3. API Reference

3.1. Connecting

mongoengine.connect(*db=None*, *alias='default'*, ***kwargs*)

Connect to the database specified by the ‘db’ argument.

Connection settings may be provided here as well if the database is not running on the default port on localhost. If authentication is needed, provide username and password arguments as well.

Multiple databases are supported by using aliases. Provide a separate *alias* to connect to a different instance of **mongod**.

See the docstring for *register_connection* for more details about all supported kwargs.

Changed in version 0.6: - added multiple database support.

mongoengine.register_connection(*alias*, *db=None*, *name=None*, *host=None*, *port=None*, *read_preference=Primary()*, *username=None*, *password=None*, *authentication_source=None*, *authentication_mechanism=None*, ***kwargs*)

Add a connection.

Parameters:

- **alias** – the name that will be used to refer to this connection throughout MongoEngine
- **name** – the name of the specific database to use
- **db** – the name of the database to use, for compatibility with connect
- **host** – the host name of the **mongod** instance to connect to
- **port** – the port that the **mongod** instance is running on
- **read_preference** – The read preference for the collection ** Added pymongo 2.1
- **username** – username to authenticate with
- **password** – password to authenticate with
- **authentication_source** – database to authenticate against
- **authentication_mechanism** – database authentication mechanisms. By default, use SCRAM-SHA-1 with MongoDB 3.0 and later, MONGODB-CR (MongoDB Challenge Response protocol) for older servers.
- **is_mock** – explicitly use mongomock for this connection (can also be done by using *mongomock://* as db host prefix)
- **kwargs** – ad-hoc parameters to be passed into the pymongo driver, for example maxpoolsize, tz_aware, etc. See the documentation for pymongo's *MongoClient* for a full list.

Changed in version 0.10.6: - added mongomock support

3.2. Documents

class `mongoengine.Document`(**args*, ***values*)

The base class used for defining the structure and properties of collections of documents stored in MongoDB. Inherit from this class, and add fields as class attributes to define a document's structure. Individual documents may then be created by making instances of the `Document` subclass.

By default, the MongoDB collection used to store documents created using a `Document` subclass will be the name of the subclass converted to lowercase. A different collection may be specified by providing `collection` to the `meta` dictionary in the class definition.

A `Document` subclass may be itself subclassed, to create a specialised version of the document that will be stored in the same collection. To facilitate this behaviour a *_cls* field is added to documents (hidden though the MongoEngine interface). To enable this behaviourset `allow_inheritance` to `True` in the `meta` dictionary.

A `Document` may use a **Capped Collection** by specifying `max_documents` and `max_size` in the `meta` dictionary. `max_documents` is the maximum number of documents that is allowed to be stored in the collection, and `max_size` is the maximum size of the collection in bytes. `max_size` is rounded

up to the next multiple of 256 by MongoDB internally and mongoengine before. Use also a multiple of 256 to avoid confusions. If `max_size` is not specified and `max_documents` is, `max_size` defaults to 10485760 bytes (10MB).

Indexes may be created by specifying `indexes` in the `meta` dictionary. The value should be a list of field names or tuples of field names. Index direction may be specified by prefixing the field names with a **+** or **-** sign.

Automatic index creation can be disabled by specifying `auto_create_index` in the `meta` dictionary. If this is set to False then indexes will not be created by MongoEngine. This is useful in production systems where index creation is performed as part of a deployment system.

By default, _cls will be added to the start of every index (that doesn't contain a list) if allow_inheritance is True. This can be disabled by either setting cls to False on the specific index or by setting index_cls to False on the meta dictionary for the document.

By default, any extra attribute existing in stored data but not declared in your model will raise a `FieldDoesNotExist` error. This can be disabled by setting `strict` to `False` in the `meta` dictionary.

Initialise a document or embedded document

Parameters:
- **__auto_convert** – Try and will cast python objects to Object types
- **values** – A dictionary of values for the document

objects

A `QuerySet` object that is created lazily on access.

cascade_save(***kwargs*)

Recursively save any references and generic references on the document.

classmethod **compare_indexes**()

Compares the indexes defined in MongoEngine with the ones existing in the database. Returns any missing/extra indexes.

classmethod **create_index**(*keys*, *background=False*, ***kwargs*)

Creates the given indexes if required.

Parameters:
- **keys** – a single index key or a list of index keys (to construct a multi-field index); keys may be prefixed with a **+** or a **-** to determine the index ordering
- **background** – Allows index creation in the background

delete(*signal_kwargs=None*, ***write_concern*)

Delete the `Document` from the database. This will only take effect if the document has been previously saved.

Parameters:
- **signal_kwargs** – (optional) kwargs dictionary to be passed to the signal calls.
- **write_concern** – Extra keyword arguments are passed down which will be used as options for the resultant `getLastError` command. For example, `save(..., w: 2, fsync: True)` will wait until at least two servers have recorded the write and will force an fsync on the primary server.

Changed in version 0.10.7: Add signal_kwargs argument

classmethod **drop_collection**()

Drops the entire collection associated with this `Document` type from the database.

Raises `OperationError` if the document has no collection set (i.e. if it is *abstract*)

Changed in version 0.10.7: `OperationError` exception raised if no collection available

classmethod **ensure_index**(*key_or_list*, *drop_dups=False*, *background=False*, ***kwargs*)

Ensure that the given indexes are in place. Deprecated in favour of create_index.

Parameters:
- **key_or_list** – a single index key or a list of index keys (to construct a multi-field index); keys may be prefixed with a **+** or a **-** to determine the index ordering
- **background** – Allows index creation in the background
- **drop_dups** – Was removed/ignored with MongoDB >2.7.5. The value will be removed if PyMongo3+ is used

classmethod **ensure_indexes**()

Checks the document meta data and ensures all the indexes exist.

Global defaults can be set in the meta - see Defining documents

Note

You can disable automatic index creation by setting *auto_create_index* to False in the documents meta data

classmethod **list_indexes**()

Lists all of the indexes that should be created for given collection. It includes all the indexes from super- and sub-classes.

modify(*query=None*, ***update*)

Perform an atomic update of the document in the database and reload the document object using updated version.

Returns True if the document has been updated or False if the document in the database doesn't match the query.

Note

All unsaved changes that have been made to the document are rejected if the method returns True.

Parameters:
- **query** – the update will be performed only if the document in the database matches the query
- **update** – Django-style update keyword arguments

my_metaclass

alias of `TopLevelDocumentMetaclass`

pk

Get the primary key.

classmethod **register_delete_rule**(*document_cls*, *field_name*, *rule*)

This method registers the delete rules to apply when removing this object.

reload(**fields*, ***kwargs*)

Reloads all attributes from the database.

Parameters:
- **fields** – (optional) args list of fields to reload
- **max_depth** – (optional) depth of dereferencing to follow

New in version 0.1.2.

Changed in version 0.6: Now chainable

Changed in version 0.9: Can provide specific fields to reload

save(*force_insert=False*, *validate=True*, *clean=True*, *write_concern=None*, *cascade=None*, *cascade_kwargs=None*, *_refs=None*, *save_condition=None*, *signal_kwargs=None*, ***kwargs*)

Save the `Document` to the database. If the document already exists, it will be updated, otherwise it will be created.

Parameters:

- **force_insert** – only try to create a new document, don't allow updates of existing documents.
- **validate** – validates the document; set to `False` to skip.
- **clean** – call the document clean method, requires *validate* to be True.
- **write_concern** – Extra keyword arguments are passed down to `save()` OR `insert()` which will be used as options for the resultant `getLastError` command. For example, `save(..., write_concern={w: 2, fsync: True}, ...)` will wait until at least two servers have recorded the write and will force an fsync on the primary server.
- **cascade** – Sets the flag for cascading saves. You can set a default by setting "cascade" in the document __meta__
- **cascade_kwargs** – (optional) kwargs dictionary to be passed throw to cascading saves. Implies `cascade=True`.
- **_refs** – A list of processed references used in cascading saves
- **save_condition** – only perform save if matching record in db satisfies condition(s) (e.g. version number). Raises `OperationError` if the conditions are not satisfied
- **signal_kwargs** – (optional) kwargs dictionary to be passed to the signal calls.

Changed in version 0.5: In existing documents it only saves changed fields using set / unset. Saves are cascaded and any `DBRef` objects that have changes are saved as well.

Changed in version 0.6: Added cascading saves

Changed in version 0.8: Cascade saves are optional and default to False. If you want fine grain control then you can turn off using document meta['cascade'] = True. Also you can pass different kwargs to the cascade save using cascade_kwargs which overwrites the existing kwargs with custom values.

Changed in version 0.8.5: Optional save_condition that only overwrites existing documents if the condition is satisfied in the current db record.

Changed in version 0.10: `OperationError` exception raised if save_condition fails.

Changed in version 0.10.1: :class: save_condition failure now raises a *SaveConditionError*

Changed in version 0.10.7: Add signal_kwargs argument

`select_related`(*max_depth=1*)

Handles dereferencing of `DBRef` objects to a maximum depth in order to cut down the number queries to mongodb.

New in version 0.5.

switch_collection(*collection_name*, *keep_created=True*)

Temporarily switch the collection for a document instance.

Only really useful for archiving off data and calling *save()*:

```
user = User.objects.get(id=user_id)
user.switch_collection('old-users')
user.save()
```

Parameters:
- **collection_name** (*str*) – The database alias to use for saving the document
- **keep_created** (*bool*) – keep self._created value after switching collection, else is reset to True

See also

Use `switch_db` if you need to read from another database

switch_db(*db_alias*, *keep_created=True*)

Temporarily switch the database for a document instance.

Only really useful for archiving off data and calling *save()*:

```
user = User.objects.get(id=user_id)
user.switch_db('archive-db')
user.save()
```

Parameters:
- **db_alias** (*str*) – The database alias to use for saving the document
- **keep_created** (*bool*) – keep self._created value after switching db, else is reset to True

See also

Use `switch_collection` if you need to read from another collection

to_dbref()

Returns an instance of `DBRef` useful in *__raw__* queries.

update(***kwargs*)

Performs an update on the `Document` A convenience wrapper to `update()`.

Raises `OperationError` if called on an object that has not yet been saved.

class **mongoengine.EmbeddedDocument**(**args*, ***kwargs*)

A `Document` that isn't stored in its own collection. `EmbeddedDocument`s should be used as fields on `Document`s through the `EmbeddedDocumentField` field type.

A `EmbeddedDocument` subclass may be itself subclassed, to create a specialised version of the embedded document that will be stored in the same collection. To facilitate this behaviour a *_cls* field is added to documents (hidden though the MongoEngine interface). To enable this behaviour set `allow_inheritance` to `True` in the `meta` dictionary.

my_metaclass

alias of `DocumentMetaclass`

class **mongoengine.DynamicDocument**(**args*, ***values*)

A Dynamic Document class allowing flexible, expandable and uncontrolled schemas. As a `Document` subclass, acts in the same way as an ordinary document but has expanded style properties. Any data passed or set against the `DynamicDocument` that is not a field is automatically converted into a `DynamicField` and data can be attributed to that field.

Note

There is one caveat on Dynamic Documents: fields cannot start with _

Initialise a document or embedded document

Parameters:

- **__auto_convert** – Try and will cast python objects to Object types
- **values** – A dictionary of values for the document

my_metaclass

alias of `TopLevelDocumentMetaclass`

class **mongoengine.DynamicEmbeddedDocument**(**args*, ***kwargs*)

A Dynamic Embedded Document class allowing flexible, expandable and uncontrolled schemas. See `DynamicDocument` for more information about dynamic documents.

my_metaclass

alias of `DocumentMetaclass`

 class **mongoengine.document.MapReduceDocument**(*document*, *collection*, *key*, *value*)

A document returned from a map/reduce query.

Parameters:
- **collection** – An instance of `Collection`
- **key** – Document/result key, often an instance of `ObjectId`. If supplied as an `ObjectId` found in the given `collection`, the object can be accessed via the `object` property.
- **value** – The result(s) for this key.

New in version 0.3.

object

Lazy-load the object referenced by `self.key`. `self.key` should be the `primary_key`.

---

class **mongoengine.ValidationError**(*message=''*, ***kwargs*)

Validation exception.

May represent an error validating a field or a document containing fields with validation errors.

Variables: **errors** – A dictionary of errors for fields within this document or list, or None if the error is for an individual field.

to_dict()

Returns a dictionary of all errors within a document

Keys are field names or list indices and values are the validation error messages, or a nested dictionary of errors for an embedded document or list.

---

class **mongoengine.FieldDoesNotExist**

Raised when trying to set a field not declared in a `Document` or an `EmbeddedDocument`.

To avoid this behavior on data loading, you should set the `strict` to `False` in the `meta` dictionary.

3.3. Context Managers

---

class **mongoengine.context_managers.switch_db**(*cls*, *db_alias*)

switch_db alias context manager.

Example

```
# Register connections
register_connection('default', 'mongoenginetest')
register_connection('testdb-1', 'mongoenginetest2')

class Group(Document):
    name = StringField()

Group(name='test').save()  # Saves in the default db

with switch_db(Group, 'testdb-1') as Group:
    Group(name='hello testdb!').save()  # Saves in testdb-1
```

Construct the switch_db context manager

Parameters:
- **cls** – the class to change the registered db
- **db_alias** – the name of the specific database to use

class mongoengine.context_managers.switch_collection(*cls*, *collection_name*)

switch_collection alias context manager.

Example

```
class Group(Document):
    name = StringField()

Group(name='test').save()  # Saves in the default db

with switch_collection(Group, 'group1') as Group:
    Group(name='hello testdb!').save()  # Saves in group1 collection
```

Construct the switch_collection context manager.

Parameters:
- **cls** – the class to change the registered db
- **collection_name** – the name of the collection to use

class mongoengine.context_managers.no_dereference(*cls*)

no_dereference context manager.

Turns off all dereferencing in Documents for the duration of the context manager:

```
with no_dereference(Group) as Group:
    Group.objects.find()
```

Construct the no_dereference context manager.

Parameters: **cls** – the class to turn dereferencing off on

class **mongoengine.context_managers.query_counter**

Query_counter context manager to get the number of queries. This works by updating the *profiling_level* of the database so that all queries get logged, resetting the db.system.profile collection at the beginnig of the context and counting the new entries.

This was designed for debugging purpose. In fact it is a global counter so queries issued by other threads/processes can interfere with it

Be aware that: - Iterating over large amount of documents (>101) makes pymongo issue *getmore* queries to fetch the next batch of

documents (https://docs.mongodb.com/manual/tutorial/iterate-a-cursor/#cursor-batches)

- Some queries are ignored by default by the counter (killcursors, db.system.indexes)

Construct the query_counter

3.4. Querying

class **mongoengine.queryset.QuerySet**(*document*, *collection*)

The default queryset, that builds queries and handles a set of results returned from a query.

Wraps a MongoDB cursor, providing `Document` objects as the results.

__call__(*q_obj=None*, *class_check=True*, *read_preference=None*, ***query*)

Filter the selected documents by calling the `QuerySet` with a query.

Parameters:

- **q_obj** – a `Q` object to be used in the query; the `QuerySet` is filtered multiple times with different `Q` objects, only the last one will be used
- **class_check** – If set to False bypass class name check when querying collection
- **read_preference** – if set, overrides connection-level read_preference from *ReplicaSetConnection*.
- **query** – Django-style query keyword arguments

aggregate(**pipeline*, ***kwargs*)

Perform a aggregate function based in your queryset params :param pipeline: list of aggregation commands, see: http://docs.mongodb.org/manual/core/aggregation-pipeline/

New in version 0.9.

all()

Returns a copy of the current QuerySet.

all_fields()

Include all fields. Reset all previously calls of .only() or .exclude().

```
post = BlogPost.objects.exclude('comments').all_fields()
```

New in version 0.5.

as_pymongo()

Instead of returning Document instances, return raw values from pymongo.

This method is particularly useful if you don't need dereferencing and care primarily about the speed of data retrieval.

average(*field*)

Average over the values of the specified field.

Parameters: **field** – the field to average over; use dot notation to refer to embedded document fields

batch_size(*size*)

Limit the number of documents returned in a single batch (each batch requires a round trip to the server).

See http://api.mongodb.com/python/current/api/pymongo/cursor.html#pymongo.cursor.Cursor.batch_siz for details.

Parameters: **size** – desired size of each batch.

clone()

Create a copy of the current queryset.

comment(*text*)

Add a comment to the query.

See https://docs.mongodb.com/manual/reference/method/cursor.comment/#cursor.comment for details.

count(*with_limit_and_skip=False*)

Count the selected elements in the query.

Parameters: **(optional)** (*with_limit_and_skip*) – take any `limit()` or `skip()` that has been applied to this cursor into account when getting the count

create(***kwargs*)

Create new object. Returns the saved object instance.

New in version 0.4.

delete(*write_concern=None*, *_from_doc_delete=False*, *cascade_refs=None*)

Delete the documents matched by the query.

Parameters:

- **write_concern** – Extra keyword arguments are passed down which will be used as options for the resultant `getLastError` command. For example, `save(..., write_concern={w: 2, fsync: True}, ...)` will wait until at least two servers have recorded the write and will force an fsync on the primary server.
- **_from_doc_delete** – True when called from document delete therefore signals will have been triggered so don't loop.

:returns number of deleted documents

distinct(*field*)

Return a list of distinct values for a given field.

Parameters: **field** – the field to select distinct values from

Note

This is a command and won't take ordering or limit into account.

New in version 0.4.

Changed in version 0.5: - Fixed handling references

Changed in version 0.6: - Improved db_field refrence handling

ensure_index(***kwargs*)

Deprecated use `Document.ensure_index()`

exclude(**fields*)

Opposite to only(), exclude some document's fields.

```
post = BlogPost.objects(...).exclude('comments')
```

Note

exclude() is chainable and will perform a union :: So with the following it will exclude both: *title* and *author.name*:

```
post = BlogPost.objects.exclude('title').exclude('author.name')
```

`all_fields()` will reset any field filters.

Parameters: **fields** – fields to exclude

New in version 0.5.

exec_js(*code*, **fields*, ***options*)

Execute a Javascript function on the server. A list of fields may be provided, which will be translated to their correct names and supplied as the arguments to the function. A few extra variables are added to the function's scope: `collection`, which is the name of the collection in use; `query`, which is an object representing the current query; and `options`, which is an object containing any options specified as keyword arguments.

As fields in MongoEngine may use different names in the database (set using the `db_field` keyword argument to a `Field` constructor), a mechanism exists for replacing MongoEngine field names with the database field names in Javascript code. When accessing a field, use square-bracket notation, and prefix the MongoEngine field name with a tilde (~).

Parameters:
- **code** – a string of Javascript code to execute
- **fields** – fields that you will be using in your function, which will be passed in to your function as arguments
- **options** – options that you want available to the function (accessed in Javascript through the `options` object)

explain(*format=False*)

Return an explain plan record for the `QuerySet`'s cursor.

Parameters: **format** – format the plan before returning it

fields(*_only_called=False*, ***kwargs*)

Manipulate how you load this document's fields. Used by *.only()* and *.exclude()* to manipulate which fields to retrieve. If called directly, use a set of kwargs similar to the MongoDB projection document. For example:

Include only a subset of fields:

```
posts = BlogPost.objects(...).fields(author=1, title=1)
```

Exclude a specific field:

```
posts = BlogPost.objects(...).fields(comments=0)
```

To retrieve a subrange of array elements:

```
posts = BlogPost.objects(...).fields(slice__comments=5)
```

Parameters: **kwargs** – A set of keyword arguments identifying what to include, exclude, or slice.

New in version 0.5.

filter(**q_objs*, ***query*)

An alias of `__call__()`

first()

Retrieve the first object matching the query.

from_json(*json_data*)

Converts json data to unsaved objects

get(**q_objs*, ***query*)

Retrieve the the matching object raising `MultipleObjectsReturned` or *DocumentName.MultipleObjectsReturned* exception if multiple results and `DoesNotExist` or *DocumentName.DoesNotExist* if no results are found.

New in version 0.3.

hint(*index=None*)

Added 'hint' support, telling Mongo the proper index to use for the query.

Judicious use of hints can greatly improve query performance. When doing a query on multiple fields (at least one of which is indexed) pass the indexed field as a hint to the query.

Hinting will not do anything if the corresponding index does not exist. The last hint applied to this cursor takes precedence over all others.

New in version 0.5.

in_bulk(*object_ids*)

Retrieve a set of documents by their ids.

Parameters: **object_ids** – a list or tuple of `ObjectId` s

Return type: dict of ObjectIds as keys and collection-specific Document subclasses as values.

New in version 0.3.

insert(*doc_or_docs*, *load_bulk=True*, *write_concern=None*, *signal_kwargs=None*)

bulk insert documents

Parameters:

- **doc_or_docs** – a document or list of documents to be inserted
- **(optional)** (*load_bulk*) – If True returns the list of document instances
- **write_concern** – Extra keyword arguments are passed down to `insert()` which will be used as options for the resultant `getLastError` command. For example, `insert(..., {w: 2, fsync: True})` will wait until at least two servers have recorded the write and will force an fsync on each server being written to.

Parm signal_kwargs:

(optional) kwargs dictionary to be passed to the signal calls.

By default returns document instances, set `load_bulk` to False to return just `ObjectIds`

New in version 0.5.

Changed in version 0.10.7: Add signal_kwargs argument

item_frequencies(*field*, *normalize=False*, *map_reduce=True*)

Returns a dictionary of all items present in a field across the whole queried set of documents, and their corresponding frequency. This is useful for generating tag clouds, or searching documents.

Note

Can only do direct simple mappings and cannot map across `ReferenceField` or `GenericReferenceField` for more complex counting a manual map reduce call is required.

If the field is a `ListField`, the items within each list will be counted individually.

Parameters:
- **field** – the field to use
- **normalize** – normalize the results so they add to 1.0
- **map_reduce** – Use map_reduce over exec_js

Changed in version 0.5: defaults to map_reduce and can handle embedded document lookups

`limit`(*n*)

Limit the number of returned documents to *n*. This may also be achieved using array-slicing syntax (e.g. `User.objects[:5]`).

Parameters: **n** – the maximum number of objects to return if n is greater than 0.

When 0 is passed, returns all the documents in the cursor

`map_reduce`(*map_f*, *reduce_f*, *output*, *finalize_f=None*, *limit=None*, *scope=None*)

Perform a map/reduce query using the current query spec and ordering. While `map_reduce` respects `QuerySet` chaining, it must be the last call made, as it does not return a maleable `QuerySet`.

See the `test_map_reduce()` and `test_map_advanced()` tests in `tests.queryset.QuerySetTest` for usage examples.

Parameters:
- **map_f** – map function, as `Code` or string
- **reduce_f** – reduce function, as `Code` or string
- **output** – output collection name, if set to 'inline' will try to use `inline_map_reduce` This can also be a dictionary containing output options see: http://docs.mongodb.org/manual/reference/command/mapReduce/#dbcmd.mapReduce
- **finalize_f** – finalize function, an optional function that performs any post-reduction processing.
- **scope** – values to insert into map/reduce global scope. Optional.
- **limit** – number of objects from current query to provide to map/reduce method

Returns an iterator yielding `MapReduceDocument`.

Note

Map/Reduce changed in server version **>= 1.7.4**. The PyMongo `map_reduce()` helper requires PyMongo version **>= 1.11**.

Changed in version 0.5: - removed `keep_temp` keyword argument, which was only relevant for MongoDB server versions older than 1.7.4

New in version 0.3.

max_time_ms(*ms*)

Wait *ms* milliseconds before killing the query on the server

Parameters: **ms** – the number of milliseconds before killing the query on the server

modify(*upsert=False*, *full_response=False*, *remove=False*, *new=False*, ***update*)

Update and return the updated document.

Returns either the document before or after modification based on *new* parameter. If no documents match the query and *upsert* is false, returns `None`. If upserting and *new* is false, returns `None`.

If the full_response parameter is `True`, the return value will be the entire response object from the server, including the 'ok' and 'lastErrorObject' fields, rather than just the modified document. This is useful mainly because the 'lastErrorObject' document holds information about the command's execution.

Parameters:

- **upsert** – insert if document doesn't exist (default `False`)
- **full_response** – return the entire response object from the server (default `False`, not available for PyMongo 3+)
- **remove** – remove rather than updating (default `False`)
- **new** – return updated rather than original document (default `False`)
- **update** – Django-style update keyword arguments

New in version 0.9.

next()

Wrap the result in a `Document` object.

no_cache()

Convert to a non-caching queryset

New in version 0.8.3: Convert to non caching queryset

no_dereference()

Turn off any dereferencing for the results of this queryset.

no_sub_classes()

Only return instances of this document and not any inherited documents

none()

Helper that just returns a list

only(**fields*)

Load only a subset of this document's fields.

```
post = BlogPost.objects(...).only('title', 'author.name')
```

Note

only() is chainable and will perform a union :: So with the following it will fetch both: *title* and *author.name*:

```
post = BlogPost.objects.only('title').only('author.name')
```

`all_fields()` will reset any field filters.

Parameters: **fields** – fields to include

New in version 0.3.

Changed in version 0.5: - Added subfield support

order_by(**keys*)

Order the `QuerySet` by the keys. The order may be specified by prepending each of the keys by a + or a -. Ascending order is assumed. If no keys are passed, existing ordering is cleared instead.

Parameters: **keys** – fields to order the query results by; keys may be prefixed with **+** or **-** to determine the ordering direction

read_preference(*read_preference*)

Change the read_preference when querying.

Parameters: **read_preference** – override ReplicaSetConnection-level preference.

rewind()

Rewind the cursor to its unevaluated state.

New in version 0.3.

scalar(**fields*)

Instead of returning Document instances, return either a specific value or a tuple of values in order.

Can be used along with `no_dereference()` to turn off dereferencing.

Note

This effects all results and can be unset by calling `scalar` without arguments. Calls `only` automatically.

Parameters: **fields** – One or more fields to return instead of a Document.

search_text(*text*, *language=None*)

Start a text search, using text indexes. Require: MongoDB server version 2.6+.

Parameters: **language** – The language that determines the list of stop words for the search and the rules for the stemmer and tokenizer. If not specified, the search uses the default language of the index. For supported languages, see *Text Search Languages <http://docs.mongodb.org/manual/reference/text-search-languages/#text-search-languages>*.

select_related(*max_depth=1*)

Handles dereferencing of `DBRef` objects or `ObjectId` a maximum depth in order to cut down the number queries to mongodb.

New in version 0.5.

skip(*n*)

Skip *n* documents before returning the results. This may also be achieved using array-slicing syntax (e.g. `User.objects[5:]`).

Parameters: **n** – the number of objects to skip before returning results

slave_okay(*enabled*)

Enable or disable the slave_okay when querying.

Parameters: **enabled** – whether or not the slave_okay is enabled

Deprecated since version Ignored: with PyMongo 3+

snapshot(*enabled*)

Enable or disable snapshot mode when querying.

Parameters: **enabled** – whether or not snapshot mode is enabled

..versionchanged:: 0.5 - made chainable .. deprecated:: Ignored with PyMongo 3+

sum(*field*)

Sum over the values of the specified field.

Parameters: **field** – the field to sum over; use dot notation to refer to embedded document fields

timeout(*enabled*)

Enable or disable the default mongod timeout when querying.

Parameters: **enabled** – whether or not the timeout is used

..versionchanged:: 0.5 - made chainable

to_json(**args*, ***kwargs*)

Converts a queryset to JSON

update(*upsert=False*, *multi=True*, *write_concern=None*, *full_result=False*, ***update*)

Perform an atomic update on the fields matched by the query.

Parameters:

- **upsert** – insert if document doesn't exist (default `False`)
- **multi** – Update multiple documents.
- **write_concern** – Extra keyword arguments are passed down which will be used as options for the resultant `getLastError` command. For example, `save(..., write_concern={w: 2, fsync: True}, ...)` will wait until at least two servers have recorded the write and will force an fsync on the primary server.
- **full_result** – Return the full result dictionary rather than just the number updated, e.g. return `{'n': 2, 'nModified': 2, 'ok': 1.0, 'updatedExisting': True}`.
- **update** – Django-style update keyword arguments

New in version 0.2.

update_one(*upsert=False*, *write_concern=None*, ***update*)

Perform an atomic update on the fields of the first document matched by the query.

Parameters:

- **upsert** – insert if document doesn't exist (default `False`)
- **write_concern** – Extra keyword arguments are passed down which will be used as options for the resultant `getLastError` command. For example, `save(..., write_concern={w: 2, fsync: True}, ...)` will wait until at least two servers have recorded the write and will force an fsync on the primary server.
- **update** – Django-style update keyword arguments

New in version 0.2.

upsert_one(*write_concern=None*, ***update*)

Overwrite or add the first document matched by the query.

Parameters:

- **write_concern** – Extra keyword arguments are passed down which will be used as options for the resultant `getLastError` command. For example, `save(..., write_concern={w: 2, fsync: True}, ...)` will wait until at least two servers have recorded the write and will force an fsync on the primary server.
- **update** – Django-style update keyword arguments

:returns the new or overwritten document

New in version 0.10.2.

using(*alias*)

This method is for controlling which database the QuerySet will be evaluated against if you are using more than one database.

Parameters: **alias** – The database alias

New in version 0.9.

values_list(**fields*)

An alias for scalar

where(*where_clause*)

Filter `QuerySet` results with a `$where` clause (a Javascript expression). Performs automatic field name substitution like `mongoengine.queryset.Queryset.exec_js()`.

Note

When using this mode of query, the database will call your function, or evaluate your predicate clause, for each object in the collection.

New in version 0.5.

with_id(*object_id*)

Retrieve the object matching the id provided. Uses *object_id* only and raises InvalidQueryError if a filter has been applied. Returns *None* if no document exists with that id.

Parameters: **object_id** – the value for the id of the document to look up

Changed in version 0.6: Raises InvalidQueryError if filter has been set

class **mongoengine.queryset.QuerySetNoCache**(*document*, *collection*)

A non caching QuerySet

__call__(*q_obj=None*, *class_check=True*, *read_preference=None*, ***query*)

Filter the selected documents by calling the `QuerySet` with a query.

Parameters:
- **q_obj** – a `Q` object to be used in the query; the `QuerySet` is filtered multiple times with different `Q` objects, only the last one will be used
- **class_check** – If set to False bypass class name check when querying collection
- **read_preference** – if set, overrides connection-level read_preference from *ReplicaSetConnection*.
- **query** – Django-style query keyword arguments

cache()

Convert to a caching queryset

New in version 0.8.3: Convert to caching queryset

mongoengine.queryset.queryset_manager(*func*)

Decorator that allows you to define custom QuerySet managers on `Document` classes. The manager must be a function that accepts a `Document` class as its first argument, and a `QuerySet` as its second argument. The method function should return a `QuerySet`, probably the same one that was passed in, but modified in some way.

3.5. Fields

class **mongoengine.base.fields.BaseField**(*db_field=None*, *name=None*, *required=False*, *default=None*, *unique=False*, *unique_with=None*, *primary_key=False*, *validation=None*, *choices=None*, *null=False*, *sparse=False*, ***kwargs*)

A base class for fields in a MongoDB document. Instances of this class may be added to subclasses of *Document* to define a document's schema.

Changed in version 0.5: - added verbose and help text

Parameters:

- **db_field** – The database field to store this field in (defaults to the name of the field)
- **name** – Deprecated - use db_field
- **required** – If the field is required. Whether it has to have a value or not. Defaults to False.
- **default** – (optional) The default value for this field if no value has been set (or if the value has been unset). It can be a callable.
- **unique** – Is the field value unique or not. Defaults to False.
- **unique_with** – (optional) The other field this field should be unique with.
- **primary_key** – Mark this field as the primary key. Defaults to False.
- **validation** – (optional) A callable to validate the value of the field. Generally this is deprecated in favour of the *FIELD.validate* method
- **choices** – (optional) The valid choices
- **null** – (optional) If the field value can be null. If no and there is a default value then the default value is set
- **sparse** – (optional) *sparse=True* combined with *unique=True* and *required=False* means that uniqueness won't be enforced for *None* values
- ****kwargs** –
  (optional) Arbitrary indirection-free metadata for this field can be supplied as additional keyword arguments and accessed as attributes of the field. Must not conflict with any existing attributes. Common metadata includes *verbose_name* and *help_text*.

---

class **mongoengine.fields.StringField**(*regex=None, max_length=None, min_length=None, **kwargs*)

A unicode string field.

---

class **mongoengine.fields.URLField**(*url_regex=None, schemes=None, **kwargs*)

A field that validates input as an URL.

New in version 0.3.

---

class **mongoengine.fields.EmailField**(*domain_whitelist=None, allow_utf8_user=False, allow_ip_domain=False, *args, **kwargs*)

A field that validates input as an email address.

New in version 0.4.

Initialize the EmailField.

Args:

domain_whitelist (list) - list of otherwise invalid domain names which you'd like to support.

allow_utf8_user (bool) - if True, the user part of the email address can contain UTF8 characters. False by default.

allow_ip_domain (bool) - if True, the domain part of the email can be a valid IPv4 or IPv6 address.

---

class **mongoengine.fields.IntField**(*min_value=None, max_value=None, **kwargs*)

32-bit integer field.

---

class **mongoengine.fields.LongField**(*min_value=None, max_value=None, **kwargs*)

64-bit integer field.

---

class **mongoengine.fields.FloatField**(*min_value=None, max_value=None, **kwargs*)

Floating point number field.

---

class **mongoengine.fields.DecimalField**(*min_value=None, max_value=None, force_string=False, precision=2, rounding='ROUND_HALF_UP', **kwargs*)

Fixed-point decimal number field. Stores the value as a float by default unless *force_string* is used. If using floats, beware of Decimal to float conversion (potential precision loss)

Changed in version 0.8.

New in version 0.3.

Parameters:

- **min_value** – Validation rule for the minimum acceptable value.
- **max_value** – Validation rule for the maximum acceptable value.
- **force_string** – Store the value as a string (instead of a float). Be aware that this affects query sorting and operation like lte, gte (as string comparison is applied) and some query operator won't work (e.g: inc, dec)
- **precision** – Number of decimal places to store.
- **rounding** –
  The rounding rule from the python decimal library:
  - decimal.ROUND_CEILING (towards Infinity)
  - decimal.ROUND_DOWN (towards zero)
  - decimal.ROUND_FLOOR (towards -Infinity)
  - decimal.ROUND_HALF_DOWN (to nearest with ties going towards zero)
  - decimal.ROUND_HALF_EVEN (to nearest with ties going to nearest even integer)
  - decimal.ROUND_HALF_UP (to nearest with ties going away from zero)
  - decimal.ROUND_UP (away from zero)
  - decimal.ROUND_05UP (away from zero if last digit after rounding towards zero would have been 0 or 5; otherwise towards zero)

  Defaults to: `decimal.ROUND_HALF_UP`

---

class **mongoengine.fields.BooleanField**(*db_field=None, name=None, required=False, default=None, unique=False, unique_with=None, primary_key=False, validation=None, choices=None, null=False, sparse=False, **kwargs*)

Boolean field type.

New in version 0.1.2.

Parameters:

- **db_field** – The database field to store this field in (defaults to the name of the field)
- **name** – Deprecated - use db_field
- **required** – If the field is required. Whether it has to have a value or not. Defaults to False.
- **default** – (optional) The default value for this field if no value has been set (or if the value has been unset). It can be a callable.
- **unique** – Is the field value unique or not. Defaults to False.
- **unique_with** – (optional) The other field this field should be unique with.
- **primary_key** – Mark this field as the primary key. Defaults to False.
- **validation** – (optional) A callable to validate the value of the field. Generally this is deprecated in favour of the *FIELD.validate* method
- **choices** – (optional) The valid choices
- **null** – (optional) If the field value can be null. If no and there is a default value then the default value is set
- **sparse** – (optional) *sparse=True* combined with *unique=True* and *required=False* means that uniqueness won't be enforced for *None* values
- ****kwargs** –
  (optional) Arbitrary indirection-free metadata for this field can be supplied as additional keyword arguments and accessed as attributes of the field. Must not conflict with any existing attributes. Common metadata includes *verbose_name* and *help_text*.

---

class **mongoengine.fields.DateTimeField**(*db_field=None, name=None, required=False, default=None, unique=False, unique_with=None, primary_key=False, validation=None, choices=None, null=False, sparse=False, **kwargs*)

Datetime field.

Uses the python-dateutil library if available alternatively use time.strptime to parse the dates. Note: python-dateutil's parser is fully featured and when installed you can utilise it to convert varying types of date formats into valid python datetime objects.

Note: To default the field to the current datetime, use: DateTimeField(default=datetime.utcnow)

> **Note: Microseconds are rounded to the nearest millisecond.**
>
> Pre UTC microsecond support is effectively broken. Use `ComplexDateTimeField` if you need accurate microsecond support.

Parameters:

- **db_field** – The database field to store this field in (defaults to the name of the field)
- **name** – Deprecated - use db_field
- **required** – If the field is required. Whether it has to have a value or not. Defaults to False.
- **default** – (optional) The default value for this field if no value has been set (or if the value has been unset). It can be a callable.
- **unique** – Is the field value unique or not. Defaults to False.
- **unique_with** – (optional) The other field this field should be unique with.
- **primary_key** – Mark this field as the primary key. Defaults to False.
- **validation** – (optional) A callable to validate the value of the field. Generally this is deprecated in favour of the *FIELD.validate* method
- **choices** – (optional) The valid choices
- **null** – (optional) If the field value can be null. If no and there is a default value then the default value is set
- **sparse** – (optional) *sparse=True* combined with *unique=True* and *required=False* means that uniqueness won't be enforced for *None* values
- ****kwargs** –
  (optional) Arbitrary indirection-free metadata for this field can be supplied as additional keyword arguments and accessed as attributes of the field. Must not conflict with any existing attributes. Common metadata includes *verbose_name* and *help_text*.

---

class **mongoengine.fields.ComplexDateTimeField**(*separator=', ', **kwargs*)

ComplexDateTimeField handles microseconds exactly instead of rounding like DateTimeField does.

Derives from a StringField so you can do *gte* and *lte* filtering by using lexicographical comparison when filtering / sorting strings.

The stored string has the following format:

YYYY,MM,DD,HH,MM,SS,NNNNNN

Where NNNNNN is the number of microseconds of the represented ***datetime***. The , as the separator can be easily modified by passing the *separator* keyword when initializing the field.

Note: To default the field to the current datetime, use: DateTimeField(default=datetime.utcnow)

New in version 0.5.

Parameters: **separator** – Allows to customize the separator used for storage (default `,`)

---

class **mongoengine.fields.EmbeddedDocumentField**(*document_type, **kwargs*)

An embedded document field - with a declared document_type. Only valid values are subclasses of `EmbeddedDocument`.

---

class **mongoengine.fields.GenericEmbeddedDocumentField**(*db_field=None, name=None, required=False, default=None, unique=False, unique_with=None, primary_key=False, validation=None, choices=None, null=False, sparse=False, **kwargs*)

A generic embedded document field - allows any `EmbeddedDocument` to be stored.

Only valid values are subclasses of `EmbeddedDocument`.

> **Note**
>
> You can use the choices param to limit the acceptable EmbeddedDocument types

Parameters:

- **db_field** – The database field to store this field in (defaults to the name of the field)
- **name** – Deprecated - use db_field
- **required** – If the field is required. Whether it has to have a value or not. Defaults to False.
- **default** – (optional) The default value for this field if no value has been set (or if the value has been unset). It can be a callable.
- **unique** – Is the field value unique or not. Defaults to False.
- **unique_with** – (optional) The other field this field should be unique with.
- **primary_key** – Mark this field as the primary key. Defaults to False.
- **validation** – (optional) A callable to validate the value of the field. Generally this is deprecated in favour of the *FIELD.validate* method
- **choices** – (optional) The valid choices
- **null** – (optional) If the field value can be null. If no and there is a default value then the default value is set
- **sparse** – (optional) *sparse=True* combined with *unique=True* and *required=False* means that uniqueness won't be enforced for *None* values
- ****kwargs** –
  (optional) Arbitrary indirection-free metadata for this field can be supplied as additional keyword arguments and accessed as attributes of the field. Must not conflict with any existing attributes. Common metadata includes *verbose_name* and *help_text*.

---

class **mongoengine.fields.DynamicField**(*db_field=None, name=None, required=False, default=None, unique=False, unique_with=None, primary_key=False, validation=None, choices=None, null=False, sparse=False, **kwargs*)

A truly dynamic field type capable of handling different and varying types of data.

Used by `DynamicDocument` to handle dynamic data

Parameters:

- **db_field** – The database field to store this field in (defaults to the name of the field)
- **name** – Deprecated - use db_field
- **required** – If the field is required. Whether it has to have a value or not. Defaults to False.
- **default** – (optional) The default value for this field if no value has been set (or if the value has been unset). It can be a callable.
- **unique** – Is the field value unique or not. Defaults to False.
- **unique_with** – (optional) The other field this field should be unique with.
- **primary_key** – Mark this field as the primary key. Defaults to False.
- **validation** – (optional) A callable to validate the value of the field. Generally this is deprecated in favour of the *FIELD.validate* method
- **choices** – (optional) The valid choices
- **null** – (optional) If the field value can be null. If no and there is a default value then the default value is set
- **sparse** – (optional) *sparse=True* combined with *unique=True* and *required=False* means that uniqueness won't be enforced for *None* values
- ****kwargs** –
  (optional) Arbitrary indirection-free metadata for this field can be supplied as additional keyword arguments and accessed as attributes of the field. Must not conflict with any existing attributes. Common metadata includes *verbose_name* and *help_text*.

---

class **mongoengine.fields.ListField**(*field=None*, ***kwargs*)

A list field that wraps a standard field, allowing multiple instances of the field to be used as a list in the database.

If using with ReferenceFields see: One to Many with ListFields

Note

Required means it cannot be empty - as the default for ListFields is []

---

class **mongoengine.fields.EmbeddedDocumentListField**(*document_type*, ***kwargs*)

A `ListField` designed specially to hold a list of embedded documents to provide additional query helpers.

Note

The only valid list values are subclasses of `EmbeddedDocument`.

New in version 0.9.

Parameters:

- **document_type** – The type of `EmbeddedDocument` the list will hold.
- **kwargs** – Keyword arguments passed directly into the parent `ListField`.

---

class **mongoengine.fields.SortedListField**(*field*, ***kwargs*)

A ListField that sorts the contents of its list before writing to the database in order to ensure that a sorted list is always retrieved.

Warning

There is a potential race condition when handling lists. If you set / save the whole list then other processes trying to save the whole list as well could overwrite changes. The safest way to append to a list is to perform a push operation.

New in version 0.4.

Changed in version 0.6: - added reverse keyword

---

class **mongoengine.fields.DictField**(*field=None*, **args*, ***kwargs*)

A dictionary field that wraps a standard Python dictionary. This is similar to an embedded document, but the structure is not defined.

Note

Required means it cannot be empty - as the default for DictFields is {}

New in version 0.3.

Changed in version 0.5: - Can now handle complex / varying types of data

---

class **mongoengine.fields.MapField**(*field=None*, **args*, ***kwargs*)

A field that maps a name to a specified field type. Similar to a DictField, except the 'value' of each item must match the specified field type.

New in version 0.5.

---

class **mongoengine.fields.ReferenceField**(*document_type*, *dbref=False*, *reverse_delete_rule=0*, ***kwargs*)

A reference to a document that will be automatically dereferenced on access (lazily).

Note this means you will get a database I/O access everytime you access this field. This is necessary because the field returns a `Document` which precise type can depend of the value of the *_cls* field present in the document in database. In short, using this type of field can lead to poor performances (especially if you access this field only to retrieve it *pk* field which is already known before dereference). To solve this you should consider using the `LazyReferenceField`.

Use the *reverse_delete_rule* to handle what should happen if the document the field is referencing is deleted. EmbeddedDocuments, DictFields and MapFields does not support reverse_delete_rule and an *InvalidDocumentError* will be raised if trying to set on one of these Document / Field types.

The options are:

- DO_NOTHING (0) - don't do anything (default).
- NULLIFY (1) - Updates the reference to null.
- CASCADE (2) - Deletes the documents associated with the reference.
- DENY (3) - Prevent the deletion of the reference object.
- PULL (4) - Pull the reference from a `ListField` of references

Alternative syntax for registering delete rules (useful when implementing bi-directional delete rules)

```
class Org(Document):
    owner = ReferenceField('User')

class User(Document):
    org = ReferenceField('Org', reverse_delete_rule=CASCADE)

User.register_delete_rule(Org, 'owner', DENY)
```

Changed in version 0.5: added *reverse_delete_rule*

Initialises the Reference Field.

Parameters:

- **dbref** – Store the reference as `DBRef` or as the `ObjectId`.id .
- **reverse_delete_rule** – Determines what to do when the referring object is deleted

Note

A reference to an abstract document type is always stored as a `DBRef`, regardless of the value of *dbref*.

---

class **mongoengine.fields.LazyReferenceField**(*document_type*, *passthrough=False*, *dbref=False*, *reverse_delete_rule=0*, ***kwargs*)

A really lazy reference to a document. Unlike the `ReferenceField` it will **not** be automatically (lazily) dereferenced on access. Instead, access will return a `LazyReference` class instance, allowing access to *pk* or manual dereference by using `fetch()` method.

New in version 0.15.

Initialises the Reference Field.

Parameters:

- **dbref** – Store the reference as `DBRef` or as the `ObjectId`.id .
- **reverse_delete_rule** – Determines what to do when the referring object is deleted
- **passthrough** – When trying to access unknown fields, the

`LazyReference` instance will automatically call *fetch()* and try to retrive the field on the fetched document. Note this only work getting field (not setting or deleting).

---

class **mongoengine.fields.GenericReferenceField**(**args*, ***kwargs*)

A reference to *any* `Document` subclass that will be automatically dereferenced on access (lazily).

Note this field works the same way as `ReferenceField`, doing database I/O access the first time it is accessed (even if it's to access it `pk` or `id` field). To solve this you should consider using the `GenericLazyReferenceField`.

Note

- Any documents used as a generic reference must be registered in the document registry. Importing the model will automatically register it.
- You can use the choices param to limit the acceptable Document types

New in version 0.3.

---

class **mongoengine.fields.GenericLazyReferenceField**(**args*, ***kwargs*)

A reference to *any* `Document` subclass. Unlike the `GenericReferenceField` it will **not** be automatically (lazily) dereferenced on access. Instead, access will return a `LazyReference` class instance, allowing access to *pk* or manual dereference by using `fetch()` method.

Note

- Any documents used as a generic reference must be registered in the document registry. Importing the model will automatically register it.
- You can use the choices param to limit the acceptable Document types

New in version 0.15.

class **mongoengine.fields.CachedReferenceField**(*document_type*, *fields=None*, *auto_sync=True*, ***kwargs*)

A referencefield with cache fields to purpose pseudo-joins

New in version 0.9.

Initialises the Cached Reference Field.

Parameters:
- **fields** – A list of fields to be cached in document
- **auto_sync** – if True documents are auto updated.

class **mongoengine.fields.BinaryField**(*max_bytes=None*, ***kwargs*)

A binary data field.

class **mongoengine.fields.FileField**(*db_alias='default'*, *collection_name='fs'*, ***kwargs*)

A GridFS storage field.

New in version 0.4.

Changed in version 0.5: added optional size param for read

Changed in version 0.6: added db_alias for multidb support

class **mongoengine.fields.ImageField**(*size=None*, *thumbnail_size=None*, *collection_name='images'*, ***kwargs*)

A Image File storage field.

Parameters:
- **size** – max size to store images, provided as (width, height, force) if larger, it will be automatically resized (ex: size=(800, 600, True))
- **thumbnail_size** – size to generate a thumbnail, provided as (width, height, force)

New in version 0.6.

class **mongoengine.fields.SequenceField**(*collection_name=None*, *db_alias=None*, *sequence_name=None*, *value_decorator=None*, **args*, ***kwargs*)

> **Provides a sequential counter see:**
>
> http://www.mongodb.org/display/DOCS/Object+IDs#ObjectIDs-SequenceNumbers

Note

Although traditional databases often use increasing sequence numbers for primary keys. In MongoDB, the preferred approach is to use Object IDs instead. The concept is that in a very large cluster of machines, it is easier to create an object ID than have global, uniformly increasing sequence numbers.

Parameters:

- **collection_name** – Name of the counter collection (default 'mongoengine.counters')
- **sequence_name** – Name of the sequence in the collection (default 'ClassName.counter')
- **value_decorator** – Any callable to use as a counter (default int)

Use any callable as *value_decorator* to transform calculated counter into any value suitable for your needs, e.g. string or hexadecimal representation of the default integer counter value.

Note

In case the counter is defined in the abstract document, it will be common to all inherited documents and the default sequence name will be the class name of the abstract document.

New in version 0.5.

Changed in version 0.8: added *value_decorator*

---

class **mongoengine.fields.ObjectIdField**(*db_field=None, name=None, required=False, default=None, unique=False, unique_with=None, primary_key=False, validation=None, choices=None, null=False, sparse=False, **kwargs*)

A field wrapper around MongoDB's ObjectIds.

Parameters:

- **db_field** – The database field to store this field in (defaults to the name of the field)
- **name** – Deprecated - use db_field
- **required** – If the field is required. Whether it has to have a value or not. Defaults to False.
- **default** – (optional) The default value for this field if no value has been set (or if the value has been unset). It can be a callable.
- **unique** – Is the field value unique or not. Defaults to False.
- **unique_with** – (optional) The other field this field should be unique with.
- **primary_key** – Mark this field as the primary key. Defaults to False.
- **validation** – (optional) A callable to validate the value of the field. Generally this is deprecated in favour of the *FIELD.validate* method
- **choices** – (optional) The valid choices
- **null** – (optional) If the field value can be null. If no and there is a default value then the default value is set
- **sparse** – (optional) *sparse=True* combined with *unique=True* and *required=False* means that uniqueness won't be enforced for *None* values
- ****kwargs** –
  (optional) Arbitrary indirection-free metadata for this field can be supplied as additional keyword arguments and accessed as attributes of the field. Must not conflict with any existing attributes. Common metadata includes *verbose_name* and *help_text*.

---

class **mongoengine.fields.UUIDField**(*binary=True*, ***kwargs*)

A UUID field.

New in version 0.6.

Store UUID data in the database

Parameters: **binary** – if False store as a string.

Changed in version 0.8.0.

Changed in version 0.6.19.

---

class **mongoengine.fields.GeoPointField**(*db_field=None*, *name=None*, *required=False*, *default=None*, *unique=False*, *unique_with=None*, *primary_key=False*, *validation=None*, *choices=None*, *null=False*, *sparse=False*, ***kwargs*)

A list storing a longitude and latitude coordinate.

Note

this represents a generic point in a 2D plane and a legacy way of representing a geo point. It admits 2d indexes but not "2dsphere" indexes in MongoDB > 2.4 which are more natural for modeling geospatial points. See Geospatial indexes

New in version 0.4.

Parameters:

- **db_field** – The database field to store this field in (defaults to the name of the field)
- **name** – Deprecated - use db_field
- **required** – If the field is required. Whether it has to have a value or not. Defaults to False.
- **default** – (optional) The default value for this field if no value has been set (or if the value has been unset). It can be a callable.
- **unique** – Is the field value unique or not. Defaults to False.
- **unique_with** – (optional) The other field this field should be unique with.
- **primary_key** – Mark this field as the primary key. Defaults to False.
- **validation** – (optional) A callable to validate the value of the field. Generally this is deprecated in favour of the *FIELD.validate* method
- **choices** – (optional) The valid choices
- **null** – (optional) If the field value can be null. If no and there is a default value then the default value is set
- **sparse** – (optional) *sparse=True* combined with *unique=True* and *required=False* means that uniqueness won't be enforced for *None* values
- ****kwargs** –
  (optional) Arbitrary indirection-free metadata for this field can be supplied as additional keyword arguments and accessed as attributes of the field. Must not conflict with any existing attributes. Common metadata includes *verbose_name* and *help_text*.

---

class **mongoengine.fields.PointField**(*auto_index=True*, **args*, ***kwargs*)

A GeoJSON field storing a longitude and latitude coordinate.

The data is represented as:

```
{'type' : 'Point' ,
 'coordinates' : [x, y]}
```

You can either pass a dict with the full information or a list to set the value.

Requires mongodb >= 2.4

New in version 0.8.

Parameters: **auto_index** (*bool*) – Automatically create a '2dsphere' index. Defaults to *True*.

---

class **mongoengine.fields.LineStringField**(*auto_index=True*, **args*, ***kwargs*)

A GeoJSON field storing a line of longitude and latitude coordinates.

The data is represented as:

```
{'type' : 'LineString' ,
 'coordinates' : [[x1, y1], [x1, y1] ... [xn, yn]]}
```

You can either pass a dict with the full information or a list of points.

Requires mongodb >= 2.4

New in version 0.8.

Parameters: **auto_index** (*bool*) – Automatically create a '2dsphere' index. Defaults to *True*.

---

class **mongoengine.fields.PolygonField**(*auto_index=True*, **args*, ***kwargs*)

A GeoJSON field storing a polygon of longitude and latitude coordinates.

The data is represented as:

```
{'type' : 'Polygon' ,
 'coordinates' : [[[x1, y1], [x1, y1] ... [xn, yn]],
                 [[x1, y1], [x1, y1] ... [xn, yn]]}
```

You can either pass a dict with the full information or a list of LineStrings. The first LineString being the outside and the rest being holes.

Requires mongodb >= 2.4

New in version 0.8.

Parameters: **auto_index** (*bool*) – Automatically create a '2dsphere' index. Defaults to *True*.

---

class **mongoengine.fields.MultiPointField**(*auto_index=True*, **args*, ***kwargs*)

A GeoJSON field storing a list of Points.

The data is represented as:

```
{'type' : 'MultiPoint' ,
 'coordinates' : [[x1, y1], [x2, y2]]}
```

You can either pass a dict with the full information or a list to set the value.

Requires mongodb >= 2.6

New in version 0.9.

Parameters: **auto_index** (*bool*) – Automatically create a '2dsphere' index. Defaults to *True*.

---

class **mongoengine.fields.MultiLineStringField**(*auto_index=True*, **args*, ***kwargs*)

A GeoJSON field storing a list of LineStrings.

The data is represented as:

```
{'type' : 'MultiLineString' ,
 'coordinates' : [[[x1, y1], [x1, y1] ... [xn, yn]],
                  [[x1, y1], [x1, y1] ... [xn, yn]]]}
```

You can either pass a dict with the full information or a list of points.

Requires mongodb >= 2.6

New in version 0.9.

Parameters: **auto_index** (*bool*) – Automatically create a '2dsphere' index. Defaults to *True*.

---

class **mongoengine.fields.MultiPolygonField**(*auto_index=True*, **args*, ***kwargs*)

A GeoJSON field storing list of Polygons.

The data is represented as:

```
{'type' : 'MultiPolygon' ,
 'coordinates' : [[
       [[x1, y1], [x1, y1] ... [xn, yn]],
       [[x1, y1], [x1, y1] ... [xn, yn]]
   ], [
       [[x1, y1], [x1, y1] ... [xn, yn]],
       [[x1, y1], [x1, y1] ... [xn, yn]]
   ]
}
```

You can either pass a dict with the full information or a list of Polygons.

Requires mongodb >= 2.6

New in version 0.9.

Parameters: **auto_index** (*bool*) – Automatically create a '2dsphere' index. Defaults to *True*.

---

class **mongoengine.fields.GridFSError**

---

class **mongoengine.fields.GridFSProxy**(*grid_id=None*, *key=None*, *instance=None*, *db_alias='default'*, *collection_name='fs'*)

Proxy object to handle writing and reading of files to and from GridFS

New in version 0.4.

Changed in version 0.5: - added optional size param to read

Changed in version 0.6: - added collection name param

---

class **mongoengine.fields.ImageGridFsProxy**(*grid_id=None, key=None, instance=None, db_alias='default', collection_name='fs'*)

Proxy for ImageField

versionadded: 0.6

---

class **mongoengine.fields.ImproperlyConfigured**

3.6. Embedded Document Querying

New in version 0.9.

Additional queries for Embedded Documents are available when using the `EmbeddedDocumentListField` to store a list of embedded documents.

A list of embedded documents is returned as a special list with the following methods:

---

class **mongoengine.base.datastructures.EmbeddedDocumentList**(*list_items, instance, name*)

count()

The number of embedded documents in the list.

Returns: The length of the list, equivalent to the result of `len()`.

create(***values*)

Creates a new embedded document and saves it to the database.

Note

The embedded document changes are not automatically saved to the database after calling this method.

Parameters: **values** – A dictionary of values for the embedded document.

Returns: The new embedded document instance.

delete()

Deletes the embedded documents from the database.

Note

The embedded document changes are not automatically saved to the database after calling this method.

Returns: The number of entries deleted.

`exclude`(***kwargs*)

Filters the list by excluding embedded documents with the given keyword arguments.

Parameters: **kwargs** – The keyword arguments corresponding to the fields to exclude on. *Multiple arguments are treated as if they are ANDed together.*

Returns: A new `EmbeddedDocumentList` containing the non-matching embedded documents.

Raises `AttributeError` if a given keyword is not a valid field for the embedded document class.

`filter`(***kwargs*)

Filters the list by only including embedded documents with the given keyword arguments.

This method only supports simple comparison (e.g: .filter(name='John Doe')) and does not support operators like __gte, __lte, __icontains like queryset.filter does

Parameters: **kwargs** – The keyword arguments corresponding to the fields to filter on. *Multiple arguments are treated as if they are ANDed together.*

Returns: A new `EmbeddedDocumentList` containing the matching embedded documents.

Raises `AttributeError` if a given keyword is not a valid field for the embedded document class.

`first`()

Return the first embedded document in the list, or `None` if empty.

`get`(***kwargs*)

Retrieves an embedded document determined by the given keyword arguments.

Parameters: **kwargs** – The keyword arguments corresponding to the fields to search on. *Multiple arguments are treated as if they are ANDed together.*

Returns: The embedded document matched by the given keyword arguments.

Raises `DoesNotExist` if the arguments used to query an embedded document returns no results. `MultipleObjectsReturned` if more than one result is returned.

save(**args*, ***kwargs*)

Saves the ancestor document.

Parameters:
- **args** – Arguments passed up to the ancestor Document's save method.
- **kwargs** – Keyword arguments passed up to the ancestor Document's save method.

update(***update*)

Updates the embedded documents with the given replacement values. This function does not support mongoDB update operators such as `inc__`.

Note

The embedded document changes are not automatically saved to the database after calling this method.

Parameters: **update** – A dictionary of update values to apply to each embedded document.

Returns: The number of entries updated.

3.7. Misc

mongoengine.common._import_class(*cls_name*)

Cache mechanism for imports.

Due to complications of circular imports mongoengine needs to do lots of inline imports in functions. This is inefficient as classes are imported repeated throughout the mongoengine code. This is compounded by some recursive functions requiring inline imports.

`mongoengine.common` provides a single point to import all these classes. Circular imports aren't an issue as it dynamically imports the class when first needed. Subsequent calls to the `_import_class()` can then directly retrieve the class from the `mongoengine.common._class_registry_cache`.