (biothings.hub.datapluging.assistant.GithubAssistant method), 280	get_dumper_dynamic_class() (biothings.hub.datapluging.assistant.ManifestBasedPluginLoader method), 281
get_classdef() (biothings.hub.datapluging.assistant.LocalAssistant method), 281	get_es() (in module <code>biothings.utils.es</code>), 177
get_client() (biothings.utils.redis.RedisClient class method), 214	get_es_client() (in module <code>biothings.web.connections</code>), 126
get_client() (biothings.web.connections._ClientPool method), 126	get_event() (in module <code>biothings.utils.es</code>), 177
get_client_for_url() (biothings.hub.dataload.dumper.LastModifiedFTPDumper method), 271	get_event() (in module <code>biothings.utils.hub_db</code>), 183
get_cmd() (in module <code>biothings.utils.es</code>), 177	get_event() (in module <code>biothings.utils.mongo</code>), 209
get_cmd() (in module <code>biothings.utils.hub_db</code>), 183	get_event() (in module <code>biothings.utils.sqlite3</code>), 216
get_cmd() (in module <code>biothings.utils.mongo</code>), 209	get_example_queries() (in module <code>biothings.cli.web_app</code>), 312
get_cmd() (in module <code>biothings.utils.sqlite3</code>), 216	get_field_notes() (biothings.utils.info.FieldNote method), 183
get_code_for_mod_name() (biothings.hub.datapluging.assistant.ManifestBasedPluginLoader method), 281	get_filter() (biothings.web.query.builder.ESUserQuery method), 142
get_compressed_outfile() (in module <code>biothings.utils.common</code>), 158	get_folder_name() (biothings.hub.standalone.AutoHubFeature method), 295
get_conn() (biothings.utils.sqlite3.Collection method), 215	get_from_id() (biothings.utils.backend.DocBackendBase method), 154
get_conn() (biothings.utils.sqlite3.Database method), 216	get_from_id() (biothings.utils.backend.DocESBackend method), 154
get_conn() (in module <code>biothings.utils.mongo</code>), 209	get_from_id() (biothings.utils.backend.DocMemoryBackend method), 155
get_converters() (in module <code>biothings.utils.inspect</code>), 186	get_from_id() (biothings.utils.backend.DocMongoBackend method), 156
get_current_and_new_master() (biothings.hub.dataload.uploader.BaseSourceUploader method), 274	get_hub_config() (in module <code>biothings.utils.es</code>), 177
get_custom_metadata() (biothings.hub.databuild.builder.DataBuilder method), 236	get_hub_config() (in module <code>biothings.utils.hub_db</code>), 183
get_data_plugin() (in module <code>biothings.utils.es</code>), 177	get_hub_config() (in module <code>biothings.utils.mongo</code>), 209
get_data_plugin() (in module <code>biothings.utils.hub_db</code>), 183	get_hub_config() (in module <code>biothings.utils.sqlite3</code>), 216
get_data_plugin() (in module <code>biothings.utils.mongo</code>), 209	get_hub_db_conn() (in module <code>biothings.utils.es</code>), 177
get_data_plugin() (in module <code>biothings.utils.sqlite3</code>), 216	get_hub_db_conn() (in module <code>biothings.utils.mongo</code>), 209
get_db() (<code>biothings.utils.redis.RedisClient</code> method), 214	get_hub_db_conn() (in module <code>biothings.utils.sqlite3</code>), 216
get_debug() (biothings.hub.datatransform.datatransform.IDStruct method), 293	get_hub_reloader() (in module <code>biothings.utils.hub</code>), 180
get_doc_type() (in module <code>biothings.utils.es</code>), 177	get_id_list() (biothings.utils.es.ESIndexer method), 175

```

        ings.utils.backend.DocBackendBase method), get_mapping() (biothings.utils.es.ESIndexer method),
        154                                         175
get_id_list()                                (bioth- get_mapping_meta() (biothings.utils.es.ESIndexer
                                                ings.utils.backend.DocESBackend method), 175
                                                154
get_id_list()                                (bioth- get_mappings() (bioth-
                                                ings.utils.backend.DocMemoryBackend ings.web.services.metadata.BiothingsMetadata
                                                method), 155                                         method), 125
get_id_list()                                (bioth- get_mappings() (bioth-
                                                ings.utils.backend.DocMongoBackend ings.web.services.metadata.BiothingsMongoMetadata
                                                method), 156                                         method), 125
get_id_list() (biothings.utils.es.ESIndexer method), get_metadata() (bioth-
        175                                         ings.hub.databuild.differ.BaseDiffer method),
get_indexes_by_name() (bioth- get_metadata() (bioth-
                                                ings.hub.dataindex.indexer.IndexManager ings.hub.databuild.differ.ColdHotDiffer
                                                method), 250                                         method), 240
get_indices_names_by_settings() (bioth- get_metadata() (bioth-
                                                ings.utils.es.ESIndexer method), 175 ings.web.services.metadata.BiothingsMetadata
get_indices_from_snapshots() (bioth- get_mode_layer() (in module biothings.utils.inspect),
                                                ings.utils.es.ESIndexer method), 175 186
get_internal_number_of_replicas() (bioth- get_mongo_client() (in module bioth-
                                                ings.utils.es.ESIndexer method), 175 ings.web.connections), 126
get_last_command() (in module biothings.utils.es), get_mongodb_uri() (in module biothings.utils.diff),
        177                                         170
get_last_command() (in module bioth- get_new_ioloop() (bioth-
                                                ings.utils.mongo), 209 ings.tests.web.BiothingsWebAppTest method),
get_last_command() (in module bioth- get_pending_processes() (bioth-
                                                ings.utils.sqlite3), 216 ings.utils.manager.JobManager method),
get_licenses() (bioth- get_pending_summary() (bioth-
                                                ings.web.services.metadata.BiothingsMetadata ings.utils.manager.JobManager
                                                method), 125                                         method),
get_licenses() (bioth- get_pid_files() (bioth-
                                                ings.web.services.metadata.BiothingsMongoMetadata ings.utils.manager.JobManager
                                                method), 125                                         method),
get_loadjson() (in module biothings.utils.jsonpatch), get_pinfo() (biothings.hub.databuild.builder.DataBuilder
        192                                         method), 236
get_log_content() (in module bioth- get_pinfo() (biothings.hub.databuild.differ.BaseDiffer
                                                ings.hub.api.handlers.log), 226                                         method), 239
get_logger() (in module biothings.cli.utils), 311 get_pinfo() (biothings.hub.databuild.differ.DifferManager
get_logger() (in module biothings.utils.loggers), 195                                         method), 242
get_loop() (in module biothings.utils.common), 159 get_pinfo() (biothings.hub.databuild.syncer.BaseSyncer
get_loop_with_max_workers() (in module bioth-                                         method), 244
                                                ings.utils.common), 159
get_manifest_content() (in module bioth- get_pinfo() (biothings.hub.dataindex.indexer.IndexManager
                                                ings.cli.utils), 311                                         method), 250
get_mapper_for_source() (bioth- get_pinfo() (biothings.hub.dataindex.indexer.ProcessInfo
                                                ings.hub.databuild.builder.DataBuilder                                         method), 252
                                                method), 236
get_mapping() (bioth- get_pinfo() (biothings.hub.dataindex.snapshooter.ProcessInfo
                                                ings.hub.databuild.builder.DataBuilder                                         method), 255
                                                method), 236
get_mapping() (bioth- get_pinfo() (biothings.hub.dataload.dumper.BaseDumper
                                                ings.hub.dataload.uploader.BaseSourceUploader                                         method), 262
                                                class method), 274
                                                get_pinfo() (biothings.hub.dataload.uploader.BaseSourceUploader

```

`method), 274`
`get_pinfo() (biothings.hub.datarelease.publisher.BasePublisher method), 282`
`get_pinfo() (biothings.hub.datarelease.publisher.ReleaseManager method), 284`
`get_plugin_name() (in module biothings.cli.utils), 311`
`get_plugin_name_from_local_manifest() (in module biothings.utils.common), 159`
`get_plugin_name_from_remote_manifest() (in module biothings.utils.common), 159`
`get_plugin_obj() (biothings.hub.dataplugin.assistant.BasePluginLoader method), 280`
`get_pre_post_previous_result() (biothings.hub.datarelease.publisher.BasePublisher method), 282`
`get_pre_post_previous_result() (biothings.hub.datarelease.publisher.DiffPublisher method), 283`
`get_pre_post_previous_result() (biothings.hub.datarelease.publisher.SnapshotPublisher method), 285`
`get_predicates() (biothings.hub.databuild.builder.DataBuilder method), 236`
`get_predicates() (biothings.hub.databuild.differ.BaseDiffer method), 239`
`get_predicates() (biothings.hub.databuild.differ.DifferManager method), 242`
`get_predicates() (biothings.hub.databuild.syncer.BaseSyncer method), 244`
`get_predicates() (biothings.hub.databuild.syncer.ThrottlerSyncer method), 246`
`get_predicates() (biothings.hub.dataindex.indexer.IndexManager method), 250`
`get_predicates() (biothings.hub.dataindex.indexer.ProcessInfo method), 252`
`get_predicates() (biothings.hub.dataindex.snapshooter.ProcessInfo method), 255`
`get_predicates() (biothings.hub.dataload.dumper.BaseDumper method), 262`
`get_predicates() (biothings.hub.dataload.uploader.BaseSourceUploader method), 274`
`get_predicates() (biothings.hub.datarelease.publisher.BasePublisher method), 283`
`get_predicates() (biothings.hub.datarelease.publisher.ReleaseManager method), 284`
`get_previous_collection() (in module biothings.utils.mongo), 209`
`get_process_summary() (biothings.utils.manager.JobManager method), 197`
`get_python_exec_version() (in module biothings.utils.version), 217`
`get_python_version() (in module biothings.utils.version), 217`
`get_query() (biothings.web.query.builder.ESUserQuery method), 142`
`get_query_for_list_merge() (biothings.hub.databuild.builder.BuilderManager method), 235`
`get_random_string() (in module biothings.utils.common), 159`
`get_release() (biothings.hub.dataload.dumper.APIDumper static method), 261`
`get_release_note(), 300`
`get_release_note() (biothings.hub.datarelease.publisher.ReleaseManager method), 284`
`get_release_note_filename() (biothings.hub.datarelease.publisher.BasePublisher method), 283`
`get_release_note_filename() (biothings.hub.datarelease.publisher.DiffPublisher method), 283`
`get_remote_file() (biothings.hub.dataload.dumper.DockerContainerDumper method), 266`
`get_remote_file() (biothings.hub.dataload.dumper.LastModifiedFTPDumper method), 271`
`get_remote_lastmodified() (biothings.hub.dataload.dumper.DockerContainerDumper method), 266`
`get_repository() (biothings.utils.es.ESIndexer method), 175`
`get_repository_information() (in module biothings.utils.version), 217`
`get_restore_status() (biothings.utils.es.ESIndexer method), 175`
`get_root_document_sources() (biothings.hub.databuild.builder.DataBuilder method), 237`
`get_s3_file() (in module biothings.utils.aws), 153`
`get_s3_file_contents() (in module biothings.utils.aws), 153`

get_s3_folder() (in module `biothings.utils.aws`), 153
get_s3_static_website_url() (in module `biothings.utils.aws`), 153
get_s3_url() (in module `biothings.utils.aws`), 153
get_schedule() (biothings.hub.dataloader.DumperManager method), 267
get_schedule() (in module `biothings.hub`), 224
get_server() (biothings.web.launcher.BiothingsAPILauncher method), 122
get_server() (biothings.web.launcher.FlaskAPILauncher method), 123
get_server() (biothings.web.launcher.TornadoAPILauncher method), 123
get_settings() (biothings.utils.es.ESIndexer method), 175
get_snapshot_repository_config() (biothings.hub.autoupdate.uploader.BiothingsUploader method), 230
get_snapshot_status() (biothings.utils.es.ESIndexer method), 175
get_snapshots() (biothings.utils.es.ESIndexer method), 175
get_software_info() (in module `biothings.utils.version`), 217
get_source() (biothings.hub.dataloader.source.SourceManager method), 273
get_source_code_info() (in module `biothings.utils.version`), 217
get_source_fullname() (in module `biothings.utils.es`), 177
get_source_fullname() (in module `biothings.utils.mongo`), 209
get_source_fullname() (in module `biothings.utils.sqlite3`), 216
get_source_fullnames() (in module `biothings.utils.mongo`), 210
get_source_ids() (biothings.hub.dataloader.dumper.DumperManager method), 267
get_source_ids() (biothings.hub.dataloader.uploader.UploaderManager method), 277
get_sources() (biothings.hub.dataloader.source.SourceManager method), 273
get_sql_client() (in module `biothings.web.connections`), 126
get_src_build() (in module `biothings.utils.es`), 177
get_src_build() (in module `biothings.utils.hub_db`), 183
get_src_build() (in module `biothings.utils.mongo`), 210
get_src_build() (in module `biothings.utils.sqlite3`), 216
get_src_build_config() (in module `biothings.utils.es`), 177
get_src_build_config() (in module `biothings.utils.hub_db`), 183
get_src_build_config() (in module `biothings.utils.mongo`), 210
get_src_build_config() (in module `biothings.utils.sqlite3`), 216
get_src_conn() (in module `biothings.utils.es`), 177
get_src_conn() (in module `biothings.utils.mongo`), 210
get_src_conn() (in module `biothings.utils.sqlite3`), 216
get_src_db() (in module `biothings.utils.es`), 177
get_src_db() (in module `biothings.utils.mongo`), 210
get_src_db() (in module `biothings.utils.sqlite3`), 216
get_src_dump() (in module `biothings.utils.es`), 177
get_src_dump() (in module `biothings.utils.hub_db`), 183
get_src_dump() (in module `biothings.utils.mongo`), 210
get_src_dump() (in module `biothings.utils.sqlite3`), 216
get_src_master() (in module `biothings.utils.es`), 177
get_src_master() (in module `biothings.utils.hub_db`), 183
get_src_master() (in module `biothings.utils.mongo`), 210
get_src_master() (in module `biothings.utils.sqlite3`), 216
get_src_master_docs() (biothings.hub.databuild.backend.SourceDocBackendBase method), 231
get_src_master_docs() (biothings.hub.databuild.backend.SourceDocMongoBackend method), 231
get_src_metadata() (biothings.hub.databuild.backend.SourceDocBackendBase method), 231
get_src_metadata() (biothings.hub.databuild.backend.SourceDocMongoBackend method), 231
get_stats() (biothings.hub.databuild.builder.DataBuilder method), 237
get_summary() (biothings.utils.manager.JobManager method), 197
get_target_backend() (biothings.hub.autoupdate.dumper.BiothingsDumper method), 229
get_target_backend() (biothings.hub.databuild.syncer.BaseSyncer method), 244
get_target_conn() (in module `biothings.utils.mongo`), 210
get_target_db() (in module `biothings.utils.mongo`), 210
get_target_master() (in module `biothings.utils.mongo`), 210

ings.utils.mongo), 210

get_target_name() (*biothings.hub.databuild.builder.DataBuilder method*), 237

get_template_path() (*biothings.web.handlers.base.BaseAPIHandler method*), 128

get_template_path() (*biothings.web.handlers.services.FrontPageHandler method*), 132

get_thread_files() (*biothings.utils.manager.JobManager method*), 197

get_thread_summary() (*biothings.utils.manager.JobManager method*), 197

get_timestamp() (*in module biothings.utils.common*), 159

get_uploaded_collections() (*in module biothings.cli.utils*), 311

get_uploader_dynamic_class() (*biothings.hub.dataplugin.assistant.ManifestBasedPluginLoader method*), 281

get_uploader_dynamic_classes() (*biothings.hub.dataplugin.assistant.ManifestBasedPluginLoader method*), 281

get_uploaders() (*in module biothings.cli.utils*), 311

get_url() (*biothings.tests.web.BiothingsWebAppTest method*), 151

get_url() (*biothings.tests.web.BiothingsWebTestBase method*), 152

get_value() (*biothings.utils.configuration.ConfigurationValue method*), 163

get_value_from_db() (*biothings.utils.configuration.ConfigurationWrapper method*), 163

get_value_from_file() (*biothings.utils.configuration.ConfigurationWrapper method*), 163

get_version() (*in module biothings.utils.version*), 217

GET_VERSION_CMD (*biothings.hub.dataload.dumper.DockerContainerDumper attribute*), 265

get_websocket_urls() (*biothings.hub.HubServer method*), 223

GIT_REPO_URL (*biothings.hub.dataload.dumper.GitDumper attribute*), 269

GitDataPlugin (*class in biothings.hub.dataplugin.manager*), 282

GitDumper (*class in biothings.hub.dataload.dumper*), 269

GithubAssistant (*class in biothings.hub.dataplugin.assistant*), 280

GoogleDriveDumper (*class in biothings.hub.dataload.dumper*), 269

ings.hub.dataload.dumper), 269

Group (*class in biothings.web.query.builder*), 142

gunzip() (*in module biothings.utils.common*), 159

gunzipall() (*in module biothings.utils.common*), 159

GZipRotator (*class in biothings.utils.loggers*), 193

H

handle() (*biothings.hub.dataplugin.assistant.BaseAssistant method*), 279

handle() (*biothings.hub.dataplugin.assistant.GithubAssistant method*), 280

handle() (*biothings.hub.dataplugin.assistant.LocalAssistant method*), 281

handle() (*biothings.utils.parallel_mp.ErrorHandler method*), 211

handle_autoreconnect() (*in module biothings.utils.mongo*), 210

HandleAutoReconnectMixin (*class in biothings.utils.mongo*), 208

handles() (*biothings.web.analytics.channels.Channel method*), 126

handles() (*biothings.web.analytics.channels.GA4Channel method*), 126

handles() (*biothings.web.analytics.channels.GAChannel method*), 127

handles() (*biothings.web.analytics.channels.SlackChannel method*), 127

has_cold_collection() (*biothings.hub.datarelease.releasenote.ReleaseNoteSrcBuildReader method*), 286

has_filter() (*biothings.web.query.builder.ESUserQuery method*), 142

has_multiple_types (*biothings.utils.inspect.FieldInspectValidation attribute*), 184

has_query() (*biothings.web.query.builder.ESUserQuery method*), 142

hash() (*biothings.web.connections._ClientPool static method*), 126

head() (*biothings.hub.api.handlers.base.GenericHandler method*), 226

head() (*biothings.web.handlers.services.StatusHandler method*), 132

head() (*biothings.web.options.openapi.OpenAPIPathItemContext method*), 140

HEADER (*biothings.utils.manager.JobManager attribute*), 197

HEADERLINE (*biothings.utils.manager.JobManager attribute*), 197

help(), 300

help() (*biothings.utils.hub.HubShell method*), 179

hidden (*biothings.utils.configuration.ConfigAttrMeta attribute*), 162

Histogram (class in *biothings.hub.datatransform.histogram*), 294
Hits (class in *biothings.web.query.formatter*), 147
HomeHandler (class in *biothings.cli.web_app*), 312
host (*biothings.tests.web.BiothingsWebTestBase* attribute), 152
http_method (*biothings.web.options.openapi.OpenAPIPathItemContext* method), 140
HTTPDumper (class in *biothings.hub.dataload.dumper*), 270
hub_memory (*biothings.utils.manager.JobManager* property), 197
hub_process (*biothings.utils.manager.JobManager* property), 197
HubCommands (class in *biothings.hub*), 217
HubDBListener (class in *biothings.hub.api.handlers.ws*), 227
HubLogDirHandler (class in *biothings.hub.api.handlers.log*), 226
HubLogFileHandler (class in *biothings.hub.api.handlers.log*), 226
HubServer (class in *biothings.hub*), 221
HubShell (class in *biothings.utils.hub*), 178
HubSSHSERVER (class in *biothings.hub*), 217
HubSSHSERVERSession (class in *biothings.hub*), 219

|
id_feeder() (in module *biothings.utils.mongo*), 210
id_lst (*biothings.hub.datatransform.datatransform.IDStruct* property), 293
id_priority_list (*biothings.hub.datatransform.datatransform.DataTransform* property), 292
id_strip() (in module *biothings.utils.dataload*), 166
IDatabase (class in *biothings.utils.hub_db*), 182
IDBaseMapper (class in *biothings.hub.databuild.mapper*), 243
IDCache (class in *biothings.hub.dataindex.idcache*), 247
IdentifiersMode (class in *biothings.utils.inspect*), 184
ids (*biothings.utils.inspect.IdentifiersMode* attribute), 184
IDStruct (class in *biothings.hub.datatransform.datatransform*), 293
IGNORE_HTTP_CODE (*biothings.hub.dataload.dumper.HTTPDumper* attribute), 270
IgnoreDuplicatedSourceUploader (class in *biothings.hub.dataload.uploader*), 275
im (built-in variable), 300
import_debug() (*biothings.hub.datatransform.datatransform.IDStruct* method), 293
include() (*biothings.web.query.formatter.FormatterDict* method), 147
INDEX (*biothings.hub.dataindex.indexer_task.Mode* attribute), 254
index(), 300
index() (*biothings.hub.dataindex.indexer.ColdHotIndexer* method), 248
index() (*biothings.hub.dataindex.indexer.Indexer* method), 251
index() (*biothings.hub.dataindex.indexer.IndexManager* method), 250
index() (*biothings.hub.dataindex.indexer_task.IndexingTask* method), 254
index() (*biothings.utils.es.ESIndexer* method), 176
index_bulk() (*biothings.utils.es.ESIndexer* method), 176
index_cleanup(), 300
index_config (built-in variable), 300
index_info(), 300
index_info() (*biothings.hub.dataindex.indexer.IndexManager* method), 250
INDEXER (*biothings.hub.dataindex.indexer.ColdHotIndexer* attribute), 248
Indexer (class in *biothings.hub.dataindex.indexer*), 251
IndexerCumulativeResult (class in *biothings.hub.dataindex.indexer*), 251
IndexerException, 177, 251
IndexerStepResult (class in *biothings.hub.dataindex.indexer*), 251
indexes_by_name(), 301
IndexingTask (class in *biothings.hub.dataindex.indexer_task*), 254
IndexJobStateRegistrar (class in *biothings.hub.dataindex.indexer_registrar*), 252
IndexManager (class in *biothings.hub.dataindex.indexer*), 249
IndexMappings (class in *biothings.hub.dataindex.indexer_payload*), 248
IndexSettings (class in *biothings.hub.dataindex.indexer_payload*), 248
inf (class in *biothings.utils.common*), 159
INFO (*biothings.utils.loggers.Colors* attribute), 192
INFO (*biothings.utils.loggers.Squares* attribute), 194
info(), 301
info() (*biothings.hub.autoupdate.dumper.BiothingsDumper* method), 229
info() (*biothings.web.options.openapi.OpenAPIContext* method), 136
ingest_hooks() (*biothings.hub.HubServer* method), 223
init_mapper() (*biothings.hub.databuild.builder.DataBuilder* method), 237
init_state() (*biothings.hub.databuild.builder.DataBuilder* method), 237
init_state() (*biothings.hub.dataload.dumper.BaseDumper*

method), 262
init_state() (*biothings.hub.dataload.uploader.BaseSourceUpload*), 260
method), 274
init_state() (*biothings.hub.datatransform.datatransform*), 187
method), 292
init_state() (*biothings.hub.datatransform.datatransform*), 308
method), 287
init_state() (*biothings.hub.datatransform.datatransform*), 309
method), 291
initialize() (*biothings.hub.api.handlers.base.BaseHandler*), 184
method), 225
initialize() (*biothings.hub.api.handlers.base.GenericHandler*), 259
method), 226
initialize() (*biothings.hub.api.handlers.base.RootHandler*), 301
method), 226
initialize() (*biothings.hub.api.handlers.log.HubLogDirHandler*), 295
method), 226
initialize() (*biothings.hub.api.handlers.shell.ShellHandler*), 167
method), 227
initialize() (*biothings.hub.api.handlers.upload.UploadHandler*), 259
method), 227
initialize() (*biothings.utils.redis.RedisClient*), 281
method), 214
INVALID_CHARACTERS_PATTERN (*biothings.utils.inspect.InspectionValidation* attribute), 185
initialize() (*biothings.web.handlers.base.BaseAPIHandler*), 128
method), 130
handle_cache() (*biothings.utils.mongo*), 210
INPUT (*biothings.utils.loggers.ShellLogger* attribute), 193
input() (*biothings.utils.loggers.ShellLogger* method), 193
inquire() (*biothings.web.options.manager.Existentialist* method), 133
insert() (*biothings.utils.backend.DocBackendBase* method), 154
insert() (*biothings.utils.backend.DocESBackend* method), 154
insert() (*biothings.utils.backend.DocMemoryBackend* method), 155
insert() (*biothings.utils.backend.DocMongoBackend* method), 156
insert() (*biothings.utils.mongo.Collection* method), 199
insert() (*biothings.utils.sqlite3.Collection* method), 215
insert_one() (*biothings.utils.es.Collection* method), 173
insert_one() (*biothings.utils.hub_db.Collection* method), 182
insert_one() (*biothings.utils.sqlite3.Collection* method), 215
inspect(), 301
inspect() (*biothings.hub.datainspect.inspector.InspectorManager* method), 259
inspect() (*in module biothings.utils.inspect*), 186
inspect_data() (*in module biothings.hub.datainspect.inspector*), 260
inspect_docs() (*in module biothings.utils.inspect*), 187
inspect_source() (*in module biothings.hub.datainspect.inspector*), 308
inspect_target() (*in module biothings.hub.datainspect.inspector*), 309
InspectorManager (*class in biothings.utils.inspect*), 184
InspectionValidation.Warning (*class in biothings.utils.inspect*), 185
InspectorError, 259
method), 226
InspectorManager (*class in biothings.hub.datainspect.inspector*), 259
method), 301
install(), 301
HubFeatureHandler (*biothings.hub.standalone.AutoHubFeature* method), 295
hart_convert() (*in module biothings.utils.dataloader*), 167
interpret_manifest() (*biothings.hub.dataplug.in.assistant.ManifestBasedPluginLoader* method), 281
INVALID_CHARACTERS_PATTERN (*biothings.utils.inspect.InspectionValidation* attribute), 185
handle_cache() (*biothings.utils.mongo*), 210
invalidate_plugin() (*biothings.hub.dataplug.in.assistant.BasePluginLoader* method), 280
InvalidJsonPatch, 188
invisible (*biothings.utils.configuration.ConfigAttrMeta* attribute), 162
is_filehandle() (*in module biothings.utils.common*), 159
is_float() (*in module biothings.utils.common*), 159
is_int() (*in module biothings.utils.common*), 159
is_jsonable() (*in module biothings.utils.configuration*), 164
is_scalar() (*in module biothings.utils.common*), 159
is_seq() (*in module biothings.utils.common*), 159
is_str() (*in module biothings.utils.common*), 159
is_valid_data_plugin_dir() (*in module biothings.cli.utils*), 311
isempty() (*biothings.utils.doc_traversal.Queue* method), 171
isempty() (*biothings.utils.doc_traversal.Stack* method), 171
ism (*built-in variable*), 302
iter_n() (*in module biothings.utils.common*), 159
jm (*built-in variable*), 302
job_info(), 302

job_info() (*biothings.utils.manager.JobManager method*), 197
JobManager (*class in biothings.utils.manager*), 196
JobRenderer (*class in biothings.hub*), 223
jobs() (*biothings.hub.dataload.uploader.ParallelizedSourceUploader* method), 277
json_array_parser() (*in module biothings.utils.parsers*), 213
json_dumps() (*in module biothings.utils.serializer*), 214
json_encode() (*in module biothings.utils.common*), 159
json_loads() (*in module biothings.utils.serializer*), 214
json_serial() (*in module biothings.utils.common*), 160
JsonArgCvter (*class in biothings.web.options.manager*), 133
jsondiff(), 302
JsonDiffer (*class in biothings.hub.databuild.differ*), 242
JsonPatch (*class in biothings.utils.jsonpatch*), 188
JsonPatchConflict, 190
JsonPatchException, 190
JsonPatchTestFailed, 190
jsonreadify() (*in module biothings.utils.hub*), 180

K

keep_archive (*biothings.hub.databuild.builder.DataBuild attribute*), 237
keep_archive (*biothings.hub.dataload.uploader.BaseSource attribute*), 274
KEEP_CONTAINER (*biothings.hub.dataload.dumper.DockerContainerDump attribute*), 265
key (*biothings.utils.inspect.BaseMode attribute*), 183
key (*biothings.utils.inspect.DeepStatsMode attribute*), 184
key (*biothings.utils.inspect.IdentifiersMode attribute*), 184
key (*biothings.utils.inspect.StatsMode attribute*), 186
key_exists() (*in module biothings.utils.aws*), 153
key_lookup_batch() (*biothings.hub.datatransform.datatransform_api.DataTransform method*), 292
key_lookup_batch() (*biothings.hub.datatransform.datatransform_api.DataTransformA attribute*), 288
key_lookup_batch() (*biothings.hub.datatransform_datatransform_mdb.DataTransformB attribute*), 290
key_value() (*in module biothings.utils.dotstring*), 172
keylookup (*biothings.hub.datapluging.assistant.BaseAssistant attribute*), 279
kwargs (*biothings.web.handlers.base.BaseAPIHandler attribute*), 128
kwargs (*biothings.web.handlers.query.MetadataFieldHandle attribute*), 131

L

Last_element() (*in module biothings.utils.dotstring*), 172
LAST_MODIFIED (*biothings.hub.dataload.dumper.LastModifiedHTTPDumper attribute*), 271
LastModifiedBaseDumper (*class in biothings.hub.dataload.dumper*), 270
LastModifiedFTPDumper (*class in biothings.hub.dataload.dumper*), 270
LastModifiedHTTPDumper (*class in biothings.hub.dataload.dumper*), 271
launch() (*biothings.utils.hub.HubShell method*), 179
launched_commands (*biothings.utils.hub.HubShell attribute*), 179
left() (*biothings.hub.datatransform.datatransform.IDStruct method*), 293
license() (*biothings.web.options.openapi.OpenAPIInfoContext method*), 137
LinkDataBuilder (*class in biothings.hub.databuild.builder*), 238
linkTargetDocMongoBackend (*class in biothings.hub.databuild.backend*), 231

D

list2dict() (*in module biothings.utils.common*), 160
list2dict() (*in module biothings.utils.dataload*), 167
list_biothings() (*biothings.hub.standalone.AutoHubFeature method*), 295
list_itemcnt() (*in module biothings.utils.dataload*), 167
list_length() (*in module biothings.utils.dotstring*), 172
list_merge() (*biothings.hub.databuild.builder.BuilderManager method*), 235
list_snapshots() (*biothings.hub.dataindex.snapshooter.SnapshotManager method*), 257
list_sources() (*biothings.hub.databuild.builder.BuilderManager method*), 235
list_split() (*in module biothings.utils.dataload*), 167
list_items() (*biothings.utils.hub_db.ChangeWatcher attribute*), 181
listing() (*in module biothings.cli.datapluging*), 308
listing() (*in module biothings.cli.datapluging_hub*), 309
listitems() (*in module biothings.utils.dataload*), 167
listsort() (*in module biothings.utils.dataload*), 167
list() (*in module biothings.utils.dataload*), 167

load() (*biothings.hub.autoupdate.uploader.BiothingsUploader*
method), 230
load() (*biothings.hub.databuild.mapper.BaseMapper*
method), 243
load() (*biothings.hub.databuild.mapper.TransparentMapper*
method), 243
load() (*biothings.hub.dataindex.idcache.IDCache*
method), 247
load() (*biothings.hub.dataindex.idcache.RedisIDCache*
method), 247
load() (*biothings.hub.dataload.uploader.BaseSourceUploader*
method), 274
load() (*biothings.hub.dataplugin.assistant.AssistantManager*
method), 279
load() (*biothings.hub.dataplugin.manager.DataPluginManager*
method), 282
load() (*in module biothings.web.settings.configs*), 149
load_build() (*biothings.hub.datarelease.publisher.BasePublisher*
method), 283
load_build() (*biothings.hub.datarelease.publisher.ReleaseManager*
method), 285
load_class() (*in module biothings.web.applications*), 124
load_class() (*in module biothings.web.services.namespace*), 125
load_data() (*biothings.hub.dataload.uploader.BaseSource*
method), 274
load_doc() (*biothings.utils.manager.BaseStatusRegisterer*
method), 196
load_json() (*in module biothings.utils.serializer*), 214
load_metadata() (*biothings.hub.databuild.syncer.BaseSyncer*
method), 244
load_module() (*in module biothings.web.settings.configs*), 149
load_plugin() (*biothings.hub.dataplugin.assistant.AdvancedPluginLoader*
method), 278
load_plugin() (*biothings.hub.dataplugin.assistant.AssistantManager*
method), 279
load_plugin() (*biothings.hub.dataplugin.assistant.BaseAssistant*
method), 279
load_plugin() (*biothings.hub.dataplugin.assistant.BasePluginLoader*
method), 280
load_plugin() (*biothings.hub.dataplugin.assistant.ManifestBasedPluginLoader*
method), 281
load_plugin() (*in module biothings.cli.utils*), 311
load_plugin_managers() (*in module biothings.cli.utils*), 311
load_remote_json()

ings.hub.autoupdate.dumper.BiothingsDumper
method), 229
loader (*biothings.hub.dataplugin.assistant.BaseAssistant*
property), 279
loader_type (*biothings.hub.dataplugin.assistant.AdvancedPluginLoader*
attribute), 278
loader_type (*biothings.hub.dataplugin.assistant.BasePluginLoader*
attribute), 280
loader_type (*biothings.hub.dataplugin.assistant.ManifestBasedPluginLoader*
attribute), 281
loaderException, 280
loaders (*biothings.hub.dataplugin.assistant.BaseAssistant*
attribute), 279
loadobj() (*in module biothings.utils.common*), 160
LocalAssistant (*class in biothings.hub.dataplugin.assistant*), 280
Locator (*class in biothings.web.options.manager*), 133
LogBuilder (*biothings.web.options.manager.OptionsManager*
method), 134
Logger (*biothings.hub.databuild.builder.DataBuilder*
property), 237
logger (*biothings.hub.dataload.dumper.BaseDumper*
property), 262
logger (*biothings.hub.datatransform.datatransform.DataTransformEdge*
property), 292
Logger (*biothings.web.query.builder.ESUserQuery*
property), 142
LogListener (*class in biothings.hub.api.handlers.ws*), 227
LogPrint (*class in biothings.utils.common*), 157
lookin() (*biothings.web.options.manager.Locator*
method), 133
lookup() (*biothings.hub.datatransform.datatransform.IDStruct*
method), 293
lookup() (*biothings.web.options.manager.ReqArgs*
method), 135
lookup_fields (*biothings.hub.datatransform.datatransform_api.DataTransformAPI*
attribute), 288
lookup_fields (*biothings.hub.datatransform.datatransform_api.DataTransformMyChe*
attribute), 288
lookup_fields (*biothings.hub.datatransform.datatransform_api.DataTransformMyGen*
attribute), 289
lookup_one() (*biothings.hub.datatransform.datatransform.DataTransform*
method), 292
LookUpList (*class in biothings.utils.loggers*), 193
loop (*built-in variable*), 302
lsmerge(), 302
M
main() (*biothings.hub.dataload.sync.MongoSync*
method), 273

`main()` (in module `biothings.cli`), 308
`main()` (in module `biothings.cli.web_app`), 312
`main()` (in module `biothings.web.launcher`), 123
`main_source` (`biothings.hub.dataload.uploader.BaseSourceUploader` attribute), 274
`MainIndexJSR` (class in `biothings.hub.dataindex.indexer_registrar`), 252
`MainIndexStep` (class in `biothings.hub.dataindex.indexer`), 251
`MainSnapshotState` (class in `biothings.hub.dataindex.snapshot_registrar`), 258
`make()` (in module `biothings.utils.jsondiff`), 188
`make_object()` (in module `biothings.utils.dotfield`), 172
`make_patch()` (in module `biothings.utils.jsonpatch`), 192
`make_temp_collection()` (`biothings.hub.dataload.uploader.BaseSourceUploader` method), 275
`ManagerError`, 198
`ManifestBasedPluginLoader` (class in `biothings.hub.dataplugin.assistant`), 281
`ManualDataPlugin` (class in `biothings.hub.dataplugin.manager`), 282
`ManualDumper` (class in `biothings.hub.dataload.dumper`), 272
`mapdb` (`biothings.utils.redis.RedisClient` property), 214
`MappingError`, 177
`mark_done()` (`biothings.hub.dataindex.IDCache` method), 247
`mark_done()` (`biothings.hub.dataindex.RedisIDCache` method), 247
`mark_success()` (`biothings.hub.dataload.dumper.BaseDumper` method), 262
`mark_success()` (`biothings.hub.dataload.dumper.DumperManager` method), 268
`markdown()` (`biothings.utils.loggers.SlackMessage` method), 194
`match()` (`biothings.utils.configuration.ConfigLine` method), 163
`matchers` (`biothings.utils.inspect.IdentifiersMode` attribute), 184
`matchers` (`biothings.utils.inspect.RegexMode` attribute), 185
`MAX_PARALLEL_DUMP` (`biothings.hub.dataload.dumper.BaseDumper` attribute), 261
`MAX_PARALLEL_DUMP` (`biothings.hub.dataload.dumper.DockerContainerDump` attribute), 265
`maxminiflist()` (`biothings.utils.inspect.StatsMode` method), 186
`MaxRetryAutoReconnectException`, 209
`md5sum()` (in module `biothings.utils.common`), 160
`mentions()` (`biothings.utils.loggers.SlackMentionPolicy` method), 194
~~`MERGE`~~ (`biothings.hub.dataindex.indexer_task.Mode` attribute), 254
`merge()`, 302
`merge()` (`biothings.hub.databuild.builder.BuilderManager` method), 235
`merge()` (`biothings.hub.databuild.builder.DataBuilder` method), 237
`merge()` (`biothings.hub.dataindex.indexer_task.IndexingTask` method), 254
`merge()` (`biothings.utils.inspect.BaseMode` method), 183
`merge()` (`biothings.utils.inspect.DeepStatsMode` method), 184
`merge()` (`biothings.utils.inspect.RegexMode` method), 185
`merge()` (`biothings.utils.inspect.StatsMode` method), 186
`merge()` (in module `biothings.utils.common`), 160
`merge_dict()` (in module `biothings.utils.dataload`), 167
`merge_duplicate_rows()` (in module `biothings.utils.dataload`), 168
`merge_field_inspections_validations()` (in module `biothings.utils.inspect`), 187
`merge_object()` (in module `biothings.utils.dotfield`), 172
`merge_order()` (`biothings.hub.databuild.builder.DataBuilder` method), 237
~~`MERGE`~~ (`biothings.utils.inspect`), 187
`merge_root_keys()` (in module `biothings.utils.dataload`), 168
`merge_scalar_list()` (in module `biothings.utils.inspect`), 187
`merge_source()` (`biothings.hub.databuild.builder.DataBuilder` method), 237
`merge_source()` (`biothings.hub.databuild.builder.LinkDataBuilder` method), 238
`merge_sources()` (`biothings.hub.databuild.builder.DataBuilder` method), 237
`merge_src_build_metadata()` (in module `biothings.hub.databuild.backend`), 232
`merge_struct()` (in module `biothings.utils.dataload`), 168
`merger_worker()` (in module `biothings.hub.databuild.builder`), 238
`MergerSourceUploader` (class in `biothings.hub.dataload.uploader`), 276
`Message` (class in `biothings.web.analytics.events`), 127
`MetadataFieldHandler` (class in `biothings.web.handlers.query`), 131

MetadataSourceHandler (class in *biothings.web.handlers.query*), 131
MetaField (class in *biothings.utils.configuration*), 164
method (*biothings.hub.dataindex.indexer.MainIndexStep* attribute), 251
method (*biothings.hub.dataindex.indexer.PostIndexStep* attribute), 251
method (*biothings.hub.dataindex.indexer.PreIndexStep* attribute), 252
method (*biothings.hub.dataindex.indexer.Step* attribute), 252
mexists() (*biothings.hub.dataindex.indexer_task.ESIndex* method), 254
mexists() (*biothings.utils.es.ESIndexer* method), 176
mget() (*biothings.hub.dataindex.indexer_task.ESIndex* method), 254
mget_from_ids() (*biothings.utils.backend.DocESBackend* method), 154
mget_from_ids() (*biothings.utils.backend.DocMongoBackend* method), 156
mindex() (*biothings.hub.dataindex.indexer_task.ESIndex* method), 254
MinType (class in *biothings.utils.dataload*), 165
mixargs() (*biothings.hub.HubServer* method), 223
Mode (class in *biothings.hub.dataindex.indexer_task*), 254
modified (*biothings.utils.configuration.ConfigurationWrapper* property), 164
module
 biothings.cli, 308
 biothings.cli.datapluging, 308
 biothings.cli.datapluging_hub, 309
 biothings.cli.utils, 310
 biothings.cli.web_app, 312
 biothings.hub, 217
 biothings.hub.api, 224
 biothings.hub.api.handlers.base, 225
 biothings.hub.api.handlers.log, 226
 biothings.hub.api.handlers.shell, 227
 biothings.hub.api.handlers.upload, 227
 biothings.hub.api.handlers.ws, 227
 biothings.hub.api.manager, 224
 biothings.hub.autoupdate.dumper, 228
 biothings.hub.autoupdate.uploader, 230
 biothings.hub.databuild.backend, 231
 biothings.hub.databuild.buildconfig, 232
 biothings.hub.databuild.builder, 234
 biothings.hub.databuild.differ, 239
 biothings.hub.databuild.mapper, 243
 biothings.hub.databuild.prebuilder, 244
 biothings.hub.databuild.syncer, 244
 biothings.hub.dataexport.ids, 247
biothings.hub.dataindex.idcache, 247
biothings.hub.dataindex.indexer, 248
biothings.hub.dataindex.indexer_cleanup, 248
biothings.hub.dataindex.indexer_payload, 248
biothings.hub.dataindex.indexer_registrar, 252
biothings.hub.dataindex.indexer_schedule, 253
biothings.hub.dataindex.indexer_task, 254
biothings.hub.dataindex.snapshooter, 255
biothings.hub.dataindex.snapshot_cleanup, 257
biothings.hub.dataindex.snapshot_registrar, 258
biothings.hub.dataindex.snapshot_repo, 258
biothings.hub.dataindex.snapshot_task, 259
biothings.hub.datainspect.inspector, 259
biothings.hub.dataload.dumper, 260
biothings.hub.dataload.source, 273
biothings.hub.dataload.storage, 273
biothings.hub.dataload.sync, 273
biothings.hub.dataload.uploader, 274
biothings.hub.datapluging.assistant, 278
biothings.hub.datapluging.manager, 282
biothings.hub.darelease, 282
biothings.hub.darelease.publisher, 282
biothings.hub.darelease.releasenote, 286
biothings.hub.datatransform.ciidstruct, 287
biothings.hub.datatransform.datatransform, 291
biothings.hub.datatransform.datatransform_api, 287
biothings.hub.datatransform.datatransform_mdb, 289
biothings.hub.datatransform.histogram, 294
biothings.hub.standalone, 294
biothings.tests, 150
biothings.tests.hub, 150
biothings.tests.web, 150
biothings.utils, 153
biothings.utils.aws, 153
biothings.utils.backend, 154
biothings.utils.common, 156
biothings.utils.configuration, 162
biothings.utils.dataload, 165
biothings.utils.diff, 170
biothings.utils.doc_traversal, 171

biothings.utils.docs, 171
biothings.utils.dotfield, 172
biothings.utils.dotstring, 172
biothings.utils.es, 173
biothings.utils.exclude_ids, 178
biothings.utils.hub, 178
biothings.utils.hub_db, 181
biothings.utils.info, 183
biothings.utils.inspect, 183
biothings.utils.jsondiff, 188
biothings.utils.jsonpatch, 188
biothings.utils.jsonschema, 192
biothings.utils.loggers, 192
biothings.utils.manager, 195
biothings.utils.mongo, 198
biothings.utils.parallel, 211
biothings.utils.parallel_mp, 211
biothings.utils.parsers, 213
biothings.utils.redis, 214
biothings.utils.serializer, 214
biothings.utils.sqlite3, 215
biothings.utils.version, 217
biothings.web, 122
biothings.web.analytics, 126
biothings.web.analytics.channels, 126
biothings.web.analytics.events, 127
biothings.web.analytics.notifiers, 128
biothings.web.applications, 123
biothings.web.connections, 126
biothings.web.handlers, 128
biothings.web.handlers.base, 128
biothings.web.handlers.query, 130
biothings.web.handlers.services, 132
biothings.web.launcher, 122
biothings.web.options, 132
biothings.web.options.manager, 132
biothings.web.options.openapi, 135
biothings.web.query, 141
biothings.web.query.builder, 141
biothings.web.query.engine, 143
biothings.web.query.formatter, 144
biothings.web.query.pipeline, 147
biothings.web.services, 124
biothings.web.services.health, 124
biothings.web.services.metadata, 124
biothings.web.services.namespace, 125
biothings.web.services.query, 124
biothings.web.settings, 148
biothings.web.settings.configs, 148
biothings.web.settings.default, 149
biothings.web.settings.validators, 149
biothings.web.templates, 149
modules (*biothings.web.settings.configs.ConfigPackage* attribute), 148

mongo (*in module biothings.web.connections*), 126
MongoDBEdge (*class in biothings.hub.datatransform*), 118
MongoDBEdge (*class in biothings.hub.datatransform.datatransform_mdb*), 290
MongoDBPreCompiledDataProvider (*class in biothings.hub.databuild.prebuilder*), 244
MongoHealth (*class in biothings.web.services.health*), 124
MongoJsonDiffSelfContainedSyncer (*class in biothings.hub.databuild.syncer*), 245
MongoJsonDiffSyncer (*class in biothings.hub.databuild.syncer*), 245
MongoParamValidator (*class in biothings.web.settings.validators*), 149
MongoQueryBackend (*class in biothings.web.query.engine*), 144
MongoQueryBuilder (*class in biothings.web.query.builder*), 142
MongoQueryPipeline (*class in biothings.web.query.pipeline*), 148
MongoResultFormatter (*class in biothings.web.query.formatter*), 147
MongoSync (*class in biothings.hub.dataload.sync*), 273
monitor() (*biothings.utils.hub.TornadoAutoReloadHubReloader* method), 180
monitor() (*biothings.utils.hub_db.ChangeWatcher* class method), 181
MoveOperation (*class in biothings.utils.jsonpatch*), 190
msgpack_ok() (*biothings.tests.web.BiothingsWebTestBase* static method), 152
multidict() (*in module biothings.utils.jsonpatch*), 192
MyChemInfoEdge (*class in biothings.hub.datatransform*), 119
MyChemInfoEdge (*class in biothings.hub.datatransform.datatransform_api*), 289
MyGeneInfoEdge (*class in biothings.hub.datatransform*), 119
MyGeneInfoEdge (*class in biothings.hub.datatransform.datatransform_api*), 289

N

name (*biothings.hub.autoupdate.uploader.BiothingsUploader* attribute), 230
name (*biothings.hub.databuild.backend.LinkTargetDocMongoBackend* attribute), 231
name (*biothings.hub.dataindex.indexer.MainIndexStep* attribute), 251
name (*biothings.hub.dataindex.indexer.PostIndexStep* attribute), 251

name (*biothings.hub.dataindex.indexer.PreIndexStep attribute*), 252

name (*biothings.hub.dataindex.indexer.Step attribute*), need_prepare() (biothings.hub.dataload.dumper.GitDumper method), 252

name (*biothings.hub.dataindex.snapshot_registrar.MainSnapshotState attribute*), 258

name (*biothings.hub.dataindex.snapshot_registrar.PostSnapshotState attribute*), 258

name (*biothings.hub.dataindex.snapshot_registrar.PreSnapshotState attribute*), 258

name (*biothings.hub.dataload.uploader.BaseSourceUploader attribute*), 275

name (*biothings.utils.backend.DocBackendBase attribute*), 154

name (*biothings.utils.backend.DocESBackend attribute*), 155

name (*biothings.utils.backend.DocMemoryBackend attribute*), 155

name (*biothings.utils.backend.DocMongoBackend attribute*), 156

name (*biothings.utils.hub_db.Collection property*), 182

name (*biothings.utils.sqlite3.Collection property*), 215

name (*biothings.web.handlers.base.BaseAPIHandler attribute*), 128

name (*biothings.web.handlers.query.BiothingHandler attribute*), 131

name (*biothings.web.handlers.query.MetadataFieldHandler attribute*), 131

name (*biothings.web.handlers.query.MetadataSourceHandler attribute*), 131

name (*biothings.web.handlers.query.QueryHandler attribute*), 131

name() (*biothings.web.options.openapi.OpenAPIContact method*), 135

name() (*biothings.web.options.openapi.OpenAPILicenseContact method*), 138

nan (class in *biothings.utils.common*), 160

ndjson_parser() (in module *biothings.utils.parsers*), 213

need_load() (*biothings.hub.databuild.mapper.IDBaseMapper method*), 243

need_prepare() (biothings.hub.dataload.dumper.APIDumper method), 261

need_prepare() (biothings.hub.dataload.dumper.BaseDumper method), 262

need_prepare() (biothings.hub.dataload.dumper.DockerContainerDumper method), 266

need_prepare() (biothings.hub.dataload.dumper.FilesystemDumper method), 268

need_prepare() (biothings.hub.dataload.dumper.FTPDumper method), 268

need_prepare() (biothings.hub.dataload.dumper.GitDumper method), 268

need_prepare() (biothings.hub.dataload.dumper.HTTPDumper method), 270

need_prepare() (biothings.hub.dataload.dumper.WgetDumper method), 272

nested_lookup() (in module *biothings.hub.datatransform.datatransform*), 294

new_data_folder (biothings.hub.dataload.dumper.BaseDumper property), 262

new_data_folder (biothings.hub.dataload.dumper.GitDumper property), 269

new_data_folder (biothings.hub.dataload.dumper.ManualDumper property), 272

newer() (in module *biothings.utils.common*), 160

NoBatchIgnoreDuplicatedSourceUploader (class in *biothings.hub.dataload.uploader*), 276

NoDataSourceUploader (class in *biothings.hub.dataload.uploader*), 276

NoResultError, 312

norm() (in module *biothings.utils.manager*), 198

normalized_value() (in module *biothings.utils.dataload*), 168

NoSuchCommand, 180

Notifier (class in *biothings.web.analytics.notifiers*), 128

NOTSET (*biothings.utils.loggers.Colors attribute*), 192

NOTSET (*biothings.utils.loggers.Squares attribute*), 194

NUMERIC_FIELDS (biothings.utils.inspect.InspectionValidation attribute), 185

Q

on_close() (*biothings.hub.api.handlers.ws.WebSocketConnection method*), 228

on_finish() (*biothings.web.analytics.notifiers.AnalyticsMixin method*), 128

on_finish() (*biothings.web.handlers.base.BaseAPIHandler method*), 128

on_message() (*biothings.hub.api.handlers.ws.WebSocketConnection method*), 228

open() (*biothings.utils.es.ESIndexer method*), 176

open_anyfile() (in module *biothings.utils.common*), 160

open_compressed_file() (in module `biothings.utils.common`), 160
OpenAPIContactContext (class in `biothings.web.options.openapi`), 135
OpenAPIContext (class in `biothings.web.options.openapi`), 136
OpenAPIDocumentBuilder (in module `biothings.web.options.openapi`), 136
OpenAPIExternalDocsContext (class in `biothings.web.options.openapi`), 136
OpenAPIInfoContext (class in `biothings.web.options.openapi`), 137
OpenAPILicenseContext (class in `biothings.web.options.openapi`), 138
OpenAPIOperation (class in `biothings.web.options.openapi`), 138
OpenAPIParameterContext (class in `biothings.web.options.openapi`), 139
OpenAPIPathItemContext (class in `biothings.web.options.openapi`), 140
operation_id() (`biothings.web.options.openapi.OpenAPIOperation` method), 138
optimize() (`biothings.utils.es.ESIndexer` method), 176
Option (class in `biothings.web.options.manager`), 133
OptionError, 134
options() (`biothings.hub.api.handlers.base.DefaultHandler` method), 225
options() (`biothings.hub.api.handlers.log.HubLogFileHandler` method), 226
options() (`biothings.web.handlers.base.BaseAPIHandler` method), 129
options() (`biothings.web.options.openapi.OpenAPIPathItemContext` method), 140
OptionSet (class in `biothings.web.options.manager`), 134
OptionsManager (class in `biothings.web.options.manager`), 134
order() (`biothings.hub.dataindex.indexer.Step` static method), 252
ORIGINAL_CONTAINER_STATUS (`biothings.hub.dataload.dumper.DockerContainerDump` attribute), 265
or_json_default() (in module `biothings.utils.serializer`), 214
OUTPUT (`biothings.utils.loggers.ShellLogger` attribute), 193
output() (`biothings.utils.loggers.ShellLogger` method), 194
output_types (`biothings.hub.datatransform.datatransform_api.DataTrainWithMySQLInfo` attribute), 288

P
Paragraph (class in `biothings.utils.configuration`), 164
ParallelizedSourceUploader (class in `biothings.hub.dataload.uploader`), 276
ParallelResult (class in `biothings.utils.parallel_mp`), 211
parse() (`biothings.utils.configuration.ConfigLines` method), 163
parse() (`biothings.web.options.manager.Option` method), 134
parse() (`biothings.web.options.manager.OptionSet` method), 134
parse() (`biothings.web.query.builder.QStringParser` method), 143
parse_dot_fields() (in module `biothings.utils.dotfield`), 172
parse_folder_name_from_url() (in module `biothings.utils.common`), 160
parse_head() (`biothings.hub.api.handlers.upload.UploadHandler` method), 227
password_auth_supported() (`biothings.hub.HubSSHSERVER` method), 218
PASSWORDS (`biothings.hub.HubSSHSERVER` attribute), 217
patch() (`biothings.web.options.openapi.OpenAPIPathItemContext` method), 140
PatchOperation (class in `biothings.utils.jsonpatch`), 190
path() (`biothings.web.options.openapi.OpenAPIContext` method), 136
PathArgCvter (class in `biothings.web.options.manager`), 134
PATTERNS (`biothings.utils.configuration.ConfigLine` attribute), 163
pause() (`biothings.utils.common.LogPrint` method), 157
payload() (`biothings.utils.loggers.WSLogHandler` method), 195
payload() (`biothings.utils.loggers.WSShellHandler` method), 195
pcchildren (`biothings.utils.manager.JobManager` property), 197
pending (built-in variable), 302
pending() (in module `biothings.hub.databuild.builder`), 238
pending_outputs (`biothings.utils.hub.HubShell` attribute), 179
pending_snapshot() (in module `biothings.hub.dataindex.snapshooter.SnapshotManager` static method), 257
pick_db() (`biothings.utils.redis.RedisClient` method), 214
plain_text() (`biothings.hub.dataindex.indexer_cleanup.Cleaner` method), 248
plain_text() (in module `biothings.hub.snapshot_cleanup`), 257
plaintext() (`biothings.utils.loggers.SlackMessage` method), 194

plugin_name (*biothings.hub.dataplugin.assistant.BaseAssistant* property), 280
post_dump() (*biothings.hub.dataload.dumper.DockerContainerDumper* method), 266
 plugin_name (*biothings.hub.dataplugin.assistant.GithubAssistant* property), 280
post_dump_delete_files() (*biothings.hub.dataload.dumper.BaseDumper* method), 266
 plugin_name (*biothings.hub.dataplugin.assistant.LocalAssistant* property), 281
post_index() (*biothings.hub.dataindex.indexer.Indexer* method), 262
 plugin_type (*biothings.hub.dataplugin.assistant.BaseAssistant* attribute), 280
post_merge() (*biothings.hub.databuild.backend.TargetDocBackend* method), 231
 plugin_type (*biothings.hub.dataplugin.assistant.GithubAssistant* attribute), 280
post_merge() (*biothings.hub.databuild.builder.DataBuilder* method), 232
 plugin_type (*biothings.hub.dataplugin.assistant.LocalAssistant* attribute), 281
post_publish() (*biothings.hub.datarelease.publisher.DiffPublisher* method), 237
poll() (*biothings.hub.databuild.builder.BuilderManager* method), 235
poll() (*biothings.hub.databuild.differ.DifferManager* method), 242
poll() (*biothings.hub.dataindex.snapshooter.SnapshotManager* method), 257
poll() (*biothings.hub.dataload.uploader.UploaderManager* method), 277
poll() (*biothings.hub.datarelease.publisher.ReleaseManager* method), 285
poll() (*biothings.utils.hub.BaseHubReloader* method), 178
poll() (*biothings.utils.manager.BaseManager* method), 195
pop() (*biothings.utils.doc_traversal.Queue* method), 171
pop() (*biothings.utils.doc_traversal.Stack* method), 171
post() (*biothings.hub.api.handlers.base.GenericHandler* method), 226
post() (*biothings.hub.api.handlers.upload.UploadHandler* method), 227
post() (*biothings.utils.inspect.BaseMode* method), 183
post() (*biothings.utils.inspect.DeepStatsMode* method), 184
post() (*biothings.web.handlers.query.BiothingHandler* method), 131
post() (*biothings.web.handlers.query.QueryHandler* method), 131
post() (*biothings.web.options.openapi.OpenAPIPathItem* method), 140
post_diff_cols() (*biothings.hub.databuild.differ.BaseDiffer* method), 239
post_diff_cols() (*biothings.hub.databuild.differ.ColdHotJsonDiffer* method), 240
post_download() (*biothings.hub.dataload.dumper.BaseDumper* method), 262
post_dump() (*biothings.hub.autoupdate.dumper.BiothingsDumper* method), 229
post_dump() (*biothings.hub.dataload.dumper.BaseDumper* method), 262
post_indexJSR() (*biothings.hub.dataindex.indexer_registrar* class in *biothings.hub.dataindex.indexer*), 253
PostIndexStep (*class* in *biothings.hub.dataindex.indexer*), 251
PostIndexStep (*class* in *biothings.hub.dataindex.snapshot_registrar*), 258
pqueue (built-in variable), 302
pre_index() (*biothings.hub.dataindex.indexer.Indexer* method), 251
pre_publish() (*biothings.hub.datarelease.publisher.DiffPublisher* method), 283
pre_publish() (*biothings.hub.datarelease.publisher.SnapshotPublisher* method), 285
pre_snapshot() (*biothings.hub.dataindex.snapshooter.SnapshotEnv* method), 256
prefix (*biothings.tests.web.BiothingsWebTestCase* attribute), 152
PreIndexJSR (*class* in *biothings.hub.dataindex.indexer_registrar*), 253
PreIndexStep (*class* in *biothings.hub.dataindex.indexer*), 252
prepare() (*biothings.hub.api.handlers.upload.UploadHandler* method), 227
prepare() (*biothings.hub.databuild.backend.ShardedTargetDocMongoBackend* method), 227

method), 231
prepare() (*biothings.hub.databuild.builder.DataBuilder method*), 238
prepare() (*biothings.hub.dataload.dumper.BaseDumper method*), 262
prepare() (*biothings.hub.dataload.dumper.ManualDumper method*), 272
prepare() (*biothings.hub.dataload.uploader.BaseSourceUploader method*), 275
prepare() (*biothings.hub.datatransform.datatransform.DataTransformer*), 291
method), 293
prepare() (*biothings.utils.backend.DocBackendBase method*), 154
prepare() (*biothings.utils.backend.DocESBackend method*), 155
prepare() (*biothings.web.handlers.base.BaseAPIHandler method*), 129
prepare() (*biothings.web.handlers.query.BaseQueryHandler method*), 130
prepare_client() (*biothings.hub.autoupdate.dumper.BiothingsDumper method*), 229
prepare_client() (*biothings.hub.dataload.dumper.APIDumper method*), 261
prepare_client() (*biothings.hub.dataload.dumper.BaseDumper method*), 262
prepare_client() (*biothings.hub.dataload.dumper.DockerContainerDumper method*), 266
prepare_client() (*biothings.hub.dataload.dumper.DummyDumper method*), 267
prepare_client() (*biothings.hub.dataload.dumper.FilesystemDumper method*), 269
prepare_client() (*biothings.hub.dataload.dumper.FTPDumper method*), 268
prepare_client() (*biothings.hub.dataload.dumper.GitDumper method*), 269
prepare_client() (*biothings.hub.dataload.dumper.GoogleDriveDumper method*), 270
prepare_client() (*biothings.hub.dataload.dumper.HTTPDumper method*), 270
prepare_client() (*biothings.hub.dataload.dumper.LastModifiedFTPDumper method*), 271
prepare_client() (*biothings.hub.dataload.dumper.ManualDumper method*), 272
prepare_client() (*biothings.hub.dataload.dumper.WgetDumper method*), 272
prepare_client() (*biothings.hub.datatransform.datatransform_api.BiothingsAPIEdge method*), 287
prepare_collection() (*biothings.hub.datatransform.datatransform_mdb.MongoDBEdge method*), 291
prepare_dumper_params() (*biothings.hub.dataload.dumper.DockerContainerDumper method*), 266
prepare_local_folders() (*biothings.hub.dataload.dumper.BaseDumper method*), 262
prepare_local_folders() (*biothings.hub.dataload.dumper.DockerContainerDumper method*), 266
prepare_remote_container() (*biothings.hub.dataload.dumper.DockerContainerDumper method*), 266
prepare_src_dump() (*biothings.hub.dataload.dumper.BaseDumper method*), 262
prepare_src_dump() (*biothings.hub.dataload.uploader.BaseSourceUploader method*), 275
prepare_src_dump() (*biothings.hub.dataload.uploader.DummySourceUploader method*), 275
prepare_src_dump() (*biothings.hub.dataplugin.manager.GitDataPlugin method*), 282
prepare_src_dump() (*biothings.hub.dataplugin.manager.ManualDataPlugin method*), 282
PreSnapshotState (*class in biothings.hub.dataindex.snapshot_registrar*), 258
print_pending_info() (*biothings.utils.manager.JobManager method*), 197
print_workers() (*biothings.utils.manager.JobManager method*), 197
process() (*biothings.hub.databuild.mapper.BaseMapper method*), 243
process() (*biothings.hub.databuild.mapper.IDBaseMapper method*), 243
process() (*biothings.hub.databuild.mapper.TransparentMapper method*), 243
process_hook_file() (*biothings.hub.HubServer method*), 223

`process_inspect()` (*in module biothings.cli.utils*), 311
`ProcessInfo` (*class in biothings.hub.dataindex.indexer*), 252
`ProcessInfo` (*class in biothings.hub.dataindex.snapshooter*), 255
`prune()` (*biothings.hub.dataindex.indexer_registrar.IndexJob* static method), 252
`publish()`, 302
`publish()` (*biothings.hub.api.handlers.ws.WebSocketConnection* method), 228
`publish()` (*biothings.hub.datarelease.publisher.DiffPublisher* method), 283
`publish()` (*biothings.hub.datarelease.publisher.ReleaseManager* method), 285
`publish()` (*biothings.hub.datarelease.publisher.SnapshotPublisher* method), 285
`publish()` (*biothings.utils.hub_db.ChangeWatcher* class method), 181
`publish_build()` (*biothings.hub.datarelease.publisher.ReleaseManager* method), 285
`publish_data_version()` (*in module biothings.utils.hub*), 180
`publish_diff()`, 303
`publish_diff()` (*biothings.hub.datarelease.publisher.ReleaseManager* method), 285
`publish_release_notes()` (*biothings.hub.datarelease.publisher.BasePublisher* method), 283
`publish_snapshot()`, 303
`publish_snapshot()` (*biothings.hub.datarelease.publisher.ReleaseManager* method), 285
`PublisherException`, 284
`PURGE` (*biothings.hub.dataindex.indexer_task.Mode* attribute), 254
`push()` (*biothings.utils.doc_traversal.Queue* method), 171
`push()` (*biothings.utils.doc_traversal.Stack* method), 171
`put()` (*biothings.hub.api.handlers.base.GenericHandler* method), 226
`put()` (*biothings.hub.api.handlers.shell.ShellHandler* method), 227
`put()` (*biothings.web.options.openapi.OpenAPIPathItemContext* method), 140

Q

`QStringParser` (*class in biothings.web.query.builder*), 142
`Query` (*class in biothings.web.query.builder*), 143
`query()` (*biothings.tests.web.BiothingsWebTestBase* method), 152

`query()` (*biothings.utils.backend.DocESBackend* method), 155
`QueryArgCvter` (*class in biothings.web.options.manager*), 134
`QueryHandler` (*class in biothings.cli.web_app*), 312
`QueryHandler` (*class in biothings.web.handlers.query*), 131
`QueryPipeline` (*class in biothings.web.query.pipeline*), 148
`QueryPipelineException`, 148
`QueryPipelineInterrupt`, 148
`Queue` (*class in biothings.utils.doc_traversal*), 171
`QueueEmptyError`, 171
`quick_index()`, 303
`quick_index()` (*biothings.hub.HubServer* method), 223

R

`range` (*biothings.utils.loggers.Record* attribute), 193
`Range` (*class in biothings.utils.loggers*), 193
`RawQueryInterrupt`, 143
`RawResultInterrupt`, 144
`read()` (*biothings.hub.api.handlers.ws.HubDBListener* method), 227
`read()` (*biothings.hub.api.handlers.ws.LogListener* method), 227
`read()` (*biothings.utils.hub_db.ChangeListener* method), 181
`readonly` (*biothings.utils.configuration.ConfigAttrMeta* attribute), 162
`readonly` (*biothings.utils.configuration.ConfigurationWrapper* property), 164
`READY` (*biothings.hub.dataindex.indexer_registrar.Stage* attribute), 253
`reapply_patch()` (*in module biothings.utils.jsonpatch*), 192
`rebuild_diff_file_list()` (*biothings.hub.databuild.differ.DifferManager* method), 242
`rec_handler()` (*in module biothings.utils.dataload*), 168
`ReceiverGroup` (*class in biothings.utils.loggers*), 193
`Record` (*class in biothings.utils.loggers*), 193
`recycle_process_queue()` (*biothings.utils.manager.JobManager* method), 197
`RedisClient` (*class in biothings.utils.redis*), 214
`RedisClientError`, 214
`RedisIDCache` (*class in biothings.hub.dataindex.idcache*), 247
`RedisPreCompiledDataProvider` (*class in biothings.hub.databuild.prebuilder*), 244
`reduce_diffs()` (*in module biothings.hub.databuild.differ*), 242

refresh() (*biothings.web.services.metadata.BiothingsESMethod*), 125
refresh() (*biothings.web.services.metadata.BiothingsMetadataMethod*), 125
refresh() (*biothings.web.services.metadata.BiothingsMongoMetadataMethod*), 125
refresh() (*biothings.web.services.metadata.BiothingsSQLMethod*), 125
refresh_commands() (*biothings.utils.hub.HubShellClass Method*), 179
regex_name (*biothings.hub.dataload.uploader.BaseSourceUploaderAttribute*), 275
RegExEdge (class in *biothings.hub.datatransform*), 119
RegExEdge (class in *biothings.hub.datatransform.datatransform*), 294
RegexMode (class in *biothings.utils.inspect*), 185
region (*biothings.hub.dataindex.snapshooter.CloudStorageAttribute*), 255
region (*biothings.hub.dataindex.snapshooter.RepositoryConfigProperty*), 255
register() (*biothings.hub.databuild.prebuilder.BasePreCompiledDataMethod*), 244
register() (*biothings.hub.databuild.prebuilder.MongoDBPreCompiledDataMethod*), 244
register() (*biothings.hub.databuild.prebuilder.RedisPreCompiledDataMethod*), 244
register_builder() (*biothings.hub.databuild.builder.BuilderManagerMethod*), 235
register_classes() (*biothings.hub.dataload.dumper.DumperManagerMethod*), 268
register_classes() (*biothings.hub.dataload.uploader.UploaderManagerMethod*), 277
register_classes() (*biothings.hub.dataplug.in.assistant.AssistantManagerMethod*), 279
register_classes() (*biothings.utils.manager.BaseSourceManagerMethod*), 196
register_command() (*biothings.utils.hub.HubShellMethod*), 179
register_differ() (*biothings.hub.databuild.differ.DifferManagerMethod*), 242
register_loader() (*biothings.hub.dataplug.in.assistant.BaseAssistantMethod*), 280
register_managers() (*biothings.utils.hub.HubShellMethod*), 179
register_source() (*biothings.utils.manager.BaseSourceManagerMethod*), 196
~~register_sources()~~ (*biothings.utils.manager.BaseSourceManagerMethod*), 196
register_status() (*biothings.hub.databuild.builder.DataBuilderMethod*), 224
~~register_status()~~ (*biothings.hub.databuild.syncer.BaseSyncerMethod*), 244
register_status() (*biothings.hub.dataload.dumper.BaseDumperMethod*), 262
~~register_status()~~ (*biothings.hub.dataload.uploader.BaseSourceUploaderMethod*), 275
register_status() (*biothings.hub.databuild.syncer.ReleaseManagerMethod*), 283
~~register_status()~~ (*biothings.hub.databuild.syncer.SyncerManagerMethod*), 246
register_url() , 303
register_url() (*biothings.hub.dataplug.in.assistant.AssistantManagerMethod*), 279
regx (*biothings.hub.dataindex.snapshot_registrar.MainSnapshotStateAttribute*), 258
regx (*biothings.hub.dataindex.snapshot_registrar.PostSnapshotStateAttribute*), 258
reindex() (*biothings.utils.es.ESIndexerMethod*), 176
release_client() (*biothings.hub.dataload.dumper.APIDumperMethod*), 261
release_client() (*biothings.hub.dataload.dumper.BaseDumperMethod*), 263
release_client() (*biothings.hub.dataload.dumper.DockerContainerDumperMethod*), 266
release_client() (*biothings.hub.dataload.dumper.FilesystemDumperMethod*), 269

release_client()	(biothings.hub.dataload.dumper.FTPDumper method), 268	remote_is_better()	(biothings.hub.dataload.dumper.FilesystemDumper method), 269
release_client()	(biothings.hub.dataload.dumper.GitDumper method), 269	remote_is_better()	(biothings.hub.dataload.dumper.FTPDumper method), 268
release_client()	(biothings.hub.dataload.dumper.HTTPDumper method), 270	remote_is_better()	(biothings.hub.dataload.dumper.GitDumper method), 269
release_client()	(biothings.hub.dataload.dumper.LastModifiedFTPDumper method), 271	remote_is_better()	(biothings.hub.dataload.dumper.GoogleDriveDumper method), 270
release_client()	(biothings.hub.dataload.dumper.WgetDumper method), 272	remote_is_better()	(biothings.hub.dataload.dumper.HTTPDumper method), 270
RELEASE_FORMAT	(biothings.hub.dataload.dumper.LastModifiedFTPDumper attribute), 271	remote_is_better()	(biothings.hub.dataload.dumper.LastModifiedFTPDumper method), 271
RELEASE_FORMAT	(biothings.hub.dataload.dumper.LastModifiedHTTPDumper attribute), 271	remote_is_better()	(biothings.hub.dataload.dumper.LastModifiedHTTPDumper method), 271
release_info()	303	remote_is_better()	(biothings.hub.dataload.dumper.WgetDumper method), 272
release_info()	(biothings.hub.datarelease.publisher.ReleaseManager method), 285	remove()	(biothings.utils.es.Collection method), 173
RELEASE_TO_BUILD	(biothings.hub.databuild.buildconfig.AutoBuildConfig attribute), 233	remove()	(biothings.utils.hub_db.Collection method), 182
RELEASE_TYPES	(biothings.hub.databuild.buildconfig.AutoBuildConfig attribute), 233	remove()	(biothings.utils.mongo.Collection method), 199
ReleaseManager	(class in biothings.hub.datarelease.publisher), 284	remove()	(biothings.utils.serializer.URL method), 214
ReleaseNoteSource	(class in biothings.hub.datarelease.releasenote), 286	remove()	(biothings.utils.sqlite3.Collection method), 215
ReleaseNoteSrcBuildReader	(class in biothings.hub.datarelease.releasenote), 286	remove_files_in_folder()	(in module biothings.cli.utils), 311
ReleaseNoteSrcBuildReaderAdapter	(class in biothings.hub.datarelease.releasenote), 286	remove_from_ids()	(biothings.utils.backend.DocESBackend method), 155
ReleaseNoteTxt	(class in biothings.hub.datarelease.releasenote), 286	remove_from_ids()	(biothings.utils.backend.DocMongoBackend method), 156
reload()	(biothings.hub.dataload.source.SourceManager method), 273	remove_key()	(in module biothings.utils.dotstring), 172
remote_is_better()	(biothings.hub.autoupdate.dumper.BiothingsDumper method), 229	RemoveOperation	(class in biothings.utils.jsonpatch), 190
remote_is_better()	(biothings.hub.dataload.dumper.APIDumper method), 261	rename()	(biothings.utils.sqlite3.Collection method), 215
remote_is_better()	(biothings.hub.dataload.dumper.BaseDumper method), 263	render()	(biothings.hub.JobRenderer method), 223
remote_is_better()	(biothings.hub.dataload.dumper.DockerContainerDump	render_cron()	(biothings.hub.JobRenderer method), 224
		render_func()	(biothings.hub.JobRenderer method), 224
		render_lambda()	(biothings.hub.JobRenderer method), 224
		render_method()	(biothings.hub.JobRenderer method), 224

224
render_partial() (*biothings.hub.JobRenderer method*), 224
render_strdelta() (*biothings.hub.JobRenderer method*), 224
RenderedStr (class in *biothings.hub.dataindex.snapshooter*), 255
replace_one() (*biothings.utils.es.Collection method*), 173
replace_one() (*biothings.utils.hub_db.Collection method*), 182
replace_one() (*biothings.utils.sqlite3.Collection method*), 215
ReplaceOperation (class in *biothings.utils.jsonpatch*), 191
repo (*biothings.hub.dataindex.snapshooter.RepositoryConfig property*), 256
report(), 303
report() (*biothings.utils.inspect.BaseMode method*), 183
report() (*biothings.utils.inspect.DeepStatsMode method*), 184
report() (*biothings.utils.inspect.RegexMode method*), 185
report() (*biothings.utils.inspect.StatsMode method*), 186
Repository (class in *biothings.hub.dataindex.snapshot_repo*), 258
RepositoryConfig (class in *biothings.hub.dataindex.snapshooter*), 255
ReqArgs (class in *biothings.web.options.manager*), 134
ReqArgs.Path (class in *biothings.web.options.manager*), 135
ReqResult (class in *biothings.web.options.manager*), 135
request() (*biothings.tests.web.BiothingsWebAppTest method*), 151
request() (*biothings.tests.web.BiothingsWebTestBase method*), 152
requires_config() (in module *biothings.utils.mongo*), 210
requires_config() (in module *biothings.utils.sqlite3*), 216
reset() (*biothings.hub.dataload.source.SourceManager method*), 273
reset() (*biothings.utils.configuration.ConfigAttrMeta method*), 163
reset() (*biothings.utils.configuration.ConfigurationWrapper method*), 164
reset_backend(), 303
reset_synced(), 303
reset_synced() (*biothings.hub.datarelease.publisher.DiffPublisher method*), 284
reset_synced() (*biothings.hub.datarelease.publisher.ReleaseManager method*), 285
reset_target_backend() (*biothings.hub.autoupdate.dumper.BiothingsDumper method*), 230
resetconf(), 304
resolve_builder_class() (*biothings.hub.databuild.builder.BuilderManager method*), 235
RESOLVE_FILENAME (*biothings.hub.dataload.dumper.HTTPDumper attribute*), 270
RESOLVE_FILENAME (*biothings.hub.dataload.dumper.LastModifiedHTTPDumper attribute*), 271
resolve_sources() (*biothings.hub.databuild.builder.DataBuilder method*), 238
ResourceError, 198, 277
ResourceNotFound, 198
ResourceNotReady, 277
restart(), 304
restart() (*biothings.utils.hub.HubShell method*), 179
restore(), 304
restore() (*biothings.utils.es.ESIndexer method*), 176
restore() (in module *biothings.utils.hub_db*), 183
restore_running_apis() (*biothings.hub.api.manager.APIManager method*), 224
restore_snapshot() (*biothings.hub.autoupdate.uploader.BiothingsUploader method*), 230
ResultFormatter (class in *biothings.web.query.formatter*), 147
ResultFormatterException, 147
ResultInterrupt, 144
RESUME (*biothings.hub.dataindex.indexer_task.Mode attribute*), 254
resume() (*biothings.hub.dataindex.indexer_task.IndexingTask method*), 254
resume() (*biothings.utils.common.LogPrint method*), 157
ResumeException, 238
right() (*biothings.hub.datatransform.datatransform.IDStruct method*), 293
rm (built-in variable), 304
rmdashfr() (in module *biothings.utils.common*), 160
rmmmerge(), 304
root (*biothings.web.settings.configs.ConfigPackage attribute*), 148
RootHandler (class in *biothings.hub.handlers.base*), 226
run_converters() (in module *biothings.utils.inspect*),

187		
run_jobs_on_ipythoncluster() (in module biothings.utils.parallel), 211		
run_jobs_on_parallel() (in module biothings.utils.parallel), 211		
run_once() (in module biothings.utils.common), 160		
run_parallel_on_ids_dir() (in module biothings.utils.parallel_mp), 211		
run_parallel_on_ids_file() (in module biothings.utils.parallel_mp), 211		
run_parallel_on_iterable() (in module biothings.utils.parallel_mp), 212		
run_parallel_on_query() (in module biothings.utils.parallel_mp), 213		
run_post_publish_diff() (biothings.hub.datarelease.publisher.DiffPublisher method), 284		
run_post_publish_snapshot() (biothings.hub.datarelease.publisher.SnapshotPublisher method), 285		
run_pre_post() (biothings.hub.datarelease.publisher.BasePublisher method), 283		
run_pre_publish_diff() (biothings.hub.datarelease.publisher.DiffPublisher method), 284		
run_pre_publish_snapshot() (biothings.hub.datarelease.publisher.SnapshotPublisher method), 285		
run_sync_or_async_job() (in module biothings.cli.utils), 311		
S		
safe_type() (in module biothings.utils.dataloader), 168		
safe_unicode() (in module biothings.utils.common), 161		
safewfile() (in module biothings.utils.common), 161		
sanitize_settings() (biothings.utils.es.ESIndexer method), 176		
sanitize_tarfile() (in module biothings.utils.common), 161		
save() (biothings.hub.databuild.differ.DiffReportRendererBase method), 241		
save() (biothings.hub.databuild.differ.DiffReportTxt method), 241		
save() (biothings.hub.datarelease.releasenote.ReleaseNoteTxt method), 286		
save() (biothings.utils.es.Collection method), 173		
save() (biothings.utils.hub_db.Collection method), 182		
save() (biothings.utils.mongo.Collection method), 199		
save() (biothings.utils.sqlite3.Collection method), 215		
save_cmd() (biothings.utils.hub.HubShell class method), 179		
save_doc_src_master() (biothings.hub.dataload.uploader.BaseSourceUploader method), 275		
save_mapping() (biothings.hub.databuild.builder.BuilderManager method), 235		
save_mapping() (biothings.hub.dataload.source.SourceManager method), 273		
sch(), 304		
SCHEDULE (biothings.hub.dataload.dumper.BaseDumper attribute), 261		
Schedule (class in biothings.hub.dataindex.indexer_schedule), 253		
schedule(), 304		
schedule() (biothings.utils.manager.JobManager method), 197		
schedule_all() (biothings.hub.dataload.dumper.DumperManager method), 268		
schema() (biothings.web.options.openapi.OpenAPIParameterContext method), 139		
scheme (biothings.tests.web.BiothingsWebTestCase attribute), 152		
scopes (biothings.web.query.builder.Group attribute), 142		
scopes (biothings.web.query.builder.Query attribute), 143		
search() (biothings.web.query.pipeline.AsyncESQueryPipeline method), 147		
search() (biothings.web.query.pipeline.ESQueryPipeline method), 148		
search() (biothings.web.query.pipeline.QueryPipeline method), 148		
secret_key (biothings.hub.dataindex.snapshooter.CloudStorage attribute), 255		
section (biothings.utils.configuration.ConfigAttrMeta attribute), 163		
SelfContainedJsonDiffer (class in biothings.hub.databuild.differ), 242		
send() (biothings.utils.loggers.SlackHandler static method), 194		
send() (biothings.web.analytics.channels.Channel method), 126		
send() (biothings.web.analytics.channels.GA4Channel method), 126		
send() (biothings.web.analytics.channels.GAChannel method), 127		
send() (biothings.web.analytics.channels.SlackChannel method), 127		
send_s3_big_file() (in module biothings.utils.aws), 153		
send_s3_file() (in module biothings.utils.aws), 153		
send_s3_folder() (in module biothings.utils.aws), 153		

serve() (in module `biothings.cli.dataplugin`), 308
serve() (in module `biothings.cli.dataplugin_hub`), 309
serve() (in module `biothings.cli.utils`), 311
server() (`biothings.web.options.openapi.OpenAPIContext` method), 136
session_requested() (`biothings.hub.HubSSHSERVER` method), 218
session_started() (`biothings.hub.HubSSHServerSession` method), 220
set_cache_header() (`biothings.web.handlers.base.BaseAPIHandler` method), 129
set_command_counter() (`biothings.utils.hub.HubShell` class method), 180
set_commands() (`biothings.utils.hub.HubShell` method), 180
set_data_path() (`biothings.hub.dataload.dumper.DockerContainerDumper` method), 266
set_debug() (`biothings.hub.datatransform.datatransform`.~~HTTPStaticWebsite~~ method), 293
set_default_folder() (in module `biothings.utils.configuration`), 164
set_default_headers() (`biothings.cli.web_app.BaseHandler` method), 312
set_default_headers() (`biothings.hub.api.handlers.base.DefaultHandler` method), 225
set_default_headers() (`biothings.hub.api.handlers.log.DefaultCORSHeaderMis` method), 226
set_default_headers() (`biothings.hub.handlers.base.BaseAPIHandler` method), 129
set_dump_command() (`biothings.hub.dataload.dumper.DockerContainerDumper` method), 266
set_get_version_cmd() (`biothings.hub.dataload.dumper.DockerContainerDumper` method), 266
set_internal_number_of_replicas() (`biothings.utils.es.ESIndexer` method), 176
set_keep_container() (`biothings.hub.dataload.dumper.DockerContainerDumper` method), 266
set_key_value() (in module `biothings.utils.dotstring`), 173
set_mapping_src_meta() (`biothings.hub.dataload.source.SourceManager` method), 273
set_pending_to_build() (in module `biothings.hub.databuild.builder`), 238
set_pending_to_diff() (in module `biothings.hub.databuild.differ`), 243
set_pending_to_publish() (in module `biothings.hub.datarelease`), 282
set_pending_to_release_note() (in module `biothings.hub.datarelease`), 282
set_pending_to_upload() (in module `biothings.hub.dataload.uploader`), 278
set_release() (`biothings.hub.dataload.dumper.DockerContainerDumper` method), 266
set_release() (`biothings.hub.dataload.dumper.LastModifiedBaseDumper` method), 270
set_release() (`biothings.hub.dataload.dumper.LastModifiedFTPDumper` method), 271
set_release() (`biothings.hub.dataload.dumper.LastModifiedHTTPDumper` method), 271
set_release() (`biothings.hub.dataload.dumper.LastModifiedMongoDumper` method), 232
set_target_name() (`biothings.hub.databuild.backend.TargetDocBackend` method), 232
set_target_name() (`biothings.hub.databuild.backend.TargetDocMongoBackend` method), 232
set_versions() (in module `biothings.utils.version`), 217
setconf(), 304
setup() (`biothings.hub.databuild.builder.DataBuilder` method), 238
setup() (`biothings.hub.datarelease.publisher.BasePublisher` method), 283
setup() (`biothings.hub.datarelease.publisher.ReleaseManager` method), 285
setup() (`biothings.web.options.manager.OptionSet` method), 134
setup() (in module `biothings.utils.hub_db`), 183
setup_class() (`biothings.tests.web.BiothingsWebTest` class method), 152
setup_config() (in module `biothings.cli`), 308
setup_default_log() (in module `biothings.utils.loggers`), 195
setup_log() (`biothings.hub.api.manager.APIManager` method), 225
setup_log() (`biothings.hub.databuild.builder.BuilderManager` method), 236
setup_log() (`biothings.hub.databuild.builder.DataBuilder` method), 238
setup_log() (`biothings.hub.databuild.differ.BaseDiffer`

method), 239
setup_log() (*biothings.hub.databuild.differ.DifferManager method*), 242
setup_log() (*biothings.hub.databuild.syncer.BaseSyncer method*), 244
setup_log() (*biothings.hub.databuild.syncer.SyncerManager method*), 246
setup_log() (*biothings.hub.dataindex.indexer.Indexer method*), 251
setup_log() (*biothings.hub.dataindex.snapshooter.Snapshot method*), 256
setup_log() (*biothings.hub.datainspect.inspector.Inspect method*), 260
setup_log() (*biothings.hub.dataload.dumper.BaseDumper method*), 263
setup_log() (*biothings.hub.dataload.uploader.BaseSource method*), 275
setup_log() (*biothings.hub.dataplugin.assistant.AssistantManager method*), 279
setup_log() (*biothings.hub.dataplugin.assistant.BaseAssistant method*), 280
setup_log() (*biothings.hub.dataplugin.assistant.BasePluginLoader static method*), 293
setup_log() (*biothings.hub.datarelease.publisher.BasePublisher method*), 134
setup_log() (*biothings.hub.datarelease.publisher.ReleaseManager method*), 187
setup_log() (*biothings.hub.datatransform.datatransform.IDStruct method*), 293
setup_logfile() (*in module biothings.utils.common*), 161
ShardedTargetDocMongoBackend (*class in biothings.hub.databuild.backend*), 231
SHELL (*biothings.hub.HubSSHServer attribute*), 217
shell_requested() (*biothings.hub.HubSSHServerSession method*), 221
ShellHandler (*class in biothings.hub.api.handlers.shell*), 227
ShellListener (*class in biothings.hub.api.handlers.ws*), 227
ShellLogger (*class in biothings.utils.loggers*), 193
should_diff_new_build() (*biothings.hub.databuild.buildconfig.AutoBuildConfig method*), 233
should_install_new_diff() (*biothings.hub.databuild.buildconfig.AutoBuildConfig method*), 233
should_install_new_release() (*biothings.hub.databuild.buildconfig.AutoBuildConfig method*), 233
should_install_new_snapshot() (*biothings.hub.databuild.buildconfig.AutoBuildConfig method*), 233

should_publish_new_diff() (*biothings.hub.databuild.buildconfig.AutoBuildConfig method*), 234
should_publish_new_snapshot() (*biothings.hub.databuild.buildconfig.AutoBuildConfig method*), 234
should_snapshot_new_build() (*biothings.hub.databuild.buildconfig.AutoBuildConfig method*), 234
show() (*biothings.utils.configuration.ConfigurationWrapper method*), 164
show_dumped_files() (*in module biothings.cli.utils*), 311
show_hubdb_content() (*in module biothings.cli.utils*), 311
show_pendings() (*biothings.utils.manager.JobManager method*), 197
show_uploaded_sources() (*in module biothings.cli.utils*), 311
size() (*biothings.hub.datatransform.datatransform.IDStruct*), 311
simplify() (*biothings.web.options.manager.OptionError method*), 134
simplify_inspection_data() (*in module biothings.utils.inspect*), 187
sizeof_fmt() (*in module biothings.utils.common*), 161
SlackChannelEdge (*class in biothings.web.analytics.channels*), 127
SlackHandler (*class in biothings.utils.loggers*), 194
SlackMentionPolicy (*class in biothings.utils.loggers*), 194
SlackMessage (*class in biothings.utils.loggers*), 194
SLEEP_BETWEEN_DOWNLOAD (*biothings.hub.dataload.dumper.BaseDumper attribute*), 261
sm (*built-in variable*), 304
Snapshot (*class in biothings.hub.dataindex.snapshot_task*), 259
snapshot(), 304
snapshot() (*biothings.hub.dataindex.snapshooter.SnapshotEnv method*), 256
snapshot() (*biothings.hub.dataindex.snapshooter.SnapshotManager method*), 257
snapshot() (*biothings.utils.es.ESIndexer method*), 176
snapshot_a_build() (*biothings.hub.dataindex.snapshooter.SnapshotManager method*), 257
snapshot_cleanup(), 305
snapshot_config (*built-in variable*), 305
snapshot_info(), 305
snapshot_info() (*biothings.hub.dataindex.snapshooter.SnapshotManager method*), 257

SnapshotEnv (class in *biothings.hub.dataindex.snapshooter*), 256
SnapshotManager (class in *biothings.hub.dataindex.snapshooter*), 256
SnapshotPublisher (class in *biothings.hub.datarelease.publisher*), 285
soft_eof_received() (*biothings.hub.HubSSHServerSession* method), 221
sort_input_by_priority_list() (*biothings.hub.datatransform.datatransform.DataTransform* method), 292
sort_output_by_priority_list() (*biothings.hub.datatransform.datatransform.DataTransform* method), 292
source_backend (*biothings.hub.databuild.builder.BuilderManager* property), 236
source_backend (*biothings.hub.databuild.builder.DataBuilder* property), 238
SOURCE_CLASS (*biothings.hub.dataload.dumper.DumperManager* attribute), 267
SOURCE_CLASS (*biothings.hub.dataload.uploader.UploaderManager* attribute), 277
SOURCE_CLASS (*biothings.utils.manager.BaseSourceManager* attribute), 195
source_config (*biothings.hub.dataload.dumper.DockerContainerDumper* property), 266
source_info(), 305
source_info() (*biothings.hub.dataload.dumper.DumperManager* method), 268
source_info() (*biothings.hub.dataload.uploader.UploaderManager* method), 277
source_reset(), 305
source_save_mapping(), 305
SourceDocBackendBase (class in *biothings.hub.databuild.backend*), 231
SourceDocMongoBackend (class in *biothings.hub.databuild.backend*), 231
SourceManager (class in *biothings.hub.dataload.source*), 273
sources(), 305
SPACE_PATTERN (*biothings.utils.inspect.InspectionValidation* attribute), 185
split_ids() (in module *biothings.utils.common*), 161
splitstr (class in *biothings.utils.common*), 161
sql (in module *biothings.web.connections*), 126
SQLHealth (class in *biothings.web.services.health*), 124
SQLQueryBackend (class in *biothings*), 144
SQLQueryBuilder (class in *biothings.web.query.builder*), 143
SQLQueryPipeline (class in *biothings.web.query.pipeline*), 148
SQLResultFormatter (class in *biothings.web.query.formatter*), 147
Squares (class in *biothings.utils.loggers*), 194
src_doc (*biothings.hub.dataload.dumper.BaseDumper* property), 263
src_dump (*biothings.hub.dataload.dumper.BaseDumper* property), 263
SRC_NAME (*biothings.hub.autoupdate.dumper.BiothingsDumper* attribute), 228
SRC_NAME (*biothings.hub.dataload.dumper.BaseDumper* attribute), 261
SRC_ROOT_FOLDER (*biothings.hub.autoupdate.dumper.BiothingsDumper* attribute), 228
SRC_ROOT_FOLDER (*biothings.hub.dataload.dumper.BaseDumper* attribute), 261
SRC_URLS (*biothings.hub.dataload.dumper.LastModifiedBaseDumper* attribute), 261
ssm (built-in variable), 305
Stack (class in *biothings.utils.doc_traversal*), 171
StackEmptyError, 171
Stage (*class in biothings.hub.dataindex.indexer_registrar*), 253
start (*biothings.utils.loggers.Range* attribute), 193
start() (*biothings.hub.HubServer* method), 223
start() (*biothings.utils.common.LogPrint* method), 157
start() (*biothings.web.launcher.BiothingsAPIBaseLauncher* method), 122
start() (*biothings.web.launcher.FlaskAPILauncher* method), 123
start() (*biothings.web.launcher.TornadoAPILauncher* method), 123
start_api(), 306
start_api() (*biothings.hub.api.manager.APIManager* method), 225
start_api() (in module *biothings.hub.api*), 224
start_ssh_server() (in module *biothings.hub*), 224
STARTED (*biothings.hub.dataindex.indexer_registrar.Stage* attribute), 253
started() (*biothings.hub.dataindex.indexer_registrar.IndexJobStateRegistration* method), 252
started() (*biothings.hub.dataindex.indexer_registrar.MainIndexJSR* method), 253
started() (*biothings.hub.dataindex.indexer_registrar.PostIndexJSR* method), 253
started() (*biothings.hub.dataindex.indexer_registrar.PreIndexJSR* method), 253
state (*biothings.hub.dataindex.indexer.MainIndexStep*

attribute), 251
state (biothings.hub.dataindex.indexer.PostIndexStep attribute), 252
state (biothings.hub.dataindex.indexer.PreIndexStep attribute), 252
state (biothings.hub.dataindex.indexer.Step attribute), 252
state() (biothings.hub.dataindex.snapshot_task.Snapshot method), 259
stats (biothings.utils.inspect.FieldInspection attribute), 184
stats() (in module biothings.utils.hub), 180
StatsMode (class in biothings.utils.inspect), 186
status(), 306
status() (in module biothings.hub), 224
StatusHandler (class in biothings.web.handlers.services), 132
step (biothings.hub.dataindex.snapshot_registrar.MainSnapshot attribute), 258
step (biothings.hub.dataindex.snapshot_registrar.PostSnapshotState attribute), 258
step (biothings.hub.dataindex.snapshot_registrar.PreSnapshotState attribute), 258
Step (class in biothings.hub.dataindex.indexer), 252
step_archive() (biothings.hub.datarelease.publisher.BasePublisher method), 283
step_upload() (biothings.hub.datarelease.publisher.BasePublisher method), 283
step_upload_s3() (biothings.hub.datarelease.publisher.BasePublisher method), 283
StepResult (class in biothings.hub.dataindex.snapshooter), 257
stop(), 306
stop() (biothings.utils.hub.HubShell method), 180
stop() (biothings.utils.manager.JobManager method), 198
stop_api(), 306
stop_api() (biothings.hub.api.manager.APIManager method), 225
storage_class (biothings.hub.dataload.uploader.BaseSourceUploader attribute), 275
storage_class (biothings.hub.dataload.uploader.IgnoreDuplicatedSource attribute), 276
storage_class (biothings.hub.dataload.uploader.MergerSourceUpload attribute), 276
storage_class (biothings.hub.dataload.uploader.NoBatchIgnoreDuplicatedSource attribute), 276
storage_class (biothings.hub.dataload.uploader.NoDataSourceUploader attribute), 276
store_metadata() (biothings.hub.databuild.builder.DataBuilder method), 238
store_value_to_db() (biothings.utils.configuration.ConfigurationWrapper method), 164
str_to_bool() (biothings.web.options.manager.Converter static method), 132
str_to_bool() (biothings.web.options.manager.QueryArgCvter class method), 134
str_to_int() (biothings.web.options.manager.Converter method), 132
str_to_list() (biothings.web.options.manager.Converter static method), 132
stringify_inspect_doc() (in module biothings.utils.inspect), 187
style() (biothings.web.options.openapi.OpenAPIParameterContext method), 139
subclasses (biothings.web.options.openapi.OpenAPIContext attribute), 135
subclasses (biothings.web.options.openapi.OpenAPIContext attribute), 136
subclasses (biothings.web.options.openapi.OpenAPIExternalDocsContext attribute), 136
subclasses (biothings.web.options.openapi.OpenAPIInfoContext attribute), 137
subclasses (biothings.web.options.openapi.OpenAPILicenseContext attribute), 138
subclasses (biothings.web.options.openapi.OpenAPIOperation attribute), 138
subclasses (biothings.web.options.openapi.OpenAPIParameterContext attribute), 139
subclasses (biothings.web.options.openapi.OpenAPIPathItemContext attribute), 140
subclasses() (biothings.web.options.manager.Converter class method), 132
submit() (biothings.hub.dataplugin.assistant.AssistantManager method), 279
submit() (biothings.utils.manager.JobManager method), 198
SubModuleValidator (class in biothings.web.settings.validators), 149
SubStr() (in module biothings.utils.common), 157
succeed() (biothings.hub.dataindex.indexer_registrar.IndexJobStateRegister method), 252
succeed() (biothings.hub.dataindex.indexer_registrar.PreIndexJSR method), 253
suffix() (biothings.hub.dataindex.indexer_schedule.Schedule

method), 253
SUFFIX_ATTR (*biothings.hub.dataloader.dumper.BaseDumper attribute*), 261
sumiflist() (*biothings.utils.inspect.StatsMode method*), 186
summary (*biothings.web.query.pipeline.QueryPipelineException attribute*), 148
sumup_source() (*biothings.hub.dataloader.source.SourceManager method*), 273
supersede() (*biothings.utils.configuration.ConfigurationWrapper method*), 164
switch_collection() (*biothings.hub.dataloader.uploader.BaseSourceUploader method*), 275
sym (*built-in variable*), 306
sync(), 306
sync() (*biothings.hub.databuild.syncer.BaseSyncer method*), 245
sync() (*biothings.hub.databuild.syncer.SyncerManager method*), 246
sync_cols() (*biothings.hub.databuild.syncer.BaseSyncer method*), 245
sync_es_coldhot_jsondiff_worker() (*in module biothings.hub.databuild.syncer*), 246
sync_es_for_update() (*in module biothings.hub.databuild.syncer*), 246
sync_es_jsondiff_worker() (*in module biothings.hub.databuild.syncer*), 246
sync_mongo_jsondiff_worker() (*in module biothings.hub.databuild.syncer*), 247
SYNCER_FUNC (*biothings.hub.autoupdate.uploader.BiothingsUploader rings.hub.databuild.syncer.MongoJsonDiffSelfContainedSyncer attribute*), 230
syncer_func (*biothings.hub.autoupdate.uploader.BiothingsUploader property*), 230
SyncerException, 245
SyncerManager (*class in biothings.hub.databuild.syncer*), 246

T

tab2dict() (*in module biothings.utils.dataloader*), 168
tab2dict_iter() (*in module biothings.utils.dataloader*), 168
tab2list() (*in module biothings.utils.dataloader*), 169
tabfile_feeder() (*in module biothings.utils.dataloader*), 169
tabfile_tester() (*in module biothings.utils.dataloader*), 169
target_alias (*biothings.utils.backend.DocESBackend property*), 155
TARGET_BACKEND (*biothings.hub.autoupdate.dumper.BiothingsDumper attribute*), 228
target_backend (*biothings.hub.autoupdate.dumper.BiothingsDumper property*), 230
TARGET_BACKEND (*biothings.hub.autoupdate.uploader.BiothingsUploader attribute*), 230
target_backend (*biothings.hub.autoupdate.uploader.BiothingsUploader property*), 230
target_backend (*biothings.hub.databuild.builder.BuilderManager property*), 236
target_backend (*biothings.hub.databuild.builder.DataBuilder property*), 238
target_backend_type (*biothings.hub.databuild.syncer.BaseSyncer attribute*), 245
target_backend_type (*biothings.hub.databuild.syncer.ESColdHotJsonDiffSelfContainedSyncer attribute*), 245
target_backend_type (*biothings.hub.databuild.syncer.ESColdHotJsonDiffSyncer attribute*), 245
target_backend_type (*biothings.hub.databuild.syncer.ESJsonDiffSyncer attribute*), 245
target_backend_type (*biothings.hub.databuild.syncer.MongoJsonDiffSyncer attribute*), 245
target_backend_type (*biothings.hub.databuild.syncer.MongoJsonDiffSyncer attribute*), 245
target_collection (*biothings.hub.databuild.backend.LinkTargetDocMongoBackend property*), 231
target_db (*biothings.utils.backend.DocMongoBackend property*), 156
target_esidixer (*biothings.utils.backend.DocESBackend property*), 155
target_name (*biothings.hub.databuild.backend.TargetDocBackend property*), 232
target_name (*biothings.utils.backend.DocBackendBase property*), 154
target_name (*biothings.utils.backend.DocESBackend property*), 155
target_name (*biothings.utils.backend.DocMemoryBackend property*), 155
target_name (*biothings.utils.backend.DocMongoBackend property*), 156

```

TargetDocBackend      (class      in      bioth-           ings.hub.dataindex.indexer_task), 254
                  (bioth-          TEST_DATA_DIR_NAME          ings.tests.web.BiothingsWebAppTest attribute),
                  (bioth-          151
TargetDocMongoBackend (class      in      bioth-          test_database_index()          (bioth-
                  (bioth-          ings.tests.hub.DatabaseCollectionTesting
                  (bioth-          method), 150
tearDown()          (biothings.tests.web.BiothingsWebTestBase
                  method), 152
template()          (biothings.utils.inspect.BaseMode attribute),
                  184
template()          (biothings.utils.inspect.DeepStatsMode
                  attribute), 184
template()          (biothings.utils.inspect.StatsMode attribute),
                  186
template_out()      (in module biothings.utils.hub), 180
template_out_conf() (bioth-          test_document_name()          (bioth-
                  ings.hub.datarelease.publisher.BasePublisher
                  method), 283
method), 283
TemplateStr         (class      in      bioth-          test_documents_taxid()          (bioth-
                  ings.hub.dataindex.snapshooter), 257
term()              (biothings.web.query.builder.Group attribute), 142
term()              (biothings.web.query.builder.Query attribute), 143
terms_of_service() (bioth-          test_field_does_not_exist()          (bioth-
                  ings.web.options.openapi.OpenAPIInfoContext
                  method), 138
method), 138
test()              (in module biothings.hub.databuild.buildconfig),
                  234
test()              (in module bioth-          test_field_taxid()          (bioth-
                  ings.hub.dataindex.snapshot_registrar),
                  258
method), 258
test()              (in module biothings.utils.jsonschema), 192
test0()             (in module bioth-          test_field_unique_id()          (bioth-
                  ings.hub.dataindex.indexer_task), 254
method), 254
test1()             (in module bioth-          test_find()          (in module bioth-
                  ings.hub.dataindex.indexer_task), 254
method), 254
test_00()            (in module bioth-          test_find()          (in module bioth-
                  ings.hub.dataindex.indexer_task), 254
method), 254
test_01()            (in module bioth-          test_find()          (in module bioth-
                  ings.hub.dataindex.indexer_schedule), 253
method), 253
test_01()            (in module bioth-          test_print()          (in module bioth-
                  ings.hub.dataindex.snapshot_repo), 258
method), 258
test_01()            (in module bioth-          test_register()          (in module bioth-
                  ings.hub.dataindex.snapshot_task), 259
method), 259
test_02()            (in module bioth-          test_register()          (in module bioth-
                  ings.hub.dataindex.indexer_schedule), 253
method), 253
test_02()            (in module bioth-          test_str()          (in module bioth-
                  ings.hub.dataindex.snapshot_repo), 258
method), 258
test_02()            (in module bioth-          test_total_document_count()          (bioth-
                  ings.hub.dataindex.snapshot_task), 259
method), 259
test_03()            (in module bioth-          TestOperation (class in biothings.utils.jsonpatch), 191
                  ings.hub.dataindex.indexer_schedule), 253
method), 253
test_04()            (in module bioth-          Text (class in biothings.utils.configuration), 164
                  ings.hub.dataindex.indexer_schedule), 253
method), 253
test_clean()          (in module bioth-          ThrottledESColdHotJsonDiffSelfContainedSyncer
                  ings.hub.dataindex.indexer_cleanup), 248
method), 248
test_clients()        (in module bioth-          (class in biothings.hub.databuild.syncer), 246
method), 127
method), 127

```

to_boolean() (in module `biothings.utils.dataloader`), 169
to_delete(`biothings.hub.dataloader.dumper.BaseDumper` attribute), 263
to_dict() (`biothings.hub.datarelease.releasenote.ReleaseNote` method), 286
to_dict() (`biothings.utils.inspect.InspectionValidation`.`WatsonQueue` method), 185
to_dict() (`biothings.web.services.metadata.BiothingHubMeta` method), 124
to_dict() (`biothings.web.services.metadata.BiothingLicense` method), 124
to_dict() (`biothings.web.services.metadata.BiothingMappings` method), 124
to_dict() (`biothings.web.services.metadata.BiothingMetaProp` method), 125
to_email_payload() (`biothings.web.analytics.events.Message` method), 127
to_float() (in module `biothings.utils.dataloader`), 169
to_GA4_payload() (`biothings.web.analytics.events.Event` method), 127
to_GA4_payload() (`biothings.web.analytics.events.GAEEvent` method), 127
to_GA_payload() (`biothings.web.analytics.events.Event` method), 127
to_GA_payload() (`biothings.web.analytics.events.GAEEvent` method), 127
to_int() (in module `biothings.utils.dataloader`), 169
to_jira_payload() (`biothings.web.analytics.events.Message` method), 127
to_json() (in module `biothings.utils.serializer`), 215
to_json_0() (in module `biothings.utils.serializer`), 215
to_json_file() (in module `biothings.utils.serializer`), 215
to_msgpack() (in module `biothings.utils.serializer`), 215
to_number() (in module `biothings.utils.dataloader`), 169
to_slack_payload() (`biothings.web.analytics.events.Message` method), 127
to_string() (`biothings.utils.jsonpatch.JsonPatch` method), 190
to_type() (`biothings.web.options.manager.Converter` static method), 132
to_type() (`biothings.web.options.manager.JsonArgCvter` method), 133
to_yaml() (in module `biothings.utils.serializer`), 215
top(), 306
top() (`biothings.utils.manager.JobManager` method), 198
TornadoAPILauncher (class in `biothings.web.launcher`), 123
TornadoAutoReloadHubReloader (class in `biothings.utils.hub`), 180
TornadoBiothingsAPI (class in `biothings.web.applications`), 123
trace() (`biothings.web.options.openapi.OpenAPIPathItemContext` method), 141
track() (in module `biothings.utils.manager`), 198
transfer_debug() (`biothings.hub.datatransform.datatransform.IDStruct` method), 293
transform() (`biothings.web.query.formatter.ESResultFormatter` method), 144
transform() (`biothings.web.query.formatter.MongoResultFormatter` method), 147
transform() (`biothings.web.query.formatter.ResultFormatter` method), 147
transform() (`biothings.web.query.formatter.SQLResultFormatter` method), 147
transform_aggs() (`biothings.web.query.formatter.ESResultFormatter` method), 145
transform_hit() (`biothings.web.query.formatter.ESResultFormatter` method), 145
transform_mapping() (`biothings.web.query.formatter.ResultFormatter` method), 146
transform_mapping() (`biothings.web.query.formatter.ResultFormatter` method), 147
translate() (`biothings.hub.databuild.mapper.IDBaseMapper` method), 243
translate() (`biothings.web.options.manager.Converter` method), 133
TransparentMapper (class in `biothings.hub.databuild.mapper`), 243
trasform_jmespath() (`biothings.web.query.formatter.ESResultFormatter` method), 146
travel() (`biothings.hub.datatransform.datatransform_mdb.DataTransform` method), 290
traverse() (`biothings.web.query.formatter.ESResultFormatter` static method), 146
traverse() (in module `biothings.utils.common`), 161
traverse_keys() (in module `biothings.utils.dataloader`), 169
trigger_diff() (`biothings.hub.databuild.differ.DifferManager` method), 242
trigger_merge() (`biothings.hub.databuild.builder.BuilderManager` method), 236

`trigger_release_note()` (*biothings.hub.datarelease.publisher.BasePublisher method*), 283

`two_docs_iterator()` (*in module biothings.utils.diff*), 170

`type` (*biothings.hub.dataindex.snapshooter.CloudStorage attribute*), 255

`type()` (*biothings.web.options.openapi.OpenAPIParameterContext method*), 140

`types` (*biothings.utils.inspect.FieldInspectValidation attribute*), 184

`types` (*biothings.web.services.metadata.BiothingsESMetadata property*), 125

`types` (*biothings.web.services.metadata.BiothingsMongoMetadata property*), 125

`types` (*biothings.web.services.metadata.BiothingsSQLMetadata property*), 125

`typify_inspect_doc()` (*in module biothings.utils.inspect*), 187

U

`um` (*built-in variable*), 306

`uncompressall()` (*in module biothings.utils.common*), 162

`unique_ids()` (*in module biothings.utils.dataloader*), 169

`UnknownResource`, 198

`unlist()` (*in module biothings.utils.dataloader*), 169

`unlist_incl()` (*in module biothings.utils.dataloader*), 169

`unprepare()` (*biothings.hub.databuild.builder.DataBuilder method*), 238

`unprepare()` (*biothings.hub.dataloader.dumper.BaseDumper method*), 263

`unprepare()` (*biothings.hub.dataloader.uploader.BaseSourceUploader method*), 275

`unprepare()` (*biothings.hub.datatransform.datatransform.DataTransformEdge method*), 293

`unregister_url()`, 306

`unregister_url()` (*biothings.hub.dataplugin.assistant.AssistantManager method*), 279

`untarall()` (*in module biothings.utils.common*), 162

`untargzall()` (*in module biothings.utils.common*), 162

`unxzall()` (*in module biothings.utils.common*), 162

`unzipall()` (*in module biothings.utils.common*), 162

`update()` (*biothings.utils.backend.DocBackendBase method*), 154

`update()` (*biothings.utils.backend.DocESBackend method*), 155

`update()` (*biothings.utils.backend.DocMemoryBackend method*), 155

`update()` (*biothings.utils.backend.DocMongoBackend method*), 156

`update()` (*biothings.utils.configuration.ConfigAttrMeta method*), 163

`update()` (*biothings.utils.es.Collection method*), 173

`update()` (*biothings.utils.es.ESIndexer method*), 176

`update()` (*biothings.utils.hub_db.Collection method*), 182

`update()` (*biothings.utils.mongo.Collection method*), 199

`update()` (*biothings.utils.sqlite3.Collection method*), 215

`update()` (*biothings.web.services.metadata.BiothingsESMetadata method*), 125

`update_alias()` (*biothings.utils.es.ESIndexer method*), 176

`update_build_conf()`, 306

`update_build_configuration()` (*biothings.hub.databuild.builder.BuilderManager method*), 236

`update_data()` (*biothings.hub.autoupdate.uploader.BiothingsUploader method*), 230

`update_data()` (*biothings.hub.dataload.uploader.BaseSourceUploader method*), 275

`update_data()` (*biothings.hub.dataload.uploader.DummySourceUploader method*), 275

`update_data()` (*biothings.hub.dataload.uploader.NoDataSourceUploader method*), 276

`update_data()` (*biothings.hub.dataload.uploader.ParallelizedSourceUploader method*), 277

`update_dict_recur()` (*in module biothings.utils.dataloader*), 170

`update_diff()` (*biothings.utils.backend.DocMongoBackend method*), 156

`update_docs()` (*biothings.utils.es.ESIndexer method*), 177

`update_edge()` (*biothings.hub.datatransform.histogram.Histogram method*), 294

`update_io()` (*biothings.hub.datatransform.histogram.Histogram method*), 294

`update_many()` (*biothings.utils.es.Collection method*), 173

`update_mapping()` (*biothings.utils.es.ESIndexer method*), 177

`update_mapping_meta()` (*biothings.utils.es.ESIndexer method*), 177

`update_master()` (*biothings.hub.dataload.uploader.BaseSourceUploader method*), 275

update_metadata(), 307
update_metadata() (biothings.hub.dataindex.indexer.IndexManager method), 250
update_one() (biothings.utils.es.Collection method), 173
update_one() (biothings.utils.hub_db.Collection method), 182
update_one() (biothings.utils.sqlite3.Collection method), 215
update_plugin_name() (biothings.hub.dataplugn.assistant.AssistantManager method), 279
update_settings() (biothings.utils.es.ESIndexer method), 177
update_source_meta(), 307
update_source_meta() (biothings.hub.dataload.uploader.UploaderManager method), 278
update_src_meta_stats() (biothings.hub.databuild.builder.DataBuilder method), 238
updated_dict() (in module biothings.utils.dataload), 170
upgrade(), 307
upload(), 307
upload_all(), 307
upload_all() (biothings.hub.dataload.uploader.UploaderManager method), 278
upload_ids() (in module biothings.hub.dataexport.ids), 247
upload_info(), 307
upload_info() (biothings.hub.dataload.uploader.UploaderManager method), 278
upload_source() (in module biothings.cli.dataplugn), 309
upload_source() (in module biothings.cli.dataplugn_hub), 310
upload_src() (biothings.hub.dataload.uploader.UploaderManager method), 278
uploader_manager (biothings.hub.dataplugn.assistant.BaseAssistant attribute), 280
UploaderManager (class in biothings.hub.dataload.uploader), 277
UploadHandler (class in biothings.hub.api.handlers.upload), 227
upsert_build_conf() (biothings.hub.databuild.builder.BuilderManager method), 236
URL (class in biothings.utils.serializer), 214
url() (biothings.web.options.openapi.OpenAPIContext method), 135
url() (biothings.web.options.openapi.OpenAPIExternalDocsContext method), 137
url() (biothings.web.options.openapi.OpenAPILicenseContext method), 138
use_curl() (biothings.web.launcher.TornadoAPILauncher static method), 123

V

validate() (biothings.utils.inspect.InspectionValidation static method), 185
validate() (biothings.web.options.manager.Validator method), 135
validate() (biothings.web.settings.validators.DBParamValidator method), 149
validate() (biothings.web.settings.validators.MongoParamValidaor method), 149
validate() (biothings.web.settings.validators.SubmoduleValidator method), 149
validate() (biothings.web.settings.validators.WebAPIValidator method), 149
validate_mapping(), 307
validate_mapping() (biothings.hub.dataindex.indexer.IndexManager method), 251
validate_password() (biothings.hub.HubSSHSERVER method), 219
validate_sources() (biothings.hub.databuild.backend.SourceDocBackendBase method), 231
validate_sources() (biothings.hub.databuild.backend.SourceDocMongoBackend method), 231
validate_W001() (biothings.utils.inspect.InspectionValidation static method), 185
validate_W002() (biothings.utils.inspect.InspectionValidation static method), 185
validate_W003() (biothings.utils.inspect.InspectionValidation static method), 185
validate_W004() (biothings.utils.inspect.InspectionValidation static method), 185
Validator (class in biothings.web.options.manager), 135
value (biothings.utils.configuration.MetaField property), 164
value (biothings.utils.configuration.Paragraph property), 164
value (biothings.utils.loggers.Record attribute), 193
value_convert() (in module biothings.utils.dataload), 170

```

value_convert_incl() (in module biothings.utils.dataloader), 170
value_convert_to_number() (in module biothings.utils.dataloader), 170
value_in_result() (biothings.tests.web.BiothingsWebTestCase method), 152
verify() (biothings.hub.dataindex.snapshot_repo.Repository method), 258
VERIFY_CERT (biothings.hub.dataloader.dumper.HTTPDump attribute), 270
verify_ids() (in module biothings.utils.es), 177
version (biothings.utils.backend.DocBackendBase property), 154
version (biothings.utils.backend.DocESBackend property), 155
version (biothings.utils.backend.DocMongoBackend property), 156
version() (biothings.web.options.openapi.OpenAPIInfoContext method), 138
VERSION_URL (biothings.hub.autoupdate.dumper.BiothingsDumper attribute), 229
versions(), 308
versions() (biothings.hub.autoupdate.dumper.BiothingsDumper method), 230

```

W

```

W001 (biothings.utils.inspect.InspectionValidation.Warning attribute), 185
W002 (biothings.utils.inspect.InspectionValidation.Warning attribute), 185
W003 (biothings.utils.inspect.InspectionValidation.Warning attribute), 185
W004 (biothings.utils.inspect.InspectionValidation.Warning attribute), 185
WARNING (biothings.utils.loggers.Colors attribute), 193
WARNING (biothings.utils.loggers.Squares attribute), 194
warnings (biothings.utils.inspect.FieldInspection attribute), 184
warnings (biothings.utils.inspect.FieldInspectValidation attribute), 184
watched_files() (biothings.utils.hub.BaseHubReloader method), 178
watched_files() (biothings.utils.hub.TornadoAutoReloadHubReloader method), 180
WebAPIValidator (class in biothings.web.settings.validators), 149
WebSocketConnection (class in biothings.hub.api.handlers.ws), 227
WgetDumper (class in biothings.hub.dataloader.dumper), 272
whatsnew(), 308

```