ffmddb Documentation

Release pre-release

Madison Scott-Clary

February 08, 2017

Contents:

A flat-file-with-metadata database. This is a reference implementation for a simple document database idea based on flat-files, each of which contains at least one field (a large text blob, the document field) and potentially many other fields formed of structured data contained in a metadata blob within the file.

In short, it turns files written in a Jekyll fashion into objects in a database. The 'post content' turns into the document field, and the metadata blob turns into other fields. Indices are built and querying becomes possible within the indices (full document querying should rely on something like elasticsearch). The same data and relations are represented, but in a format easily edited in any text editor, easily readible or served from something like Jekyll, and easily stored in a VCS repo. The goal is not speed, but flexibility for manually interfacing with smaller datasets.

CHAPTER 1

Using ffmddb

1.1 The ffmddb shell

1.2 The ffmddb API

CHAPTER 2

ffmddb Configuration

`ffmddb` relies on a single configuration file to figure out how to interact with the database. This file contains a YAML blob, which describes a few things about the structure of the data. It informs the database of where

Configuration file (.ffmddbrc) example:

```
my_log_files:
    version: 1
    collections:
        - name: logs
          path: ./logs
        - name: participants
          path: ./participants
    indices:
        - name: log_tag
          from: ['logs', 'metadata:tag']
        - name: participants_logs
          from: ['participants', 'name:']
          to: ['logs', 'metadata:participants']
    fence: ['<!--ffmddb', '-->']
```

Config entry	Default	Explanation
<Root level key>	N/A	The name of the database.
`collections`	N/A	A list of YAML objects. Each object should containa `name` and a `path` entry. The `name` should contain letters, numbers, and underscores and start with a letter. The `path` should be relative to the configuration file. Can be empty.
`indices`	N/A	A list of YAML objects. Each object should contain a name, a from field, and an optional to field. The name should contain letters, numbers, and underscores and start with a letter. The from and to fields should contain an array with the first item being the name of a collection and the second being a query for selecting one field. Can be empty, cannot contain data if `collections` is empty.
`fence`	`['---+', '---+']`	The fence that delineates the metadata field from the document field. Should be a two-item array, with the two items being strings containing the open fence and the close fence. The strings will be interpreted as regular expressions (the default being an example, specifying both fences as three or more hyphens), so be careful to escape where needed. fences occur on a line by themselves. Multiple metadata blocks may occur in a file; they will be merged before parsing.
`index_path`	`.ffmddb_idx`	The folder relative to the configuration file which contains the indices.
`multiple_metadata`	False	Whether or not to collect metadata from multiple fenced blocks.

some more

CHAPTER 3

The ffmddb Database

3.1 Documents and indices

3.2 File format

Files which will be documents in the database should be textual. They can be of any format, so long as, when read, the fenced metadata may be found. For example, you could have a markdown file with a metadata block:

```
---
layout: post
title: My great document
tags:
    - foxes
    - cats
    - dogs
---

Wow, foxes and dogs and cats are all *really great*!
```

In this instance, you can see that the file itself is an actual Jekyll file.

Fences do not need to be Jekyll style (three or more hyphens), but may be anything, so long as they're specified in the configuration file. For example, you can specify the fence to be an XML comment if you're storing XML-based documents.

In the configuration file:

```
mydb:
    ...
    fence: ['<!--ffmddb', '-->']
```

And in the document:

```
<mydoc>
    <!--ffmddb
    foo: bar
    baz: qux
    -->
    ...
</mydoc>
```

CHAPTER 4

ffmddb

4.1 ffmddb package

4.1.1 Subpackages

ffmddb.core package

Subpackages

ffmddb.core.models package Models, to ffmddb are any thing that maps from a string or file to a python object.

For files, this includes:

- a *document*, which maps to one of the files ffmddb knows about
- a folder acting as a *collection* of documents
- an *index* file

For strings, this includes:

- a *configuration* YAML blob (which may come from a file)
- a json *query* against the database (which may be a python dict)
- a *field* spec

Submodules

ffmddb.core.models.config module

class ffmddb.core.models.config.**Configuration**(*name*, *collections*, *indices*, *options*)

Stores database configuration read from a file or the user.

exception MalformedConfiguration

Bases: exceptions.Exception

classmethod Configuration.**from_object**(*config_obj*)

Parses a configuration object (as generated by loading a yaml configuration file) into an internal object used by the database

Configuration.**marshal**()

marshals the configuration object back to YAML

ffmddb.core.models.document module

class ffmddb.core.models.document.**Collection**(*name*, *path*)
Stores a reference to a collection of documents

marshal()

class ffmddb.core.models.document.**CollectionField**(*collection*, *field*)
Stores an abstract reference to a field which should exist on most/all documents in a collection, used for indexing

marshal()

class ffmddb.core.models.document.**Document**(*db*, *collection*, *name*, *document_field=None*, *metadata=None*)
Stores a reference to a single document

marshal()

ffmddb.core.models.field module

class ffmddb.core.models.field.**Field**(*field_spec*)

exception MalformedSpec
Bases: exceptions.Exception

Field.**marshal**()

classmethod Field.**parse_spec**(*spec*)

Field.**run**(*document*)

ffmddb.core.models.index module

class ffmddb.core.models.index.**CoreIndex**
Bases: *ffmddb.core.models.index.Index*

Represents the core index, which tracks documents and metadata field names, as well as indices

class ffmddb.core.models.index.**CrossCollectionIndex**(*name*, *from_collection_field*, *to_collection_field*)
Bases: *ffmddb.core.models.index.Index*

Represents an index on one field common to a collection which maps to a field on another (or the same) collection

marshal()

class ffmddb.core.models.index.**Index**
Provides an interface of common methods for collection types

read()

write()

class ffmddb.core.models.index.**SingleCollectionIndex**(*name*, *collection_field*)
Bases: *ffmddb.core.models.index.Index*

Represents an index on one field common to a collection

marshal()

ffmddb.core.models.query module

class ffmddb.core.models.query.**Filter**(*filter_obj*)

Stores a single filter for comparing a field to a value

OPERATORS = {'le': <function <lambda> at 0x7fb3eb375140>, 'lt': <function <lambda> at 0x7fb3eb4825f0>, 'gt': <funct

classmethod is_filter(*obj*)

duck-types a dict to see if it looks like a filter object

run(*document*)

runs the test against the document, comparing the metadata field specified by the filter's field against the provided value using the provided operation

class ffmddb.core.models.query.**FilterGroup**(*conjunction*, *filter_list*)

Stores a list of filter objects joined by a conjunction

CONJUNCTIONS = {'and': <function <lambda> at 0x7fb3eb375410>, 'not': <function <lambda> at 0x7fb3eb375500>, 'or

classmethod is_filter_group(*obj*)

duck-types a dict to see if it looks like a filter-group

run(*document*)

runs each specified filter in the group and reduces the results to a single value with the provided conjunction

class ffmddb.core.models.query.**Query**(*query_obj*)

Stores an arbitrarily complex query

exception MalformedQuery

Bases: exceptions.Exception

Query.**run**(*document*)

runs the core filter group which contains all filters and groups in the query

Submodules

ffmddb.core.database module

class ffmddb.core.database.**Database**(*config_obj*, *config_file=None*)

Stores a reference to a database (a configuration file and the files it specifies), providing methods to interact with it

close()

create_collection(*name*, *path*, *mkdir_if_needed=True*, *keep_file=True*)

create_document(*document*)

delete_collection(*name*, *cascade=False*)

delete_document(*document*)

classmethod from_file(*config_file*)

classmethod from_string(*config_str*, *config_file=None*)

get_collection(*name*)

get_documents(*collection_name*, *query*)

update_document(*document*, *field*, *value*)

4.1.2 Submodules

ffmddb.client module

ffmddb.client.**run**()

ffmddb.server module

ffmddb.server.**run**()

CHAPTER 5

ffmddb Use Case Scenario

ffmddb was born from the idea that the best tool for editing a textfile is a text editor, and yet even text files benefit from managed metadata and relations between objects, as shown by a case study:

> I've been on the 'net for well over twenty years now, and over that period of time, I've amassed hundreds of log files. Some are notes, some are important conversations that led to relationship, some are inane conversations with individuals who have since passed away.
>
> In that time, I've run through several different organizational schemes, databases, and projects to manage these files. I wanted the organizational benefits of a relational database, the freedom of a document database, and the flexibility of editing the files by hand in whatever editor I choose. Finally, I wanted the ability to keep the files in a repository.

For the above problem space, the relation solution would be:

- A table of participants (name, about)
- A table of logs (name, text, date)
- A mapping table (log, participant)

In *ffmddb*, that maps to:

- A folder of participant files, text files with any document data, named after the participant
- A folder of log files, text files with any document data, and metadata containing a list of participants and the date of the log
- An index file containing mapping between logs by participant for faster queries

The same data and relations are represented, but in a format easily edited in any text editor, easily readible or served from something like Jekyll, and easily stored in a VCS repo. The goal is not speed, but flexibility for manually interfacing with smaller datasets.

f

C

D

F

G

I

M

O

P

Q

R

S

U

W