A computational EXFOR database

User manual

Georg Schnabel
georg.schnabel@nucleardata.com

July 22, 2019

The EXFOR library compiled and maintained by the International Network of Nuclear Reaction Data Centers (NRDC) is an essential source of information in the domain of nuclear physics. This document describes the setup of the EXFOR library in the form of a MongoDB database on the local computer. Using the Docker technology for virtualization, the setup is easy to perform on Linux, Windows, and MacOS. Some examples to interact with the database in the programming languages R and Python are provided. Due to the popularity of MongoDB, equivalent functionality is also available in many other programming languages, such as C, C++, Java, and Perl.

Contents

1 Introduction

The EXFOR library [1] is an important source of information in the domain of nuclear physics. Interfaces, such as the IAEA search form[2], enable experimentalists and evaluators to conveniently locate and retrieve data. Ongoing efforts to make evaluations reproducible suggest that a streamlined and programmatic way to access the data including bibliographical details would be a helpful complement to existing interfaces and tools. Anticipating the increasing need of programmatic access in the future, the complete EXFOR library has been converted to a MongoDB database. As most of the popular high-level languages provide facilities to retrieve and modify data in a MongoDB database, programmatic access to any data (cross sections, bibliography, etc.) becomes very convenient.

This document describes the installation of the EXFOR library as a MongoDB database (occasionally referred to as computational EXFOR database) on your local computer or cluster and how to use it from R and Python. The installation is described in section 2. Some introductory usage examples for R and Python are provided in section 3. Details about the choices made for the conversion from the original EXFOR library to a MongoDB database are discussed in section 4. Final remarks and an outlook in section 5 end this document.

As a final introductory note, please do not hesitate to get in touch if you have found a bug or have ideas for improvements of the manual or the database. All software components needed for installing and running the database are open-source. Feel free to inspect them and to modify them according to your specific needs.

2 Installation of the computational EXFOR database

During the installation process, several tasks are performed:

1. Installation of the MongoDB database software
2. Download of the EXFOR library
3. Conversion of EXFOR entries to JSON objects
4. Insertion of the JSON objects into the MongoDB database

Because the scripts for conversion are written in the programming language R, the installation process for the time being requires the availability of the R interpreter and some R packages. To avoid cluttering your system with libraries you don't need and to reduce the need for user intervention in the installation process to a minimum, the installation as described in this document relies on the Docker software. Therefore,

[1] N. Otsuka, E. Dupont, V. Semkova et al., Towards a More Complete and Accurate Experimental Nuclear Reaction Data Library (EXFOR): International Collaboration Between Nuclear Reaction Data Centres (NRDC), Nuclear Data Sheets 120, 272-276, 2014, DOI: 10.1016/j.nds.2014.07.065

[2] `https://www-nds.iaea.org/exfor/`

the Docker software must be installed on your computer first before you can attempt the installation of the EXFOR database. A brief introduction to Docker and pointers to resources on the web concerning its installation are provided in the next section. Afterwards building and launching the database are described.

2.1 About Docker

The Docker software is a prerequisite for the installation process of the computational EXFOR database as outlined in this document. The Docker software can be loosely described as an application to manage light-weight virtual machines on your computer. A virtual machine is the simulation of a specific computer system on top of your base system. For instance, within a virtual machine you may run Linux even though the operating system on your computer is Linux. In the language of Docker, the two relevant concepts to know about are *images* and *containers*. A Docker container is a virtual machine, which may run Linux with a specific selection of libraries and applications. A Docker image is a template for the creation of Docker containers and defines which operating system, libraries, and applications should be available in a Docker container based on that image. Therefore, every time a Docker container (i.e., virtual machine) is started up, a Docker image needs to be provided to set up the container.

This mechanism involving images and containers is very powerful. It facilitates the setup procedure because all dependencies of an application can be shipped as a bundled package. For instance, a program written in a script language needs an interpreter to be executed. The Docker mechanism allows to ship the script and the necessary interpreter as one unit—a Docker image. Furthermore, the Docker mechanism completely separates and decouples the software in the container from the operating system, applications and data on your computer. Problems such as with conflicting library versions are therefore avoided. Since software in the container cannot access anything outside the container, the deployment of software as containers has also advantages in terms of system security.

The official Docker website is located at `https://www.docker.com`. Please consult the information on this website for more information about Docker. In particular, the installation of the Docker Community Edition (CE), which is free of charge, is detailed at `https://docs.docker.com/install/`. The Docker application is an essential requirement to perform the installation of the computational EXFOR database as described in this document. Whether Docker is installed on your system can be checked by running in a terminal

```
docker version
```

If Docker is installed, you should see information about the installed Docker version and you can proceed to the next section.

2.2 Building the database

First, download the file at

```
https://github.com/gschnabel/compEXFOR-docker/archive/master.zip
```

Unpacking creates a new folder `compEXFOR-docker-master`. If you want to enable authentication to access the computational EXFOR database to be created, open the file `install_all.sh` present in this folder in a text editor. At the beginning of this file are the two lines

```
db_user=""
db_password=""
```

Change the username and the password between the quotation marks to your liking. These credentials will be needed to connect to the computational EXFOR database and to access information. In the default configuration above, authentication is disabled and everyone on the local computer can access and manipulate the information in the computational EXFOR database.

Depending on your internet speed, the following instruction may take a while (say half an hour). Open a terminal and change into the folder `compEXFOR-docker-master`. Make sure that this folder, now the current working directory, contains the file `Dockerfile`. Then in the terminal run

```
docker build --t compexfor-img:latest .
```

Please note that the final point is part of the command. This instruction creates a so-called *Docker image* by downloading all required components (e.g., MongoDb, R, conversion scripts, etc.) from various resources on the net and assembling them as a Docker image. If everything went well, the computational EXFOR database is now successfully installed on your computer. If you wish, you can delete the folder `compEXFOR-docker-master` now because it was only needed for the creation of the Docker image. Another good reason to delete this folder is if you have specified authentication credentials and don't want them to reside on your disk in clear text.

2.3 Launching the database

As a first verification step, we test whether the Docker image has been successfully created. To that end, open a terminal window and run the command

```
docker images
```

It will display a listing of available Docker images. If a row is present with *compexfor-img* as the repository name, the creation of the Docker image was successful.

Now let us create a container based on the Docker image. In a terminal window run the command

```
docker run --it -p 27017:27017 --name compexfor-cont compexfor-img:latest
```

Please note that the first number, 27017, is your choice. If you have already a MongoDB server running on your computer and listening on the default port, you need to change this number to something else, e.g., 27018. After a few seconds, you will see a command prompt starting with `root@` followed by some long hexadecimal string. You are in a

Linux session *inside* the container. The container is running and the computational EXFOR database should be accessible. At this stage, you can jump to the section with examples to get a brief introduction how to interact with the database from R or Python. To shutdown the container, run the command (inside the container)

```
exit
```

You can see whether the container is running or not by executing in a terminal

```
docker ps -a
```

If you want to start up again the container at a later point, execute in a terminal

```
docker start compexfor-cont
```

You will not automatically enter a shell inside the container. If you want that, you need to run the command

```
docker attach compexfor-cont
```

Finally, to stop a container from the terminal (*not* inside the container), run

```
docker stop compexfor-cont
```

These are the essential commands provided by the Docker application necessary to set up the computational EXFOR database. If you want to learn more about Docker and how to manage images and containers, consult the websites referenced in section 2.1.

3 Usage examples

Support to access, search, retrieve, and manipulate data in a MongoDB database is available in many programming languages. In this section simple usage examples for Python and R are provided. For more information about the capabilities of MongoDB databases and how to make use of them from your favorite programming language, please consult the numerous resources in form of tutorials and documentation available on the web.

3.1 Accessing data using Python

MongoDB can be accessed from Python via the `pymongo` module. In the following examples, we further assume that the modules `numpy` and `pandas` are installed.

The following code provides a skeleton to connect with the EXFOR database. It is assumed that the Docker container runs on the machine where the code is executed and you have specified the port 27017 during the creation of the container, see section 2.3.

```
import pymongo

client = MongoClient("localhost", 27017,
                     username = "user", password = "pw")
db = client["exfor"]
entries = db["entries"]
# place code here to interact
# with the data in EXFOR
client.close()
```

The credentials `user` and `pw` passed to the function `MongoClient` need to your choice of credentials during the setup in section 2.2. In the default setup, authentication is disabled and the `username` and `password` argument can be removed.

Having the connection to the database established, let's see how we can retrieve a dataset with a specific subentry ID:

```
subent = entries.find_one({ "ID": "11701004" })
```

The variable `subent` is a dictionary with the information of the dataset of of entry 11701 and subentry 004. To explore it a bit, run for instance

```
subent["BIB"]["AUTHOR"]
subent["DATA"]["UNIT"]
subent["DATA"]["TABLE"]
```

The query language of MongoDB is very expressive and it is impossible to go through all the possibilities in this document. Just as an example, the following code snippet shows how a regular expressions can be used to match specific reaction strings:

```
import re
regex = re.compile("^\(26-FE-56\(N,[^)]+\)[^,]*,,SIG\)")
subents = entries.find({ "BIB.REACTION" : regex })
for subent in subents:
    print(subent)
```

Sometimes one is not interested in retrieving all the data in each dataset and the possibility to specify which fields should be included in the result comes handy:

```
import re
regex = re.compile("^\(26-FE-56\(N,[^)]+\)[^,]*,,SIG\)")
search_spec = {"BIB.REACTION" : regex,
               "DATA.TABLE.DATA": { "$exists" : True}}
filter_spec = {"ID": 1, "BIB.AUTHOR": 1,
               "DATA.TABLE.DATA": 1, "_id": 0 }
subents = entries.find(search_spec, filter_spec)
```

The dictionary `search_spec` defines that we search for datasets whose reaction string matches the regular expression `regex` and that have a column named `DATA` in their data table. The dictionary `filter_spec` defines which fields should be included in the result. Here we specified that the subentry ID (named `ID`), the author field `BIB.AUTHOR`, and the `DATA` column should be included in the result but not `_id`. The latter field is an internal unique identification number automatically assigned by MongoDB. Because the subentry `ID` already uniquely identifies the dataset, the automatically generated `_id` field is usually neither needed nor of interest.

Building on the last code snippet above, it may be convenient to assemble a `pandas` DataFrame as a basis for efficient grouping operations and convenient data analysis:

```
import pandas as pd
import numpy as np
from pandas.io.json import json_normalize
rawdf = json_normalize(list(subents))

lst_col = "DATA.TABLE.DATA"
rep_len = df[lst_col].str.len()
rep_len[np.isnan(rep_len)] = 1
rep_len = rep_len.astype(int)

df = pd.DataFrame({
    col: np.repeat(rawdf[col].values, rep_len)
         for col in rawdf.columns.drop(lst_col)
}).assign(**{lst_col:np.hstack(rawdf[lst_col].values)})[df.columns]
```

The resulting `pandas` DataFrame `df` contains the subentry ID, the author list, and the values in the `DATA` column in a tabular format.

3.2 Accessing data using R

A MongoDB database can be accessed from R using the `mongolite` package available on the CRAN network. Assuming that the container with the EXFOR MongoDB runs on the local computer, the following code establishes a connection to the database:

```
library(mongolite)
db <- mongo(collection = "entries",
            db = "exfor",
            url = "mongodb://user:password@localhost:27017")
# place code here to interact
# with data in EXFOR
db$disconnect()
```

Replace `user` and `password` by the credentials you provided during the installation of the database described in section 2.2. If you did not enable authentication by providing

a username and password, you can remove the substring `user:password@` from the command to connect.

Having the connection to the database established, let's see how we can retrieve a dataset with a specific subentry ID:

```
it <- db$iterate('{ "ID": "11701004" }')
subent <- it$one()
```

The variable `subent` is a list with the information of the dataset identified by entry ID 11701 and subentry ID 004. To explore it a bit, run for instance

```
# alternative:
# subent[["BIB"]][["AUTHOR"]]
subent$BIB$AUTHOR
subent$DATA$UNIT
subent$DATA$TABLE
```

The query language of MongoDB is very expressive and it is impossible to go through all the possibilities in this document. Just as an example, the following code snippet shows how a regular expressions can be used to match specific reaction strings:

```
regex <- "^\\\\(26-FE-56\\\\(N,[^)]+\\\\)[^,]*,,SIG\\\\)"
query <- paste0('{ "BIB.REACTION" :
                   { "$regex": "', regex, '", "$options" : "" }}')
it <- db$iterate(query)
while (!is.null(subent <- it$one())) {
    print(subent)
}
```

The string `fields` containing a JSON object as a string defines that we search for datasets whose reaction string matches the regular expression `regex`.

Sometimes one is not interested in retrieving all the data in each dataset and the possibility to specify which fields should be included in the result comes handy. To filter fields, modify the previous code snippet in the following way:

```
fields <- '{ "ID": 1,
             "BIB.AUTHOR": 1,
             "DATA.TABLE.DATA": 1,
             "_id": 0 }'
it <- db$find(query, fields)
```

The string `fields` containing a JSON object defines which fields should be included in the result. Here we specified that the subentry ID (named `ID`), the author field `BIB.AUTHOR`, and the `DATA` column should be included in the result but not `_id`. The latter field is an internal unique identification number automatically assigned by MongoDB. Because the subentry `ID` already uniquely identifies the dataset, the automatically generated `_id` field is usually neither needed nor of interest.

The `mongolite` package provides a low-level interface to access a MongoDB database. A package called `MongoEXFOR` is available from `https://github.com/gschnabel/MongoEXFOR` which is a shallow wrapper around the `mongolite` package and makes access to the EXFOR stored as MongoDB database more convenient.

4 Details on the conversion of the EXFOR library

This section lists some choices made for the conversion of the original EXFOR library to a MongoDB database. The MongoDB databases possesses the following properties:

- Datasets are subentries and not entries.
- Each subentry is augmented with the information of the first subentry containing bibliographical information.
- If a subentry contains a field name that is also present in the first subentry, the suffix `_firstSub` is added to the field name of the first subentry before merging.
- If the text in a free text field extends over several lines in the original EXFOR format, the strings of each line are concatenated to one string with an additional space inserted instead of a new line character.
- Some fields, such as the `REACTION` field contain both processable data enclosed by brackets and free text. No transformations are applied to such fields.
- The original EXFOR format supports fields that contain lists. For instance, several reactions may be provided together in the `REACTION` field. In the MongoDB database, these lists are stored as arrays. Fields can therefore either contain a scalar quantity (i.e., string or number) or an array.
- The top-level field `DATA` contains the three fields `UNIT`, `DESCR`, and `TABLE`.
- A field `UNIT` is introduced as a subelement of the `DATA` field, i.e. `DATA.UNIT`. It is a dictionary with keys given by by the name of the quantity (e.g., `EN`, `DATA-ERR`, `DATA`) and associated with it strings indicating the associated units.
- The field `DATA.TABLE` contains the table with the cross section data. It is a dictionary where the keys denote the name of the quantity and the associated content is an array if the cross section table contains more than one row. In the case of just one row, the content associated with the keys are scalars, i.e., just numbers.
- The field `DATA.DESCR` contains the names of the quantities referenced in `DATA.TABLE` in the order given in the original EXFOR entry.
- The information in the `COMMON` blocks is added to `DATA.TABLE` as additional columns. The `COMMON` fields are also preserved in the subentry as a separate field.

- Some units in the data table are converted. Specifically, energy and cross section units are standardized. The energy unit is MeV (`MEV`) and the unit for cross sections is millibarn (`MB`).
- The original EXFOR format allows several columns for the same quantity, e.g., several `DATA` columns in `DATA.TABLE`. In the MongoDB database, the index is appended as a suffix, e.g., `DATA-1`.
- Sometimes values are missing in the data table of an EXFOR entry. The value `null` is used in the MongoDB database to indicate a missing value.
- A top-level field `META` is introduced which among others contains subfields, such as the charge number `Z` and mass number `A`.

5 Concluding remarks and outlook

Programmatic access to all the available information in the EXFOR library helps to accelerate data analysis and automate evaluations. Storing the EXFOR library as a MongoDB database is one possible route towards this goal. Because the MongoDB database software is open-source, free of charge, and widely supported in a variety of programming languages, it is a good fit for the use in research. However, due to a recent change of its license, running such a MongoDB database as a web application accessible by the world would require to make the server application which embeds the database open-source as well, which is certainly not desired in many cases. This can be a reason to refrain from using *MongoDB* and to pick an alternative with a better license. At the time of this writing, such an alternative could be *CouchDB*.

Irrespective of such technical details, building on software and technology that is widely supported and used by many avoids reinventing the wheel. In this way, new software solutions adding value by removing technical obstacles in the research workflow can be implemented in months rather than years.

The EXFOR library exists since a long time and many people rely on its information and structural correctness. The introduction of new fields or new representations of data is therefore not taken light-heartedly. A prudent approach in this matter is reasonable but bears the disadvantage that it takes time until useful extensions find their way into the format. A local copy of the EXFOR library as a *document-oriented database*, be it *MongoDB*, *CouchDB* or something else, puts users into the position to easily modify the structure of their local copy in any way they want. As an example, currently the `COV` field containing a covariance matrix is not parsed and its content in the MongoDB database is a long character string with numbers. The conversion of this information to an array in the MongoDB database can be achieved by the user with a small Python script with maybe not more than hundred lines of code. Users can share their innovations added to the format and if they prove useful for a larger group, this may help to decide which extensions should be incorporated into the official EXFOR library.