pandagg Documentation

Release 0.1

Léonard Binet

Apr 14, 2020

Contents

CHAPTER 1

Principles

Note: This is a work in progress. Some sections still need to be furnished.

pandagg is designed for both for "regular" code repository usage, and "interactive" usage (ipython or jupyter notebook usage with autocompletion features inspired by pandas design).

This library focuses on two principles:

- stick to the **tree** structure of Elasticsearch objects
- provide simple and flexible interfaces to make it easy and intuitive to use in an interactive usage

1.1 Elasticsearch tree structures

Many Elasticsearch objects have a **tree** structure, ie they are built from a hierarchy of **nodes**:

- a mapping (tree) is a hierarchy of fields (nodes)
- a query (tree) is a hierarchy of query clauses (nodes)
- an aggregation (tree) is a hierarchy of aggregation clauses (nodes)
- an aggregation response (tree) is a hierarchy of response buckets (nodes)

This library aims to stick to that structure by providing a flexible syntax distinguishing **trees** and **nodes**.

1.2 Interactive usage

Some classes are not intended to be used elsewhere than in interactive mode (ipython), since their purpose is to serve auto-completion features and convenient representations.

They won't serve you for any other usage than interactive ones.

Namely:

- *pandagg.mapping.IMapping*: used to interactively navigate in mapping and run quick aggregations on some fields
- *pandagg.client.Elasticsearch*: used to discover cluster indices, and eventually navigate their mappings, or run quick access aggregations or queries.
- *pandagg.agg.AggResponse*: used to interactively navigate in an aggregation response

These use case will be detailed in following sections.

CHAPTER 2

User Guide

Note: Examples will be based on *IMDB dataset* data. This is a work in progress. Some sections still need to be furnished.

2.1 Query

The *Query* class allows multiple ways to declare and udpate an Elasticsearch query.

Let's explore the multiple ways we have to declare the following query:

```
>>> expected_query = {'bool': {'must': [
>>>     {'terms': {'genres': ['Action', 'Thriller']}},
>>>     {'range': {'rank': {'gte': 7}}},
>>>     {'nested': {
>>>         'path': 'roles',
>>>         'query': {'bool': {'must': [
>>>             {'term': {'roles.gender': {'value': 'F'}}},
>>>             {'term': {'roles.role': {'value': 'Reporter'}}}]}}
>>>          }
>>>     }}
>>> ]}}
```

2.1.1 Pandagg DSL

Pandagg provides a DSL to declare this query in a quite similar fashion:

```
>>> from pandagg.query import Nested, Bool, Query, Range, Term, Terms
```

```
>>> q = Query(
>>>     Bool(must=[
>>>         Terms('genres', terms=['Action', 'Thriller']),
>>>         Range('rank', gte=7),
>>>         Nested(
>>>             path='roles',
>>>             query=Bool(must=[
>>>                 Term('roles.gender', value='F'),
>>>                 Term('roles.role', value='Reporter')
>>>             ])
>>>         )
>>>     ])
>>>)
```

The serialized query is then available with *query_dict* method:

```
>>> q.query_dict() == expected_query
True
```

A visual representation of the query helps to have a clearer view:

```
>>> q
<Query>
bool
└── must
    ├── nested
    │   ├── path="roles"
    │   └── query
    │       └── bool
    │           └── must
    │               ├── term, field=roles.gender, value="F"
    │               └── term, field=roles.role, value="Reporter"
    ├── range, field=rank, gte=7
    └── terms, field=genres, values=['Action', 'Thriller']
```

2.1.2 Chaining

Another way to declare this query is through chaining:

```
>>> from pandagg.utils import equal_queries
>>> from pandagg.query import Nested, Bool, Query, Range, Term, Terms
```

```
>>> q = Query()\
>>>     .query({'terms': {'genres': ['Action', 'Thriller']}})\
>>>     .nested(path='roles', _name='nested_roles', query=Term('roles.gender', value=
↪'F'))\
>>>     .query(Range('rank', gte=7))\
>>>     .query(Term('roles.role', value='Reporter'), parent='nested_roles')
```

```
>>> equal_queries(q.query_dict(), expected_query)
True
```

Note: *equal_queries* function won't consider order of clauses in must/should parameters since it actually doesn't matter in Elasticsearch execution, ie

```
>>> equal_queries({'must': [A, B]}, {'must': [B, A]})
True
```

2.1.3 Regular syntax

Eventually, you can also use regular Elasticsearch dict syntax:

```
>>> q = Query(expected_query)
>>> q
<Query>
bool
└── must
    ├── nested
    │   ├── path="roles"
    │   └── query
    │       └── bool
    │           └── must
    │               ├── term, field=roles.gender, value="F"
    │               └── term, field=roles.role, value="Reporter"
    ├── range, field=rank, gte=7
    └── terms, field=genres, values=['Action', 'Thriller']
```

2.2 Aggregation

2.2.1 Aggregation declaration

2.2.2 Aggregation response

TODO

2.3 Mapping

Here is a portion of *IMDB dataset* example mapping:

```
>>> imdb_mapping = {
>>>     'dynamic': False,
>>>     'properties': {
>>>         'movie_id': {'type': 'integer'},
>>>         'name': {
>>>             'type': 'text',
>>>             'fields': {
>>>                 'raw': {'type': 'keyword'}
>>>             }
>>>         },
>>>         'year': {
>>>             'type': 'date',
>>>             'format': 'yyyy'
>>>         },
>>>         'rank': {'type': 'float'},
```

(continues on next page)

(continued from previous page)

```
>>>         'genres': {'type': 'keyword'},
>>>         'roles': {
>>>             'type': 'nested',
>>>             'properties': {
>>>                 'role': {'type': 'keyword'},
>>>                 'actor_id': {'type': 'integer'},
>>>                 'gender': {'type': 'keyword'},
>>>                 'first_name':  {
>>>                     'type': 'text',
>>>                     'fields': {
>>>                         'raw': {'type': 'keyword'}
>>>                     }
>>>                 },
>>>                 'last_name':  {
>>>                     'type': 'text',
>>>                     'fields': {
>>>                         'raw': {'type': 'keyword'}
>>>                     }
>>>                 }
>>>             }
>>>         }
>>>     }
>>> }
```

2.3.1 Mapping DSL

The *Mapping* class provides a more compact view, which can help when dealing with large mappings:

```
>>> from pandagg.mapping import Mapping
>>> m = Mapping(imdb_mapping)
<Mapping>
                                                        {Object}
├── genres                                               Keyword
├── movie_id                                             Integer
├── name                                                 Text
│   └── raw                                           ~ Keyword
├── rank                                                 Float
├── roles                                               [Nested]
│   ├── actor_id                                         Integer
│   ├── first_name                                       Text
│   │   └── raw                                       ~ Keyword
│   ├── gender                                           Keyword
│   ├── last_name                                        Text
│   │   └── raw                                       ~ Keyword
│   └── role                                             Keyword
└── year                                                 Date
```

With pandagg DSL, an equivalent declaration would be the following:

```
>>> from pandagg.mapping import Mapping, Object, Nested, Float, Keyword, Date,␣
↪Integer, Text
>>>
>>> dsl_mapping = Mapping(properties=[
>>>     Integer('movie_id'),
>>>     Text('name', fields=[
```

(continues on next page)

(continued from previous page)

```
>>>         Keyword('raw')
>>>     ]),
>>>     Date('year', format='yyyy'),
>>>     Float('rank'),
>>>     Keyword('genres'),
>>>     Nested('roles', properties=[
>>>         Keyword('role'),
>>>         Integer('actor_id'),
>>>         Keyword('gender'),
>>>         Text('first_name', fields=[
>>>             Keyword('raw')
>>>         ]),
>>>         Text('last_name', fields=[
>>>             Keyword('raw')
>>>         ])
>>>     ])
>>> ])
```

Which is exactly equivalent to initial mapping:

```
>>> dsl_mapping.serialize() == imdb_mapping
True
```

2.3.2 Interactive mapping

In interactive context, the *IMapping* class provides navigation features with autocompletion to quickly discover a large mapping:

```
>>> from pandagg.mapping import IMapping
>>> m = IMapping(imdb_mapping)
>>> m.roles
<IMapping subpart: roles>
roles                                                    [Nested]
├── actor_id                                              Integer
├── first_name                                            Text
│   └── raw                                            ~ Keyword
├── gender                                                Keyword
├── last_name                                             Text
│   └── raw                                            ~ Keyword
└── role                                                  Keyword
>>> m.roles.first_name
<IMapping subpart: roles.first_name>
first_name                                             Text
└── raw                                             ~ Keyword
```

To get the complete field definition, just call it:

```
>>> m.roles.first_name()
<Mapping Field first_name> of type text:
{
    "type": "text",
    "fields": {
        "raw": {
            "type": "keyword"
        }
```

(continues on next page)

(continued from previous page)

```
    }
}
```

A **IMapping** instance can be bound to an Elasticsearch client to get quick access to aggregations computation on mapping fields.

Suppose you have the following client:

```
>>> from elasticsearch import Elasticsearch
>>> client = Elasticsearch(hosts=['localhost:9200'])
```

Client can be bound either at initiation:

```
>>> m = IMapping(imdb_mapping, client=client, index_name='movies')
```

or afterwards through *bind* method:

```
>>> m = IMapping(imdb_mapping)
>>> m.bind(client=client, index_name='movies')
```

Doing so will generate a **a** attribute on mapping fields, this attribute will list all available aggregation for that field type (with autocompletion):

```
>>> m.roles.gender.a.terms()
[('M', {'key': 'M', 'doc_count': 2296792}),
('F', {'key': 'F', 'doc_count': 1135174})]
```

Note: Nested clauses will be automatically taken into account.

2.4 Cluster indices discovery

TODO

CHAPTER 3

Advanced usage

Note: This is a work in progress. Some sections still need to be furnished.

- node and tree deserialization order
- compound query insertion

CHAPTER 4

IMDB dataset

You might know the Internet Movie Database, commonly called IMDB.

Well it's a good simple example to showcase ElasticSearch capabilities.

In this case, relational databases (SQL) are a good fit to store with consistence this kind of data. Yet indexing some of this data in a optimized search engine will allow more powerful queries.

4.1 Query requirements

In this example, we'll suppose most usage/queries requirements will be around the concept of movie (rather than usages focused on fetching actors or directors, even though it will still be possible with this data structure).

The index should provide good performances trying to answer these kind question (non-exhaustive):

- in which movies this actor played?
- what movies genres were most popular among decades?
- which actors have played in best-rated movies, or worst-rated movies?
- which actors movies directors prefer to cast in their movies?
- which are best ranked movies of last decade in Action or Documentary genres?
- ...

4.2 Data source

I exported following SQL tables from MariaDB following these instructions.

Relational schema is the following:

imdb tables

4.3 Index mapping

4.3.1 Overview

The base unit (document) will be a movie, having a name, rank (ratings), year of release, a list of actors and a list of directors.

Schematically:

```
Movie:
 - name
 - year
 - rank
 - [] genres
 - [] directors
 - [] actor roles
```

4.3.2 Which fields require nesting?

Since genres contain a single keyword field, in no case we need it to be stored as a nested field. On the contrary, actor roles and directors require a nested mapping if we consider applying multiple simultanous query clauses on their sub-fields (for instance search movie in which actor is a woman AND whose role is nurse). More information on distinction between array and nested fields here.

4.3.3 Text or keyword fields?

Some fields are easy to choose, in no situation gender will require a full text search, thus we'll store it as a keyword. On the other hand actors and directors names (first and last) will require full-text search, we'll thus opt for a text field. Yet we might want to aggregate on exact keywords to count number of movies per actor for instance. More inforamtion on distinction between text and keyword fields here

4.3.4 Final mapping

*TODO -> use copy_to parameter to build **full_name***

```
.
├── directors                       [Nested]
│   ├── director_id                  Integer
│   ├── first_name                   Text
│   │   └── raw                    ~ Keyword
│   ├── full_name                    Text
│   │   └── raw                    ~ Keyword
│   ├── genres                       Keyword
│   └── last_name                    Text
│       └── raw                    ~ Keyword
├── genres                           Keyword
├── movie_id                         Integer
├── name                             Text
│   └── raw                        ~ Keyword
├── rank                             Float
├── roles                           [Nested]
│   ├── actor_id                     Integer
```

(continues on next page)

(continued from previous page)

```
│   ├── first_name                    Text
│   │   └── raw                     ~ Keyword
│   ├── full_name                     Text
│   │   └── raw                     ~ Keyword
│   ├── gender                        Keyword
│   ├── last_name                     Text
│   │   └── raw                     ~ Keyword
│   └── role                          Keyword
└── year                              Date
```

4.4 Steps to start playing with your index

Note to Elastic, if you have a spare cluster to prepare demo indices on which you could let your community perform read operations we could skip this step ;)

4.4.1 Dump tables

Follow instruction on bottom of https://relational.fit.cvut.cz/dataset/IMDb page and dump following tables in a directory:

- movies.csv
- movies_genres.csv
- movies_directors.csv
- directors.csv
- directors_genres.csv
- roles.csv
- actors.csv

4.4.2 Clone pandagg and setup environment

```
git clone git@github.com:alkemics/pandagg.git
cd pandagg

virtualenv env
python setup.py develop
pip install pandas simplejson jupyter seaborn
```

Then copy `conf.py.dist` file into `conf.py` and edit variables as suits you, for instance:

```
# your cluster address
ES_HOST = 'localhost:9200'

# where your table dumps are stored, and where serialized output will be written
DATA_DIR = '/path/to/dumps/'
OUTPUT_FILE_NAME = 'serialized.json'
```

4.4.3 Serialize movie documents and insert them

```
# generate serialized movies documents, ready to be inserted in ES
# can take a while
python examples/imdb/serialize.py

# create index with mapping if necessary, bulk insert documents in ES
python examples/imdb/load.py
```

4.4.4 Explore pandagg notebooks

An example notebook is available to showcase some of `pandagg` functionalities: here it is.

Code is present in `examples/imdb/IMDB exploration.py` file.

CHAPTER 5

pandagg package

5.1 Subpackages

5.1.1 pandagg.interactive package

5.1.1.1 Submodules

pandagg.interactive.abstract module

class pandagg.interactive.abstract.**Obj**(***kwargs*)

Bases: object

Object class that allows to get items both by attribute *__getattribute__* access: *obj.attribute* or by dict *__getitem__* access: >>> obj = Obj(key='value') >>> obj.key 'value' >>> obj['key'] 'value'

In Ipython interpreter, attributes will be available in autocomplete (except private ones): >>> obj = Obj(key='value', key2='value2') >>> obj.k # press tab for autocompletion key key2

Items names that are not compliant with python attributes (accepted characters are [**a-zA-Z0-9_**] without beginning with a figure), will be only available through dict *__getitem__* access.

class pandagg.interactive.abstract.**TreeBasedObj**(*tree*, *root_path=None*, *depth=1*, *initial_tree=None*)

Bases: *pandagg.interactive.abstract.Obj*

Recursive Obj whose structure is defined by a treelib.Tree object.

The main purpose of this object is to iteratively expand the tree as attributes of this object. To avoid creating useless instances, only direct children of accessed nodes are expanded.

pandagg.interactive.abstract.**is_valid_attr_name**(*item*)

pandagg.interactive.client module

class pandagg.interactive.client.**Elasticsearch**(*hosts=None*, *transport_class=<class 'elasticsearch.transport.Transport'>*, ***kwargs*)

Bases: elasticsearch.client.Elasticsearch

fetch_indices(*index='*'*)

Parameters **index** – Comma-separated list or wildcard expression of index names used to limit the request.

pandagg.interactive.index module

class pandagg.interactive.index.**Aliases**(***kwargs*)

Bases: *pandagg.interactive.abstract.Obj*

class pandagg.interactive.index.**Index**(*name*, *settings*, *mapping*, *aliases*, *client=None*)

Bases: *pandagg.interactive.abstract.Obj*

agg(*arg*, *insert_below=None*, *output='dataframe'*, *execute=True*, ***kwargs*)

groupby(*by*, *insert_below=None*, ***kwargs*)

query(*query*, *validate=False*, ***kwargs*)

set_mapping(*mapping*)

class pandagg.interactive.index.**Indices**(***kwargs*)

Bases: *pandagg.interactive.abstract.Obj*

pandagg.interactive.mapping module

class pandagg.interactive.mapping.**IMapping**(*from_=None*, *properties=None*, *dynamic=False*, *client=None*, *root_path=None*, *depth=1*, *initial_tree=None*, *index_name=None*)

Bases: *pandagg.interactive.abstract.TreeBasedObj*

Interactive wrapper upon mapping tree.

pandagg.interactive.mapping.**as_mapping**(*mapping*)

pandagg.interactive.response module

class pandagg.interactive.response.**IResponse**(*tree*, *client=None*, *index_name=None*, *root_path=None*, *depth=None*, *initial_tree=None*, *query=None*)

Bases: *pandagg.interactive.abstract.TreeBasedObj*

Interactive aggregation response.

get_bucket_filter()

Build filters to select documents belonging to that bucket

list_documents(*size=None*, *execute=True*, *_source=None*, *compact=False*, ***kwargs*)

Return ES aggregation query to list documents belonging to given bucket. :param size: number of returned documents (ES default: 20) :param execute: if set to False, return aggregation query :param _source: list of

desired documents attributes :param compact: provide more compact ES response :param kwargs: query arguments passed to aggregation body :return:

5.1.1.2 Module contents

5.1.2 pandagg.node package

5.1.2.1 Subpackages

pandagg.node.agg package

Submodules

pandagg.node.agg.abstract module

class pandagg.node.agg.abstract.**AggNode**(*name*, *meta=None*, ***body*)
Bases: pandagg.node._node.Node

Wrapper around elasticsearch aggregation concept. https://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations.html

Each aggregation can be seen both a Node that can be encapsulated in a parent agg.

Define a method to build aggregation request.

BLACKLISTED_MAPPING_TYPES = None

KEY = NotImplementedError()

VALUE_ATTRS = NotImplementedError()

WHITELISTED_MAPPING_TYPES = None

classmethod deserialize(*name*, *body*, *meta=None*)

classmethod extract_bucket_value(*response*, *value_as_dict=False*)

extract_buckets(*response_value*)

get_filter(*key*)
Return filter query to list documents having this aggregation key. :param key: string :return: elasticsearch filter query

query_dict(*with_name=False*)
ElasticSearch aggregation queries follow this formatting: {

“<aggregation_name>” [{]

“<aggregation_type>” [{] <aggregation_body>

} [,”meta” : { [<meta_data_body>] }]?

}

}

Query dict returns the following part (without aggregation name): {

“<aggregation_type>” [{] <aggregation_body>

} [,”meta” : { [<meta_data_body>] }]?

}

tag
The readable node name for human. This attribute can be accessed and modified with . and = operator respectively.

classmethod valid_on_field_type(*field_type*)

class pandagg.node.agg.abstract.**BucketAggNode**(*name*, *meta=None*, *aggs=None*, ***body*)
Bases: *pandagg.node.agg.abstract.AggNode*

Bucket aggregation have special abilities: they can encapsulate other aggregations as children. Each time, the extracted value is a 'doc_count'.

Provide methods: - to build aggregation request (with children aggregations) - to to extract buckets from raw response - to build query to filter documents belonging to that bucket

Note: the aggs attribute's only purpose is for children initiation with the following syntax: >>> from pandagg.agg import Terms, Avg >>> agg = Terms(>>> name='term_agg', >>> field='some_path', >>> aggs=[>>> Avg(agg_name='avg_agg', field='some_other_path') >>>] >>>) Yet, the aggs attribute will then be reset to None to avoid confusion since the real hierarchy is stored in the bpointer/fpointer attributes inherited from treelib.Tree class.

VALUE_ATTRS = NotImplementedError()

extract_buckets(*response_value*)

get_filter(*key*)
Provide filter to get documents belonging to document of given key.

class pandagg.node.agg.abstract.**FieldOrScriptMetricAgg**(*name*, *meta=None*, ***body*)
Bases: *pandagg.node.agg.abstract.MetricAgg*

Metric aggregation based on single field.

VALUE_ATTRS = NotImplementedError()

class pandagg.node.agg.abstract.**MetricAgg**(*name*, *meta=None*, ***body*)
Bases: *pandagg.node.agg.abstract.AggNode*

Metric aggregation are aggregations providing a single bucket, with value attributes to be extracted.

VALUE_ATTRS = NotImplementedError()

extract_buckets(*response_value*)

get_filter(*key*)
Return filter query to list documents having this aggregation key. :param key: string :return: elasticsearch filter query

class pandagg.node.agg.abstract.**MultipleBucketAgg**(*name*, *keyed=None*, *key_path='key'*, *meta=None*, *aggs=None*, ***body*)
Bases: *pandagg.node.agg.abstract.BucketAggNode*

IMPLICIT_KEYED = False

VALUE_ATTRS = NotImplementedError()

extract_buckets(*response_value*)

get_filter(*key*)
Provide filter to get documents belonging to document of given key.

class pandagg.node.agg.abstract.**Pipeline**(*name*, *buckets_path*, *gap_policy=None*, *meta=None*, *aggs=None*, ***body*)

Bases: *pandagg.node.agg.abstract.UniqueBucketAgg*

VALUE_ATTRS = NotImplementedError()

get_filter(*key*)

Provide filter to get documents belonging to document of given key.

class pandagg.node.agg.abstract.**ScriptPipeline**(*name*, *script*, *buckets_path*, *gap_policy=None*, *meta=None*, *aggs=None*, ***body*)

Bases: *pandagg.node.agg.abstract.Pipeline*

KEY = NotImplementedError()

VALUE_ATTRS = 'value'

class pandagg.node.agg.abstract.**ShadowRoot**

Bases: *pandagg.node.agg.abstract.BucketAggNode*

Not a real aggregation.

KEY = 'shadow_root'

tag

The readable node name for human. This attribute can be accessed and modified with . and = operator respectively.

class pandagg.node.agg.abstract.**UniqueBucketAgg**(*name*, *meta=None*, *aggs=None*, ***body*)

Bases: *pandagg.node.agg.abstract.BucketAggNode*

Aggregations providing a single bucket.

VALUE_ATTRS = NotImplementedError()

extract_buckets(*response_value*)

get_filter(*key*)

Provide filter to get documents belonging to document of given key.

pandagg.node.agg.bucket module

Not implemented aggregations include: - children agg - geo-distance - geo-hash grid - ipv4 - sampler - significant terms

class pandagg.node.agg.bucket.**DateHistogram**(*name*, *field*, *interval=None*, *calendar_interval=None*, *fixed_interval=None*, *meta=None*, *keyed=False*, *key_as_string=True*, *aggs=None*, ***body*)

Bases: *pandagg.node.agg.abstract.MultipleBucketAgg*

KEY = 'date_histogram'

VALUE_ATTRS = ['doc_count']

WHITELISTED_MAPPING_TYPES = ['date']

get_filter(*key*)

Provide filter to get documents belonging to document of given key.

class pandagg.node.agg.bucket.**DateRange**(*name*, *field*, *key_as_string=True*, *aggs=None*, *meta=None*, ***body*)

Bases: *pandagg.node.agg.bucket.Range*

KEY = 'date_range'

KEY_SEP = '::'

VALUE_ATTRS = ['doc_count']

WHITELISTED_MAPPING_TYPES = ['date']

class pandagg.node.agg.bucket.**Filter**(*name*, *filter*, *meta=None*, *aggs=None*, ***body*)

Bases: *pandagg.node.agg.abstract.UniqueBucketAgg*

KEY = 'filter'

VALUE_ATTRS = ['doc_count']

get_filter(*key*)

Provide filter to get documents belonging to document of given key.

class pandagg.node.agg.bucket.**Filters**(*name*, *filters*, *other_bucket=False*, *other_bucket_key=None*, *meta=None*, *aggs=None*, ***body*)

Bases: *pandagg.node.agg.abstract.MultipleBucketAgg*

DEFAULT_OTHER_KEY = '_other_'

IMPLICIT_KEYED = True

KEY = 'filters'

VALUE_ATTRS = ['doc_count']

get_filter(*key*)

Provide filter to get documents belonging to document of given key.

class pandagg.node.agg.bucket.**Global**(*name*, *meta=None*, *aggs=None*)

Bases: *pandagg.node.agg.abstract.UniqueBucketAgg*

KEY = 'global'

VALUE_ATTRS = ['doc_count']

get_filter(*key*)

Provide filter to get documents belonging to document of given key.

class pandagg.node.agg.bucket.**Histogram**(*name*, *field*, *interval*, *meta=None*, *aggs=None*, ***body*)

Bases: *pandagg.node.agg.abstract.MultipleBucketAgg*

KEY = 'histogram'

VALUE_ATTRS = ['doc_count']

WHITELISTED_MAPPING_TYPES = ['long', 'integer', 'short', 'byte', 'double', 'float', 'h

get_filter(*key*)

Provide filter to get documents belonging to document of given key.

class pandagg.node.agg.bucket.**MatchAll**(*name*, *meta=None*, *aggs=None*)

Bases: *pandagg.node.agg.bucket.Filter*

class pandagg.node.agg.bucket.**Missing**(*name*, *field*, *meta=None*, *aggs=None*, ***body*)

Bases: *pandagg.node.agg.abstract.UniqueBucketAgg*

BLACKLISTED_MAPPING_TYPES = []

KEY = 'missing'

VALUE_ATTRS = ['doc_count']

get_filter(*key*)
Provide filter to get documents belonging to document of given key.

class pandagg.node.agg.bucket.**Nested**(*name*, *path*, *meta=None*, *aggs=None*)
Bases: *pandagg.node.agg.abstract.UniqueBucketAgg*

KEY = 'nested'

VALUE_ATTRS = ['doc_count']

WHITELISTED_MAPPING_TYPES = ['nested']

get_filter(*key*)
Provide filter to get documents belonging to document of given key.

class pandagg.node.agg.bucket.**Range**(*name*, *field*, *ranges*, *keyed=False*, *meta=None*, *aggs=None*, ***body*)
Bases: *pandagg.node.agg.abstract.MultipleBucketAgg*

KEY = 'range'

KEY_SEP = '-'

VALUE_ATTRS = ['doc_count']

WHITELISTED_MAPPING_TYPES = ['long', 'integer', 'short', 'byte', 'double', 'float', 'h

from_key

get_filter(*key*)
Provide filter to get documents belonging to document of given key.

to_key

class pandagg.node.agg.bucket.**ReverseNested**(*name*, *path=None*, *meta=None*, *aggs=None*, ***body*)
Bases: *pandagg.node.agg.abstract.UniqueBucketAgg*

KEY = 'reverse_nested'

VALUE_ATTRS = ['doc_count']

WHITELISTED_MAPPING_TYPES = ['nested']

get_filter(*key*)
Provide filter to get documents belonging to document of given key.

class pandagg.node.agg.bucket.**Terms**(*name*, *field*, *missing=None*, *size=None*, *aggs=None*, *meta=None*, ***body*)
Bases: *pandagg.node.agg.abstract.MultipleBucketAgg*

Terms aggregation.

BLACKLISTED_MAPPING_TYPES = []

KEY = 'terms'

VALUE_ATTRS = ['doc_count', 'doc_count_error_upper_bound', 'sum_other_doc_count']

get_filter(*key*)
Provide filter to get documents belonging to document of given key.

pandagg.node.agg.deserializer module

pandagg.node.agg.deserializer.**deserialize_agg**(*d*)

pandagg.node.agg.metric module

class pandagg.node.agg.metric.**Avg**(*name*, *meta=None*, ***body*)
Bases: *pandagg.node.agg.abstract.FieldOrScriptMetricAgg*

KEY = 'avg'

VALUE_ATTRS = ['value']

WHITELISTED_MAPPING_TYPES = ['long', 'integer', 'short', 'byte', 'double', 'float', 'ha

class pandagg.node.agg.metric.**Cardinality**(*name*, *meta=None*, ***body*)
Bases: *pandagg.node.agg.abstract.FieldOrScriptMetricAgg*

KEY = 'cardinality'

VALUE_ATTRS = ['value']

class pandagg.node.agg.metric.**ExtendedStats**(*name*, *meta=None*, ***body*)
Bases: *pandagg.node.agg.abstract.FieldOrScriptMetricAgg*

KEY = 'extended_stats'

VALUE_ATTRS = ['count', 'min', 'max', 'avg', 'sum', 'sum_of_squares', 'variance', 'std

WHITELISTED_MAPPING_TYPES = ['long', 'integer', 'short', 'byte', 'double', 'float', 'ha

class pandagg.node.agg.metric.**GeoBound**(*name*, *meta=None*, ***body*)
Bases: *pandagg.node.agg.abstract.FieldOrScriptMetricAgg*

KEY = 'geo_bounds'

VALUE_ATTRS = ['bounds']

WHITELISTED_MAPPING_TYPES = ['geo_point']

class pandagg.node.agg.metric.**GeoCentroid**(*name*, *meta=None*, ***body*)
Bases: *pandagg.node.agg.abstract.FieldOrScriptMetricAgg*

KEY = 'geo_centroid'

VALUE_ATTRS = ['location']

WHITELISTED_MAPPING_TYPES = ['geo_point']

class pandagg.node.agg.metric.**Max**(*name*, *meta=None*, ***body*)
Bases: *pandagg.node.agg.abstract.FieldOrScriptMetricAgg*

KEY = 'max'

VALUE_ATTRS = ['value']

WHITELISTED_MAPPING_TYPES = ['long', 'integer', 'short', 'byte', 'double', 'float', 'ha

class pandagg.node.agg.metric.**Min**(*name*, *meta=None*, ***body*)
Bases: *pandagg.node.agg.abstract.FieldOrScriptMetricAgg*

KEY = 'min'

VALUE_ATTRS = ['value']

WHITELISTED_MAPPING_TYPES = ['long', 'integer', 'short', 'byte', 'double', 'float', 'ha

class pandagg.node.agg.metric.**PercentileRanks**(*name*, *field*, *values*, *meta=None*, ***body*)
Bases: *pandagg.node.agg.abstract.FieldOrScriptMetricAgg*

KEY = 'percentile_ranks'

VALUE_ATTRS = ['values']

WHITELISTED_MAPPING_TYPES = ['long', 'integer', 'short', 'byte', 'double', 'float', 'ha

class pandagg.node.agg.metric.**Percentiles**(*name*, *meta=None*, ***body*)
Bases: *pandagg.node.agg.abstract.FieldOrScriptMetricAgg*

Percents body argument can be passed to specify which percentiles to fetch.

KEY = 'percentiles'

VALUE_ATTRS = ['values']

WHITELISTED_MAPPING_TYPES = ['long', 'integer', 'short', 'byte', 'double', 'float', 'ha

class pandagg.node.agg.metric.**Stats**(*name*, *meta=None*, ***body*)
Bases: *pandagg.node.agg.abstract.FieldOrScriptMetricAgg*

KEY = 'stats'

VALUE_ATTRS = ['count', 'min', 'max', 'avg', 'sum']

WHITELISTED_MAPPING_TYPES = ['long', 'integer', 'short', 'byte', 'double', 'float', 'ha

class pandagg.node.agg.metric.**Sum**(*name*, *meta=None*, ***body*)
Bases: *pandagg.node.agg.abstract.FieldOrScriptMetricAgg*

KEY = 'sum'

VALUE_ATTRS = ['value']

WHITELISTED_MAPPING_TYPES = ['long', 'integer', 'short', 'byte', 'double', 'float', 'ha

class pandagg.node.agg.metric.**TopHits**(*name*, *meta=None*, ***body*)
Bases: *pandagg.node.agg.abstract.MetricAgg*

KEY = 'top_hits'

VALUE_ATTRS = ['hits']

class pandagg.node.agg.metric.**ValueCount**(*name*, *meta=None*, ***body*)
Bases: *pandagg.node.agg.abstract.FieldOrScriptMetricAgg*

BLACKLISTED_MAPPING_TYPES = []

KEY = 'value_count'

VALUE_ATTRS = ['value']

pandagg.node.agg.pipeline module

Pipeline aggregations: https://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations-pipeline.html

class pandagg.node.agg.pipeline.**AvgBucket**(*name*, *buckets_path*, *gap_policy=None*, *meta=None*, *aggs=None*, ***body*)
Bases: *pandagg.node.agg.abstract.Pipeline*

KEY = 'avg_bucket'

VALUE_ATTRS = ['value']

class pandagg.node.agg.pipeline.**BucketScript**(*name*, *script*, *buckets_path*, *gap_policy=None*, *meta=None*, *aggs=None*, ***body*)

Bases: *pandagg.node.agg.abstract.ScriptPipeline*

KEY = 'bucket_script'

VALUE_ATTRS = ['value']

class pandagg.node.agg.pipeline.**BucketSelector**(*name*, *script*, *buckets_path*, *gap_policy=None*, *meta=None*, *aggs=None*, ***body*)

Bases: *pandagg.node.agg.abstract.ScriptPipeline*

KEY = 'bucket_selector'

VALUE_ATTRS = None

class pandagg.node.agg.pipeline.**BucketSort**(*name*, *script*, *buckets_path*, *gap_policy=None*, *meta=None*, *aggs=None*, ***body*)

Bases: *pandagg.node.agg.abstract.ScriptPipeline*

KEY = 'bucket_sort'

VALUE_ATTRS = None

class pandagg.node.agg.pipeline.**CumulativeSum**(*name*, *buckets_path*, *gap_policy=None*, *meta=None*, *aggs=None*, ***body*)

Bases: *pandagg.node.agg.abstract.Pipeline*

KEY = 'cumulative_sum'

VALUE_ATTRS = ['value']

class pandagg.node.agg.pipeline.**Derivative**(*name*, *buckets_path*, *gap_policy=None*, *meta=None*, *aggs=None*, ***body*)

Bases: *pandagg.node.agg.abstract.Pipeline*

KEY = 'derivative'

VALUE_ATTRS = ['value']

class pandagg.node.agg.pipeline.**ExtendedStatsBucket**(*name*, *buckets_path*, *gap_policy=None*, *meta=None*, *aggs=None*, ***body*)

Bases: *pandagg.node.agg.abstract.Pipeline*

KEY = 'extended_stats_bucket'

VALUE_ATTRS = ['count', 'min', 'max', 'avg', 'sum', 'sum_of_squares', 'variance', 'std

class pandagg.node.agg.pipeline.**MaxBucket**(*name*, *buckets_path*, *gap_policy=None*, *meta=None*, *aggs=None*, ***body*)

Bases: *pandagg.node.agg.abstract.Pipeline*

KEY = 'max_bucket'

VALUE_ATTRS = ['value']

class pandagg.node.agg.pipeline.**MinBucket**(*name*, *buckets_path*, *gap_policy=None*, *meta=None*, *aggs=None*, ***body*)

Bases: *pandagg.node.agg.abstract.Pipeline*

KEY = 'min_bucket'

VALUE_ATTRS = ['value']

class pandagg.node.agg.pipeline.**MovingAvg**(*name*, *buckets_path*, *gap_policy=None*, *meta=None*, *aggs=None*, ***body*)

Bases: *pandagg.node.agg.abstract.Pipeline*

KEY = 'moving_avg'

VALUE_ATTRS = ['value']

class pandagg.node.agg.pipeline.**PercentilesBucket**(*name*, *buckets_path*, *gap_policy=None*, *meta=None*, *aggs=None*, ***body*)

Bases: *pandagg.node.agg.abstract.Pipeline*

KEY = 'percentiles_bucket'

VALUE_ATTRS = ['values']

class pandagg.node.agg.pipeline.**SerialDiff**(*name*, *buckets_path*, *gap_policy=None*, *meta=None*, *aggs=None*, ***body*)

Bases: *pandagg.node.agg.abstract.Pipeline*

KEY = 'serial_diff'

VALUE_ATTRS = ['value']

class pandagg.node.agg.pipeline.**StatsBucket**(*name*, *buckets_path*, *gap_policy=None*, *meta=None*, *aggs=None*, ***body*)

Bases: *pandagg.node.agg.abstract.Pipeline*

KEY = 'stats_bucket'

VALUE_ATTRS = ['count', 'min', 'max', 'avg', 'sum']

class pandagg.node.agg.pipeline.**SumBucket**(*name*, *buckets_path*, *gap_policy=None*, *meta=None*, *aggs=None*, ***body*)

Bases: *pandagg.node.agg.abstract.Pipeline*

KEY = 'sum_bucket'

VALUE_ATTRS = ['value']

Module contents

pandagg.node.mapping package

Submodules

pandagg.node.mapping.abstract module

class pandagg.node.mapping.abstract.**Field**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: pandagg.node._node.Node

DISPLAY_PATTERN = ' %s'

KEY = NotImplementedError()

body(*with_children=False*)

classmethod deserialize(*name*, *body*, *depth=0*, *is_subfield=False*)

reset_data()

tree_repr

pandagg.node.mapping.deserializer module

pandagg.node.mapping.deserializer.**deserialize_field**(*name*, *body*, *depth=0*, *is_subfield=False*)

pandagg.node.mapping.field_datatypes module

https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping-types.html

class pandagg.node.mapping.field_datatypes.**Alias**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

Defines an alias to an existing field.

KEY = 'alias'

class pandagg.node.mapping.field_datatypes.**Binary**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'binary'

class pandagg.node.mapping.field_datatypes.**Boolean**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'boolean'

class pandagg.node.mapping.field_datatypes.**Byte**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'byte'

class pandagg.node.mapping.field_datatypes.**Completion**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

To provide auto-complete suggestions

KEY = 'completion'

class pandagg.node.mapping.field_datatypes.**Date**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'date'

class pandagg.node.mapping.field_datatypes.**DateNanos**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'date_nanos'

class pandagg.node.mapping.field_datatypes.**DateRange**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'date_range'

class pandagg.node.mapping.field_datatypes.**DenseVector**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

Record dense vectors of float values.

KEY = 'dense_vector'

class pandagg.node.mapping.field_datatypes.**Double**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'double'

class pandagg.node.mapping.field_datatypes.**DoubleRange**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'double_range'

class pandagg.node.mapping.field_datatypes.**Flattened**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

Allows an entire JSON object to be indexed as a single field.

KEY = 'flattened'

class pandagg.node.mapping.field_datatypes.**Float**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'float'

class pandagg.node.mapping.field_datatypes.**FloatRange**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'float_range'

class pandagg.node.mapping.field_datatypes.**GeoPoint**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

For lat/lon points

KEY = 'geo_point'

class pandagg.node.mapping.field_datatypes.**GeoShape**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

For complex shapes like polygons

KEY = 'geo_shape'

class pandagg.node.mapping.field_datatypes.**HalfFloat**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'half_float'

class pandagg.node.mapping.field_datatypes.**Histogram**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

For pre-aggregated numerical values for percentiles aggregations.

KEY = 'histogram'

class pandagg.node.mapping.field_datatypes.**IP**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

for IPv4 and IPv6 addresses

KEY = 'IP'

class pandagg.node.mapping.field_datatypes.**Integer**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'integer'

class pandagg.node.mapping.field_datatypes.**IntegerRange**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'integer_range'

class pandagg.node.mapping.field_datatypes.**Join**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

Defines parent/child relation for documents within the same index

KEY = 'join'

class pandagg.node.mapping.field_datatypes.**Keyword**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'keyword'

class pandagg.node.mapping.field_datatypes.**Long**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'long'

class pandagg.node.mapping.field_datatypes.**LongRange**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'long_range'

class pandagg.node.mapping.field_datatypes.**MapperAnnotatedText**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

To index text containing special markup (typically used for identifying named entities)

KEY = 'annotated-text'

class pandagg.node.mapping.field_datatypes.**MapperMurMur3**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

To compute hashes of values at index-time and store them in the index

KEY = 'murmur3'

class pandagg.node.mapping.field_datatypes.**Nested**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

DISPLAY_PATTERN = ' [%s]'

KEY = 'nested'

class pandagg.node.mapping.field_datatypes.**Object**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

DISPLAY_PATTERN = ' {%s}'

KEY = 'object'

class pandagg.node.mapping.field_datatypes.**Percolator**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

Accepts queries from the query-dsl

KEY = 'percolator'

class pandagg.node.mapping.field_datatypes.**RankFeature**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

Record numeric feature to boost hits at query time.

KEY = 'rank_feature'

class pandagg.node.mapping.field_datatypes.**RankFeatures**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

Record numeric features to boost hits at query time.

KEY = 'rank_features'

class pandagg.node.mapping.field_datatypes.**ScaledFloat**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'scaled_float'

class pandagg.node.mapping.field_datatypes.**SearchAsYouType**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

A text-like field optimized for queries to implement as-you-type completion

KEY = 'search_as_you_type'

class pandagg.node.mapping.field_datatypes.**Shape**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

For arbitrary cartesian geometries.

KEY = 'shape'

class pandagg.node.mapping.field_datatypes.**Short**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'short'

class pandagg.node.mapping.field_datatypes.**SparseVector**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

Record sparse vectors of float values.

KEY = 'sparse_vector'

class pandagg.node.mapping.field_datatypes.**Text**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'text'

class pandagg.node.mapping.field_datatypes.**TokenCount**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

To count the number of tokens in a string

KEY = 'token_count'

pandagg.node.mapping.meta_fields module

class pandagg.node.mapping.meta_fields.**FieldNames**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

All fields in the document which contain non-null values.

KEY = '_field_names'

class pandagg.node.mapping.meta_fields.**Id**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

The document's ID.

KEY = '_id'

class pandagg.node.mapping.meta_fields.**Ignored**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

All fields in the document that have been ignored at index time because of ignore_malformed.

KEY = '_ignored'

class pandagg.node.mapping.meta_fields.**Index**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

The index to which the document belongs.

KEY = '_index'

class pandagg.node.mapping.meta_fields.**Meta**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

Application specific metadata.

KEY = '_meta'

class pandagg.node.mapping.meta_fields.**Routing**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

A custom routing value which routes a document to a particular shard.

KEY = '_routing'

class pandagg.node.mapping.meta_fields.**Size**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

The size of the _source field in bytes, provided by the mapper-size plugin.

KEY = '_size'

class pandagg.node.mapping.meta_fields.**Source**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

The original JSON representing the body of the document.

KEY = '_source'

class pandagg.node.mapping.meta_fields.**Type**(*name*, *depth=0*, *is_subfield=False*, ***body*)

Bases: *pandagg.node.mapping.abstract.Field*

The document's mapping type.

KEY = '_type'

Module contents

pandagg.node.query package

Submodules

pandagg.node.query.abstract module

class pandagg.node.query.abstract.**LeafQueryClause**(*_name=None*, ***body*)

Bases: *pandagg.node.query.abstract.QueryClause*

class pandagg.node.query.abstract.**MultiFieldsQueryClause**(*fields*, *_name=None*, ***body*)

Bases: *pandagg.node.query.abstract.LeafQueryClause*

tag

The readable node name for human. This attribute can be accessed and modified with `.` and = operator respectively.

class pandagg.node.query.abstract.**QueryClause**(*_name=None*, ***body*)

Bases: pandagg.node._node.Node

KEY = NotImplementedError()

NID_SIZE = 6

classmethod deserialize(***body*)

name

serialize(*named=False*)

tag
The readable node name for human. This attribute can be accessed and modified with . and = operator respectively.

class pandagg.node.query.abstract.**SingleFieldQueryClause**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.LeafQueryClause*

FLAT = False

SHORT_TAG = None

classmethod deserialize(***body*)

tag
The readable node name for human. This attribute can be accessed and modified with . and = operator respectively.

pandagg.node.query.compound module

class pandagg.node.query.compound.**Bool**(**args*, ***kwargs*)
Bases: *pandagg.node.query.compound.CompoundClause*

DEFAULT_OPERATOR
alias of pandagg.node.query._parameter_clause.Must

KEY = 'bool'

PARAMS_WHITELIST = ['should', 'must', 'must_not', 'filter', 'boost', 'minimum_should_m

class pandagg.node.query.compound.**Boosting**(**args*, ***kwargs*)
Bases: *pandagg.node.query.compound.CompoundClause*

DEFAULT_OPERATOR
alias of pandagg.node.query._parameter_clause.Positive

KEY = 'boosting'

PARAMS_WHITELIST = ['positive', 'negative', 'negative_boost']

class pandagg.node.query.compound.**CompoundClause**(**args*, ***kwargs*)
Bases: *pandagg.node.query.abstract.QueryClause*

Compound clauses can encapsulate other query clauses.

Note: the children attribute's only purpose is for initiation with the following syntax: >>> from pandagg.query import Bool, Term >>> query = Bool(>>> filter=Term(field='some_path', value=3), >>> _name='term_agg', >>>) Yet, the children attribute will then be reset to None to avoid confusion since the real hierarchy is stored in the bpointer/fpointer attributes inherited from treelib.Tree class.

DEFAULT_OPERATOR = NotImplementedError()

PARAMS_WHITELIST = None

classmethod deserialize(**args*, ***body*)

classmethod operator(*key*)

classmethod params(*parent_only=False*)
Return map of key -> params that handle children leaves.

class pandagg.node.query.compound.**ConstantScore**(**args*, ***kwargs*)
Bases: *pandagg.node.query.compound.CompoundClause*

DEFAULT_OPERATOR
alias of pandagg.node.query._parameter_clause.Filter

KEY = 'constant_score'

PARAMS_WHITELIST = ['filter', 'boost']

class pandagg.node.query.compound.**DisMax**(**args*, ***kwargs*)
Bases: *pandagg.node.query.compound.CompoundClause*

DEFAULT_OPERATOR
alias of pandagg.node.query._parameter_clause.Queries

KEY = 'dis_max'

PARAMS_WHITELIST = ['queries', 'tie_breaker']

class pandagg.node.query.compound.**FunctionScore**(**args*, ***kwargs*)
Bases: *pandagg.node.query.compound.CompoundClause*

DEFAULT_OPERATOR
alias of pandagg.node.query._parameter_clause.QueryP

KEY = 'function_score'

PARAMS_WHITELIST = ['query', 'boost', 'random_score', 'boost_mode', 'functions', 'max_

pandagg.node.query.deserializer module

pandagg.node.query.deserializer.**deserialize_node**(*k*, *body*, *accept_leaf=True*, *accept_compound=True*, *accept_param=True*)

pandagg.node.query.full_text module

class pandagg.node.query.full_text.**Common**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'common'

class pandagg.node.query.full_text.**Intervals**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'intervals'

class pandagg.node.query.full_text.**Match**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'match'

SHORT_TAG = 'query'

class pandagg.node.query.full_text.**MatchBoolPrefix**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'match_bool_prefix'

SHORT_TAG = 'query'

class pandagg.node.query.full_text.**MatchPhrase**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'match_phrase'

SHORT_TAG = 'query'

class pandagg.node.query.full_text.**MatchPhrasePrefix**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'match_phrase_prefix'

SHORT_TAG = 'query'

class pandagg.node.query.full_text.**MultiMatch**(*fields*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.MultiFieldsQueryClause*

KEY = 'multi_match'

class pandagg.node.query.full_text.**QueryString**(*_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.LeafQueryClause*

KEY = 'query_string'

class pandagg.node.query.full_text.**SimpleQueryString**(*_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.LeafQueryClause*

KEY = 'simple_string'

pandagg.node.query.geo module

class pandagg.node.query.geo.**GeoBoundingBox**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'geo_bounding_box'

class pandagg.node.query.geo.**GeoDistance**(*field*, *location*, *distance*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.LeafQueryClause*

KEY = 'geo_distance'

classmethod deserialize(***body*)

tag
The readable node name for human. This attribute can be accessed and modified with . and = operator respectively.

class pandagg.node.query.geo.**GeoPolygone**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'geo_polygon'

class pandagg.node.query.geo.**GeoShape**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'geo_shape'

pandagg.node.query.joining module

class pandagg.node.query.joining.**HasChild**(**args*, ***kwargs*)
Bases: *pandagg.node.query.compound.CompoundClause*

DEFAULT_OPERATOR
alias of pandagg.node.query._parameter_clause.QueryP

KEY = 'has_child'

PARAMS_WHITELIST = ['query', 'type', 'max_children', 'min_children', 'score_mode', 'ig

class pandagg.node.query.joining.**HasParent**(**args*, ***kwargs*)
Bases: *pandagg.node.query.compound.CompoundClause*

DEFAULT_OPERATOR
alias of pandagg.node.query._parameter_clause.QueryP

KEY = 'has_parent'

PARAMS_WHITELIST = ['query', 'parent_type', 'score', 'ignore_unmapped']

class pandagg.node.query.joining.**Nested**(**args*, ***kwargs*)
Bases: *pandagg.node.query.compound.CompoundClause*

DEFAULT_OPERATOR
alias of pandagg.node.query._parameter_clause.QueryP

KEY = 'nested'

PARAMS_WHITELIST = ['path', 'query', 'score_mode', 'ignore_unmapped']

class pandagg.node.query.joining.**ParentId**(**args*, ***kwargs*)
Bases: *pandagg.node.query.compound.CompoundClause*

KEY = 'parent_id'

pandagg.node.query.shape module

class pandagg.node.query.shape.**Shape**(*_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.LeafQueryClause*

KEY = 'shape'

pandagg.node.query.span module

pandagg.node.query.specialized module

class pandagg.node.query.specialized.**DistanceFeature**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

FLAT = True

KEY = 'distance_feature'

class pandagg.node.query.specialized.**MoreLikeThis**(*fields*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.MultiFieldsQueryClause*

KEY = 'more_like_this'

class pandagg.node.query.specialized.**Percolate**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

FLAT = True

KEY = 'percolate'

class pandagg.node.query.specialized.**RankFeature**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

FLAT = True

KEY = 'rank_feature'

class pandagg.node.query.specialized.**Script**(*_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.LeafQueryClause*

KEY = 'script'

class pandagg.node.query.specialized.**Wrapper**(*_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.LeafQueryClause*

KEY = 'wrapper'

pandagg.node.query.specialized_compound module

class pandagg.node.query.specialized_compound.**PinnedQuery**(**args*, ***kwargs*)
Bases: *pandagg.node.query.compound.CompoundClause*

DEFAULT_OPERATOR
alias of pandagg.node.query._parameter_clause.Organic

KEY = 'pinned'

PARAMS_WHITELIST = ['ids', 'organic']

class pandagg.node.query.specialized_compound.**ScriptScore**(**args*, ***kwargs*)
Bases: *pandagg.node.query.compound.CompoundClause*

DEFAULT_OPERATOR
alias of pandagg.node.query._parameter_clause.QueryP

KEY = 'script_score'

PARAMS_WHITELIST = ['query', 'script', 'min_score']

pandagg.node.query.term_level module

class pandagg.node.query.term_level.**Exists**(*field*, *_name=None*)
Bases: *pandagg.node.query.abstract.LeafQueryClause*

KEY = 'exists'

tag
The readable node name for human. This attribute can be accessed and modified with . and = operator respectively.

class pandagg.node.query.term_level.**Fuzzy**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'fuzzy'

class pandagg.node.query.term_level.**Ids**(*values*, *_name=None*)
Bases: *pandagg.node.query.abstract.LeafQueryClause*

KEY = 'ids'

serialize(*named=False*)

tag
The readable node name for human. This attribute can be accessed and modified with . and = operator respectively.

class pandagg.node.query.term_level.**Prefix**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'prefix'

class pandagg.node.query.term_level.**Range**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'range'

class pandagg.node.query.term_level.**Regexp**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'regexp'

class pandagg.node.query.term_level.**Term**(*field*, *value*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'term'

SHORT_TAG = 'value'

class pandagg.node.query.term_level.**Terms**(*field*, *terms*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.LeafQueryClause*

KEY = 'terms'

classmethod deserialize(***body*)

tag
The readable node name for human. This attribute can be accessed and modified with `.` and = operator respectively.

class pandagg.node.query.term_level.**TermsSet**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'terms_set'

class pandagg.node.query.term_level.**Type**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'type'

class pandagg.node.query.term_level.**Wildcard**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'wildcard'

Module contents

pandagg.node.response package

Submodules

pandagg.node.response.bucket module

class pandagg.node.response.bucket.**Bucket**(*depth*, *value*, *key=None*, *level=None*)
Bases: pandagg.node._node.Node

ROOT_NAME = 'root'

attr_name
Determine under which attribute name the bucket will be available in response tree. Dots are replaced by _ characters so that they don't prevent from accessing as attribute.

Resulting attribute unfit for python attribute name syntax is still possible and will be accessible through item access (dict like), see more in 'utils.Obj' for more details.

display_name

display_name_with_value

Module contents

5.1.2.2 Submodules

pandagg.node.mixins module

class pandagg.node.mixins.**FieldValidityMixin**
Bases: object

BLACKLISTED_MAPPING_TYPES = None

WHITELISTED_MAPPING_TYPES = None

classmethod valid_on_field_type(*field_type*)

pandagg.node.types module

5.1.2.3 Module contents

5.1.3 pandagg.tree package

5.1.3.1 Submodules

pandagg.tree.agg module

class pandagg.tree.agg.**Agg**(*from_=None*, *mapping=None*, *identifier=None*, *client=None*, *query=None*, *index_name=None*)
Bases: pandagg.tree._tree.Tree

Tree combination of aggregation nodes.

Mapping declaration is optional, but doing so validates aggregation validity.

DEFAULT_OUTPUT = 'dataframe'

add_node(*node*, *pid=None*)
If mapping is provided, nested and outnested are automatically applied.

agg(*arg*, *insert_below=None*, ***kwargs*)
Arrange passed aggregations in *arg* arguments "horizontally".

Those will be placed under the *insert_below* aggregation clause id if provided, else under the deepest linear bucket aggregation if there is no ambiguity: OK: A—> B –> C –> arg KO: A—> B

└—> C

arg argument accepts single occurrence or sequence of following formats: - string (for terms agg concise declaration) - regular Elasticsearch dict syntax - AggNode instance (for instance Terms, Filters etc)

Parameters

- **arg** – aggregation(s) clauses to insert "horizontally"
- **insert_below** – parent aggregation id under which these aggregations should be placed
- **kwargs** – agg body arguments when using "string" syntax for terms aggregation

Return type *pandagg.agg.Agg*

applied_nested_path_at_node(*nid*)

bind(*client*, *index_name=None*)

deepest_linear_bucket_agg

Return deepest bucket aggregation node (pandagg.nodes.abstract.BucketAggNode) of that aggregation that neither has siblings, nor has an ancestor with siblings.

classmethod deserialize(*from_*)

execute(*index=None*, *output='dataframe'*, ***kwargs*)

groupby(*by*, *insert_below=None*, *insert_above=None*, ***kwargs*)

Arrange passed aggregations in *by* arguments "vertically" (nested manner), above or below another agg clause.

Given the initial aggregation: A—> B └—> C

If *insert_below* = 'A': A—> by—> B

└—> C

If *insert_above* = 'B': A—> by—> B └—> C

by argument accepts single occurrence or sequence of following formats: - string (for terms agg concise declaration) - regular Elasticsearch dict syntax - AggNode instance (for instance Terms, Filters etc)

If *insert_below* nor *insert_above* is provided by will be placed between the the deepest linear bucket aggregation if there is no ambiguity, and its children: A—> B : OK generates A—> B –> C –> by

A—> B : KO, ambiguous, must precise either A, B or C └—> C

Parameters

- **by** – aggregation(s) clauses to insert "vertically"
- **insert_below** – parent aggregation id under which these aggregations should be placed
- **insert_above** – aggregation id above which these aggregations should be placed
- **kwargs** – agg body arguments when using "string" syntax for terms aggregation

Return type *pandagg.agg.Agg*

node_class

alias of `pandagg.node.agg.abstract.AggNode`

paste(*nid*, *new_tree*, *deep=False*)

Pastes a tree handling nested implications if mapping is provided. The provided tree should be validated beforehands.

query(*query*, *validate=False*, ***kwargs*)

query_dict(*from_=None*, *depth=None*, *with_name=True*)

serialize_response(*aggs*, *output*, ***kwargs*)

set_mapping(*mapping*)

validate_tree(*exc=False*)
Validate tree definition against defined mapping. :param exc: if set to True, will raise exception if tree is invalid :return: boolean

pandagg.tree.mapping module

class pandagg.tree.mapping.**Mapping**(*from_=None*, *identifier=None*, *properties=None*, *dynamic=False*)
Bases: pandagg.tree._tree.Tree

contains(*nid*)
Check if the tree contains node of given id

classmethod deserialize(*from_*, *depth=0*)

list_nesteds_at_field(*field_path*)

mapping_type_of_field(*field_path*)

nested_at_field(*field_path*)

node_class
alias of *pandagg.node.mapping.abstract.Field*

node_path(*nid*)

serialize()

show(*data_property='pretty'*, ***kwargs*)
Print the tree structure in hierarchy style.

You have three ways to output your tree data, i.e., stdout with `show()`, plain text file with `save2file()`, and json string with `to_json()`. The former two use the same backend to generate a string of tree structure in a text graph.

- Version >= 1.2.7a*: you can also specify the `line_type` parameter, such as 'ascii' (default), 'ascii-ex', 'ascii-exr', 'ascii-em', 'ascii-emv', 'ascii-emh') to the change graphical form.

Parameters

- **nid** – the reference node to start expanding.
- **level** – the node level in the tree (root as level 0).
- **idhidden** – whether hiding the node ID when printing.
- **filter** – the function of one variable to act on the `Node` object. When this parameter is specified, the traversing will not continue to following children of node whose condition does not pass the filter.
- **key** – the `key` param for sorting `Node` objects in the same level.
- **reverse** – the `reverse` param for sorting `Node` objects in the same level.
- **line_type** –
- **data_property** – the property on the node data object to be printed.

Returns None

validate_agg_node(*agg_node*, *exc=True*)
Ensure if node has field or path that it exists in mapping, and that required aggregation type if allowed on this kind of field. :param agg_node: AggNode you want to validate on this mapping :param exc: boolean, if set to True raise exception if invalid :rtype: boolean

pandagg.tree.query module

class pandagg.tree.query.**Query**(*from_=None*, *mapping=None*, *identifier=None*, *client=None*, *index_name=None*)
Bases: pandagg.tree._tree.Tree

Tree combination of query nodes.

Mapping declaration is optional, but doing so validates query validity.

add_node(*node*, *pid=None*)
Add a new node object to the tree and make the parent as the root by default.

The 'node' parameter refers to an instance of Class::Node.

bind(*client*, *index_name=None*)

bool(**args*, ***kwargs*)

boost(**args*, ***kwargs*)

constant_score(**args*, ***kwargs*)

classmethod deserialize(*from_*)

dis_max(**args*, ***kwargs*)

execute(*index=None*, ***kwargs*)

filter(**args*, ***kwargs*)

function_score(**args*, ***kwargs*)

has_child(**args*, ***kwargs*)

has_parent(**args*, ***kwargs*)

must(**args*, ***kwargs*)

must_not(**args*, ***kwargs*)

nested(**args*, ***kwargs*)

node_class
alias of *pandagg.node.query.abstract.QueryClause*

parent_id(**args*, ***kwargs*)

pinned_query(**args*, ***kwargs*)

query(*q*, *parent=None*, *child=None*, *parent_param=None*, *child_param=None*, *mode='add'*)
Place query below a given parent.

query_dict(*from_=None*, *named=False*)
Return None if no query clause.

script_score(**args*, ***kwargs*)

set_mapping(*mapping*)

should(**args*, ***kwargs*)

pandagg.tree.response module

class pandagg.tree.response.**ResponseTree**(*agg_tree*, *identifier=None*)
Bases: pandagg.tree._tree.Tree

Tree representation of an ES response. ES response format is determined by the aggregation query.

bucket_properties(*bucket*, *properties=None*, *end_level=None*, *depth=None*)
Recursive method returning a given bucket's properties in the form of an ordered dictionnary. Travel from current bucket through all ancestors until reaching root. :param bucket: instance of pandagg.buckets.buckets.Bucket :param properties: OrderedDict accumulator of 'level' -> 'key' :param end_level: optional parameter to specify until which level properties are fetched :param depth: optional parameter to specify a limit number of levels which are fetched :return: OrderedDict of structure 'level' -> 'key'

get_bucket_filter(*nid*)
Build query filtering documents belonging to that bucket. Suppose the following configuration:

Base <- filter on base |— Nested_A no filter on A (nested still must be applied for children) | |— SubNested A1 | └— SubNested A2 <- filter on A2 └— Nested_B <- filter on B

parse_aggregation(*raw_response*)
Build response tree from ES response :param raw_response: ES aggregation response :return: self

Note: if the root aggregation node can generate multiple buckets, a response root is crafted to avoid having multiple roots.

show(*data_property='pretty'*, ***kwargs*)
Print the tree structure in hierarchy style.

You have three ways to output your tree data, i.e., stdout with `show()`, plain text file with `save2file()`, and json string with `to_json()`. The former two use the same backend to generate a string of tree structure in a text graph.

- Version >= 1.2.7a*: you can also specify the `line_type` parameter, such as 'ascii' (default), 'ascii-ex', 'ascii-exr', 'ascii-em', 'ascii-emv', 'ascii-emh') to the change graphical form.

Parameters

- **nid** – the reference node to start expanding.
- **level** – the node level in the tree (root as level 0).
- **idhidden** – whether hiding the node ID when printing.
- **filter** – the function of one variable to act on the `Node` object. When this parameter is specified, the traversing will not continue to following children of node whose condition does not pass the filter.
- **key** – the `key` param for sorting `Node` objects in the same level.
- **reverse** – the `reverse` param for sorting `Node` objects in the same level.
- **line_type** –
- **data_property** – the property on the node data object to be printed.

Returns None

5.1.3.2 Module contents

5.2 Submodules

5.2.1 pandagg.agg module

class pandagg.agg.**Agg**(*from_=None*, *mapping=None*, *identifier=None*, *client=None*, *query=None*, *index_name=None*)

Bases: pandagg.tree._tree.Tree

Tree combination of aggregation nodes.

Mapping declaration is optional, but doing so validates aggregation validity.

DEFAULT_OUTPUT = 'dataframe'

add_node(*node*, *pid=None*)

If mapping is provided, nested and outnested are automatically applied.

agg(*arg*, *insert_below=None*, ***kwargs*)

Arrange passed aggregations in *arg* arguments "horizontally".

Those will be placed under the *insert_below* aggregation clause id if provided, else under the deepest linear bucket aggregation if there is no ambiguity: OK: A—> B –> C –> arg KO: A—> B

└—> C

arg argument accepts single occurrence or sequence of following formats: - string (for terms agg concise declaration) - regular Elasticsearch dict syntax - AggNode instance (for instance Terms, Filters etc)

Parameters

- **arg** – aggregation(s) clauses to insert "horizontally"
- **insert_below** – parent aggregation id under which these aggregations should be placed
- **kwargs** – agg body arguments when using "string" syntax for terms aggregation

Return type *pandagg.agg.Agg*

applied_nested_path_at_node(*nid*)

bind(*client*, *index_name=None*)

deepest_linear_bucket_agg

Return deepest bucket aggregation node (pandagg.nodes.abstract.BucketAggNode) of that aggregation that neither has siblings, nor has an ancestor with siblings.

classmethod deserialize(*from_*)

execute(*index=None*, *output='dataframe'*, ***kwargs*)

groupby(*by*, *insert_below=None*, *insert_above=None*, ***kwargs*)

Arrange passed aggregations in *by* arguments "vertically" (nested manner), above or below another agg clause.

Given the initial aggregation: A—> B └—> C

If *insert_below* = 'A': A—> by—> B

└—> C

If *insert_above* = 'B': A—> by—> B └—> C

by argument accepts single occurrence or sequence of following formats: - string (for terms agg concise declaration) - regular Elasticsearch dict syntax - AggNode instance (for instance Terms, Filters etc)

If *insert_below* nor *insert_above* is provided by will be placed between the the deepest linear bucket aggregation if there is no ambiguity, and its children: A—> B : OK generates A—> B –> C –> by

A—> B : KO, ambiguous, must precise either A, B or C └—> C

Parameters

- **by** – aggregation(s) clauses to insert "vertically"
- **insert_below** – parent aggregation id under which these aggregations should be placed
- **insert_above** – aggregation id above which these aggregations should be placed
- **kwargs** – agg body arguments when using "string" syntax for terms aggregation

Return type *pandagg.agg.Agg*

node_class
alias of *pandagg.node.agg.abstract.AggNode*

paste(*nid*, *new_tree*, *deep=False*)
Pastes a tree handling nested implications if mapping is provided. The provided tree should be validated beforehands.

query(*query*, *validate=False*, ***kwargs*)

query_dict(*from_=None*, *depth=None*, *with_name=True*)

serialize_response(*aggs*, *output*, ***kwargs*)

set_mapping(*mapping*)

validate_tree(*exc=False*)
Validate tree definition against defined mapping. :param exc: if set to True, will raise exception if tree is invalid :return: boolean

class pandagg.agg.**MatchAll**(*name*, *meta=None*, *aggs=None*)
Bases: *pandagg.node.agg.bucket.Filter*

class pandagg.agg.**Terms**(*name*, *field*, *missing=None*, *size=None*, *aggs=None*, *meta=None*, ***body*)
Bases: *pandagg.node.agg.abstract.MultipleBucketAgg*

Terms aggregation.

BLACKLISTED_MAPPING_TYPES = []

KEY = 'terms'

VALUE_ATTRS = ['doc_count', 'doc_count_error_upper_bound', 'sum_other_doc_count']

get_filter(*key*)
Provide filter to get documents belonging to document of given key.

class pandagg.agg.**Filters**(*name*, *filters*, *other_bucket=False*, *other_bucket_key=None*, *meta=None*, *aggs=None*, ***body*)
Bases: *pandagg.node.agg.abstract.MultipleBucketAgg*

DEFAULT_OTHER_KEY = '_other_'

IMPLICIT_KEYED = True

KEY = 'filters'

VALUE_ATTRS = ['doc_count']

get_filter(*key*)
Provide filter to get documents belonging to document of given key.

class pandagg.agg.**Histogram**(*name*, *field*, *interval*, *meta=None*, *aggs=None*, ***body*)
Bases: *pandagg.node.agg.abstract.MultipleBucketAgg*

KEY = 'histogram'

VALUE_ATTRS = ['doc_count']

WHITELISTED_MAPPING_TYPES = ['long', 'integer', 'short', 'byte', 'double', 'float', 'ha

get_filter(*key*)
Provide filter to get documents belonging to document of given key.

class pandagg.agg.**DateHistogram**(*name*, *field*, *interval=None*, *calendar_interval=None*, *fixed_interval=None*, *meta=None*, *keyed=False*, *key_as_string=True*, *aggs=None*, ***body*)
Bases: *pandagg.node.agg.abstract.MultipleBucketAgg*

KEY = 'date_histogram'

VALUE_ATTRS = ['doc_count']

WHITELISTED_MAPPING_TYPES = ['date']

get_filter(*key*)
Provide filter to get documents belonging to document of given key.

class pandagg.agg.**Range**(*name*, *field*, *ranges*, *keyed=False*, *meta=None*, *aggs=None*, ***body*)
Bases: *pandagg.node.agg.abstract.MultipleBucketAgg*

KEY = 'range'

KEY_SEP = '-'

VALUE_ATTRS = ['doc_count']

WHITELISTED_MAPPING_TYPES = ['long', 'integer', 'short', 'byte', 'double', 'float', 'ha

from_key

get_filter(*key*)
Provide filter to get documents belonging to document of given key.

to_key

class pandagg.agg.**Global**(*name*, *meta=None*, *aggs=None*)
Bases: *pandagg.node.agg.abstract.UniqueBucketAgg*

KEY = 'global'

VALUE_ATTRS = ['doc_count']

get_filter(*key*)
Provide filter to get documents belonging to document of given key.

class pandagg.agg.**Filter**(*name*, *filter*, *meta=None*, *aggs=None*, ***body*)
Bases: *pandagg.node.agg.abstract.UniqueBucketAgg*

KEY = 'filter'

VALUE_ATTRS = ['doc_count']

get_filter(*key*)
Provide filter to get documents belonging to document of given key.

class pandagg.agg.**Nested**(*name*, *path*, *meta=None*, *aggs=None*)
Bases: *pandagg.node.agg.abstract.UniqueBucketAgg*

KEY = 'nested'

VALUE_ATTRS = ['doc_count']

WHITELISTED_MAPPING_TYPES = ['nested']

get_filter(*key*)
Provide filter to get documents belonging to document of given key.

class pandagg.agg.**ReverseNested**(*name*, *path=None*, *meta=None*, *aggs=None*, ***body*)
Bases: *pandagg.node.agg.abstract.UniqueBucketAgg*

KEY = 'reverse_nested'

VALUE_ATTRS = ['doc_count']

WHITELISTED_MAPPING_TYPES = ['nested']

get_filter(*key*)
Provide filter to get documents belonging to document of given key.

class pandagg.agg.**Avg**(*name*, *meta=None*, ***body*)
Bases: *pandagg.node.agg.abstract.FieldOrScriptMetricAgg*

KEY = 'avg'

VALUE_ATTRS = ['value']

WHITELISTED_MAPPING_TYPES = ['long', 'integer', 'short', 'byte', 'double', 'float', 'ha

class pandagg.agg.**Max**(*name*, *meta=None*, ***body*)
Bases: *pandagg.node.agg.abstract.FieldOrScriptMetricAgg*

KEY = 'max'

VALUE_ATTRS = ['value']

WHITELISTED_MAPPING_TYPES = ['long', 'integer', 'short', 'byte', 'double', 'float', 'ha

class pandagg.agg.**Sum**(*name*, *meta=None*, ***body*)
Bases: *pandagg.node.agg.abstract.FieldOrScriptMetricAgg*

KEY = 'sum'

VALUE_ATTRS = ['value']

WHITELISTED_MAPPING_TYPES = ['long', 'integer', 'short', 'byte', 'double', 'float', 'ha

class pandagg.agg.**Min**(*name*, *meta=None*, ***body*)
Bases: *pandagg.node.agg.abstract.FieldOrScriptMetricAgg*

KEY = 'min'

VALUE_ATTRS = ['value']

WHITELISTED_MAPPING_TYPES = ['long', 'integer', 'short', 'byte', 'double', 'float', 'ha

class pandagg.agg.**Cardinality**(*name*, *meta=None*, ***body*)
Bases: *pandagg.node.agg.abstract.FieldOrScriptMetricAgg*

KEY = 'cardinality'

VALUE_ATTRS = ['value']

class pandagg.agg.**Stats**(*name*, *meta=None*, ***body*)
Bases: *pandagg.node.agg.abstract.FieldOrScriptMetricAgg*

KEY = 'stats'

VALUE_ATTRS = ['count', 'min', 'max', 'avg', 'sum']

WHITELISTED_MAPPING_TYPES = ['long', 'integer', 'short', 'byte', 'double', 'float', 'ha

class pandagg.agg.**ExtendedStats**(*name*, *meta=None*, ***body*)
Bases: *pandagg.node.agg.abstract.FieldOrScriptMetricAgg*

KEY = 'extended_stats'

VALUE_ATTRS = ['count', 'min', 'max', 'avg', 'sum', 'sum_of_squares', 'variance', 'std

WHITELISTED_MAPPING_TYPES = ['long', 'integer', 'short', 'byte', 'double', 'float', 'ha

class pandagg.agg.**Percentiles**(*name*, *meta=None*, ***body*)
Bases: *pandagg.node.agg.abstract.FieldOrScriptMetricAgg*

Percents body argument can be passed to specify which percentiles to fetch.

KEY = 'percentiles'

VALUE_ATTRS = ['values']

WHITELISTED_MAPPING_TYPES = ['long', 'integer', 'short', 'byte', 'double', 'float', 'ha

class pandagg.agg.**PercentileRanks**(*name*, *field*, *values*, *meta=None*, ***body*)
Bases: *pandagg.node.agg.abstract.FieldOrScriptMetricAgg*

KEY = 'percentile_ranks'

VALUE_ATTRS = ['values']

WHITELISTED_MAPPING_TYPES = ['long', 'integer', 'short', 'byte', 'double', 'float', 'ha

class pandagg.agg.**GeoBound**(*name*, *meta=None*, ***body*)
Bases: *pandagg.node.agg.abstract.FieldOrScriptMetricAgg*

KEY = 'geo_bounds'

VALUE_ATTRS = ['bounds']

WHITELISTED_MAPPING_TYPES = ['geo_point']

class pandagg.agg.**GeoCentroid**(*name*, *meta=None*, ***body*)
Bases: *pandagg.node.agg.abstract.FieldOrScriptMetricAgg*

KEY = 'geo_centroid'

VALUE_ATTRS = ['location']

WHITELISTED_MAPPING_TYPES = ['geo_point']

class pandagg.agg.**TopHits**(*name*, *meta=None*, ***body*)
Bases: *pandagg.node.agg.abstract.MetricAgg*

KEY = 'top_hits'

VALUE_ATTRS = ['hits']

class pandagg.agg.**ValueCount**(*name*, *meta=None*, ***body*)
Bases: *pandagg.node.agg.abstract.FieldOrScriptMetricAgg*

BLACKLISTED_MAPPING_TYPES = []

KEY = 'value_count'

VALUE_ATTRS = ['value']

class pandagg.agg.**AvgBucket**(*name*, *buckets_path*, *gap_policy=None*, *meta=None*, *aggs=None*, ***body*)

Bases: *pandagg.node.agg.abstract.Pipeline*

KEY = 'avg_bucket'

VALUE_ATTRS = ['value']

class pandagg.agg.**Derivative**(*name*, *buckets_path*, *gap_policy=None*, *meta=None*, *aggs=None*, ***body*)

Bases: *pandagg.node.agg.abstract.Pipeline*

KEY = 'derivative'

VALUE_ATTRS = ['value']

class pandagg.agg.**MaxBucket**(*name*, *buckets_path*, *gap_policy=None*, *meta=None*, *aggs=None*, ***body*)

Bases: *pandagg.node.agg.abstract.Pipeline*

KEY = 'max_bucket'

VALUE_ATTRS = ['value']

class pandagg.agg.**MinBucket**(*name*, *buckets_path*, *gap_policy=None*, *meta=None*, *aggs=None*, ***body*)

Bases: *pandagg.node.agg.abstract.Pipeline*

KEY = 'min_bucket'

VALUE_ATTRS = ['value']

class pandagg.agg.**SumBucket**(*name*, *buckets_path*, *gap_policy=None*, *meta=None*, *aggs=None*, ***body*)

Bases: *pandagg.node.agg.abstract.Pipeline*

KEY = 'sum_bucket'

VALUE_ATTRS = ['value']

class pandagg.agg.**StatsBucket**(*name*, *buckets_path*, *gap_policy=None*, *meta=None*, *aggs=None*, ***body*)

Bases: *pandagg.node.agg.abstract.Pipeline*

KEY = 'stats_bucket'

VALUE_ATTRS = ['count', 'min', 'max', 'avg', 'sum']

class pandagg.agg.**ExtendedStatsBucket**(*name*, *buckets_path*, *gap_policy=None*, *meta=None*, *aggs=None*, ***body*)

Bases: *pandagg.node.agg.abstract.Pipeline*

KEY = 'extended_stats_bucket'

VALUE_ATTRS = ['count', 'min', 'max', 'avg', 'sum', 'sum_of_squares', 'variance', 'std

class pandagg.agg.**PercentilesBucket**(*name*, *buckets_path*, *gap_policy=None*, *meta=None*, *aggs=None*, ***body*)

Bases: *pandagg.node.agg.abstract.Pipeline*

KEY = 'percentiles_bucket'

VALUE_ATTRS = ['values']

class pandagg.agg.**MovingAvg**(*name*, *buckets_path*, *gap_policy=None*, *meta=None*, *aggs=None*, ***body*)
Bases: *pandagg.node.agg.abstract.Pipeline*

KEY = 'moving_avg'

VALUE_ATTRS = ['value']

class pandagg.agg.**CumulativeSum**(*name*, *buckets_path*, *gap_policy=None*, *meta=None*, *aggs=None*, ***body*)
Bases: *pandagg.node.agg.abstract.Pipeline*

KEY = 'cumulative_sum'

VALUE_ATTRS = ['value']

class pandagg.agg.**BucketScript**(*name*, *script*, *buckets_path*, *gap_policy=None*, *meta=None*, *aggs=None*, ***body*)
Bases: *pandagg.node.agg.abstract.ScriptPipeline*

KEY = 'bucket_script'

VALUE_ATTRS = ['value']

class pandagg.agg.**BucketSelector**(*name*, *script*, *buckets_path*, *gap_policy=None*, *meta=None*, *aggs=None*, ***body*)
Bases: *pandagg.node.agg.abstract.ScriptPipeline*

KEY = 'bucket_selector'

VALUE_ATTRS = None

class pandagg.agg.**BucketSort**(*name*, *script*, *buckets_path*, *gap_policy=None*, *meta=None*, *aggs=None*, ***body*)
Bases: *pandagg.node.agg.abstract.ScriptPipeline*

KEY = 'bucket_sort'

VALUE_ATTRS = None

class pandagg.agg.**SerialDiff**(*name*, *buckets_path*, *gap_policy=None*, *meta=None*, *aggs=None*, ***body*)
Bases: *pandagg.node.agg.abstract.Pipeline*

KEY = 'serial_diff'

VALUE_ATTRS = ['value']

5.2.2 pandagg.client module

class pandagg.client.**Elasticsearch**(*hosts=None*, *transport_class=<class 'elasticsearch.transport.Transport'>*, ***kwargs*)
Bases: elasticsearch.client.Elasticsearch

fetch_indices(*index='*'*)

Parameters **index** – Comma-separated list or wildcard expression of index names used to limit the request.

5.2.3 pandagg.exceptions module

exception pandagg.exceptions.**AbsentMappingFieldError**
Bases: *pandagg.exceptions.MappingError*

Field is not present in mapping.

exception pandagg.exceptions.**InvalidAggregation**
Bases: Exception

Wrong aggregation definition

exception pandagg.exceptions.**InvalidOperationMappingFieldError**
Bases: *pandagg.exceptions.MappingError*

Invalid aggregation type on this mapping field.

exception pandagg.exceptions.**MappingError**
Bases: Exception

Basic Mapping Error

exception pandagg.exceptions.**VersionIncompatibilityError**
Bases: Exception

Pandagg is not compatible with this ElasticSearch version.

5.2.4 pandagg.mapping module

class pandagg.mapping.**Mapping**(*from_=None*, *identifier=None*, *properties=None*, *dynamic=False*)
Bases: pandagg.tree._tree.Tree

contains(*nid*)
Check if the tree contains node of given id

classmethod deserialize(*from_*, *depth=0*)

list_nesteds_at_field(*field_path*)

mapping_type_of_field(*field_path*)

nested_at_field(*field_path*)

node_class
alias of *pandagg.node.mapping.abstract.Field*

node_path(*nid*)

serialize()

show(*data_property='pretty'*, ***kwargs*)
Print the tree structure in hierarchy style.

You have three ways to output your tree data, i.e., stdout with `show()`, plain text file with `save2file()`, and json string with `to_json()`. The former two use the same backend to generate a string of tree structure in a text graph.

- Version >= 1.2.7a*: you can also specify the `line_type` parameter, such as 'ascii' (default), 'ascii-ex', 'ascii-exr', 'ascii-em', 'ascii-emv', 'ascii-emh') to the change graphical form.

Parameters

- **nid** – the reference node to start expanding.
- **level** – the node level in the tree (root as level 0).
- **idhidden** – whether hiding the node ID when printing.

- **filter** – the function of one variable to act on the `Node` object. When this parameter is specified, the traversing will not continue to following children of node whose condition does not pass the filter.
- **key** – the `key` param for sorting `Node` objects in the same level.
- **reverse** – the `reverse` param for sorting `Node` objects in the same level.
- **line_type** –
- **data_property** – the property on the node data object to be printed.

Returns None

validate_agg_node(*agg_node*, *exc=True*)
Ensure if node has field or path that it exists in mapping, and that required aggregation type if allowed on this kind of field. :param agg_node: AggNode you want to validate on this mapping :param exc: boolean, if set to True raise exception if invalid :rtype: boolean

class pandagg.mapping.**IMapping**(*from_=None*, *properties=None*, *dynamic=False*, *client=None*, *root_path=None*, *depth=1*, *initial_tree=None*, *index_name=None*)
Bases: *pandagg.interactive.abstract.TreeBasedObj*

Interactive wrapper upon mapping tree.

class pandagg.mapping.**Text**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'text'

class pandagg.mapping.**Keyword**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'keyword'

class pandagg.mapping.**Long**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'long'

class pandagg.mapping.**Integer**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'integer'

class pandagg.mapping.**Short**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'short'

class pandagg.mapping.**Byte**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'byte'

class pandagg.mapping.**Double**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'double'

class pandagg.mapping.**HalfFloat**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'half_float'

class pandagg.mapping.**ScaledFloat**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'scaled_float'

class pandagg.mapping.**Date**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'date'

class pandagg.mapping.**DateNanos**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'date_nanos'

class pandagg.mapping.**Boolean**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'boolean'

class pandagg.mapping.**Binary**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'binary'

class pandagg.mapping.**IntegerRange**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'integer_range'

class pandagg.mapping.**Float**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'float'

class pandagg.mapping.**FloatRange**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'float_range'

class pandagg.mapping.**LongRange**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'long_range'

class pandagg.mapping.**DoubleRange**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'double_range'

class pandagg.mapping.**DateRange**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

KEY = 'date_range'

class pandagg.mapping.**Object**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

DISPLAY_PATTERN = ' {%s}'

KEY = 'object'

class pandagg.mapping.**Nested**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

DISPLAY_PATTERN = ' [%s]'

KEY = 'nested'

class pandagg.mapping.**GeoPoint**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

For lat/lon points

KEY = 'geo_point'

class pandagg.mapping.**GeoShape**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

For complex shapes like polygons

KEY = 'geo_shape'

class pandagg.mapping.**IP**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

for IPv4 and IPv6 addresses

KEY = 'IP'

class pandagg.mapping.**Completion**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

To provide auto-complete suggestions

KEY = 'completion'

class pandagg.mapping.**TokenCount**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

To count the number of tokens in a string

KEY = 'token_count'

class pandagg.mapping.**MapperMurMur3**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

To compute hashes of values at index-time and store them in the index

KEY = 'murmur3'

class pandagg.mapping.**MapperAnnotatedText**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

To index text containing special markup (typically used for identifying named entities)

KEY = 'annotated-text'

class pandagg.mapping.**Percolator**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

Accepts queries from the query-dsl

KEY = 'percolator'

class pandagg.mapping.**Join**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

Defines parent/child relation for documents within the same index

KEY = 'join'

class pandagg.mapping.**RankFeature**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

Record numeric feature to boost hits at query time.

KEY = 'rank_feature'

class pandagg.mapping.**RankFeatures**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

Record numeric features to boost hits at query time.

KEY = 'rank_features'

class pandagg.mapping.**DenseVector**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

Record dense vectors of float values.

KEY = 'dense_vector'

class pandagg.mapping.**SparseVector**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

Record sparse vectors of float values.

KEY = 'sparse_vector'

class pandagg.mapping.**SearchAsYouType**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

A text-like field optimized for queries to implement as-you-type completion

KEY = 'search_as_you_type'

class pandagg.mapping.**Alias**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

Defines an alias to an existing field.

KEY = 'alias'

class pandagg.mapping.**Flattened**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

Allows an entire JSON object to be indexed as a single field.

KEY = 'flattened'

class pandagg.mapping.**Shape**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

For arbitrary cartesian geometries.

KEY = 'shape'

class pandagg.mapping.**Histogram**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

For pre-aggregated numerical values for percentiles aggregations.

KEY = 'histogram'

class pandagg.mapping.**Index**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

The index to which the document belongs.

KEY = '_index'

class pandagg.mapping.**Type**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

The document's mapping type.

KEY = '_type'

class pandagg.mapping.**Id**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

The document's ID.

KEY = '_id'

class pandagg.mapping.**Source**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

The original JSON representing the body of the document.

KEY = '_source'

class pandagg.mapping.**Size**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

The size of the _source field in bytes, provided by the mapper-size plugin.

KEY = '_size'

class pandagg.mapping.**FieldNames**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

All fields in the document which contain non-null values.

KEY = '_field_names'

class pandagg.mapping.**Ignored**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

All fields in the document that have been ignored at index time because of ignore_malformed.

KEY = '_ignored'

class pandagg.mapping.**Routing**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

A custom routing value which routes a document to a particular shard.

KEY = '_routing'

class pandagg.mapping.**Meta**(*name*, *depth=0*, *is_subfield=False*, ***body*)
Bases: *pandagg.node.mapping.abstract.Field*

Application specific metadata.

KEY = '_meta'

5.2.5 pandagg.query module

class pandagg.query.**Query**(*from_=None*, *mapping=None*, *identifier=None*, *client=None*, *index_name=None*)
Bases: pandagg.tree._tree.Tree

Tree combination of query nodes.

Mapping declaration is optional, but doing so validates query validity.

add_node(*node*, *pid=None*)
Add a new node object to the tree and make the parent as the root by default.

The 'node' parameter refers to an instance of Class::Node.

bind(*client*, *index_name=None*)

bool(**args*, ***kwargs*)

boost(**args*, ***kwargs*)

constant_score(**args*, ***kwargs*)

classmethod deserialize(*from_*)

dis_max(**args*, ***kwargs*)

execute(*index=None*, ***kwargs*)

filter(**args*, ***kwargs*)

function_score(**args*, ***kwargs*)

has_child(**args*, ***kwargs*)

has_parent(**args*, ***kwargs*)

must(**args*, ***kwargs*)

must_not(**args*, ***kwargs*)

nested(**args*, ***kwargs*)

node_class
alias of *pandagg.node.query.abstract.QueryClause*

parent_id(**args*, ***kwargs*)

pinned_query(**args*, ***kwargs*)

query(*q*, *parent=None*, *child=None*, *parent_param=None*, *child_param=None*, *mode='add'*)
Place query below a given parent.

query_dict(*from_=None*, *named=False*)
Return None if no query clause.

script_score(**args*, ***kwargs*)

set_mapping(*mapping*)

should(**args*, ***kwargs*)

class pandagg.query.**Exists**(*field*, *_name=None*)
Bases: *pandagg.node.query.abstract.LeafQueryClause*

KEY = 'exists'

tag
The readable node name for human. This attribute can be accessed and modified with `.` and = operator respectively.

class pandagg.query.**Fuzzy**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'fuzzy'

class pandagg.query.**Ids**(*values*, *_name=None*)
Bases: *pandagg.node.query.abstract.LeafQueryClause*

KEY = 'ids'

serialize(*named=False*)

tag
The readable node name for human. This attribute can be accessed and modified with . and = operator respectively.

class pandagg.query.**Prefix**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'prefix'

class pandagg.query.**Range**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'range'

class pandagg.query.**Regexp**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'regexp'

class pandagg.query.**Term**(*field*, *value*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'term'

SHORT_TAG = 'value'

class pandagg.query.**Terms**(*field*, *terms*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.LeafQueryClause*

KEY = 'terms'

classmethod deserialize(***body*)

tag
The readable node name for human. This attribute can be accessed and modified with . and = operator respectively.

class pandagg.query.**TermsSet**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'terms_set'

class pandagg.query.**Type**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'type'

class pandagg.query.**Wildcard**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'wildcard'

class pandagg.query.**Intervals**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'intervals'

class pandagg.query.**Match**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'match'

SHORT_TAG = 'query'

class pandagg.query.**MatchBoolPrefix**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'match_bool_prefix'

SHORT_TAG = 'query'

class pandagg.query.**MatchPhrase**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'match_phrase'

SHORT_TAG = 'query'

class pandagg.query.**MatchPhrasePrefix**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'match_phrase_prefix'

SHORT_TAG = 'query'

class pandagg.query.**MultiMatch**(*fields*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.MultiFieldsQueryClause*

KEY = 'multi_match'

class pandagg.query.**Common**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'common'

class pandagg.query.**QueryString**(*_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.LeafQueryClause*

KEY = 'query_string'

class pandagg.query.**SimpleQueryString**(*_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.LeafQueryClause*

KEY = 'simple_string'

class pandagg.query.**Bool**(**args*, ***kwargs*)
Bases: *pandagg.node.query.compound.CompoundClause*

DEFAULT_OPERATOR
alias of pandagg.node.query._parameter_clause.Must

KEY = 'bool'

PARAMS_WHITELIST = ['should', 'must', 'must_not', 'filter', 'boost', 'minimum_should_m

class pandagg.query.**Boosting**(**args*, ***kwargs*)
Bases: *pandagg.node.query.compound.CompoundClause*

DEFAULT_OPERATOR
alias of pandagg.node.query._parameter_clause.Positive

KEY = 'boosting'

PARAMS_WHITELIST = ['positive', 'negative', 'negative_boost']

class pandagg.query.**ConstantScore**(**args*, ***kwargs*)
Bases: *pandagg.node.query.compound.CompoundClause*

DEFAULT_OPERATOR
alias of pandagg.node.query._parameter_clause.Filter

KEY = 'constant_score'

PARAMS_WHITELIST = ['filter', 'boost']

class pandagg.query.**FunctionScore**(**args*, ***kwargs*)
Bases: *pandagg.node.query.compound.CompoundClause*

DEFAULT_OPERATOR
alias of pandagg.node.query._parameter_clause.QueryP

KEY = 'function_score'

PARAMS_WHITELIST = ['query', 'boost', 'random_score', 'boost_mode', 'functions', 'max_

class pandagg.query.**DisMax**(**args*, ***kwargs*)
Bases: *pandagg.node.query.compound.CompoundClause*

DEFAULT_OPERATOR
alias of pandagg.node.query._parameter_clause.Queries

KEY = 'dis_max'

PARAMS_WHITELIST = ['queries', 'tie_breaker']

class pandagg.query.**Nested**(**args*, ***kwargs*)
Bases: *pandagg.node.query.compound.CompoundClause*

DEFAULT_OPERATOR
alias of pandagg.node.query._parameter_clause.QueryP

KEY = 'nested'

PARAMS_WHITELIST = ['path', 'query', 'score_mode', 'ignore_unmapped']

class pandagg.query.**HasParent**(**args*, ***kwargs*)
Bases: *pandagg.node.query.compound.CompoundClause*

DEFAULT_OPERATOR
alias of pandagg.node.query._parameter_clause.QueryP

KEY = 'has_parent'

PARAMS_WHITELIST = ['query', 'parent_type', 'score', 'ignore_unmapped']

class pandagg.query.**HasChild**(**args*, ***kwargs*)
Bases: *pandagg.node.query.compound.CompoundClause*

DEFAULT_OPERATOR
alias of pandagg.node.query._parameter_clause.QueryP

KEY = 'has_child'

PARAMS_WHITELIST = ['query', 'type', 'max_children', 'min_children', 'score_mode', 'ig

class pandagg.query.**ParentId**(**args*, ***kwargs*)
Bases: *pandagg.node.query.compound.CompoundClause*

KEY = 'parent_id'

class pandagg.query.**Shape**(*_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.LeafQueryClause*

KEY = 'shape'

class pandagg.query.**GeoShape**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'geo_shape'

class pandagg.query.**GeoPolygone**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'geo_polygon'

class pandagg.query.**GeoDistance**(*field*, *location*, *distance*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.LeafQueryClause*

KEY = 'geo_distance'

classmethod deserialize(***body*)

tag
The readable node name for human. This attribute can be accessed and modified with . and = operator respectively.

class pandagg.query.**GeoBoundingBox**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

KEY = 'geo_bounding_box'

class pandagg.query.**DistanceFeature**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

FLAT = True

KEY = 'distance_feature'

class pandagg.query.**MoreLikeThis**(*fields*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.MultiFieldsQueryClause*

KEY = 'more_like_this'

class pandagg.query.**Percolate**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

FLAT = True

KEY = 'percolate'

class pandagg.query.**RankFeature**(*field*, *_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.SingleFieldQueryClause*

FLAT = True

KEY = 'rank_feature'

class pandagg.query.**Script**(*_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.LeafQueryClause*

KEY = 'script'

class pandagg.query.**Wrapper**(*_name=None*, ***body*)
Bases: *pandagg.node.query.abstract.LeafQueryClause*

KEY = 'wrapper'

class pandagg.query.**ScriptScore**(**args*, ***kwargs*)
Bases: *pandagg.node.query.compound.CompoundClause*

DEFAULT_OPERATOR
alias of pandagg.node.query._parameter_clause.QueryP

KEY = 'script_score'

PARAMS_WHITELIST = ['query', 'script', 'min_score']

class pandagg.query.**PinnedQuery**(**args*, ***kwargs*)
Bases: *pandagg.node.query.compound.CompoundClause*

DEFAULT_OPERATOR
alias of pandagg.node.query._parameter_clause.Organic

KEY = 'pinned'

PARAMS_WHITELIST = ['ids', 'organic']

5.2.6 pandagg.utils module

class pandagg.utils.**PrettyNode**(*pretty*)
Bases: object

pretty

pandagg.utils.**bool_if_required**(*conditions*, *operator='must'*)

pandagg.utils.**equal_queries**(*d1*, *d2*)
Compares if two queries are equivalent (do not consider nested list orders).

pandagg.utils.**ordered**(*obj*)

5.3 Module contents

CHAPTER 6

Contributing to Pandagg

We want to make contributing to this project as easy and transparent as possible.

6.1 Our Development Process

We use github to host code, to track issues and feature requests, as well as accept pull requests.

6.2 Pull Requests

We actively welcome your pull requests.

1. Fork the repo and create your branch from `master`.
2. If you've added code that should be tested, add tests.
3. If you've changed APIs, update the documentation.
4. Ensure the test suite passes.
5. Make sure your code lints.

6.3 Any contributions you make will be under the MIT Software License

In short, when you submit code changes, your submissions are understood to be under the same MIT License that covers the project. Feel free to contact the maintainers if that's a concern.

6.4 Issues

We use GitHub issues to track public bugs. Please ensure your description is clear and has sufficient instructions to be able to reproduce the issue.

6.5 Report bugs using Github's issues

We use GitHub issues to track public bugs. Report a bug by opening a new issue; it's that easy!

6.6 Write bug reports with detail, background, and sample code

Great Bug Reports tend to have:

- A quick summary and/or background
- Steps to reproduce
  - Be specific!
  - Give sample code if you can.
- What you expected would happen
- What actually happens
- Notes (possibly including why you think this might be happening, or stuff you tried that didn't work)

6.7 License

By contributing, you agree that your contributions will be licensed under its MIT License.

6.8 References

This document was adapted from the open-source contribution guidelines of briandk's gist

pandagg is a Python package providing a simple interface to manipulate ElasticSearch queries and aggregations. It brings the following features:

- flexible aggregation and search queries declaration
- query validation based on provided mapping
- parsing of aggregation results in handy format: interactive bucket tree, normalized tree or tabular breakdown
- mapping interactive navigation

CHAPTER 7

Installing

pandagg can be installed with pip:

```
$ pip install pandagg
```

Alternatively, you can grab the latest source code from GitHub:

```
$ git clone git://github.com/alkemics/pandagg.git
$ python setup.py install
```

CHAPTER 8

Usage

The *User Guide* is the place to go to learn how to use the library and accomplish common tasks. The more in-depth *Advanced usage* guide is the place to go for deeply nested queries.

An example based on publicly available IMDB data is documented in repository *examples/imdb* directory, with a jupyter notebook to showcase some of *pandagg* functionalities: here it is.

The *pandagg package* documentation provides API-level documentation.

CHAPTER 9

License

pandagg is made available under the MIT License. For more details, see LICENSE.txt.

CHAPTER 10

Contributing

We happily welcome contributions, please see *Contributing to Pandagg* for details.

Python Module Index

p

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W