

---

# **ESORM - ElasticSearch ORM**

***Release 0.4.4***

**Adam Wallner**

**May 18, 2024**



# CONTENTS

<b>1</b>	<b>ESORM - Python Elasticsearch ORM based on Pydantic</b>	<b>1</b>
1.1	Table of Contents	1
1.2	Installation	2
1.3	Features	2
1.3.1	Supported Elasticsearch versions	3
1.3.2	Supported Python versions	3
1.4	Usage	3
1.4.1	Define a model	3
1.4.2	Connecting to Elasticsearch	8
1.4.3	Create index templates	9
1.4.4	Create indices and mappings	9
1.4.5	Model instances	10
1.4.6	CRUD: Create	10
1.4.7	CRUD: Read	11
1.4.8	CRUD: Update	11
1.4.9	CRUD: Delete	11
1.4.10	Bulk operations	12
1.4.11	Search	12
1.4.12	Aggregations	14
1.4.13	Pagination and sorting	15
1.5	Testing	17
1.6	License	17
1.7	Citation	17
<b>2</b>	<b>Advanced usage</b>	<b>19</b>
2.1	Optimistic concurrency control	19
2.1.1	Retry on conflict	21
2.2	Lazy properties	21
2.3	Shard routing	22
2.4	Watchers	23
2.5	FastAPI integration	24
2.5.1	FastAPI pagination	25
<b>3</b>	<b>Reference</b>	<b>27</b>
3.1	Module contents	27
3.2	Submodules	36
3.2.1	esorm.aggs module	36
3.2.2	esorm.bulk module	39
3.2.3	esorm.error module	39
3.2.4	esorm.esorm module	40

3.2.5	esorm.fastapi module . . . . .	40
3.2.6	esorm.fields module . . . . .	41
3.2.7	esorm.logger module . . . . .	46
3.2.8	esorm.model module . . . . .	46
3.2.9	esorm.query module . . . . .	54
3.2.10	esorm.response module . . . . .	61
3.2.11	esorm.utils module . . . . .	61
3.2.12	esorm.watcher module . . . . .	62
<b>4</b>	<b>Changelog . . . . .</b>	<b>67</b>
4.1	v0.4.4 . . . . .	67
4.2	v0.4.3 . . . . .	67
4.3	v0.4.2 . . . . .	67
4.4	v0.4.1 . . . . .	68
4.5	v0.4.0 . . . . .	68
4.6	v0.3.2 . . . . .	68
4.7	v0.3.1 . . . . .	68
4.8	v0.3.0 . . . . .	69
4.9	v0.2.1 . . . . .	69
4.10	v0.2.0 . . . . .	69
4.11	v0.1.2 . . . . .	69
4.12	v0.1.1 . . . . .	69
<b>5</b>	<b>Indices and tables . . . . .</b>	<b>71</b>
	<b>Python Module Index . . . . .</b>	<b>73</b>
	<b>Index . . . . .</b>	<b>75</b>

# **ESORM - PYTHON ELASTICSEARCH ORM BASED ON PYDANTIC**

Some ideas come from [Pydantic](#) library, which is similar, but not as advanced (yet).

## **1.1 Table of Contents**

- *Installation*
- *Features*
  - *Supported Elasticsearch versions*
  - *Supported Python versions*
- *Usage*
  - *Define a model*
    - \* *Python basic types*
    - \* *ESORM field types*
    - \* *Nested documents*
    - \* *ESBaseModel*
    - \* *Id field*
    - \* *Model Settings*
    - \* *Describe fields*
    - \* *ESModelTimestamp*
  - *Connecting to Elasticsearch*
    - \* *Client*
  - *Create index templates*
  - *Create indices and mappings*
  - *Model instances*
  - *CRUD: Create*
  - *CRUD: Read*
  - *CRUD: Update*
  - *CRUD: Delete*
  - *Bulk operations*

- *Search*
  - \* *General search*
  - \* *Search with field value terms (dictioanry search)*
- *Aggregations*
- *Pagination and sorting*
- *Advanced usage*
  - *Optimistic concurrency control*
  - *Lazy properties*
  - *Shard routing*
  - *Watchers*
  - *FastAPI integration*
- *Testing*
- *License*
- *Citation*

## 1.2 Installation

```
pip install pyesorm
```

## 1.3 Features

- Pydantic model representation of ElasticSearch documents
- Automatic mapping and index creation
- CRUD operations
- Full async support (no sync version at all)
- Mapping to and from ElasticSearch types
- Support for nested documents
- Automatic optimistic concurrency control
- Custom id field
- Context for bulk operations
- Supported IDE autocompletion and type checking (PyCharm tested)
- Everything in the source code is documented and annotated
- TypedDicts for ElasticSearch queries and aggregations
- Docstring support for fields

- Shard routing support
- Lazy properties
- Support  $\geq$  Python 3.8 (tested with 3.8 through 3.12)
- Support for Elasticsearch 8.x and 7.x
- Watcher support (You may need Elasticsearch subscription license for this)
- Pagination and sorting
- FastAPI integration

Not all Elasticsearch features are supported yet, pull requests are welcome.

### 1.3.1 Supported Elasticsearch versions

It is tested with Elasticsearch 7.x and 8.x.

### 1.3.2 Supported Python versions

Tested with Python 3.8 through 3.12.

## 1.4 Usage

### 1.4.1 Define a model

You can use all [Pydantic](#) model features, because `ESModel` is a subclass of `pydantic.BaseModel`. (Actually it is a subclass of `ESBaseModel`, see more *below...*)

`ESModel` extends `pydantic BaseModel` with Elasticsearch specific features. It serializes and deserializes documents to and from Elasticsearch types and handle Elasticsearch operations in the background.

#### Python basic types

```
from esorm import ESModel

class User(ESModel):
    name: str
    age: int
```

This is how the python types are converted to ES types:

Python type	ES type
str	text
int	long
float	double
bool	boolean
datetime.datetime	date
datetime.date	date
datetime.time	date

## ESORM field types

You can specify Elasticsearch special fields using `esorm.fields` module.

```
from esorm import ESModel
from esorm.fields import keyword, text, byte, geo_point

class User(ESModel):
    name: text
    email: keyword
    age: byte
    location: geo_point
    ...
```

The supported fields are:

Field name	ES type
keyword	keyword
text	text
binary	binary
byte	byte
short	short
integer or int32	integer
long or int64	long
float16 or half_float	half_float
float32	float
double	double
boolean	boolean
geo_point	geo_point

The binary field accepts **base64** encoded strings. However, if you provide bytes to it, they will be automatically converted to a **base64** string during serialization. When you retrieve the field, it will always be a **base64** encoded string. You can easily convert it back to bytes using the `bytes()` method: `binary_field.bytes()`.



## Nested documents

```
from esorm import ESModel
from esorm.fields import keyword, text, byte

class User(ESModel):
    name: text
    email: keyword
    age: byte = 18

class Post(ESModel):
    title: text
    content: text
    writer: User # User is a nested document
```

## ESBaseModel

ESBaseModel is the base of ESModel, also it is useful to use it for nested documents, because by using it will not be included in the ElasticSearch index.

```
from esorm import ESModel, ESBaseModel
from esorm.fields import keyword, text, byte

# This way `User` model won't be in the index
class User(ESBaseModel): # <-----
    name: text
    email: keyword
    age: byte = 18

class Post(ESModel):
    title: text
    content: text
    writer: User # User is a nested document
```

## Id field

You can specify id field in *model settings*:

```
from esorm import ESModel
from esorm.fields import keyword, text, byte

class User(ESModel):
    class ESConfig:
        id_field = 'email'

    name: text
    email: keyword
    age: byte = 18
```

This way the field specified in `id_field` will be removed from the document and used as the document `_id` in the index.

If you specify a field named `id` in your model, it will be used as the document `_id` in the index (it will automatically override the `id_field` setting):

```
from esorm import ESModel

class User(ESModel):
    id: int # This will be used as the document _id in the index
    name: str
```

You can also create an `__id__` property in your model to return a custom id:

```
from esorm import ESModel
from esorm.fields import keyword, text, byte

class User(ESModel):
    name: text
    email: keyword
    age: byte = 18

    @property
    def __id__(self) -> str:
        return self.email
```

NOTE: annotation of `__id__` method is important, and it must be declared as a property.

## Model Settings

You can specify model settings using ESConfig child class.

```
from typing import Optional, List, Dict, Any
from esorm import ESModel

class User(ESModel):
    class ESConfig:
        """ ESModel Config """
        # The index name
        index: Optional[str] = None
        # The name of the 'id' field
        id_field: Optional[str] = None
        # Default sort
        default_sort: Optional[List[Dict[str, Dict[str, str]]]] = None
        # Elasticsearch index settings (https://www.elastic.co/guide/en/elasticsearch/reference/current/index-modules.html)
        settings: Optional[Dict[str, Any]] = None
        # Maximum recursion depth of lazy properties
        lazy_property_max_recursion_depth: int = 1
```

## ESModelTimestamp

You can use ESModelTimestamp class to add created\_at and updated\_at fields to your model:

```
from esorm import ESModelTimestamp

class User(ESModelTimestamp):
    name: str
    age: int
```

These fields will be automatically updated to the actual datetime when you create or update a document. The created\_at field will be set only when you create a document. The updated\_at field will be set when you create or update a document.

## Describe fields

You can use the usual Pydantic field description, but you can also use docstrings like this:

```
from esorm import ESModel
from esorm.fields import TextField

class User(ESModel):
    name: str = 'John Doe'
    """ The name of the user """
```

(continues on next page)

(continued from previous page)

```
age: int = 18
    """ The age of the user """

    # This is the usual Pydantic way, but I think docstrings are more intuitive and
    ↪readable
    address: str = TextField(description="The address of the user")
```

The documentation is useful if you create an API and you want to generate documentation from the model. It can be used in [FastAPI](#) for example.

## 1.4.2 Connecting to Elasticsearch

You can connect with a simple connection string:

```
from esorm import connect

async def es_init():
    await connect('localhost:9200')
```

Also you can connect to multiple hosts if you have a cluster:

```
from esorm import connect

async def es_init():
    await connect(['localhost:9200', 'localhost:9201'])
```

You can wait for node or cluster to be ready (recommended):

```
from esorm import connect

async def es_init():
    await connect('localhost:9200', wait=True)
```

This will ping the node in 2 seconds intervals until it is ready. It can be a long time.

You can pass any arguments that AsyncElasticsearch supports:

```
from esorm import connect

async def es_init():
    await connect('localhost:9200', wait=True, sniff_on_start=True, sniff_on_connection_
    ↪fail=True)
```

## Client

The `connect` function is a wrapper for the `AsyncElasticsearch` constructor. It creates and stores a global instance of a proxy to an `AsyncElasticsearch` instance. The model operations will use this instance to communicate with Elasticsearch. You can retrieve the proxy client instance and you can use the same way as `AsyncElasticsearch` instance:

```
from esorm import es

async def es_init():
    await es.ping()
```

### 1.4.3 Create index templates

You can create index templates easily:

```
from esorm import model as esorm_model

# Create index template
async def prepare_es():
    await esorm_model.create_index_template('default_template',
                                           prefix_name='esorm_',
                                           shards=3,
                                           auto_expand_replicas='1-5')
```

Here this will be applied all `esorm_` prefixed (default) indices.

All indices created by ESORM have a prefix, which you can modify globally if you want:

```
from esorm.model import set_default_index_prefix

set_default_index_prefix('custom_prefix_')
```

The default prefix is `esorm_`.

### 1.4.4 Create indices and mappings

You can create indices and mappings automatically from your models:

```
from esorm import setup_mappings

# Create indices and mappings
async def prepare_es():
    import models # Import your models
    # Here models argument is not needed, but you can pass it to prevent unused import.
    ↪ warning
    await setup_mappings(models)
```

First you must create (import) all model classes. Model classes will be registered into a global registry. Then you can call `setup_mappings` function to create indices and mappings for all registered models.

**IMPORTANT:** This method will ignore mapping errors if you already have an index with the same name. It can update the indices by new fields, but cannot modify or delete fields! For that you need to reindex your ES database. It is an Elasticsearch limitation.

### 1.4.5 Model instances

When you get a model instance from elasticsearch by `search` or `get` methods, you will get the following private attributes filled automatically:

Attribute	Description
<code>_id</code>	The ES id of the document
<code>_routing</code>	The routing value of the document
<code>_version</code>	Version of the document
<code>_primary_term</code>	The primary term of the document
<code>_seq_no</code>	The sequence number of the document

### 1.4.6 CRUD: Create

```
from esorm import ESModel

# Here the model have automatically generated id
class User(ESModel):
    name: str
    age: int

async def create_user():
    # Create a new user
    user = User(name='John Doe', age=25)
    # Save the user to Elasticsearch
    new_user_id = await user.save()
    print(new_user_id)
```

### 1.4.7 CRUD: Read

```
from esorm import ESModel

# Here the model have automatically generated id
class User(ESModel):
    name: str
    age: int

async def get_user(user_id: str):
    user = await User.get(user_id)
    print(user.name)
```

### 1.4.8 CRUD: Update

On update race conditions are checked automatically (with the help of `_primary_term` and `_seq_no` fields). This way an optimistic locking mechanism is implemented.

```
from esorm import ESModel

# Here the model have automatically generated id
class User(ESModel):
    name: str
    age: int

async def update_user(user_id: str):
    user = await User.get(user_id)
    user.name = 'Jane Doe'
    await user.save()
```

### 1.4.9 CRUD: Delete

```
from esorm import ESModel

# Here the model have automatically generated id
class User(ESModel):
    name: str
    age: int

async def delete_user(user_id: str):
```

(continues on next page)

(continued from previous page)

```
user = await User.get(user_id)
await user.delete()
```

### 1.4.10 Bulk operations

Bulk operations could be much faster than single operations, if you have lot of documents to create, update or delete.

You can use context for bulk operations:

```
from typing import List
from esorm import ESModel, ESBulk

# Here the model have automatically generated id
class User(ESModel):
    name: str
    age: int

async def bulk_create_users():
    async with ESBulk() as bulk:
        # Creating or modifying models
        for i in range(10):
            user = User(name=f'User {i}', age=i)
            await bulk.save(user)

async def bulk_delete_users(users: List[User]):
    async with ESBulk(wait_for=True) as bulk: # Here we wait for the bulk operation to
    ↪ finish
        # Deleting models
        for user in users:
            await bulk.delete(user)
```

The wait\_for argument is optional. If it is True, the context will wait for the bulk operation to finish.

### 1.4.11 Search



## General search

You can search for documents using `search` method, where an ES query can be specified as a dictionary. You can use `res_dict=True` argument to get the result as a dictionary instead of a list. The key will be the id of the document: `await User.search(query, res_dict=True)`.

If you only need one result, you can use `search_one` method.

```
from esorm import ESModel

# Here the model have automatically generated id
class User(ESModel):
    name: str
    age: int

async def search_users():
    # Search for users at least 18 years old
    users = await User.search(
        query={
            'bool': {
                'must': [{
                    'range': {
                        'age': {
                            'gte': 18
                        }
                    }
                }]
            }
        }
    )
    for user in users:
        print(user.name)

async def search_one_user():
    # Search a user named John Doe
    user = await User.search_one(
        query={
            'bool': {
                'must': [{
                    'match': {
                        'name': {
                            'query': 'John Doe'
                        }
                    }
                }]
            }
        }
    )
    print(user.name)
```

Queries are type checked, because they are annotated as `TypedDicts`. You can use IDE autocompletion and type checking.

### Search with field value terms (dictionary search)

You can search for documents using `search_by_fields` method, where you can specify a field and a value. It also has a `res_dict` argument and `search_one_by_fields` variant.

```
from esorm import ESModel

# Here the model have automatically generated id
class User(ESModel):
    name: str
    age: int

async def search_users():
    # Search users age is 18
    users = await User.search_by_fields({'age': 18})
    for user in users:
        print(user.name)
```

### 1.4.12 Aggregations

You can use `aggregate` method to get aggregations. You can specify an ES aggregation query as a dictionary. It also accepts normal ES queries, to be able to filter which documents you want to aggregate. Both the `aggs` parameter and the `query` parameter are type checked, because they are annotated as `TypedDicts`. You can use IDE autocompletion and type checking.

```
from esorm import ESModel

# Here the model have automatically generated id
class User(ESModel):
    name: str
    age: int
    country: str

async def aggregate_avg():
    # Get average age of users
    aggs_def = {
        'avg_age': {
            'avg': {
                'field': 'age'
            }
        }
    }
    aggs = await User.aggregate(aggs_def)
    print(aggs['avg_age']['value'])

async def aggregate_avg_by_country(country = 'Hungary'):
```

(continues on next page)

(continued from previous page)

```

# Get average age of users by country
aggs_def = {
    'avg_age': {
        'avg': {
            'field': 'age'
        }
    }
}
query = {
    'bool': {
        'must': [{
            'match': {
                'country': {
                    'query': country
                }
            }
        }]
    }
}
aggs = await User.aggregate(aggs_def, query)
print(aggs['avg_age']['value'])

async def aggregate_terms():
    # Get number of users by country
    aggs_def = {
        'countries': {
            'terms': {
                'field': 'country'
            }
        }
    }
    aggs = await User.aggregate(aggs_def)
    for bucket in aggs['countries']['buckets']:
        print(bucket['key'], bucket['doc_count'])

```

### 1.4.13 Pagination and sorting

You can use `Pagination` and `Sort` classes to decorate your models. They simply wrap your models and add pagination and sorting functionality to them.

## Pagination

You can add a callback parameter to the `Pagination` class which will be invoked after the search with the total number of documents found.

```
from esorm.model import ESModel, Pagination

class User(ESModel):
    id: int # This will be used as the document _id in the index
    name: str
    age: int

def get_users(page = 1, page_size = 10):

    def pagination_callback(total: int):
        # You may set a header value or something else here
        print(f'Total users: {total}')

    # 1st create the decorator itself
    pagination = Pagination(page=page, page_size=page_size)

    # Then decorate your model
    res = pagination(User).search_by_fields(age=18)

    # Here the result has maximum 10 items
    return res
```

## Sorting

It is similar to pagination:

```
from esorm.model import ESModel, Sort

class User(ESModel):
    id: int # This will be used as the document _id in the index
    name: str
    age: int

def get_users():
    # 1st create the decorator itself
    sort = Sort(sort=[
        {'age': {'order': 'desc'}},
        {'name': {'order': 'asc'}}
    ])

    # Then decorate your model
    res = sort(User).search_by_fields(age=18)
```

(continues on next page)

(continued from previous page)

```

    # Here the result is sorted by age ascending
    return res

def get_user_sorted_by_name():
    # You can also use this simplified syntax
    sort = Sort(sort='name')

    # Then decorate your model
    res = sort(User).all()

    # Here the result is sorted by age descending
    return res

```

## 1.5 Testing

For testing you can use the `test.sh` in the root directory. It is a script to running tests on multiple python interpreters in virtual environments. At the top of the file you can specify which python interpreters you want to test. The ES versions are specified in `tests/docker-compose.yml` file.

If you already have a virtual environment, simply use `pytest` to run the tests.

## 1.6 License

This project is licensed under the terms of the [Mozilla Public License 2.0](#) (MPL 2.0) license.

## 1.7 Citation

If you use this project in your research, please cite it using the following BibTeX entry:

```

@misc{esorm,
  author = {Adam Wallner},
  title = {ESORM: Elasticsearch Object Relational Mapper},
  year = {2023},
  publisher = {GitHub},
  journal = {GitHub repository},
  howpublished = {\url{https://github.com/wallneradam/esorm}},
}

```



## ADVANCED USAGE

### 2.1 Optimistic concurrency control

ESORM uses optimistic concurrency control automatically, to prevent race conditions and data losses when multiple updates are made to the same document at the same time.

When you save a document, ESORM checks if the document has been changed since it was loaded and raises an exception if it has. In the background it uses the `_seq_no` and `_primary_term` fields for checking. More information in [ES documentation](#).

This is how to handle race conditions:

```
from esorm import ESModel

class User(ESModel):
    first_name: str
    last_name: str

async def test_race_condition():
    """ Update the user """
    import asyncio
    from elasticsearch import ConflictError

    # Load the user
    user_load1 = await User.get(id = 1)
    # Load again
    user_load2 = await User.get(id = 1)
    # Here both have the same _seq_no and _primary_term

    try:
        # Update the 1st loaded user
        user_load1.first_name = "John"
        await user_load1.save()
        # Update the 2nd loaded user
        user_load2.first_name = "Jane"
        await user_load2.save() # This will raise a ConflictError
    except ConflictError:
```

(continues on next page)

(continued from previous page)

```
print("Conflict error")

# Load the user
user_load1 = await User.get(id = 1)
# Load again
user_load2 = await User.get(id = 1)
# Here both have the same _seq_no and _primary_term

# Delete operation
try:
    user_load1.first_name = "John"
    await user_load1.save()
    await user_load2.delete() # This will raise a ConflictError
except ConflictError:
    print("Conflict error")
```

This is how to use it in bulk operations:

```
from esorm import ESModel, ESBulk, error

class User(ESModel):
    first_name: str
    last_name: str

async def test_bulk():
    # Load the user
    user_load1 = await User.get(id = 1)
    # Load again
    user_load2 = await User.get(id = 1)
    # Here both have the same _seq_no and _primary_term
    try:
        async with ESBulk(wait_for=True) as bulk:
            user_load1.first_name = "John"
            await bulk.save(user_load1)
            user_load2.first_name = "Jane"
            await bulk.save(user_load2) #
    except error.BulkError as e:
        # You can get the failed operations from e.failed_operations
        print("Bulk error:", e.failed_operations)
```



### 2.1.1 Retry on conflict

You can use the `retry_on_conflict` decorator to automatically retry the operation(s) on conflict:

```
import asyncio
from esorm import ESModel, retry_on_conflict

class User(ESModel):
    first_name: str
    last_name: str
    logins: int = 0

async def test_retry_on_conflict(user: User):
    @retry_on_conflict(3) # Retry 3 times on conflict
    async def login(user_id):
        _user = await User.get(id=user_id)
        _user.logins += 1

    # This won't raise a ConflictError
    await asyncio.gather(
        login(user._id),
        login(user._id),
        login(user._id),
    )
```

## 2.2 Lazy properties

ESORM is based on `pydantic` which is fully synchronous, so it's not possible to use async functions to calculate property values. Because of this, you also can't query another model in a computed field. To solve this problem, ESORM provides lazy properties.

You can create lazy properties like this:

```
from typing import List
from esorm import ESModel, lazy_property
from pydantic import computed_field

class User(ESModel):
    first_name: str
    last_name: str

    # This is classic pydantic computed field, which can be only synchronous
    @computed_field
    @property
    def full_name(self) -> str:
        return f"{self.first_name} {self.last_name}"

    # This is lazy property, which can be async
```

(continues on next page)

(continued from previous page)

```
@lazy_property
async def same_first_name(self) -> List["User"]:
    return await self.search_by_fields(first_name=self.first_name)
```

Lazy properties in the background work like the following:

- they are registered in the model by storing the async function
- replace it to a real property on model creation
- after a query is executed (which is always async), lazy properties are calculated and stored in the model
- these stored values are used when accessing the property, e.g, when they are serialized to JSON

Lazy properties are computed parallelly, you can configure the number of parallel query tasks by `set_max_lazy_property_concurrency` function:

```
from esorm.model import set_max_lazy_property_concurrency

set_max_lazy_property_concurrency(10) # Set the number of parallel tasks to 10
```

The above example is recursive, because `User` model is used in the `same_first_name` property. This could lead to an infinite loop. Because of this, ESORM restricts the depth of recursion to 1 by default.

If you want to change the recursion depth, you can do it by setting the `max_lazy_property_depth` in the `ESConfig`:

```
from esorm import ESModel

class User(ESModel):
    class ESConfig:
        lazy_property_max_recursion_depth = 2 # Set the recursion depth to 2

    first_name: str
    last_name: str
    ...
```

## 2.3 Shard routing

Shard routing is a feature of Elasticsearch which allows you to store documents in a specific shard. This can be useful if you want to store documents of a specific type in a specific shard, e.g, you want to store all documents of a specific region. When using shard routing, Elasticsearch does not need to search all shards, but only the shards which contain the documents you are looking for.

More info: <https://www.elastic.co/guide/en/elasticsearch/reference/current/search-shard-routing.html>

In ESORM shard routing looks like this:

```
from typing import List
from esorm import ESModel

class User(ESModel):
    first_name: str
```

(continues on next page)

(continued from previous page)

```

last_name: str
region: str

@property
def __routing__(self) -> str:
    """ Return the routing value for this document """
    return self.region + '_routing' # Calculate the routing value from the region_
↪field

async def get_user_by_region(region: str = 'europe') -> List[User]:
    """ Search for users by region using shard routing """
    return await User.search_by_fields(region=region, routing=f"{region}_routing")

```

## 2.4 Watchers

You can add watches to automatically perform an action when certain conditions are met. The conditions are generally based on data you've loaded into the watch, also known as the Watch Payload. This payload can be loaded from different sources - from Elasticsearch, an external HTTP service, or even a combination of the two.

More info: <https://www.elastic.co/guide/en/elasticsearch/reference/current/how-watcher-works.html>

The following example shows how to create a watcher which deletes all draft documents older than 1 hour:

```

from esorm.watcher import DeleteWatcher
from esorm import query

TIMEOUT = 60 * 60 # 1 hour

class PurgeDraft(DeleteWatcher):
    """
    Purge draft data after TIMEOUT
    """
    trigger = {
        "schedule": {
            "interval": f"30s"
        }
    }

    _index = f"draft"
    _query: query.ESQuery = {
        "bool": {
            "must": [
                # Search for all documents with id starting with "_"
                {
                    "wildcard": {
                        "id": {
                            "value": "_*"
                        }
                    }
                }
            ]
        }
    }

```

(continues on next page)

(continued from previous page)

```

        }
    },
    # Filter documents which are older than TIMEOUT + 30s
    {
        "range": {
            "created_at": {
                "lt": f"now-{TIMEOUT + 30}s", # 30 sec buffer
            }
        }
    }
]
}

```

For more info check the [reference](#), or Elasticsearch documentation, or the source code.

## 2.5 FastAPI integration

Because ESORM is based on pydantic, it can be easily integrated with FastAPI:

```

from typing import List, Optional
from esorm import ESModelTimestamp
from fastapi import FastAPI

class User(ESModelTimestamp):
    """ The User model """
    first_name: str
    last_name: str

app = FastAPI()

@app.post("/users")
async def create_user(first_name: str, last_name: str) -> User:
    """ Create a new user """
    user = User(first_name=first_name, last_name=last_name)
    await user.save()
    return user

@app.get("/users")
async def users(first_name: Optional[str] = None, last_name: Optional[str] = None) ->
↳ List[User]:
    """ Search users """
    return await User.search_by_fields(first_name=first_name, last_name=last_name)

```

### 2.5.1 FastAPI pagination

You can add pagination and sort parameters as a dependency in endpoint arguments: The pagination dependency set the X-Total-Hits header in the response to the total number of hits. So in your frontend, you can get the total number of hits from this header.

```
from typing import List
from esorm import ESModelTimestamp, Pagination, Sort
from fastapi import FastAPI, Depends

from esorm.fastapi import make_dep_sort, make_dep_pagination

class User(ESModelTimestamp):
    """ The User model """
    first_name: str
    last_name: str

app = FastAPI()

@app.get("/all_users")
async def all_users(
    # This will create a _page, and a _page_size query parameter for the endpoint
    pagination: Pagination = Depends(make_dep_pagination(default_page=1, default_page_
    ↪size=10)),
    # This will create a _sort enum query parameter for the endpoint, so it is
    ↪selectable in swagger UI
    sort: Sort = Depends(make_dep_sort(
        first_name_last_name_asc= # This is the name of the 1st sort option
        # Definition of the sort options
        [
            {'first_name': {"order": "asc"}},
            {'last_name': {"order": "asc"}},
        ],
        last_name_first_name_asc= # This is the name of the 2nd sort option
        # Definition of the sort options
        [
            {'last_name': {"order": "asc"}},
            {'first_name': {"order": "asc"}},
        ]
    )),
) -> List[User]:
    """ Get all users """
    return await sort(pagination(User)).all()
```



## REFERENCE

### 3.1 Module contents

ESORM is an elasticsearch python ORM based on Pydantic

**class** esorm.ESBaseModel(\*\*data)

Bases: BaseModel

Base class for Elastic

It is useful for nested models, if you don't need the model in ES mappings

**class** ESConfig

Bases: object

ESBaseModel Config

This is just for lazy properties, to make ESBasemodel compatible with them

**lazy\_property\_max\_recursion\_depth:** int = 1

Maximum recursion depth of lazy properties

**async** calc\_lazy\_properties()

(re)Calculate lazy properties

**model\_computed\_fields:** ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

**model\_config:** ClassVar[ConfigDict] = {'arbitrary\_types\_allowed': True, 'extra':  
'forbid', 'populate\_by\_name': True, 'ser\_json\_bytes': 'base64',  
'str\_strip\_whitespace': True, 'validate\_assignment': True}

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

**model\_fields:** ClassVar[dict[str, FieldInfo]] = {}

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.\_\_fields\_\_* from Pydantic V1.

**class** esorm.ESBulk(wait\_for=False, \*\*bulk\_kwargs)

Bases: object

Bulk operation for ElasticSearch

**async delete**(*model*)

Add the model to the bulk for deletion

**Parameters**

**model** (TypeVar(TModel, bound= ESModel)) – The model to add for deletion

**async save**(*model*)

Add the model to the bulk for saving

If the model is from ES (get or search, so it has `_seq_no` and `_primary_term`), it will use optimistic concurrency check, so it will only update the document if the `_seq_no` and `_primary_term` are the same as the document in the index.

If the model is an ESModelTimestamp, it will update the `modified_at` field to the current time and if the `created_at` field is not already set, it will set it to the current time too.

**Parameters**

**model** (TypeVar(TModel, bound= ESModel)) – The model to add for saving

**class** esorm.ESModel(\*\*data)

Bases: [ESBaseModel](#)

ElasticSearch Base Model

**class** ESConfig

Bases: object

ESModel Config

**default\_sort:** Optional[List[Dict[str, Dict[str, str]]]] = None

Default sort

**id\_field:** Optional[str] = None

The name of the 'id' field

**index:** Optional[str] = None

The index name

**lazy\_property\_max\_recursion\_depth:** int = 1

Maximum recursion depth of lazy properties

**settings:** Optional[Dict[str, Any]] = None

Index settings

**async classmethod aggregate**(aggs, \*, query=None, routing=None, \*\*kwargs)

Aggregate Model with aggregation dict Before aggregation the model can be filtered by query dict.

**Parameters**

- **aggs** (Dict[str, [ESAgg](#)]) – Aggregation dict
- **query** (Optional[[ESQuery](#)]) – ElasticSearch query dict
- **routing** (Optional[str]) – Shard routing value
- **kwargs** – Other search API params

**Return type**

Dict[str, Union[[ESAggValueResponse](#), [ESAggTermsResponse](#), [ESAggHistogramResponse](#)]]

**Returns**

The result list



**async classmethod all**(\*\*kwargs)

Get all documents

**Parameters**

**kwargs** – Other search API params

**Return type**

List[TypeVar(TModel, bound= ESModel)]

**Returns**

The result list

**async classmethod call**(method\_name, \*, wait\_for=None, \*\*kwargs)

Call an elasticsearch method

This is a low level ES method call, it is not recommended to use this directly.

**Parameters**

- **method\_name** – The name of the method to call
- **wait\_for** – Waits for all shards to sync before returning response
- **kwargs** – The arguments to pass to the method

**Return type**

dict

**Returns**

The result dictionary from ElasticSearch

**static create\_query\_from\_dict**(fields)

Creates a query dict from a dictionary of fields and values

**Parameters**

**fields** (Dict[str, Union[str, int, float]]) – A dictionary of fields and values to search by

**Return type**

*ESQuery*

**Returns**

A query dict

**async delete**(\*, wait\_for=False, routing=None)

Deletes document from ElasticSearch.

**Parameters**

- **wait\_for** – Waits for all shards to sync before returning response - useful when writing tests. Defaults to False.
- **routing** (Optional[str]) – Shard routing value

**Raises**

- *esorm.error.NotFoundError* – Returned if document not found
- **ValueError** – Returned when id attribute missing from instance

**classmethod from\_es**(data)

Returns an ESModel from an elasticsearch document that has \_id, \_source

**Parameters**

**data** (Dict[str, Any]) – Elasticsearch document that has \_id, \_source

**Raises**

*esorm.error.InvalidResponseError* – Returned when `_id` or `_source` is missing from data

**Return type**

`Optional[TypeVar(TModel, bound= ESModel)]`

**Returns**

The ESModel instance

**async classmethod** `get(id, *, routing=None)`

Fetches document and returns ESModel instance populated with properties.

**Parameters**

- **id** (`Union[str, int, float]`) – Document id
- **routing** (`Optional[str]`) – Shard routing value

**Raises**

*esorm.error.NotFoundError* – Returned if document not found

**Return type**

`TypeVar(TModel, bound= ESModel)`

**Returns**

ESModel object

**model\_computed\_fields:** `ClassVar[dict[str, ComputedFieldInfo]] = {}`

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

**model\_config:** `ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'extra': 'forbid', 'populate_by_name': True, 'ser_json_bytes': 'base64', 'str_strip_whitespace': True, 'validate_assignment': True}`

Configuration for the model, should be a dictionary conforming to `[ConfigDict][pydantic.config.ConfigDict]`.

**model\_fields:** `ClassVar[dict[str, FieldInfo]] = {}`

Metadata about the fields defined on the model, mapping of field names to `[FieldInfo][pydantic.fields.FieldInfo]`.

This replaces `Model.__fields__` from Pydantic V1.

**model\_post\_init**(`__context`)

This function is meant to behave like a BaseModel method to initialise private attributes.

It takes context as an argument since that's what pydantic-core passes when calling it.

**Return type**

None

**Args:**

self: The BaseModel instance. `__context`: The context.

**async** `reload(*, routing=None)`

Reloads the document from Elasticsearch

**Parameters**

**routing** (`Optional[str]`) – Shard routing value

**Raises**

*esorm.error.NotFoundError* – Returned if document not found

**Return type**

TypeVar(TModel, bound= ESModel)

**async save**(\*, wait\_for=False, pipeline=None, routing=None)

Save document into elasticsearch.

If document already exists, existing document will be updated as per native elasticsearch index operation. If model has id (Config.id\_field or \_\_id\_\_), this will be used as the elasticsearch \_id. The id field will be removed from the document before indexing. If no id is provided, then document will be indexed and elasticsearch will generate a suitable id that will be populated on the returned model.

**Parameters**

- **wait\_for** – Waits for all shards to sync before returning response - useful when writing tests. Defaults to False.
- **pipeline** (Optional[str]) – Pipeline to use for indexing
- **routing** (Optional[str]) – Shard routing value

**Return type**

str

**Returns**

The new document's ID, it is always a string, even if the id field is an integer

**async classmethod search**(query, \*, page\_size=None, page=None, sort=None, routing=None, res\_dict=False, \*\*kwargs)

Search Model with query dict

**Parameters**

- **query** ([ESQuery](#)) – ElasticSearch query dict
- **page\_size** (Optional[int]) – Pagination page size
- **page** (Optional[int]) – Pagination page num, 1st page is 1
- **sort** (Union[list, str, None]) – Name of field to be sorted, or sort term list of dict, if not specified, model's default sort will be used, or no sorting
- **routing** (Optional[str]) – Shard routing value
- **res\_dict** (bool) – If the result should be a dict with id as key and model as value instead of a list of models
- **kwargs** – Other search API params

**Return type**

Union[List[TypeVar(TModel, bound= ESModel)], Dict[str, TypeVar(TModel, bound= ESModel)]]

**Returns**

The result list

**async classmethod search\_by\_fields**(fields, \*, page\_size=None, page=None, sort=None, routing=None, aggs=None, res\_dict=False, \*\*kwargs)

Search Model by fields as key-value pairs

**Parameters**

- **fields** (Dict[str, Union[str, int, float]]) – A dictionary of fields and values to search by
- **page\_size** (Optional[int]) – Pagination page size

- **page** (Optional[int]) – Pagination page num, 1st page is 1
- **sort** (Union[list, str, None]) – Name of field to be sorted, or sort term list of dict, if not specified, model's default sort will be used, or no sorting
- **routing** (Optional[str]) – Shard routing value
- **aggs** (Optional[Dict[str, [ESAgg](#)]]) – Aggregations
- **res\_dict** (bool) – If the result should be a dict with id as key and model as value instead of a list of models
- **kwargs** – Other search API params

**Return type**

List[TypeVar(TModel, bound= ESModel)]

**Returns**

The result list

**async classmethod search\_one**(*query*, \*, *routing=None*, *\*\*kwargs*)

Search Model and return the first result

**Parameters**

- **query** ([ESQuery](#)) – Elasticsearch query dict
- **routing** (Optional[str]) – Shard routing value
- **kwargs** – Other search API params

**Return type**

Optional[TypeVar(TModel, bound= ESModel)]

**Returns**

The first result or None if no result

**async classmethod search\_one\_by\_fields**(*fields*, \*, *routing=None*, *aggs=None*, *\*\*kwargs*)

Search Model by fields as key-value pairs and return the first result

**Parameters**

- **fields** (Dict[str, Union[str, int, float]]) – A dictionary of fields and values to search by
- **routing** (Optional[str]) – Shard routing value
- **aggs** (Optional[Dict[str, [ESAgg](#)]]) – Aggregations
- **kwargs** – Other search API params

**Return type**

Optional[TypeVar(TModel, bound= ESModel)]

**Returns**

The first result or None if no result

**to\_es**(*\*\*kwargs*)

Generates a dictionary equivalent to what Elasticsearch returns in the ‘\_source’ property of a response.

It automatically removes the id field from the document if it is set in ESConfig.id\_field to prevent duplication of the id field.

**Parameters**

**kwargs** – Pydantic's model\_dump parameters

**Return type**

dict

**Returns**

The dictionary for Elasticsearch

**update\_from\_es**(*data*)

Update the model from Elasticsearch data

**Parameters****data** (Dict[str, Any]) – The Elasticsearch data**Raises****esorm.error.InvalidResponseError** – Returned when `_id` or `_source` is missing from data**class** esorm.ESModelTimestamp(\*\**data*)Bases: *ESModel*Model which stores *created\_at* and *modified\_at* fields automatically.**created\_at**: Optional[datetime]**model\_computed\_fields**: ClassVar[dict[str, ComputedFieldInfo]] = {}A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.**model\_config**: ClassVar[ConfigDict] = {'arbitrary\_types\_allowed': True, 'extra': 'forbid', 'populate\_by\_name': True, 'ser\_json\_bytes': 'base64', 'str\_strip\_whitespace': True, 'validate\_assignment': True}

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

**model\_fields**: ClassVar[dict[str, FieldInfo]] = {'created\_at': FieldInfo(annotation=Union[datetime, NoneType], required=False, default=None, description='Creation date and time'), 'modified\_at': FieldInfo(annotation=Union[datetime, NoneType], required=False, default\_factory=utcnow, description='Modification date and time')}

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.\_\_fields\_\_* from Pydantic V1.**model\_post\_init**(*\_ModelMetaclass\_\_context: Any*) → NoneWe need to both initialize private attributes and call the user-defined *model\_post\_init* method.**Return type**

None

**modified\_at**: Optional[datetime]**async save**(\*, *wait\_for=False*, *force\_new=False*, *pipeline=None*, *routing=None*)

Save document into elasticsearch.

If document already exists, existing document will be updated as per native elasticsearch index operation. If model has id (*Meta.id\_field* or `__id__`), this will be used as the elasticsearch `_id`. The id field will be removed from the document before indexing. If no id is provided, then document will be indexed and elasticsearch will generate a suitable id that will be populated on the returned model.**Parameters**

- **wait\_for** – Waits for all shards to sync before returning response - useful when writing tests. Defaults to False.
- **force\_new** – It is assumed to be a new document, so created\_at will be set to current time (it is no more necessary, because created\_at is set to current time if it is None. It is here for backward compatibility)
- **pipeline** (Optional[str]) – Pipeline to use for indexing
- **routing** (Optional[str]) – Shard routing value

**Return type**

str

**Returns**

The new document's ID

`esorm.Field(default, *, index=True, alias=None, title=None, description=None, exclude=None, include=None, frozen=False, **extra)`

Basic Field Info

**Parameters**

- **default** (Any) – since this is replacing the field's default, its first argument is used to set the default, use ellipsis ( . . . ) to indicate the field is required
- **index** (bool) – if this field should be indexed or not
- **alias** (Optional[str]) – the public name of the field
- **title** (Optional[str]) – can be any string, used in the schema
- **description** (Optional[str]) – can be any string, used in the schema
- **exclude** (Optional[bool]) – exclude this field while dumping. Takes same values as the include and exclude arguments on the .dict method.
- **include** (Optional[bool]) – include this field while dumping. Takes same values as the include and exclude arguments on the .dict method.
- **frozen** (bool) – if this field should be frozen or not
- **extra** – any additional keyword arguments will be added as is to the schema

**Return type**

FieldInfo

**Returns**

A field info object

**exception** `esorm.InvalidModelError`

Bases: Exception

Raised when a model is invalid.

**exception** `esorm.InvalidResponseError`

Bases: Exception

Raised when the response from Elasticsearch is invalid.

**exception** `esorm.NotFoundError`

Bases: Exception

Raised when a model is not found.

```

class esorm.Pagination(**data)
    Bases: BaseModel

    Pagination parameters

    callback: Optional[Callable[[int], Awaitable[None]]]
        Callback after the search is done with the total number of hits

    model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
        A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

    model_config: ClassVar[ConfigDict] = {}
        Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

    model_fields: ClassVar[dict[str, FieldInfo]] = {'callback':
        FieldInfo(annotation=Union[Callable[[int], Awaitable[NoneType]], NoneType],
        required=False, default=None, description='Callback after the search is done with
        the total number of hits'), 'page': FieldInfo(annotation=int, required=False,
        default=1, description='The page number'), 'page_size': FieldInfo(annotation=int,
        required=False, default=10, description='The page size')}

        Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

        This replaces Model.__fields__ from Pydantic V1.

    page: int
        The page number

    page_size: int
        The page size

class esorm.Sort(**data)
    Bases: BaseModel

    Sort parameters

    model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
        A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

    model_config: ClassVar[ConfigDict] = {}
        Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

    model_fields: ClassVar[dict[str, FieldInfo]] = {'sort':
        FieldInfo(annotation=Union[List[Dict[str, esorm.model.SortOrder]], str, NoneType],
        required=True)}

        Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

        This replaces Model.__fields__ from Pydantic V1.

    sort: Union[List[Dict[str, SortOrder]], str, None]

async esorm.connect(hosts, *args, wait=False, **kwargs)
    Connect to Elasticsearch

    Parameters

```

- **hosts** (Union[str, List[Union[str, Mapping[str, Union[str, int]]], NodeConfig]]))  
– Elasticsearch hosts to connect, either a list a mapping, or a single string
- **args** – Other AsyncElasticsearch arguments
- **wait** – Wait for AsyncElasticsearch to be ready
- **kwargs** – Other AsyncElasticsearch keyword arguments

**Return type**

Optional[AsyncElasticsearch]

**Returns**

AsyncElasticsearch client instance

**async** `esorm.get_es_version()`

Get Elasticsearch version

**Return type**

str

**Returns**

ElasticSearch version

`esorm.lazy_property(func)`

Decorator for lazy properties

Lazy properties computed after search from ES

**Parameters**

**func** (Callable[[], Awaitable[Any]]) – The async function to decorate

**Returns**

The decorated function

`esorm.retry_on_conflict(max_retries=-1)`

Decorator for optimistic concurrency control

**Parameters**

**max\_retries** – The maximum number of retries, -1 for infinite

**Returns**

The decorated function

**async** `esorm.setup_mappings(*_, debug=False)`

Create mappings for indices or try to extend it if there are new fields

## 3.2 Submodules

### 3.2.1 esorm.aggs module

ElasticSearch aggregation type definitions for ESORM

**class** `esorm.aggs.ESAgg(*args, **kwargs)`

Bases: dict

Holds all types of aggregations supported

**avg:** [\*ESAggFieldParams\*](#)

Average aggregation



**histogram:** *ESAggHistogramParams*  
Histogram aggregation

**max:** *ESAggFieldParams*  
Maximum aggregation

**min:** *ESAggFieldParams*  
Minimum aggregation

**sum:** *ESAggFieldParams*  
Sum aggregation

**terms:** *ESAggTermParams*  
Terms aggregation

**class** esorm.aggs.**ESAggBucketResponse**(\*args, \*\*kwargs)  
Bases: dict  
Represents a single bucket in a bucket aggregation.

**doc\_count:** **int**  
The number of documents in this bucket.

**key:** **str**  
The key of the bucket.

**class** esorm.aggs.**ESAggExtendedBounds**(\*args, \*\*kwargs)  
Bases: dict  
Represents the parameters for extended bounds in Elasticsearch.

**max:** **int**  
The maximum value.

**min:** **int**  
The minimum value.

**class** esorm.aggs.**ESAggFieldParams**(\*args, \*\*kwargs)  
Bases: dict  
Represents field parameter in Elasticsearch.

**field:** **str**  
The field to aggregate on.

**class** esorm.aggs.**ESAggHistogramBucketresponse**(\*args, \*\*kwargs)  
Bases: dict  
Represents a bucket in a histogram aggregation.

**doc\_count:** **int**  
The number of documents in this bucket.

**key:** **float**  
Numeric key corresponding to the bucket's range.

**class** esorm.aggs.**ESAggHistogramParams**(\*args, \*\*kwargs)  
Bases: dict  
Represents the parameters for a histogram aggregation in Elasticsearch.

**extended\_bounds:** [\*ESAggExtendedBounds\*](#)

The extended bounds of the histogram.

**field:** **str**

The field to aggregate on.

**interval:** **int**

The interval of the histogram.

**min\_doc\_count:** **int**

The minimum number of documents in a bucket.

**class** esorm.aggs.**ESAggHistogramResponse**(\*args, \*\*kwargs)

Bases: dict

Represents the response for a histogram aggregation.

**buckets:** **List**[[\*ESAggHistogramBucketresponse\*](#)]

A list of buckets in the histogram aggregation.

**class** esorm.aggs.**ESAggTermParams**(\*args, \*\*kwargs)

Bases: dict

Represents the parameters for a terms aggregation in Elasticsearch.

**field:** **str**

The field to aggregate on.

**order:** **Dict**[**str**, **str**]

The order of the buckets.

**size:** **int**

The number of buckets to return.

**class** esorm.aggs.**ESAggTermsResponse**(\*args, \*\*kwargs)

Bases: dict

Represents the response for a terms aggregation.

**buckets:** **List**[[\*ESAggBucketResponse\*](#)]

A list of buckets in the terms aggregation.

**class** esorm.aggs.**ESAggValueResponse**(\*args, \*\*kwargs)

Bases: dict

Represents the response for an average, min, or max aggregation.

**value:** **float**

The average, min, or max value.

esorm.aggs.**ESAggs**

ElasticSearch aggregations type definition

alias of Dict[str, [\*ESAgg\*](#)]

esorm.aggs.**ESAggsResponse**

ElasticSearch aggregations response type definition

alias of Dict[str, Union[[\*ESAggValueResponse\*](#), [\*ESAggTermsResponse\*](#), [\*ESAggHistogramResponse\*](#)]]

### 3.2.2 esorm.bulk module

Bulk operation for ElasticSearch

**class** esorm.bulk.**ESBulk**(*wait\_for=False, \*\*bulk\_kwargs*)

Bases: object

Bulk operation for ElasticSearch

**async delete**(*model*)

Add the model to the bulk for deletion

**Parameters**

**model** (TypeVar(TModel, bound= ESModel)) – The model to add for deletion

**async save**(*model*)

Add the model to the bulk for saving

If the model is from ES (get or search, so it has `_seq_no` and `_primary_term`), it will use optimistic concurrency check, so it will only update the document if the `_seq_no` and `_primary_term` are the same as the document in the index.

If the model is an ESModelTimestamp, it will update the `modified_at` field to the current time and if the `created_at` field is not already set, it will set it to the current time too.

**Parameters**

**model** (TypeVar(TModel, bound= ESModel)) – The model to add for saving

### 3.2.3 esorm.error module

This module contains all the exceptions that can be raised by ESORM.

**exception** esorm.error.**BulkError**(*failed\_operations*)

Bases: Exception

Exception for handling bulk operation errors.

**failed\_operations:** List[[BulkOperationError](#)]

**class** esorm.error.**BulkOperationError**(\*args, \*\*kwargs)

Bases: dict

A dictionary type to represent an error in a bulk operation response from Elasticsearch.

**model:** [ESModel](#)

**reason:** str

**status:** int

**type:** str

**exception** esorm.error.**IndexDoesNotFoundError**

Bases: Exception

Raised when an index does not exist.

**exception** esorm.error.**InvalidModelError**

Bases: Exception

Raised when a model is invalid.

**exception** `esorm.error.InvalidResponseError`

Bases: `Exception`

Raised when the response from Elasticsearch is invalid.

**exception** `esorm.error.NotFoundError`

Bases: `Exception`

Raised when a model is not found.

### 3.2.4 `esorm.esorm` module

ElasticSearch ORM main module

**async** `esorm.esorm.connect`(*hosts*, \**args*, *wait=False*, \*\**kwargs*)

Connect to ElasticSearch

**Parameters**

- **hosts** (`Union[str, List[Union[str, Mapping[str, Union[str, int]]], NodeConfig]]`)  
– ElasticSearch hosts to connect, either a list a mapping, or a single string
- **args** – Other AsyncElasticsearch arguments
- **wait** – Wait for AsyncElasticsearch to be ready
- **kwargs** – Other AsyncElasticsearch keyword arguments

**Return type**

`Optional[AsyncElasticsearch]`

**Returns**

AsyncElasticsearch client instance

**async** `esorm.esorm.get_es_version`()

Get ElasticSearch version

**Return type**

`str`

**Returns**

ElasticSearch version

### 3.2.5 `esorm.fastapi` module

FastaAPI utilities for ESORM

`esorm.fastapi.make_dep_pagination`(*default\_page=1*, *default\_page\_size=10*, *set\_headers=True*)

Create a pagination dependency with default values

**Parameters**

- **default\_page** (`int`) – Default page number, the first page is 1
- **default\_page\_size** (`int`) – Default page size, the default is 10
- **set\_headers** (`bool`) – Set X-Total-Hits header after search

**Return type**

`callable`

**Returns**

Pagination dependency

`esorm.fastapi.make_dep_sort(**kwargs)`

Create a sort dependency with sort definitions

**Parameters**

**kwargs** (Union[List[Dict[str, dict]], Dict[str, any]]) – Sort definitions

**Return type**

callable

**Returns**

Sort dependency

`esorm.fastapi.set_max_page_size(max_page_size)`

Set the maximum page size for queries

**Parameters**

**max\_page\_size** (int) – The maximum page size

**Returns**

None

### 3.2.6 esorm.fields module

`class esorm.fields.Binary`

Bases: str

Stores binary data as base64 encoded strings

**property bytes:** bytes

**classmethod validate\_binary**(v, \_)

**Return type**

str

`class esorm.fields.Byte`

Bases: int

Byte Field

`class esorm.fields.Double(x=0, /)`

Bases: float

Double Field

`esorm.fields.Field(default, *, index=True, alias=None, title=None, description=None, exclude=None, include=None, frozen=False, **extra)`

Basic Field Info

**Parameters**

- **default** (Any) – since this is replacing the field's default, its first argument is used to set the default, use ellipsis ( . . . ) to indicate the field is required
- **index** (bool) – if this field should be indexed or not
- **alias** (Optional[str]) – the public name of the field

- **title** (Optional[str]) – can be any string, used in the schema
- **description** (Optional[str]) – can be any string, used in the schema
- **exclude** (Optional[bool]) – exclude this field while dumping. Takes same values as the include and exclude arguments on the .dict method.
- **include** (Optional[bool]) – include this field while dumping. Takes same values as the include and exclude arguments on the .dict method.
- **frozen** (bool) – if this field should be frozen or not
- **extra** – any additional keyword arguments will be added as is to the schema

**Return type**

FieldInfo

**Returns**

A field info object

**class** esorm.fields.**Float**(x=0, /)

Bases: float

Float Field

**class** esorm.fields.**HalfFloat**(x=0, /)

Bases: float

Half Float Field

**class** esorm.fields.**Integer**

Bases: int

Integer Field

**class** esorm.fields.**Keyword**

Bases: str

Keyword Field

**class** esorm.fields.**LatLon**(\*\*data)

Bases: BaseModel

Geo Point Field - Latitude and Longitude

**lat:** float

Latitude Coordinate

**lon:** float

Longitude Coordinate

**model\_computed\_fields:** ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

**model\_config:** ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

**model\_fields:** ClassVar[dict[str, FieldInfo]] = {'lat': FieldInfo(annotation=float, required=True, description='Latitude Coordinate'), 'lon': FieldInfo(annotation=float, required=True, description='Longitude Coordinate')}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.\_\_fields\_\_* from Pydantic V1.

```
class esorm.fields.Long
```

Bases: int

Long Field

```
esorm.fields.NumericField(default, *, index=None, alias=None, gt=None, ge=None, lt=None, le=None,
                           multiple_of=None, allow_inf_nan=None, max_digits=None,
                           decimal_places=None, title=None, description=None, exclude=None,
                           include=None, frozen=False, **extra)
```

Numeric Field Info

#### Parameters

- **default** (Union[int, float]) – since this is replacing the field’s default, its first argument is used to set the default, use ellipsis (...) to indicate the field is required
- **index** (Optional[bool]) – if this field should be indexed or not
- **alias** (Optional[str]) – the public name of the field
- **gt** (Optional[float]) – only applies to numbers, requires the field to be “greater than”. The schema will have an `exclusiveMinimum` validation keyword
- **ge** (Optional[float]) – only applies to numbers, requires the field to be “greater than or equal to”. The schema will have a `minimum` validation keyword
- **lt** (Optional[float]) – only applies to numbers, requires the field to be “less than”. The schema will have an `exclusiveMaximum` validation keyword
- **le** (Optional[float]) – only applies to numbers, requires the field to be “less than or equal to”. The schema will have a `maximum` validation keyword
- **multiple\_of** (Optional[float]) – only applies to numbers, requires the field to be “a multiple of”. The schema will have a `multipleOf` validation keyword
- **allow\_inf\_nan** (Optional[bool]) – only applies to numbers, allows the field to be NaN or infinity (+inf or -inf), which is a valid Python float. Default True, set to False for compatibility with JSON.
- **max\_digits** (Optional[int]) – only applies to Decimals, requires the field to have a maximum number of digits within the decimal. It does not include a zero before the decimal point or trailing decimal zeroes.
- **decimal\_places** (Optional[int]) – only applies to Decimals, requires the field to have at most a number of decimal places allowed. It does not include trailing decimal zeroes.
- **title** (Optional[str]) – can be any string, used in the schema
- **description** (Optional[str]) – can be any string, used in the schema
- **exclude** (Optional[bool]) – exclude this field while dumping. Takes same values as the `include` and `exclude` arguments on the `.dict` method.
- **include** (Optional[bool]) – include this field while dumping. Takes same values as the `include` and `exclude` arguments on the `.dict` method.
- **frozen** (bool) – if this field should be frozen or not

- **extra** – any additional keyword arguments will be added as is to the schema

**Return type**

FieldInfo

**Returns**

A field info object

**class** esorm.fields.Short

Bases: int

Short Field

**class** esorm.fields.Text

Bases: str

Text Field

esorm.fields.TextField(*default*, \*, *index=True*, *alias=None*, *min\_length=None*, *max\_length=None*, *regex=None*, *title=None*, *description=None*, *exclude=None*, *include=None*, *frozen=False*, \*\**extra*)

Text Field Info

**Parameters**

- **default** (str) – since this is replacing the field’s default, its first argument is used to set the default, use ellipsis ( . . . ) to indicate the field is required
- **index** (bool) – if this field should be indexed or not
- **alias** (Optional[str]) – the public name of the field
- **min\_length** (Optional[int]) – only applies to strings, requires the field to have a minimum length. The schema will have a minLength validation keyword
- **max\_length** (Optional[int]) – only applies to strings, requires the field to have a maximum length. The schema will have a maxLength validation keyword
- **regex** (Optional[str]) – only applies to strings, requires the field match against a regular expression pattern string. The schema will have a pattern validation keyword
- **title** (Optional[str]) – can be any string, used in the schema
- **description** (Optional[str]) – can be any string, used in the schema
- **exclude** (Optional[bool]) – exclude this field while dumping. Takes same values as the include and exclude arguments on the .dict method.
- **include** (Optional[bool]) – include this field while dumping. Takes same values as the include and exclude arguments on the .dict method.
- **frozen** (bool) – if this field should be frozen or not
- **extra** – any additional keyword arguments will be added as is to the schema

**Return type**

FieldInfo

**Returns**

A field info object

esorm.fields.binary

Binary type

alias of Union[*Binary*, str]



`esorm.fields.boolean`

alias of `bool`

`esorm.fields.byte`

Byte type

alias of `Union[Byte, int]`

`esorm.fields.double`

64 bit float (double) type

alias of `Union[Double, float]`

`esorm.fields.float16`

16 bit float type

alias of `Union[HalfFloat, float]`

`esorm.fields.float32`

32 bit float type

alias of `Union[Float, float]`

`esorm.fields.geo_point`

Geo Point type

`esorm.fields.int32`

32 bit integer type

alias of `Union[Integer, int]`

`esorm.fields.keyword`

Keyword type

alias of `Union[Keyword, str]`

`esorm.fields.long`

64 bit integer (long) type

alias of `Union[Long, int]`

`esorm.fields.short`

Short type

alias of `Union[Short, int]`

`esorm.fields.text`

Text type

alias of `Union[Text, str]`

### 3.2.7 esorm.logger module

### 3.2.8 esorm.model module

This module contains the ESMModel classes and related functions

```
class esorm.model.ESBaseModel(**data)
```

Bases: BaseModel

Base class for Elastic

It is useful for nested models, if you don't need the model in ES mappings

```
class ESConfig
```

Bases: object

ESBaseModel Config

This is just for lazy properties, to make ESBasemodel compatible with them

```
lazy_property_max_recursion_depth: int = 1
```

Maximum recursion depth of lazy properties

```
async calc_lazy_properties()
```

(re)Calculate lazy properties

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'extra':  
'forbid', 'populate_by_name': True, 'ser_json_bytes': 'base64',  
'str_strip_whitespace': True, 'validate_assignment': True}
```

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

```
model_fields: ClassVar[dict[str, FieldInfo]] = {}
```

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.\_\_fields\_\_* from Pydantic V1.

```
class esorm.model.ESModel(**data)
```

Bases: *ESBaseModel*

ElasticSearch Base Model

```
class ESConfig
```

Bases: object

ESModel Config

```
default_sort: Optional[List[Dict[str, Dict[str, str]]]] = None
```

Default sort

```
id_field: Optional[str] = None
```

The name of the 'id' field

```
index: Optional[str] = None
```

The index name

**lazy\_property\_max\_recursion\_depth:** `int = 1`

Maximum recursion depth of lazy properties

**settings:** `Optional[Dict[str, Any]] = None`

Index settings

**async classmethod aggregate**(*aggs*, \*, *query=None*, *routing=None*, \*\**kwargs*)

Aggregate Model with aggregation dict Before aggregation the model can be filtered by query dict.

**Parameters**

- **aggs** (`Dict[str, ESagg]`) – Aggregation dict
- **query** (`Optional[ESQuery]`) – ElasticSearch query dict
- **routing** (`Optional[str]`) – Shard routing value
- **kwargs** – Other search API params

**Return type**

`Dict[str, Union[ESaggValueResponse, ESaggTermsResponse, ESaggHistogramResponse]]`

**Returns**

The result list

**async classmethod all**(\*\**kwargs*)

Get all documents

**Parameters**

**kwargs** – Other search API params

**Return type**

`List[TypeVar(TModel, bound= ESModel)]`

**Returns**

The result list

**async classmethod call**(*method\_name*, \*, *wait\_for=None*, \*\**kwargs*)

Call an elasticsearch method

This is a low level ES method call, it is not recommended to use this directly.

**Parameters**

- **method\_name** – The name of the method to call
- **wait\_for** – Waits for all shards to sync before returning response
- **kwargs** – The arguments to pass to the method

**Return type**

dict

**Returns**

The result dictionary from ElasticSearch

**static create\_query\_from\_dict**(*fields*)

Creates a query dict from a dictionary of fields and values

**Parameters**

**fields** (`Dict[str, Union[str, int, float]]`) – A dictionary of fields and values to search by

**Return type**

*ESQuery*

**Returns**

A query dict

**async delete**(\*, *wait\_for=False*, *routing=None*)

Deletes document from Elasticsearch.

**Parameters**

- **wait\_for** – Waits for all shards to sync before returning response - useful when writing tests. Defaults to False.
- **routing** (Optional[str]) – Shard routing value

**Raises**

- *esorm.error.NotFoundError* – Returned if document not found
- **ValueError** – Returned when id attribute missing from instance

**classmethod from\_es**(data)

Returns an ESMModel from an elasticsearch document that has \_id, \_source

**Parameters**

**data** (Dict[str, Any]) – Elasticsearch document that has \_id, \_source

**Raises**

*esorm.error.InvalidResponseError* – Returned when \_id or \_source is missing from data

**Return type**

Optional[TypeVar(TModel, bound= ESMModel)]

**Returns**

The ESMModel instance

**async classmethod get**(id, \*, *routing=None*)

Fetches document and returns ESMModel instance populated with properties.

**Parameters**

- **id** (Union[str, int, float]) – Document id
- **routing** (Optional[str]) – Shard routing value

**Raises**

*esorm.error.NotFoundError* – Returned if document not found

**Return type**

TypeVar(TModel, bound= ESMModel)

**Returns**

ESModel object

**model\_computed\_fields:** ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

**model\_config:** ClassVar[ConfigDict] = {'arbitrary\_types\_allowed': True, 'extra': 'forbid', 'populate\_by\_name': True, 'ser\_json\_bytes': 'base64', 'str\_strip\_whitespace': True, 'validate\_assignment': True}

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

**model\_fields:** `ClassVar[dict[str, FieldInfo]] = {}`

Metadata about the fields defined on the model, mapping of field names to `[FieldInfo][pydantic.fields.FieldInfo]`.

This replaces `Model.__fields__` from Pydantic V1.

**model\_post\_init**(`__context`)

This function is meant to behave like a BaseModel method to initialise private attributes.

It takes context as an argument since that's what pydantic-core passes when calling it.

**Return type**

None

**Args:**

self: The BaseModel instance. `__context`: The context.

**async reload**(`*`, `routing=None`)

Reloads the document from ElasticSearch

**Parameters**

**routing** (Optional[str]) – Shard routing value

**Raises**

`esorm.error.NotFoundError` – Returned if document not found

**Return type**

TypeVar(TModel, bound= ESModel)

**async save**(`*`, `wait_for=False`, `pipeline=None`, `routing=None`)

Save document into elasticsearch.

If document already exists, existing document will be updated as per native elasticsearch index operation. If model has id (`Config.id_field` or `__id__`), this will be used as the elasticsearch `_id`. The id field will be removed from the document before indexing. If no id is provided, then document will be indexed and elasticsearch will generate a suitable id that will be populated on the returned model.

**Parameters**

- **wait\_for** – Waits for all shards to sync before returning response - useful when writing tests. Defaults to False.
- **pipeline** (Optional[str]) – Pipeline to use for indexing
- **routing** (Optional[str]) – Shard routing value

**Return type**

str

**Returns**

The new document's ID, it is always a string, even if the id field is an integer

**async classmethod search**(`query, *`, `page_size=None`, `page=None`, `sort=None`, `routing=None`, `res_dict=False`, `**kwargs`)

Search Model with query dict

**Parameters**

- **query** (`ESQuery`) – ElasticSearch query dict
- **page\_size** (Optional[int]) – Pagination page size
- **page** (Optional[int]) – Pagination page num, 1st page is 1

- **sort** (Union[list, str, None]) – Name of field to be sorted, or sort term list of dict, if not specified, model's default sort will be used, or no sorting
- **routing** (Optional[str]) – Shard routing value
- **res\_dict** (bool) – If the result should be a dict with id as key and model as value instead of a list of models
- **kwargs** – Other search API params

**Return type**

Union[List[TypeVar(TModel, bound= ESModel)], Dict[str, TypeVar(TModel, bound= ESModel)]]

**Returns**

The result list

**async classmethod search\_by\_fields**(fields, \*, page\_size=None, page=None, sort=None, routing=None, aggs=None, res\_dict=False, \*\*kwargs)

Search Model by fields as key-value pairs

**Parameters**

- **fields** (Dict[str, Union[str, int, float]]) – A dictionary of fields and values to search by
- **page\_size** (Optional[int]) – Pagination page size
- **page** (Optional[int]) – Pagination page num, 1st page is 1
- **sort** (Union[list, str, None]) – Name of field to be sorted, or sort term list of dict, if not specified, model's default sort will be used, or no sorting
- **routing** (Optional[str]) – Shard routing value
- **aggs** (Optional[Dict[str, [ESAgg](#)]]) – Aggregations
- **res\_dict** (bool) – If the result should be a dict with id as key and model as value instead of a list of models
- **kwargs** – Other search API params

**Return type**

List[TypeVar(TModel, bound= ESModel)]

**Returns**

The result list

**async classmethod search\_one**(query, \*, routing=None, \*\*kwargs)

Search Model and return the first result

**Parameters**

- **query** ([ESQuery](#)) – Elasticsearch query dict
- **routing** (Optional[str]) – Shard routing value
- **kwargs** – Other search API params

**Return type**

Optional[TypeVar(TModel, bound= ESModel)]

**Returns**

The first result or None if no result

**async classmethod search\_one\_by\_fields**(*fields*, \*, *routing=None*, *aggs=None*, *\*\*kwargs*)

Search Model by fields as key-value pairs and return the first result

**Parameters**

- **fields** (Dict[str, Union[str, int, float]]) – A dictionary of fields and values to search by
- **routing** (Optional[str]) – Shard routing value
- **aggs** (Optional[Dict[str, *ESAgg*]]) – Aggregations
- **kwargs** – Other search API params

**Return type**

Optional[TypeVar(TModel, bound= ESModel)]

**Returns**

The first result or None if no result

**to\_es**(*\*\*kwargs*)

Generates a dictionary equivalent to what ElasticSearch returns in the ‘\_source’ property of a response.

It automatically removes the id field from the document if it is set in ESConfig.id\_field to prevent duplication of the id field.

**Parameters**

**kwargs** – Pydantic’s model\_dump parameters

**Return type**

dict

**Returns**

The dictionary for ElasticSearch

**update\_from\_es**(*data*)

Update the model from ElasticSearch data

**Parameters**

**data** (Dict[str, Any]) – The ElasticSearch data

**Raises**

*esorm.error.InvalidResponseError* – Returned when \_id or \_source is missing from data

**class** esorm.model.ESModelTimestamp(*\*\*data*)

Bases: *ESModel*

Model which stores *created\_at* and *modified\_at* fields automatically.

**created\_at**: Optional[datetime]

**model\_computed\_fields**: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

**model\_config**: ClassVar[ConfigDict] = {'arbitrary\_types\_allowed': True, 'extra': 'forbid', 'populate\_by\_name': True, 'ser\_json\_bytes': 'base64', 'str\_strip\_whitespace': True, 'validate\_assignment': True}

Configuration for the model, should be a dictionary conforming to [Config-Dict][pydantic.config.ConfigDict].

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'created_at':
FieldInfo(annotation=Union[datetime, NoneType], required=False, default=None,
description='Creation date and time'), 'modified_at':
FieldInfo(annotation=Union[datetime, NoneType], required=False,
default_factory=utcnow, description='Modification date and time')}
```

Metadata about the fields defined on the model, mapping of field names to *[Field-Info][pydantic.fields.FieldInfo]*.

This replaces *Model.\_\_fields\_\_* from Pydantic V1.

**model\_post\_init**(*\_ModelMetaclass\_\_context: Any*) → None

We need to both initialize private attributes and call the user-defined *model\_post\_init* method.

**Return type**

None

**modified\_at:** Optional[datetime]

**async save**(\*, *wait\_for=False, force\_new=False, pipeline=None, routing=None*)

Save document into elasticsearch.

If document already exists, existing document will be updated as per native elasticsearch index operation. If model has id (*Meta.id\_field* or *\_\_id\_\_*), this will be used as the elasticsearch *\_id*. The id field will be removed from the document before indexing. If no id is provided, then document will be indexed and elasticsearch will generate a suitable id that will be populated on the returned model.

**Parameters**

- **wait\_for** – Waits for all shards to sync before returning response - useful when writing tests. Defaults to False.
- **force\_new** – It is assumed to be a new document, so *created\_at* will be set to current time (it is no more necessary, because *created\_at* is set to current time if it is None. It is here for backward compatibility)
- **pipeline** (Optional[str]) – Pipeline to use for indexing
- **routing** (Optional[str]) – Shard routing value

**Return type**

str

**Returns**

The new document's ID

**class** esorm.model.Pagination(\*\**data*)

Bases: BaseModel

Pagination parameters

**callback:** Optional[Callable[[int], Awaitable[None]]]

Callback after the search is done with the total number of hits

**model\_computed\_fields:** ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

**model\_config:** ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to *[Config-Dict][pydantic.config.ConfigDict]*.



```
model_fields: ClassVar[dict[str, FieldInfo]] = {'callback':
FieldInfo(annotation=Union[Callable[[int], Awaitable[NoneType]], NoneType],
required=False, default=None, description='Callback after the search is done with
the total number of hits'), 'page': FieldInfo(annotation=int, required=False,
default=1, description='The page number'), 'page_size': FieldInfo(annotation=int,
required=False, default=10, description='The page size')}
```

Metadata about the fields defined on the model, mapping of field names to *[Field-Info]*[pydantic.fields.FieldInfo].

This replaces *Model.\_\_fields\_\_* from Pydantic V1.

**page: int**

The page number

**page\_size: int**

The page size

```
class esorm.model.Sort(**data)
```

Bases: BaseModel

Sort parameters

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

```
model_config: ClassVar[ConfigDict] = {}
```

Configuration for the model, should be a dictionary conforming to *[Config-Dict]*[pydantic.config.ConfigDict].

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'sort':
FieldInfo(annotation=Union[List[Dict[str, esorm.model.SortOrder]], str, NoneType],
required=True)}
```

Metadata about the fields defined on the model, mapping of field names to *[Field-Info]*[pydantic.fields.FieldInfo].

This replaces *Model.\_\_fields\_\_* from Pydantic V1.

```
sort: Union[List[Dict[str, SortOrder]], str, None]
```

```
async esorm.model.create_index_template(name, *, prefix_name, shards=1, replicas=0, **other_settings)
```

Create index template

#### Parameters

- **name** (str) – The name of the template
- **prefix\_name** (str) – The prefix of index pattern
- **shards** – Number of shards
- **replicas** – Number of replicas
- **other\_settings** (Any) – Other settings

#### Return type

object

#### Returns

The result object from ES

`esorm.model.lazy_property(func)`

Decorator for lazy properties

Lazy properties computed after search from ES

**Parameters**

**func** (Callable[[], Awaitable[Any]]) – The async function to decorate

**Returns**

The decorated function

`esorm.model.retry_on_conflict(max_retries=-1)`

Decorator for optimistic concurrency control

**Parameters**

**max\_retries** – The maximum number of retries, -1 for infinite

**Returns**

The decorated function

`esorm.model.set_default_index_prefix(default_index_prefix)`

Set default index prefix we use for model and index creation

**Parameters**

**default\_index\_prefix** (str) – The default index prefix

`esorm.model.set_max_lazy_property_concurrency(concurrency)`

Set the maximum concurrency of processing lazy properties

If this is not set, the default is 16.

**Parameters**

**concurrency** (int) – The maximum concurrency

**async** `esorm.model.setup_mappings(*_, debug=False)`

Create mappings for indices or try to extend it if there are new fields

### 3.2.9 esorm.query module

Elasticsearch query type definitions for ESORM

**class** `esorm.query.ESBool(*args, **kwargs)`

Bases: dict

Bool query structure

**boost:** float

Boosting value for the query

**filter:** List[Union[FieldRange, FieldTerm, FieldTerms, FieldMatch, FieldMatchPhrase, FieldExists, FieldWildcard, FieldPrefix, FieldFuzzy, FieldGeoDistance, FieldMatchAll, FieldESMatchNone, FieldBool]]

Filter queries

**minimum\_should\_match:** Union[int, str]

Minimum number of should queries to match

```
must: List[Union[FieldRange, FieldTerm, FieldTerms, FieldMatch, FieldMatchPhrase,
FieldExists, FieldWildcard, FieldPrefix, FieldFuzzy, FieldGeoDistance,
FieldMatchAll, FieldESMatchNone, FieldBool]]
```

Must queries

```
must_not: List[Union[FieldRange, FieldTerm, FieldTerms, FieldMatch,
FieldMatchPhrase, FieldExists, FieldWildcard, FieldPrefix, FieldFuzzy,
FieldGeoDistance, FieldMatchAll, FieldESMatchNone, FieldBool]]
```

Must not queries

```
should: List[Union[FieldRange, FieldTerm, FieldTerms, FieldMatch, FieldMatchPhrase,
FieldExists, FieldWildcard, FieldPrefix, FieldFuzzy, FieldGeoDistance,
FieldMatchAll, FieldESMatchNone, FieldBool]]
```

Should queries

```
class esorm.query.ESExists(*args, **kwargs)
```

Bases: dict

Represents an exists query to check if a field exists.

```
field: str
```

The field to check.

```
esorm.query.ESFilter
```

Represents filter queries in Elasticsearch

```
alias of List[Union[FieldRange, FieldTerm, FieldTerms, FieldMatch, FieldMatchPhrase,
FieldExists, FieldWildcard, FieldPrefix, FieldFuzzy, FieldGeoDistance, FieldMatchAll,
FieldESMatchNone, FieldBool]]
```

```
class esorm.query.ESFuzzy(*args, **kwargs)
```

Bases: dict

Represents a fuzzy query for approximate matching in Elasticsearch.

```
boost: float
```

Optional boosting value for the query

```
fuzziness: Union[int, str]
```

Fuzziness value for the query

```
max_expansions: int
```

Maximum number of expansions for the query

```
prefix_length: int
```

Prefix length for the query

```
transpositions: bool
```

Whether to allow transpositions for the query

```
value: str
```

The value to search for.

```
class esorm.query.ESGeoDistance(*args, **kwargs)
```

Bases: dict

Represents a geo\_distance query for distance-based geospatial queries in Elasticsearch.

**distance:** Union[str, float]

The distance to search for.

**distance\_type:** str

The distance type to use for the query.

**location:** Union[Dict[str, float], str]

The location to search from.

**location\_field:** str

The field containing the location to search from.

**validation\_method:** str

The validation method to use for the query.

**class** esorm.query.ESMatch(\*args, \*\*kwargs)

Bases: dict

Represents the parameters for a match query in Elasticsearch.

**analyzer:** str

Optional analyzer to use for the query.

**boost:** Union[int, float]

Optional boosting value for the query.

**fuzziness:** Union[int, str]

Optional fuzziness value for the query.

**max\_expansions:** int

Optional maximum number of expansions for the query.

**operator:** str

The operator to use for the query.

**prefix\_length:** int

Optional prefix length for the query.

**query:** Union[str, int, float]

The value to search for.

**zero\_terms\_query:** str

Optional zero terms query for the query.

**class** esorm.query.ESMatchAll(\*args, \*\*kwargs)

Bases: dict

Represents a match\_all query for matching all documents in Elasticsearch.

**boost:** float

Optional boosting value for the query

**class** esorm.query.ESMatchNone(\*args, \*\*kwargs)

Bases: dict

Represents a match\_none query for matching no documents in Elasticsearch.

```
class esorm.query.ESMatchPhrase(*args, **kwargs)
```

Bases: dict

Represents the parameters for a match\_phrase query in Elasticsearch.

**analyzer:** str

Optional analyzer to use for the query.

**boost:** Union[int, float]

Optional boosting value for the query.

**query:** str

The value to search for.

**slop:** int

Optional slop value for the query.

```
esorm.query.ESMust
```

Represents must queries in Elasticsearch

alias of List[Union[FieldRange, FieldTerm, FieldTerms, FieldMatch, FieldMatchPhrase, FieldExists, FieldWildcard, FieldPrefix, FieldFuzzy, FieldGeoDistance, FieldMatchAll, FieldESMatchNone, FieldBool]]

```
esorm.query.ESMustNot
```

Represents must\_not queries in Elasticsearch

alias of List[Union[FieldRange, FieldTerm, FieldTerms, FieldMatch, FieldMatchPhrase, FieldExists, FieldWildcard, FieldPrefix, FieldFuzzy, FieldGeoDistance, FieldMatchAll, FieldESMatchNone, FieldBool]]

```
class esorm.query.ESPrefix(*args, **kwargs)
```

Bases: dict

Represents a prefix query for prefix matching in Elasticsearch.

**boost:** float

Optional boosting value for the query

**rewrite:** str

Optional, method used to rewrite the query (e.g., “constant\_score”, “scoring\_boolean”)

**value:** str

The prefix to search for.

```
class esorm.query.ESQuery(*args, **kwargs)
```

Bases: dict

Elasticsearch query structure

**aggs:** Dict[str, ESagg]

Aggregations query structure

**bool:** ESBool

Bool query structure

**exists:** ESExists

Exists query structure

**fuzzy:** Dict[str, *ESFuzzy*]

Fuzzy query structure

**geo\_distance:** Dict[str, *ESGeoDistance*]

Geo distance query structure

**match:** Dict[str, *ESMatch*]

Match query structure

**match\_all:** *ESMatchAll*

Match all query structure

**match\_none:** *ESMatchNone*

Match none query structure

**match\_phrase:** Dict[str, *ESMatchPhrase*]

Match phrase query structure

**prefix:** Dict[str, *ESPrefix*]

Prefix query structure

**term:** Dict[str, *ESTerm*]

Term query structure

**wildcard:** Dict[str, *ESWildcard*]

Wildcard query structure

**class** esorm.query.ESRange(\*args, \*\*kwargs)

Bases: dict

Range query structure

**gt:** Union[int, float, str]

Greater than

**gte:** Union[int, float, str]

Greater than or equal

**lt:** Union[int, float, str]

Less than

**lte:** Union[int, float, str]

Less than or equal

esorm.query.ESShould

Represents should queries in Elasticsearch

alias of List[Union[*FieldRange*, *FieldTerm*, *FieldTerms*, *FieldMatch*, *FieldMatchPhrase*, *FieldExists*, *FieldWildcard*, *FieldPrefix*, *FieldFuzzy*, *FieldGeoDistance*, *FieldMatchAll*, *FieldESMatchNone*, *FieldBool*]]

**class** esorm.query.ESTerm(\*args, \*\*kwargs)

Bases: dict

Represents the parameters for a term query in Elasticsearch.

**boost:** Union[int, float]

Optional boosting value for the query.

**value:** Union[str, int, float]

The value to search for.

**class** esorm.query.**ESWildcard**(\*args, \*\*kwargs)

Bases: dict

Represents a wildcard query for pattern matching in Elasticsearch.

**boost:** float

Optional boosting value for the query

**case\_insensitive:** bool

Optional, whether the query is case insensitive.

**rewrite:** str

Optional, method used to rewrite the query (e.g., “constant\_score”, “scoring\_boolean”)

**value:** str

The pattern to search for. e.g., “te?t” or “test\*”

**class** esorm.query.**FieldBool**(\*args, \*\*kwargs)

Bases: dict

Represents a bool query for combining other queries in Elasticsearch.

**bool:** [ESBool](#)

Bool query structure

**class** esorm.query.**FieldESMatchNone**(\*args, \*\*kwargs)

Bases: dict

Represents a match\_none query for matching no documents in Elasticsearch.

**match\_none:** [ESMatchNone](#)

Match none query structure

**class** esorm.query.**FieldExists**(\*args, \*\*kwargs)

Bases: dict

Represents an exists query to check if a field exists in Elasticsearch.

**exists:** [ESExists](#)

Exists query structure

**class** esorm.query.**FieldFuzzy**(\*args, \*\*kwargs)

Bases: dict

Represents a fuzzy query for approximate matching in Elasticsearch.

**fuzzy:** Dict[str, [ESFuzzy](#)]

Fuzzy query structure

**class** esorm.query.**FieldGeoDistance**(\*args, \*\*kwargs)

Bases: dict

Represents a geo\_distance query for distance-based geospatial queries in Elasticsearch.

**geo\_distance:** Dict[str, [ESGeoDistance](#)]

Geo distance query structure

```
class esorm.query.FieldMatch(*args, **kwargs)
    Bases: dict
    Represents a match query for matching based on the provided text in Elasticsearch.
    match: Dict[str, ESMatch]
        Match query structure
```

```
class esorm.query.FieldMatchAll(*args, **kwargs)
    Bases: dict
    Represents a match_all query for matching all documents in Elasticsearch.
    match_all: ESMatchAll
        Match all query structure
```

```
class esorm.query.FieldMatchPhrase(*args, **kwargs)
    Bases: dict
    Represents a match_phrase query for exact phrase matching in Elasticsearch.
    match_phrase: Dict[str, ESMatchPhrase]
        Match phrase query structure
```

```
class esorm.query.FieldPrefix(*args, **kwargs)
    Bases: dict
    Represents a prefix query for prefix matching in Elasticsearch.
    prefix: Dict[str, ESPrefix]
        Prefix query structure
```

```
class esorm.query.FieldRange(*args, **kwargs)
    Bases: dict
    Range query field
    range: Dict[str, ESRange]
        Range query structure
```

```
class esorm.query.FieldTerm(*args, **kwargs)
    Bases: dict
    Represents a term query for exact value matching in Elasticsearch.
    term: Dict[str, ESTerm]
        Term query structure
```

```
class esorm.query.FieldTerms(*args, **kwargs)
    Bases: dict
    Represents a terms query for exact value matching in Elasticsearch.
    terms: Dict[str, List[Union[str, int, float]]]
        Terms query structure
```

```
class esorm.query.FieldWildcard(*args, **kwargs)
    Bases: dict
    Represents a wildcard query for pattern matching in Elasticsearch.
    wildcard: Dict[str, ESWildcard]
        Wildcard query structure
```



### 3.2.10 esorm.response module

This module contains type definitions for the response from Elasticsearch.

**class** esorm.response.ESResponse(\*args, \*\*kwargs)

Bases: dict

Represents the overall structure of an Elasticsearch response.

**aggregations:** Dict[str, Union[ESAggValueResponse, ESaggTermsResponse, ESaggHistogramResponse]]

The aggregations section of the response.

**hits:** Hits

The hits section of the response.

**timed\_out:** bool

Whether the query timed out.

**took:** int

The time in milliseconds it took to execute the query.

**class** esorm.response.Hit(\*args, \*\*kwargs)

Bases: dict

Represents a single hit (result) from Elasticsearch.

**class** esorm.response.Hits(\*args, \*\*kwargs)

Bases: dict

Represents the hits section of the Elasticsearch response.

**hits:** List[Hit]

List of hits.

**max\_score:** Optional[float]

The maximum score of the hits.

**total:** Dict[str, int]

The total number of hits.

### 3.2.11 esorm.utils module

Utility functions

esorm.utils.camel\_case(snake\_str, capitalize\_first=False)

Convert to camel case

**Parameters**

- **snake\_str** (str) – The string to convert to camel case
- **capitalize\_first** (bool) – Capitalize the first letter

**Returns**

Converted string

`esorm.utils.snake_case(camel_str)`

Convert to snake case

**Parameters**

**camel\_str** (str) – The string to convert to snake case

**Returns**

Converted string

`esorm.utils.utcnow()`

Get current UTC time

**Returns**

Current UTC time

### 3.2.12 `esorm.watcher` module

ElasticSearch Watcher support for ESORM

**class** `esorm.watcher.Action(*args, **kwargs)`

Bases: dict

Action definition

**email:** Dict[str, Any]

**index:** Dict[str, Any]

**logging:** Dict[str, Any]

**pagerduty:** Dict[str, Any]

**slack:** Dict[str, Any]

**throttle\_period:** str

**transform:** Dict[str, [Transform](#)]

**webhook:** [ActionWebhook](#)

**class** `esorm.watcher.ActionWebhook(*args, **kwargs)`

Bases: dict

Action webhook definition

**auth:** Dict[str, Any]

**body:** str

**connection\_timeout:** str

**headers:** Dict[str, Any]

**host:** str

**method:** str

**params:** Dict[str, Any]

**path:** str

```

    port: int
    proxy: Dict[str, Any]
    read_timeout: str
    retries: int
    retry_on_status: List[int]
    scheme: str
    ssl: Dict[str, Any]
    timeout: str
    webhook: Dict[str, Any]
class esorm.watcher.ArrayCompare(*args, **kwargs)
    Bases: dict
    Array compare definition
    eq: Any
    gt: Union[int, float, str, Dict[str, Any]]
    gte: Union[int, float, str, Dict[str, Any]]
    lt: Union[int, float, str, Dict[str, Any]]
    lte: Union[int, float, str, Dict[str, Any]]
    not_eq: Any
    path: str
class esorm.watcher.Body(*args, **kwargs)
    Bases: dict
    Body definition
    query: ESQuery
    size: int
    sort: Dict[str, Order]
class esorm.watcher.Compare(*args, **kwargs)
    Bases: dict
    Compare definition
    eq: Any
    gt: Union[int, float, str, Dict[str, Any]]
    gte: Union[int, float, str, Dict[str, Any]]
    lt: Union[int, float, str, Dict[str, Any]]

```

```
    lte: Union[int, float, str, Dict[str, Any]]

    not_eq: Any

class esorm.watcher.Condition(*args, **kwargs)
    Bases: dict
    Condition definition
    always: EmptyDict
    array_compare: Dict[str, Any]
    compare: Dict[str, Compare]
    never: EmptyDict
    script: Dict[str, Any]

class esorm.watcher.DeleteWatcher
    Bases: Watcher
    Watcher for deleting documents matching a query

class esorm.watcher.EmptyDict(*args, **kwargs)
    Bases: dict
    Empty dict definition

class esorm.watcher.Order(*args, **kwargs)
    Bases: dict
    Order definition
    order: str

class esorm.watcher.Request(*args, **kwargs)
    Bases: dict
    Request definition
    body: ESQuery
    indices: Union[List[str], str]
    template: Dict[str, Any]

class esorm.watcher.Schedule(*args, **kwargs)
    Bases: dict
    Schedule definition
    interval: str

class esorm.watcher.Search(*args, **kwargs)
    Bases: dict
    Search definition
    extract: List[str]
    request: Request
```

```
class esorm.watcher.Transform(*args, **kwargs)
    Bases: dict
    Transform definition
    chain: List[Dict[str, Any]]
    script: Dict[str, Any]
    search: Dict[str, Any]

class esorm.watcher.Trigger(*args, **kwargs)
    Bases: dict
    Trigger definition
    schedule: Schedule

class esorm.watcher.Watcher
    Bases: object
    Watcher definition
    actions: Optional[Dict[str, Action]] = None
    condition: Optional[Condition] = None
    input: Optional[Search] = None
    metadata: Optional[Dict[str, Any]] = None
    to_es()
    trigger: Optional[Trigger] = None

class esorm.watcher.WatcherMeta(name, bases, attrs)
    Bases: type
    Watcher metaclass

async esorm.watcher.setup_watchers(*_, debug=False)
    Setup watchers :param _: Unused :type debug: :param debug: Whether to print the watcher definition
```



## CHANGELOG

### 4.1 v0.4.4

*Released on:* 2024-05-18

Fixed index template

- The default prefix now really “**esorm\_**”
- No more fixed “-” prefix separator
- Now it works as it is in the documentation
- Replaced the deprecated method for creating index templates in newer ESs
- Test for index templates
- Improved tests

### 4.2 v0.4.3

*Released on:* 2024-05-07

Fix: get method is static method, so it is not compatible with model’s `__routing__`

### 4.3 v0.4.2

*Released on:* 2024-05-07

- Fixed get method when id field is an integer.
- Test for integer ids
- Test for property ids

## 4.4 v0.4.1

*Released on:* 2024-05-07

- Retry on conflict decorator to automatically retry conflicted operations
- Simplified ESModelTimestamp save. The logic is simple now: created\_at is filled only if it is a new model instance (created\_at is None).

## 4.5 v0.4.0

*Released on:* 2024-05-07

Support for Optimistic Concurrency Check (OCC)

- It uses \_primary\_term and \_seq\_no for save operations to ensure consistency
- The ES private fields are accessible through the model (\_version, \_primary\_term, \_seq\_no)
- \_id and private fields are stored after save
- Bulk operations uses OCC too
- Bulk operations raise BulkError, which stores all documents which had errors
- New documents of ESModelTimestamp with bulk operations will have created\_at filled automatically
- Reload documents
- Fixed routing for delete operation
- Improved documentation

## 4.6 v0.3.2

*Released on:* 2024-05-03

- Lazy properties are now work with ESBaseModel too
- Lazy properties are now supports nested documents (truly recursive)
- Python 3.8 support back

## 4.7 v0.3.1

*Released on:* 2024-04-29

The most important change is to introduce ESBaseModel.

**Full Changelog:** <https://github.com/wallneradam/esorm/compare/v0.3.0...v0.3.1>



## 4.8 v0.3.0

*Released on:* 2024-01-25

- Replaced deprecated `datetime.utcnow`
- `.all()` method for getting all documents in an index
- Reworked lazy properties, which has recursion protection
- Improved documentation: - pagination and sorting - added `advanced.md`, which contains advanced features documentation - Fixed dark mode colors of reference
- Improved unit tests

## 4.9 v0.2.1

*Released on:* 2024-01-22

Aggregation TypeDicts are refactored.

## 4.10 v0.2.0

*Released on:* 2024-01-18

- Aggregation support
- Improved documentation

## 4.11 v0.1.2

*Released on:* 2024-01-17

Some minor fixes

## 4.12 v0.1.1

*Released on:* 2023-11-01

1st release



## INDICES AND TABLES

- genindex
- modindex
- search



## PYTHON MODULE INDEX

### e

- `esorm`, 27
- `esorm.aggs`, 36
- `esorm.bulk`, 39
- `esorm.error`, 39
- `esorm.esorm`, 40
- `esorm.fastapi`, 40
- `esorm.fields`, 41
- `esorm.logger`, 46
- `esorm.model`, 46
- `esorm.query`, 54
- `esorm.response`, 61
- `esorm.utils`, 61
- `esorm.watcher`, 62



## A

Action (class in *esorm.watcher*), 62  
actions (*esorm.watcher.Watcher* attribute), 65  
ActionWebhook (class in *esorm.watcher*), 62  
aggregate() (*esorm.ESModel* class method), 28  
aggregate() (*esorm.model.ESModel* class method), 47  
aggregations (*esorm.response.ESResponse* attribute), 61  
aggs (*esorm.query.ESQuery* attribute), 57  
all() (*esorm.ESModel* class method), 28  
all() (*esorm.model.ESModel* class method), 47  
always (*esorm.watcher.Condition* attribute), 64  
analyzer (*esorm.query.ESMatch* attribute), 56  
analyzer (*esorm.query.ESMatchPhrase* attribute), 57  
array\_compare (*esorm.watcher.Condition* attribute), 64  
ArrayCompare (class in *esorm.watcher*), 63  
auth (*esorm.watcher.ActionWebhook* attribute), 62  
avg (*esorm.aggs.ESAgg* attribute), 36

## B

Binary (class in *esorm.fields*), 41  
binary (in module *esorm.fields*), 44  
Body (class in *esorm.watcher*), 63  
body (*esorm.watcher.ActionWebhook* attribute), 62  
body (*esorm.watcher.Request* attribute), 64  
bool (*esorm.query.ESQuery* attribute), 57  
bool (*esorm.query.FieldBool* attribute), 59  
boolean (in module *esorm.fields*), 44  
boost (*esorm.query.ESBool* attribute), 54  
boost (*esorm.query.ESFuzzy* attribute), 55  
boost (*esorm.query.ESMatch* attribute), 56  
boost (*esorm.query.ESMatchAll* attribute), 56  
boost (*esorm.query.ESMatchPhrase* attribute), 57  
boost (*esorm.query.ESPrefix* attribute), 57  
boost (*esorm.query.ESTerm* attribute), 58  
boost (*esorm.query.ESWildcard* attribute), 59  
buckets (*esorm.aggs.ESAggHistogramResponse* attribute), 38  
buckets (*esorm.aggs.ESAggTermsResponse* attribute), 38  
BulkError, 39  
BulkOperationError (class in *esorm.error*), 39

Byte (class in *esorm.fields*), 41  
byte (in module *esorm.fields*), 45  
bytes (*esorm.fields.Binary* property), 41

## C

calc\_lazy\_properties() (*esorm.ESBaseModel* method), 27  
calc\_lazy\_properties() (*esorm.model.ESBaseModel* method), 46  
call() (*esorm.ESModel* class method), 29  
call() (*esorm.model.ESModel* class method), 47  
callback (*esorm.model.Pagination* attribute), 52  
callback (*esorm.Pagination* attribute), 35  
camel\_case() (in module *esorm.utils*), 61  
case\_insensitive (*esorm.query.ESWildcard* attribute), 59  
chain (*esorm.watcher.Transform* attribute), 65  
Compare (class in *esorm.watcher*), 63  
compare (*esorm.watcher.Condition* attribute), 64  
Condition (class in *esorm.watcher*), 64  
condition (*esorm.watcher.Watcher* attribute), 65  
connect() (in module *esorm*), 35  
connect() (in module *esorm.esorm*), 40  
connection\_timeout (*esorm.watcher.ActionWebhook* attribute), 62  
create\_index\_template() (in module *esorm.model*), 53  
create\_query\_from\_dict() (*esorm.ESModel* static method), 29  
create\_query\_from\_dict() (*esorm.model.ESModel* static method), 47  
created\_at (*esorm.ESModelTimestamp* attribute), 33  
created\_at (*esorm.model.ESModelTimestamp* attribute), 51

## D

default\_sort (*esorm.ESModel.ESConfig* attribute), 28  
default\_sort (*esorm.model.ESModel.ESConfig* attribute), 46  
delete() (*esorm.bulk.ESBulk* method), 39  
delete() (*esorm.ESBulk* method), 27  
delete() (*esorm.ESModel* method), 29

delete() (*esorm.model.ESModel* method), 48  
DeleteWatcher (*class in esorm.watcher*), 64  
distance (*esorm.query.ESGeoDistance* attribute), 55  
distance\_type (*esorm.query.ESGeoDistance* attribute), 56  
doc\_count (*esorm.aggs.ESAggBucketResponse* attribute), 37  
doc\_count (*esorm.aggs.ESAggHistogramBucketresponse* attribute), 37  
Double (*class in esorm.fields*), 41  
double (*in module esorm.fields*), 45

## E

email (*esorm.watcher.Action* attribute), 62  
EmptyDict (*class in esorm.watcher*), 64  
eq (*esorm.watcher.ArrayCompare* attribute), 63  
eq (*esorm.watcher.Compare* attribute), 63  
ESAgg (*class in esorm.aggs*), 36  
ESAggBucketResponse (*class in esorm.aggs*), 37  
ESAggExtendedBounds (*class in esorm.aggs*), 37  
ESAggFieldParams (*class in esorm.aggs*), 37  
ESAggHistogramBucketresponse (*class in esorm.aggs*), 37  
ESAggHistogramParams (*class in esorm.aggs*), 37  
ESAggHistogramResponse (*class in esorm.aggs*), 38  
ESAggs (*in module esorm.aggs*), 38  
ESAggsResponse (*in module esorm.aggs*), 38  
ESAggTermParams (*class in esorm.aggs*), 38  
ESAggTermsResponse (*class in esorm.aggs*), 38  
ESAggValueResponse (*class in esorm.aggs*), 38  
ESBaseModel (*class in esorm*), 27  
ESBaseModel (*class in esorm.model*), 46  
ESBaseModel.ESConfig (*class in esorm*), 27  
ESBaseModel.ESConfig (*class in esorm.model*), 46  
ESBool (*class in esorm.query*), 54  
ESBulk (*class in esorm*), 27  
ESBulk (*class in esorm.bulk*), 39  
ESExists (*class in esorm.query*), 55  
ESFilter (*in module esorm.query*), 55  
ESFuzzy (*class in esorm.query*), 55  
ESGeoDistance (*class in esorm.query*), 55  
ESMatch (*class in esorm.query*), 56  
ESMatchAll (*class in esorm.query*), 56  
ESMatchNone (*class in esorm.query*), 56  
ESMatchPhrase (*class in esorm.query*), 56  
ESModel (*class in esorm*), 28  
ESModel (*class in esorm.model*), 46  
ESModel.ESConfig (*class in esorm*), 28  
ESModel.ESConfig (*class in esorm.model*), 46  
ESModelTimestamp (*class in esorm*), 33  
ESModelTimestamp (*class in esorm.model*), 51  
ESMust (*in module esorm.query*), 57  
ESMustNot (*in module esorm.query*), 57  
esorm

module, 27  
esorm.aggs  
module, 36  
esorm.bulk  
module, 39  
esorm.error  
module, 39  
esorm.esorm  
module, 40  
esorm.fastapi  
module, 40  
esorm.fields  
module, 41  
esorm.logger  
module, 46  
esorm.model  
module, 46  
esorm.query  
module, 54  
esorm.response  
module, 61  
esorm.utils  
module, 61  
esorm.watcher  
module, 62  
ESPrefix (*class in esorm.query*), 57  
ESQuery (*class in esorm.query*), 57  
ESRange (*class in esorm.query*), 58  
ESResponse (*class in esorm.response*), 61  
ESShould (*in module esorm.query*), 58  
ESTerm (*class in esorm.query*), 58  
ESWildcard (*class in esorm.query*), 59  
exists (*esorm.query.ESQuery* attribute), 57  
exists (*esorm.query.FieldExists* attribute), 59  
extended\_bounds (*esorm.aggs.ESAggHistogramParams* attribute), 37  
extract (*esorm.watcher.Search* attribute), 64

## F

failed\_operations (*esorm.error.BulkError* attribute), 39  
field (*esorm.aggs.ESAggFieldParams* attribute), 37  
field (*esorm.aggs.ESAggHistogramParams* attribute), 38  
field (*esorm.aggs.ESAggTermParams* attribute), 38  
field (*esorm.query.ESExists* attribute), 55  
Field() (*in module esorm*), 34  
Field() (*in module esorm.fields*), 41  
FieldBool (*class in esorm.query*), 59  
FieldESMatchNone (*class in esorm.query*), 59  
FieldExists (*class in esorm.query*), 59  
FieldFuzzy (*class in esorm.query*), 59  
FieldGeoDistance (*class in esorm.query*), 59



FieldMatch (class in *esorm.query*), 59  
 FieldMatchAll (class in *esorm.query*), 60  
 FieldMatchPhrase (class in *esorm.query*), 60  
 FieldPrefix (class in *esorm.query*), 60  
 FieldRange (class in *esorm.query*), 60  
 FieldTerm (class in *esorm.query*), 60  
 FieldTerms (class in *esorm.query*), 60  
 FieldWildcard (class in *esorm.query*), 60  
 filter (*esorm.query.ESBool* attribute), 54  
 Float (class in *esorm.fields*), 42  
 float16 (in module *esorm.fields*), 45  
 float32 (in module *esorm.fields*), 45  
 from\_es() (*esorm.ESModel* class method), 29  
 from\_es() (*esorm.model.ESModel* class method), 48  
 fuzziness (*esorm.query.ESFuzzy* attribute), 55  
 fuzziness (*esorm.query.ESMatch* attribute), 56  
 fuzzy (*esorm.query.ESQuery* attribute), 57  
 fuzzy (*esorm.query.FieldFuzzy* attribute), 59

## G

geo\_distance (*esorm.query.ESQuery* attribute), 58  
 geo\_distance (*esorm.query.FieldGeoDistance* attribute), 59  
 geo\_point (in module *esorm.fields*), 45  
 get() (*esorm.ESModel* class method), 30  
 get() (*esorm.model.ESModel* class method), 48  
 get\_es\_version() (in module *esorm*), 36  
 get\_es\_version() (in module *esorm.esorm*), 40  
 gt (*esorm.query.ESRange* attribute), 58  
 gt (*esorm.watcher.ArrayCompare* attribute), 63  
 gt (*esorm.watcher.Compare* attribute), 63  
 gte (*esorm.query.ESRange* attribute), 58  
 gte (*esorm.watcher.ArrayCompare* attribute), 63  
 gte (*esorm.watcher.Compare* attribute), 63

## H

HalfFloat (class in *esorm.fields*), 42  
 headers (*esorm.watcher.ActionWebhook* attribute), 62  
 histogram (*esorm.aggs.ESAgg* attribute), 36  
 Hit (class in *esorm.response*), 61  
 Hits (class in *esorm.response*), 61  
 hits (*esorm.response.ESResponse* attribute), 61  
 hits (*esorm.response.Hits* attribute), 61  
 host (*esorm.watcher.ActionWebhook* attribute), 62

## I

id\_field (*esorm.ESModel.ESConfig* attribute), 28  
 id\_field (*esorm.model.ESModel.ESConfig* attribute), 46  
 index (*esorm.ESModel.ESConfig* attribute), 28  
 index (*esorm.model.ESModel.ESConfig* attribute), 46  
 index (*esorm.watcher.Action* attribute), 62  
 IndexDoesNotFoundError, 39  
 indices (*esorm.watcher.Request* attribute), 64

input (*esorm.watcher.Watcher* attribute), 65  
 int32 (in module *esorm.fields*), 45  
 Integer (class in *esorm.fields*), 42  
 interval (*esorm.aggs.ESAggHistogramParams* attribute), 38  
 interval (*esorm.watcher.Schedule* attribute), 64  
 InvalidModelError, 34, 39  
 InvalidResponseError, 34, 39

## K

key (*esorm.aggs.ESAggBucketResponse* attribute), 37  
 key (*esorm.aggs.ESAggHistogramBucketresponse* attribute), 37  
 Keyword (class in *esorm.fields*), 42  
 keyword (in module *esorm.fields*), 45

## L

lat (*esorm.fields.LatLon* attribute), 42  
 LatLon (class in *esorm.fields*), 42  
 lazy\_property() (in module *esorm*), 36  
 lazy\_property() (in module *esorm.model*), 53  
 lazy\_property\_max\_recursion\_depth (*esorm.ESBaseModel.ESConfig* attribute), 27  
 lazy\_property\_max\_recursion\_depth (*esorm.ESModel.ESConfig* attribute), 28  
 lazy\_property\_max\_recursion\_depth (*esorm.model.ESBaseModel.ESConfig* attribute), 46  
 lazy\_property\_max\_recursion\_depth (*esorm.model.ESModel.ESConfig* attribute), 46  
 location (*esorm.query.ESGeoDistance* attribute), 56  
 location\_field (*esorm.query.ESGeoDistance* attribute), 56  
 logging (*esorm.watcher.Action* attribute), 62  
 lon (*esorm.fields.LatLon* attribute), 42  
 Long (class in *esorm.fields*), 43  
 long (in module *esorm.fields*), 45  
 lt (*esorm.query.ESRange* attribute), 58  
 lt (*esorm.watcher.ArrayCompare* attribute), 63  
 lt (*esorm.watcher.Compare* attribute), 63  
 lte (*esorm.query.ESRange* attribute), 58  
 lte (*esorm.watcher.ArrayCompare* attribute), 63  
 lte (*esorm.watcher.Compare* attribute), 63

## M

make\_dep\_pagination() (in module *esorm.fastapi*), 40  
 make\_dep\_sort() (in module *esorm.fastapi*), 41  
 match (*esorm.query.ESQuery* attribute), 58  
 match (*esorm.query.FieldMatch* attribute), 60  
 match\_all (*esorm.query.ESQuery* attribute), 58  
 match\_all (*esorm.query.FieldMatchAll* attribute), 60  
 match\_none (*esorm.query.ESQuery* attribute), 58

- `match_none` (*esorm.query.FieldESMatchNone* attribute), 59
  - `match_phrase` (*esorm.query.ESQuery* attribute), 58
  - `match_phrase` (*esorm.query.FieldMatchPhrase* attribute), 60
  - `max` (*esorm.aggs.ESAgg* attribute), 37
  - `max` (*esorm.aggs.ESAggExtendedBounds* attribute), 37
  - `max_expansions` (*esorm.query.ESFuzzy* attribute), 55
  - `max_expansions` (*esorm.query.ESMatch* attribute), 56
  - `max_score` (*esorm.response.Hits* attribute), 61
  - `metadata` (*esorm.watcher.Watcher* attribute), 65
  - `method` (*esorm.watcher.ActionWebhook* attribute), 62
  - `min` (*esorm.aggs.ESAgg* attribute), 37
  - `min` (*esorm.aggs.ESAggExtendedBounds* attribute), 37
  - `min_doc_count` (*esorm.aggs.ESAggHistogramParams* attribute), 38
  - `minimum_should_match` (*esorm.query.ESBool* attribute), 54
  - `model` (*esorm.error.BulkOperationError* attribute), 39
  - `model_computed_fields` (*esorm.ESBaseModel* attribute), 27
  - `model_computed_fields` (*esorm.ESModel* attribute), 30
  - `model_computed_fields` (*esorm.ESModelTimestamp* attribute), 33
  - `model_computed_fields` (*esorm.fields.LatLon* attribute), 42
  - `model_computed_fields` (*esorm.model.ESBaseModel* attribute), 46
  - `model_computed_fields` (*esorm.model.ESModel* attribute), 48
  - `model_computed_fields` (*esorm.model.ESModelTimestamp* attribute), 51
  - `model_computed_fields` (*esorm.model.Pagination* attribute), 52
  - `model_computed_fields` (*esorm.model.Sort* attribute), 53
  - `model_computed_fields` (*esorm.Pagination* attribute), 35
  - `model_computed_fields` (*esorm.Sort* attribute), 35
  - `model_config` (*esorm.ESBaseModel* attribute), 27
  - `model_config` (*esorm.ESModel* attribute), 30
  - `model_config` (*esorm.ESModelTimestamp* attribute), 33
  - `model_config` (*esorm.fields.LatLon* attribute), 42
  - `model_config` (*esorm.model.ESBaseModel* attribute), 46
  - `model_config` (*esorm.model.ESModel* attribute), 48
  - `model_config` (*esorm.model.ESModelTimestamp* attribute), 51
  - `model_config` (*esorm.model.Pagination* attribute), 52
  - `model_config` (*esorm.model.Sort* attribute), 53
  - `model_config` (*esorm.Pagination* attribute), 35
  - `model_config` (*esorm.Sort* attribute), 35
  - `model_fields` (*esorm.ESBaseModel* attribute), 27
  - `model_fields` (*esorm.ESModel* attribute), 30
  - `model_fields` (*esorm.ESModelTimestamp* attribute), 33
  - `model_fields` (*esorm.fields.LatLon* attribute), 42
  - `model_fields` (*esorm.model.ESBaseModel* attribute), 46
  - `model_fields` (*esorm.model.ESModel* attribute), 49
  - `model_fields` (*esorm.model.ESModelTimestamp* attribute), 51
  - `model_fields` (*esorm.model.Pagination* attribute), 52
  - `model_fields` (*esorm.model.Sort* attribute), 53
  - `model_fields` (*esorm.Pagination* attribute), 35
  - `model_fields` (*esorm.Sort* attribute), 35
  - `model_post_init()` (*esorm.ESModel* method), 30
  - `model_post_init()` (*esorm.ESModelTimestamp* method), 33
  - `model_post_init()` (*esorm.model.ESModel* method), 49
  - `model_post_init()` (*esorm.model.ESModelTimestamp* method), 52
  - `modified_at` (*esorm.ESModelTimestamp* attribute), 33
  - `modified_at` (*esorm.model.ESModelTimestamp* attribute), 52
  - `module`
    - `esorm`, 27
    - `esorm.aggs`, 36
    - `esorm.bulk`, 39
    - `esorm.error`, 39
    - `esorm.esorm`, 40
    - `esorm.fastapi`, 40
    - `esorm.fields`, 41
    - `esorm.logger`, 46
    - `esorm.model`, 46
    - `esorm.query`, 54
    - `esorm.response`, 61
    - `esorm.utils`, 61
    - `esorm.watcher`, 62
  - `must` (*esorm.query.ESBool* attribute), 54
  - `must_not` (*esorm.query.ESBool* attribute), 55
- ## N
- `never` (*esorm.watcher.Condition* attribute), 64
  - `not_eq` (*esorm.watcher.ArrayCompare* attribute), 63
  - `not_eq` (*esorm.watcher.Compare* attribute), 64
  - `NotFoundError`, 34, 40
  - `NumericField()` (in module *esorm.fields*), 43
- ## O
- `operator` (*esorm.query.ESMatch* attribute), 56
  - `Order` (class in *esorm.watcher*), 64
  - `order` (*esorm.aggs.ESAggTermParams* attribute), 38
  - `order` (*esorm.watcher.Order* attribute), 64

## P

page (*esorm.model.Pagination* attribute), 53  
 page (*esorm.Pagination* attribute), 35  
 page\_size (*esorm.model.Pagination* attribute), 53  
 page\_size (*esorm.Pagination* attribute), 35  
 pagerduty (*esorm.watcher.ActionWebhook* attribute), 62  
 Pagination (class in *esorm*), 34  
 Pagination (class in *esorm.model*), 52  
 params (*esorm.watcher.ActionWebhook* attribute), 62  
 path (*esorm.watcher.ActionWebhook* attribute), 62  
 path (*esorm.watcher.ArrayCompare* attribute), 63  
 port (*esorm.watcher.ActionWebhook* attribute), 62  
 prefix (*esorm.query.ESQuery* attribute), 58  
 prefix (*esorm.query.FieldPrefix* attribute), 60  
 prefix\_length (*esorm.query.ESFuzzy* attribute), 55  
 prefix\_length (*esorm.query.ESMatch* attribute), 56  
 proxy (*esorm.watcher.ActionWebhook* attribute), 63

## Q

query (*esorm.query.ESMatch* attribute), 56  
 query (*esorm.query.ESMatchPhrase* attribute), 57  
 query (*esorm.watcher.Body* attribute), 63

## R

range (*esorm.query.FieldRange* attribute), 60  
 read\_timeout (*esorm.watcher.ActionWebhook* attribute), 63  
 reason (*esorm.error.BulkOperationError* attribute), 39  
 reload() (*esorm.ESModel* method), 30  
 reload() (*esorm.model.ESModel* method), 49  
 Request (class in *esorm.watcher*), 64  
 request (*esorm.watcher.Search* attribute), 64  
 retries (*esorm.watcher.ActionWebhook* attribute), 63  
 retry\_on\_conflict() (in module *esorm*), 36  
 retry\_on\_conflict() (in module *esorm.model*), 54  
 retry\_on\_status (*esorm.watcher.ActionWebhook* attribute), 63  
 rewrite (*esorm.query.ESPrefix* attribute), 57  
 rewrite (*esorm.query.ESWildcard* attribute), 59

## S

save() (*esorm.bulk.ESBulk* method), 39  
 save() (*esorm.ESBulk* method), 28  
 save() (*esorm.ESModel* method), 31  
 save() (*esorm.ESModelTimestamp* method), 33  
 save() (*esorm.model.ESModel* method), 49  
 save() (*esorm.model.ESModelTimestamp* method), 52  
 Schedule (class in *esorm.watcher*), 64  
 schedule (*esorm.watcher.Trigger* attribute), 65  
 scheme (*esorm.watcher.ActionWebhook* attribute), 63  
 script (*esorm.watcher.Condition* attribute), 64  
 script (*esorm.watcher.Transform* attribute), 65  
 Search (class in *esorm.watcher*), 64

search (*esorm.watcher.Transform* attribute), 65  
 search() (*esorm.ESModel* class method), 31  
 search() (*esorm.model.ESModel* class method), 49  
 search\_by\_fields() (*esorm.ESModel* class method), 31  
 search\_by\_fields() (*esorm.model.ESModel* class method), 50  
 search\_one() (*esorm.ESModel* class method), 32  
 search\_one() (*esorm.model.ESModel* class method), 50  
 search\_one\_by\_fields() (*esorm.ESModel* class method), 32  
 search\_one\_by\_fields() (*esorm.model.ESModel* class method), 50  
 set\_default\_index\_prefix() (in module *esorm.model*), 54  
 set\_max\_lazy\_property\_concurrency() (in module *esorm.model*), 54  
 set\_max\_page\_size() (in module *esorm.fastapi*), 41  
 settings (*esorm.ESModel.ESConfig* attribute), 28  
 settings (*esorm.model.ESModel.ESConfig* attribute), 47  
 setup\_mappings() (in module *esorm*), 36  
 setup\_mappings() (in module *esorm.model*), 54  
 setup\_watchers() (in module *esorm.watcher*), 65  
 Short (class in *esorm.fields*), 44  
 short (in module *esorm.fields*), 45  
 should (*esorm.query.ESBool* attribute), 55  
 size (*esorm.aggs.ESAggTermParams* attribute), 38  
 size (*esorm.watcher.Body* attribute), 63  
 slack (*esorm.watcher.Action* attribute), 62  
 slop (*esorm.query.ESMatchPhrase* attribute), 57  
 snake\_case() (in module *esorm.utils*), 61  
 Sort (class in *esorm*), 35  
 Sort (class in *esorm.model*), 53  
 sort (*esorm.model.Sort* attribute), 53  
 sort (*esorm.Sort* attribute), 35  
 sort (*esorm.watcher.Body* attribute), 63  
 ssl (*esorm.watcher.ActionWebhook* attribute), 63  
 status (*esorm.error.BulkOperationError* attribute), 39  
 sum (*esorm.aggs.ESAgg* attribute), 37

## T

template (*esorm.watcher.Request* attribute), 64  
 term (*esorm.query.ESQuery* attribute), 58  
 term (*esorm.query.FieldTerm* attribute), 60  
 terms (*esorm.aggs.ESAgg* attribute), 37  
 terms (*esorm.query.FieldTerms* attribute), 60  
 Text (class in *esorm.fields*), 44  
 text (in module *esorm.fields*), 45  
 TextField() (in module *esorm.fields*), 44  
 throttle\_period (*esorm.watcher.Action* attribute), 62  
 timed\_out (*esorm.response.ESResponse* attribute), 61  
 timeout (*esorm.watcher.ActionWebhook* attribute), 63  
 to\_es() (*esorm.ESModel* method), 32

`to_es()` (*esorm.model.ESModel* method), 51  
`to_es()` (*esorm.watcher.Watcher* method), 65  
`took` (*esorm.response.ESResponse* attribute), 61  
`total` (*esorm.response.Hits* attribute), 61  
`Transform` (class in *esorm.watcher*), 64  
`transform` (*esorm.watcher.Action* attribute), 62  
`transpositions` (*esorm.query.ESFuzzy* attribute), 55  
`Trigger` (class in *esorm.watcher*), 65  
`trigger` (*esorm.watcher.Watcher* attribute), 65  
`type` (*esorm.error.BulkOperationError* attribute), 39

## U

`update_from_es()` (*esorm.ESModel* method), 33  
`update_from_es()` (*esorm.model.ESModel* method), 51  
`utcnow()` (in module *esorm.utils*), 62

## V

`validate_binary()` (*esorm.fields.Binary* class method), 41  
`validation_method` (*esorm.query.ESGeoDistance* attribute), 56  
`value` (*esorm.aggs.ESAggValueResponse* attribute), 38  
`value` (*esorm.query.ESFuzzy* attribute), 55  
`value` (*esorm.query.ESPrefix* attribute), 57  
`value` (*esorm.query.ESTerm* attribute), 58  
`value` (*esorm.query.ESWildcard* attribute), 59

## W

`Watcher` (class in *esorm.watcher*), 65  
`WatcherMeta` (class in *esorm.watcher*), 65  
`webhook` (*esorm.watcher.Action* attribute), 62  
`webhook` (*esorm.watcher.ActionWebhook* attribute), 63  
`wildcard` (*esorm.query.ESQuery* attribute), 58  
`wildcard` (*esorm.query.FieldWildcard* attribute), 60

## Z

`zero_terms_query` (*esorm.query.ESMatch* attribute), 56