Flask-Simon Documentation

Release 0.2.0

Andy Dirnberger

July 31, 2013

CONTENTS

Simon is a library to help make working with MongoDB easier. Flask-Simon was created to make it even easier to use Simon with your Flask applications.

CHAPTER

ONE

INSTALLATION

To install the latest stable version of Flask-Simon:

```
$ pip install Flask-Simon
```

or, if you must:

```
$ easy_install Flask-Simon
```

To install the latest development version:

```
$ git clone git@github.com:dirn/Flask-Simon.git
$ cd Flask-Simon
$ python setup.py install
```

In addition to Flask-Simon, this will also install:

- Flask (0.8 or later)
- PyMongo (2.1 or later)
- Simon

CHAPTER

TWO

QUICKSTART

After installing Flask-Simon, import it where you create your Flask app.

```
from flask import Flask
from flask.ext.simon import Simon

app = Flask(__name__)
Simon(app)
```

`Simon` will establish a connection to the database that will be used as the default database for any `Model` classes that you define.

CHAPTER

THREE

CONFIGURATION

Simon looks for the following in your Flask app's configuration:

MONGO_URI	A MongoDB URI connection string specifying the database connection.
MONGO_HOST	The hostname or IP address of the MongoDB server. default: 'localhost'
MONGO_PORT	The port of the MongoDB server. default: 27017
MONGO_DNAME	The name of the database on MONGO_HOST. Default: app.name
MONGO_USERNAME	The username for authentication.
MONGO_PASSWORD	The password for authentication.
MONGO_REPLICA_SET	The name of the replica set.

The MONGO_URI configuration setting will be used before checking any other settings. If it's not present, the others will be used.

By default, Simon and init_app() will use MONGO as the prefix for all configuration settings. This can be overridden by using the prefix argument.

Specifying a value for prefix will allow for the use of multiple databases.

```
app = Flask(__name__)

app.config['MONGO_URI'] = 'mongodb://localhost/mongo'
app.config['SIMON_URI'] = 'mongodb://localhost/simon'

Simon(app)
Simon(app, prefix='SIMON')
```

This will allow for the use of the mongo and simon databases on localhost. mongo will be available to models through the aliases default and mongo. simon will be available through the alias simon. This alias can be changed by using the alias argument.

```
Simon(app, prefix='SIMON', alias='other-database')
```

CHAPTER

FOUR

ROUTING

Flask-Simon provides a custom converter to allow for the use of Object IDs in URLs.

```
@app.route('/<objectid:id>')
```

More information about converters is available in the Flask API.

CHAPTER

FIVE

API

class flask_simon.**Simon**(*app=None*, *prefix='MONGO'*, *alias=None*)
Automatically creates a connection for Simon models.

init_app(*app*, *prefix='MONGO'*, *alias=None*)
Initializes the Flask app for use with Simon.

This method will automatically be called if the app is passed into `__init__()`.

Parameters

- **app** (`flask.Flask`) – the Flask application.
- **prefix** (*str*) – (optional) the prefix of the config settings
- **alias** (*str*) – (optional) the alias to use for the database connection

Changed in version 0.2.0: Added support for multiple databasesNew in version 0.1.0.

flask_simon.**get_or_404**(*model*, **qs*, ***fields*)
Finds and returns a single document, or raises a 404 exception.

This method will find a single document within the specified model. If the specified query matches zero or multiple documents, a `404 Not Found` exception will be raised.

Parameters

- **model** (`simon.Model`) – the model class.
- ***qs** (`simon.query.Q`) – logical queries.
- ****fields** (*kwargs*) – keyword arguments specifying the query.

Returns `Model` – an instance of a model.

class flask_simon.**Model**(***fields*)
Base class for all Simon models.

exception MultipleDocumentsFound
Raised when more than one document is found.

exception Model.**NoDocumentFound**
Raised when an object matching a query is not found.

classmethod Model.**all**()
Return all documents in the collection.

If `sort` has been defined on the `Meta` class it will be used to order the records.

classmethod Model.**create**(***fields*)
Create a new document and saves it to the database.

This is a convenience method to create a new document. It will instantiate a new Model from the keyword arguments, call save(), and return the instance.

If the model has the required_fields options set, a TypeError will be raised if any of the fields are not provided.

Parameters

- **safe** (*bool.*) – (optional) **DEPRECATED** Use w instead.
- **w** (*int.*) – (optional) The number of servers that must receive the update for it to be successful.
- ****fields** (***kwargs.*) – Keyword arguments to add to the document.

Returns Model – the new document.

Raises TypeError

Model.**delete**(***kwargs*)
Delete a single document from the database.

This will delete the document associated with the instance object. If the document does not have an _id– this will most likely indicate that the document has never been saved– a TypeError will be raised.

Parameters

- **safe** (*bool.*) – (optional) **DEPRECATED** Use w instead.
- **w** (*int.*) – (optional) The number of servers that must receive the update for it to be successful.

Raises TypeError

classmethod Model.**find**(*q=None*, **qs*, ***fields*)
Return multiple documents from the database.

This will find a return multiple documents matching the query specified through **fields. If sort has been defined on the Meta class it will be used to order the records.

Parameters

- **q** (Q.) – (optional) A logical query to use with the query.
- ***qs** (**args.*) – **DEPRECATED** Use q instead.
- ****fields** (***kwargs.*) – Keyword arguments specifying the query.

Returns QuerySet – query set containing objects matching query.

Changed in version 0.3.0: qs is being deprecated in favor of q

classmethod Model.**get**(*q=None*, **qs*, ***fields*)
Return a single document from the database.

This will find and return a single document matching the query specified through **fields. An exception will be raised if any number of documents other than one is found.

Parameters

- **q** (Q.) – (optional) A logical query to use with the query.
- ***qs** (**args.*) – **DEPRECATED** Use q instead.
- ****fields** (***kwargs.*) – Keyword arguments specifying the query.

Returns Model – object matching query.

Raises MultipleDocumentsFound, NoDocumentFound

Changed in version 0.3.0: qs is being deprecated in favor of q

classmethod Model.**get_or_create**(***fields*)
Return an existing or create a new document.

This will find and return a single document matching the query specified through **fields. If no document is found, a new one will be created.

Along with returning the Model instance, a boolean value will also be returned to indicate whether or not the document was created.

Parameters

- **safe** (*bool.*) – (optional) **DEPRECATED** Use w instead.
- **w** (*int.*) – (optional) The number of servers that must receive the update for it to be successful.
- ****fields** (***kwargs.*) – Keyword arguments specifying the query.

Returns tuple – the Model and whether the document was created.

Raises MultipleDocumentsFound

Model.**increment**(*field=None*, *value=1*, ***fields*)
Perform an atomic increment.

This can be used to update a single field:

```
>>> obj.increment(field, value)
```

or to update multiple fields at a time:

```
>>> obj.increment(field1=value1, field2=value2)
```

Note that the latter does **not** set the values of the fields, but rather specifies the values they should be incremented by.

If the document does not have an _id–this will most likely indicate that the document has never been saved– a TypeError will be raised.

If no fields are indicated–either through field or through **fields, a ValueError will be raised.

Parameters

- **field** (*str.*) – (optional) Name of the field to increment.
- **value** (*int.*) – (optional) Value to increment field by.
- **safe** (*bool.*) – (optional) **DEPRECATED** Use w instead.
- **w** (*int.*) – (optional) The number of servers that must receive the update for it to be successful.
- ****fields** (***kwargs.*) – Keyword arguments specifying fields and increment values.

Raises TypeError, ValueError

Model.**pop**(*fields*, ***kwargs*)
Perform an atomic pop.

Values can be popped from either the end or the beginning of a list. To pop a value from the end of a list, specify the name of the field. The pop a value from the beginning of a list, specify the name of the field with a − in front of it.

If the document does not have an `_id`–this will most likely indicate that the document has never been saved– a `TypeError` will be raised.

Parameters

- **fields** (*str, list, or tuple.*) – The names of the fields to pop from.
- **safe** (*bool.*) – (optional) **DEPRECATED** Use `w` instead.
- **w** (*int.*) – (optional) The number of servers that must receive the update for it to be successful.

Raises `TypeError`

New in version 0.5.0.

`Model.`**`pull`**(*field=None*, *value=None*, ***fields*)

Perform an atomic pull.

With MongoDB there are two types of pull operations: `$pull` and `$pullAll`. As the name implies, `$pullAll` is intended to pull all values in a list from the field, while `$pull` is meant for single values.

This method will determine the correct operator(s) to use based on the value(s) being pulled. Updates can consist of either operator alone or both together.

This can be used to update a single field:

```
>>> obj.pull(field, value)
```

or to update multiple fields at a time:

```
>>> obj.pull(field1=value1, field2=value2)
```

If the document does not have an `_id`–this will most likely indicate that the document has never been saved– a `TypeError` will be raised.

If no fields are indicated–either through `field` or through `**fields`, a `ValueError` will be raised.

Parameters

- **field** (*str.*) – (optional) Name of the field to pull from.
- **value** (*scalar or list.*) – (optional) Value to pull from `field`.
- **safe** (*bool.*) – (optional) **DEPRECATED** Use `w` instead.
- **w** (*int.*) – (optional) The number of servers that must receive the update for it to be successful.
- ****fields** (***kwargs.*) – Keyword arguments specifying fields and the values to pull.

Raises `TypeError`, `ValueError`

New in version 0.5.0.

`Model.`**`push`**(*field=None*, *value=None*, *allow_duplicates=True*, ***fields*)

Perform an atomic push.

With MongoDB there are three types of push operations: `$push`, `$pushAll`, add `$addToSet`. As the name implies, `$pushAll` is intended to push all values from a list to the field, while `$push` is meant for single values. `$addToSet` can be used with either type of value, but it will only add a value to the list if it doesn't already contain the value.

This method will determine the correct operator(s) to use based on the value(s) being pushed. Setting `allow_duplicates` to `False` will use `$addToSet` instead of `$push` and `$pushAll`. Updates that allow duplicates can combine `$push` and `$pushAll` together.

This can be used to update a single field:

```
>>> obj.push(field, value)
```

or to update multiple fields at a time:

```
>>> obj.push(field1=value1, field2=value2)
```

If the document does not have an `_id`–this will most likely indicate that the document has never been saved– a `TypeError` will be raised.

If no fields are indicated–either through `field` or through `**fields`, a `ValueError` will be raised.

Parameters

- **field** (*str.*) – (optional) Name of the field to push to.
- **value** (*scalar or list.*) – (optional) Value to push to `field`.
- **allow_duplicates** (*bool.*) – (optional) Whether to allow duplicate values to be added to the list
- **safe** (*bool.*) – (optional) **DEPRECATED** Use `w` instead.
- **w** (*int.*) – (optional) The number of servers that must receive the update for it to be successful.
- ****fields** (***kwargs.*) – Keyword arguments specifying fields and the values to push.

Raises `TypeError`, `ValueError`

New in version 0.5.0.

`Model.`**`raw_update`**(*fields*, ***kwargs*)

Perform an update using a raw document.

This method should be used carefully as it will perform the update exactly, potentially performing a full document replacement.

Also, for simple updates, it is preferred to use the `save()` or `update()` methods as they will usually result in less data being transferred back from the database.

If the document does not have an `_id`–this will most likely indicate that the document has never been saved– a `TypeError` will be raised.

Unlike `save()`, `modified` will not be updated.

Parameters

- **fields** (*dict.*) – The document to save to the database.
- **safe** (*bool.*) – (optional) **DEPRECATED** Use `w` instead.
- **w** (*int.*) – (optional) The number of servers that must receive the update for it to be successful.

Raises `TypeError`

`Model.`**`remove_fields`**(*fields*, ***kwargs*)

Remove the specified fields from the document.

The specified fields will be removed from the document in the database as well as the object. This operation cannot be undone.

If the document does not have an `_id`–this will most likely indicate that the document has never been saved– a `TypeError` will be raised.

Unlike `save()`, `modified` will not be updated.

If the model has the `required_fields` options set, a `TypeError` will be raised if attempting to remove one of the required fields.

Parameters

- **fields** (*str, list, or tuple.*) – The names of the fields to remove.
- **safe** (*bool.*) – (optional) **DEPRECATED** Use `w` instead.
- **w** (*int.*) – (optional) The number of servers that must receive the update for it to be successful.

Raises `TypeError`

`Model.`**`rename`**(*field_from=None*, *field_to=None*, ***fields*)

Perform an atomic rename.

This can be used to update a single field:

```
>>> obj.rename(original, new)
```

or to update multiple fields at a time:

```
>>> obj.increment(original1=new1, original2=new2)
```

Note that the latter does **not** set the values of the fields, but rather specifies the name they should be renamed to.

If the document does not have an `_id`–this will most likely indicate that the document has never been saved– a `TypeError` will be raised.

If no fields are indicated–either through `field_from` and `field_to` or through `**fields`, a `ValueError` will be raised.

Parameters

- **field_from** (*str.*) – (optional) Name of the field to rename.
- **field_to** (*int.*) – (optional) New name for `field_from`.
- **safe** (*bool.*) – (optional) **DEPRECATED** Use `w` instead.
- **w** (*int.*) – (optional) The number of servers that must receive the update for it to be successful.
- ****fields** (***kwargs.*) – Keyword arguments specifying fields and their new names.

Raises `TypeError`, `ValueError`

New in version 0.5.0.

`Model.`**`save`**(***kwargs*)

Save the document to the database.

When saving a new document for a model with `auto_timestamp` set to `True`, `created` will be added with the current datetime in UTC. `modified` will always be set with the current datetime in UTC.

If the model has the `required_fields` options set, a `TypeError` will be raised if any of the fields have not been associated with the instance.

Parameters

- **safe** (*bool.*) – (optional) **DEPRECATED** Use `w` instead.
- **w** (*int.*) – (optional) The number of servers that must receive the update for it to be successful.

Raises `TypeError`

Changed in version 0.4.0: `created` is always added to inserted documents when `auto_timestamp` is `True`

`Model.`**`save_fields`**(*fields*, ***kwargs*)

Save the specified fields.

If only a select number of fields need to be updated, an atomic update is preferred over a document replacement. `save_fields()` takes either a single field name or a list of field names to update.

All of the specified fields must exist or an `AttributeError` will be raised. To add a field to the document with a blank value, make sure to assign it through `object.attribute = ''` or something similar before calling `save_fields()`.

If the document does not have an `_id`–this will most likely indicate that the document has never been saved– a `TypeError` will be raised.

Unlike `save()`, `modified` will not be updated.

Parameters

- **fields** (*str, list, or tuple.*) – The names of the fields to update.
- **safe** (*bool.*) – (optional) **DEPRECATED** Use `w` instead.
- **w** (*int.*) – (optional) The number of servers that must receive the update for it to be successful.

Raises `AttributeError`, `TypeError`

`Model.`**`update`**(***fields*)

Perform an atomic update.

If only a select number of fields need to be updated, an atomic update is preferred over a document replacement. `update()` takes a series of fields and values through its keyword arguments. This fields will be updated both in the database and on the instance.

If the document does not have an `_id`–this will most likely indicate that the document has never been saved– a `TypeError` will be raised.

Unlike `save()`, `modified` will not be updated.

Parameters

- **safe** (*bool.*) – (optional) **DEPRECATED** Use `w` instead.
- **w** (*int.*) – (optional) The number of servers that must receive the update for it to be successful.
- ****fields** (***kwargs.*) – The fields to update.

Raises `TypeError`

class `flask_simon.`**`ObjectIDConverter`**(*map*)

Convert Object IDs for use in view routing URLs.

Full details of how to query using `get_or_404()` can be found in the `Simon API`.

CHAPTER

SIX

FURTHER READING

For more information, check out the Simon docs and the MongoDB docs.

CHAPTER

SEVEN

CHANGELOG

7.1 0.2.0 (2013-02-04)

- Add Object ID URL converter
- Support importing most of Simon's functionality through `flask.ext.simon`
- Add support for multiple databases
- Add support for settings other than `MONGO_URI`
- Require Simon 0.2 or greater

7.2 0.1.0 (2013-01-21)

- Initial release

PYTHON MODULE INDEX