pymongoimport Documentation

Release 1.5.2

Joe Drumgoole

Mar 16, 2020

pymongoimport command-line programs:

`pymongoimport` is a collection of python programs for importing CSV files into MongoDB.

Why do we have `pymongoimport`?

MongoDB already has a perfectly good (and much faster) mongoimport program that is available for free in the standard MongoDB community download.

Well `pymonogoimport` does a few things that `mongoimport` doesn't do (yet). For people with new CSV files there is the `--genfieldfile` option which will automatically generate a typed field file for the specified input file. Even with a field file `pymongoimport` will fall back to the string type if type conversion fails on any input column.

`pymongoimport` allows you to use the `--addlocator` argument to automatically include a locator in each document that is inserted. This locator will indicate the file name and the line number of the line that was the input for the generated document.

pymongoimport also has the ability to restart an upload from the point where is finished. This restart capability is recorded in an `audit` collection in the current database. An audit record is stored for each upload in progress and each completed upload. Thus the audit collection gives you a record of all uploads by filename and date time.

Finally pymongoimport is more forgiving of *dirty* data. So if your actual data doesn't match your field type definitions then the type converter will fall back to using a string type.

On the other hand mongoimport supports the more extensive security options of the MongoDB Enterprise Advanced product and because it is written in Go it can use threads more effectively and so is generally faster.

CHAPTER 1

Mongoimport

Mongoimport is the command line program for importing CSV files into MongoDB.

1.1 Field Files

Each file you intend to upload must have a field file defining the contents of the CSV file you plan to upload.

If a `--fieldfile` arguement is not explicitly passed in the program will look for a fieldfile corresponding to the file name with the extension replaced by *.ff*. So for an input file `inventory.csv` the corresponding field file would be `inventory.ff`.

If there is no corresponding field file the upload will fail.

Field files (normally expected to have the extension `.ff`) define the names of columns and their types for the importer. A field file is formatted line a python config file with each section defined by a name inside square brackets (`[` and `]`) and values for the section defined by `key=value` pairs.

Here is an example CSV file inventory.csv defined by the following format,

Inventory Item	Amount	Last Order
Screws	300	1-Jan-2016
Bolts	150	3-Feb-2017
Nails	25	31-Dec-2017
Nuts	75	29-Feb-2016

The field file generated by **–genfieldfile** is

```
[Inventory Item]
type=str
[Amount]
type=int
[Last Order]
type=datetime
```

The **–genfieldfile** file function uses the first line after the header line to guess the type of each column. It trys in order for each column to successfully convert the type to a string (str), integer (int), float (float) or date (datetime).

The generate function may guess wrong if the first line is not correctly parseable. In this case the user can edit the .ff file to correct the types.

In any case if the type conversion fails when reading the actual data-file the program will fall back to converting to a string without failing (unless **–onerror** *fail* is specified).

Each file in the input list must correspond to a fieldfile format that is common across all the files. The fieldfile is specified by the **–fieldfile** parameter.

Once you have generated a fiels dilw you can pass it in on the command line by using the **–fieldfile** argument.

1.2 Restart

if a user specifies the **–restart** argument the program will keep track of what has been uploaded by maintaining a document for each file in an *audit* collection. If the upload fails or is interrupted for any reason, the user can restart the upload at the last line of the file by including **–restart** on the command line. This will force the program to check the audit collection for a corresponding record and restart the upload from the last written location. The audit records are keyed by *filename* so for **–restart** to work correctly the same file path must be used for the the original upload and the restart.

1.3 Examples

How to generate a field file

```
$pymongoimport --genfieldfile inventory.csv
Creating 'inventory.ff' from 'inventory.csv'
```

An example run:

```
$pymongoimport --delimiter '|' --database demo --collection demo --fieldfile mot_test_
↪set_small.ff mot_test_set_small.csv
Using database: demo, collection: demo
processing 1 files
Processing : mot_test_set_small.csv
using field file: 'mot_test_set_small.ff'
Input: 'mot_test_set_small.csv' : Inserted 100 records
Total elapsed time to upload 'mot_test_set_small.csv' : 0.047
```

An example run where we want the upload to restart

1.4 Arguments

1.4.1 Positional arguments

filenames List of files to be processed These files are expected to be in CSV format with fields delimited by the `--delimiter` argument (defaults to `,`.

1.4.2 Optional arguments

-h –help

Show the help message and exit.

–database *name*

Specify the *name* of the database to use [default: *test*]

–collection *name*

Specify the *name* of the collection to use [default : *test*]

–host *mongodb URI*

Specify the URI for connecting to the database. The full connection URI syntax is documented on the MongoDB docs website

The default is *mongodb://localhost:27017/test*

–batchsize *batchsize*

set batch os_size for bulk inserts. This is the amount of docs the client will add to a batch before trying to upload the whole chunk to the server (reduces server round trips). The default *batchsize* is 500.

For larger documents you may find a smaller *batchsize* is more efficient.

–restart

For large batches you may want to restart the batch if uploading is interrupted. Restarts are stored in the current database in a collection called *restartlog*. Each file to be uploaded has its own record in the *restartlog*. The restart log record format is

```
{ "name"            : <name of file being uploaded>,
  "timestamp"       : <datetime that this doc was inserted>,
  "batch_size"      : <the batchsize specified by --batchsize>,
  "count"           : <the total number of documents inserted from <name>␣
↪file to <timestamp> >,
  "doc_id"          : <The mongodb _id field for the last record inserted in␣
↪this batch> }
```

The restart log is keyed of the filename so each filename must be unique otherwise imports that are running in parallel will overwrite each others restart logs. use record count insert to restart at last write also enable restart logfile [default: False]

Restarts will happen from the last record that was written. You must specify restart when restarting an upload or when you wish to use the restart option.

–drop drop collection before loading [default: False]

–ordered forced ordered inserts

–fieldfile ***FIELDFILE*** field and type mappings. Defaults to the input file with the extension replaced by `.ff`.

–delimiter ***DELIMITER*** The delimiter string used to split fields [default: ‘,’]

–version show program’s version number and exit

–addfilename Add file name field to every entry. This allows records to be associated with their input file. [default : None]

–addtimestamp ***{none,now,gen}*** Add a timestamp to each record. If timestamp value is *none* don’t add a timestamp. If timestamp value is *now* add a single time stamp for all records. If timestamp is *gen* create timestamp for each

batch of records as they are inserted. Note that each batch of records (defined by –batchsize) will have the same timestamp when *gen* is the arguement. [default: none]

–has_header

The input file has a header line. We can use header line for column names [default: False]

–genfieldfile

Generate a fieldfile from the data file, we set has_header to true [default: False]

–id {mongodb,gen}

Autogenerate ID default [mongodb]

–onerror ***{fail,warn,ignore}*** What to do when we hit an error parsing a csv file [default: warn]

CHAPTER 2

multiimport

CHAPTER 3

splitfile

splitfile does what it says on the tin. Given a file it will split that file into a preset number of chunks on line boundaries. Its primary use is to split files for processing by `multiimport`.

CHAPTER 4

pwc

CHAPTER 5

Indices and tables

- genindex
- modindex
- search