Mongo Transform (mongot)

Mongo Transform is a command line utility for performing transformations on MongoDB collections using the MongoDB aggregation framework. It receives one or several JSON files containing the definition of the transformation(s) and carries them out in sequence.

Definition Files (defFiles)

Definition files (defFiles) are JSON's containing definitions for the transformations to be executed (transDefs). One or several defFiles may be passed on the command line like this:

```
node mongot hosts-to-users.json
```

If no file names are passed, ***`transform.json`*** is assumed.

Inside defFiles, the transformation definitions are represented by an array of one or more objects, each one defining a single transformation on a specified set of input documents. The transformations are executed in the order in which they appear in the defFiles.

transDef objects may contain any of the following keys:

Option (key)	**Type**	**Description**
connetionStr	String (optional)[1]	A valid connection string to the database which will be used for both the source and destination databases and collections. See Connection String URI Format.
sourceConnetionStr	String (optional)[1]	A valid connection string to the database which will be used for the source database and collection.
sourceDb	String (optional)[2]	The name of the source database.
sourceCollection	String (optional)[2]	The name of the source collection.
destConnetionStr	String (optional)[1]	A valid connection string to the database

		which will be used for the destination database and collection.
destDb	String (optional)[3]	The name of the destination database.
destCollection	String (optional)[3]	The name of the destination collection.
stages	Object (optional)	A list of comma separated objects containing pipeline aggregation stages. These aggregations are pushed into an array which serves as the parameter for the mongo DB db.collection.aggregate() method.
csvDef	Object (optional)	A definition for a Comma Separated Value (CSV) output containing a path and a projection object. See - **CSV Definition Objects (csvDef)**.
msg	String (optional)	A message to output to the console when processing this transformation.
log	String (optional)	The name of a log file into which a log will be written. If no file name is supplied, ***mongot.log*** will be assumed.

[1] ***connetionStr*** is shorthand for ***sourceConnetionStr*** and ***destConnetionStr*** combined i.e. if the source and destination use the same connection string, you can specify ***connetionStr*** instead of specifing both. If either ***sourceConnetionStr*** or ***destConnetionStr*** are missing, ***connetionStr*** must be supplied. You may however supply command line arguments to specify some or all of these properties instead.

[2] ***sourceDb*** and ***sourceCollection*** must be specified either in the defFile or in the command line arguments.

[3] ***destDb*** and ***destCollection*** must be specified either in the defFile or in the command line arguments.

Example 1

```
[
    {
        "sourceDb": "sample_airbnb",
        "sourceCollection": "listingsAndReviews",
        "destDb": "Airbnb",
        "destCollection": "user",
```

```
        "stages" : [
            {
                "$match" : { "name" : { "$gt" : "N"}}
            },
        ],
        "csvDef": {
            "path" : "./hosts.csv",
            "projection" : {
                "name" : 1,
                "username" : 1,
                "password" : 1
            }
        }
    }
]
```

The above example specifies the following transDef:

- The input is the `listingsAndReviews` collection from the `sample_airbnb` database.
- The resulting transformation is written to the `user` collection from the `Airbnb` database.
- The stages array contains a single aggregation stage which filters the input documents to those with the `name` key being greater than “N”.
- The resulting transformation is also projected into `./hosts.csv` with the fields `name`, `username` and `password`

Example 2

```
[
    {
        "sourceDb": "sample_airbnb",
        "sourceCollection": "listingsAndReviews",
        "destDb": "Airbnb",
        "destCollection": "user",

        "stages" : [
```

```
            {
                "$match" : { "name" : { "$gt" : "N"}}
            },
        ],
    },
    {
        "sourceDb": "Airbnb",
        "sourceCollection": "user",
        "destDb": "Airbnb",
        "destCollection": "hosts",

        "stages" : [
            {
                "$match" : { "isHost" : "true"}}
            },
            {
                "$unwind" : "$reviews"
            }
        ],
    }
]
```

The above example has two transDefs defined in it.

- In the first transDef the input is the **`listingsAndReviews`** colltection from the **`sample_airbnb`** database, and the output gets written to the **`user`** collection in the **`Airbnb`** database.
- There is one aggregation stage which filters the input documents to those with the **`name`** key being greater than “N”.
- In the second transDef the input is the **`user`** collection from the **`Airbnb`** database, and the output gets written to the **`hosts`** collection in the same database.
- The stages array contains two aggregation stages which filter the input documents and unwind the reviews array.

Command line arguments

Mongo Transform supports the command line arguments listed in the table below. These arguments take precedence over their definition file equivalents,

i.e. if the sourceDb option is given in both the definition file and the command line, the command line value will be observed.

Command line arguments override the options set in all of the transDefs in the defFile(s) i.e. if mongot is activated with one or more defFiles which contain several transDefs with various options set, the command line arguments will override the options for each of the transDefs.

Finally, there is no significance to the order in which the command line arguments are specified and they can be combined freely.

Option	**Flag**	**Description**
defFile	-f	The definition file or files. The "-f " can be omitted. If no definition file is supplied, ***transform.json*** is assumed. If the file extension is omitted, ***.json*** is assumed.
connectionStr	-s	A valid connection string to the database which will be used for both the source and destination databases and collections. See Connection String URI Format.
sourceConnectionStr	-i	A valid connection string to the database which will be used for the source database and collection.
sourceDB	-D	The name of the source database.
sourceCollection	-T	The name of the source collection.
destConnectionStr	-o	A valid connection string to the database which will be used for the destination database and collection.
destDB	-d	The name of the destination database.
destCollection	-t	The name of the destination collection.
csvPath	-c	The name of an output CSV file.
noCsv	-C	Suppress CSV output.
noCsvHeadings	-H	Suppress column headings in CSV output.
noWrite	-N	Suppress output to database.
msg	-m	A message to output to the console when processing this transformation.
log	-l	A file name for logging. If none is supplied, ***mongot.log*** is used.
help	-h	Displays a usage summary.

Example 1

```
node mongot
```

If no defFile is specified, Mongo Transform is run with **`transform.json`** as its defFile.

Example 2

```
node mongot -f hosts-to-users.json
```

```
node mongot hosts-to-users.json
```

In the above examples, Mongo Transform is run with **`hosts-to-users.json`** as its defFile. These two lines produce the same result since **`-f`** is the default option and can be omitted.

Example 3

```
node mongot stay hosts-to-users.json
```

In the above example, Mongo Transform is run with two defFiles - **`stay.json`** & **`hosts-to-users.json`** . The defFiles are parsed in the same order as they appear on the command line and their transDefs are processed in sequence.

Example 4

```
node mongot stay.json -C
```

Suppress output to a CSV file if such was specified in the transDef

Example 5

```
node mongot stay.json -C -N
```

```
node mongot stay.json -CN
```

Suppress output to a CSV file (`-c`) and to an output collection (`-N`) if such were specified in the transDef. These two lines produce the same result since boolean options can be grouped with a single hyphen.

Example 6

```
node mongot stay.json -c stay.csv -t mycoll -d mydb
```

Override the CSV outfile path (`-c`), and the output collection and database (`-t` and `-d`) specified in the transDef.

Example 7

```
node mongot stay.json -c stay.csv -T mycoll
```

Suppress output to a CSV file specified in the transDef (`-c`), and set the input collection to **`mycoll`** (`-T`).

Example 8

```
node mongot stay user.json -T mycollection
```

Set the input collection of all transDefs in **`stay.json`** and **`user.json`** to **`mycollection`**.

Example 9

```
node mongot -h
```

Show usage summary (`-h`).

CSV Definition Objects (csvDef)

A transformation definition (transDef) may specify a csvDef key whose value is and object with the following structure:

Option (key)	Type	Description
path	String	A full path and file name for the output.
project	Object (optional)	An object defining the output projection, i.e. the list of fields to be written to the CSV file. It follows the basic MongoDB $project syntax - {key : value} pairs, where the key is the name of a field in the transDef output.

Links and references

If you want to use Mongo Transform, you might want to get familiar with the MongoDB Aggregation Framework. Start here.

It might also be useful to play around with the Available Sample Datasets for Atlas Clusters from MongoDB. Check out this link for information on how to install them.

Mongo Transform uses command-line-args and terminal-kit which are two cute and easy to use npm packages.