my Accounting

Manuel Escriche Vicente

April 10, 2025

Abstract

Contents

1 Setup python environment

Get a list of python installation:

```
>> type -a python
python is /Users/MANEV/.pyenv/shims/python
python is /usr/bin/python
```

From this list, the first one is used:

```
>> which python
/Users/MANEV/.pyenv/shims/python
```

Check what version of python is in use:

```
>> python -V
Python 3.8.6
```

Check available versions

```
>> pyenv versions
  system
  3.6.1
* 3.8.6 (set by /Users/MANEV/.pyenv/version)
  3.9.16
```

Set proper version in working folder.

```
 >> pyenv local 3.9.16
 >> python -V
 Python 3.9.16
```

Install pipenv to setup our virtual python environment. Notice pipenv doesn't get the version established by pyenv local previously, but the global one. Therefore, you have to care to set it up again in next steps.

```
 >> pip install pipenv
 >> pipenv install --python 3.9.16
 >> pipenv shell
```

In case you have to remove your virtual environment

```
 >> pipenv --rm
```

Install sqlalchemy

```
 >> pipenv install sqlalchemy
```

Now you're ready to configure the application for any user.

2 Configuration and administration

There are five tools available for users configuration and for fixing emerging issues in the application:

config_user.py used to create the user's folder, and to copy the generic profile chosen into the specific `<user>_profile.json`. The user profile file declares what accounts are available for use in your accounting, and are moved into `accounts.json` file to create the data base.

showtree.py is able to display accounts as a tree, both in the accounts.json file and the database, and to compare them as well.

db_tool.py used to create user's data base, and populate it with those accounts declared in the `accounts.json` file, and other useful utilities.

remake.py used to rebuild the user data base and their related yearly closing and opening seats consistently over the years.

cmp_files.py used to compare two seats files. It is helpful to verify changes in the seat files after having saved them with `db_tool save`

2.1 config_user.py

Run `>> python tools/config_user.py <user> create <profile>` **only one time** in order to create your `<user>_profile.json` file, and its subsequent operative application files at due place `\users\<user>\configfiles`:

$$\begin{tikzcd}\boxed{\texttt{<user>\_profile.json}} \arrow[r] & \boxed{\texttt{accounts.json}}\end{tikzcd}$$

Thereafter, any modification by hand in your `<user>_profile.json` file, requires you to run `>> python tools/config_user.py <user> refresh` to regenerate consistently the `accounts.json` file.

Commands

```
 >> python tools/config_user.py <user> create <profile>
 >> python tools/config_user.py <user> refresh
```

2.2 db_tool.py

Before starting you have to use this tool to create and prepare the user database. Execute the following sequence:

1. `db_tool.py <user> create` to create user database file

2. `db_tool.py <user> init` to create user database structure

3. `db_tool.py <user> setup` to populate user database with accounts

4. `db_tool.py <user> query` to verify accounts creation

5. `db_tool.py <user> save` to save yearly seats stored in the data base into files named with the tag `app` like `YEAR_app_seats.json`. It's helpful when having to evolve the data base accounts, in companion of the `remake` tool

6. `db_tool.py <user> backup` to create a database file backup

7. `db_tool.py <user> remove` to remove the database file

Commands

```
>> python tools/db_tool.py <user> create
>> python tools/db_tool.py <user> init
>> python tools/db_tool.py <user> setup
>> python tools/db_tool.py <user> query
>> python tools/db_tool.py <user> save
>> python tools/db_tool.py <user> remove
```

2.3 remake.py

It's an important tool that allows to `remake` the user data base and its corresponding accounting yearly closing and opening seat files from the user profile, starting opening seat, and yearly seats file available. It's helpful when having to update the `<user>_opening_seat.json` file, which requires to work it out all data, or when having to modify the `<user>_profile.json` file because new accounts have been created, or its code modified.

Commands

```
>> python tools/remake.py <user>
>> python tools/remake.py <user> -t
>> python tools/remake.py <user> -t new
```

2.4 cmp_files.py

Tool useful to compare json transactions files; especifically, `year_<tag>_seats.json` file against `<year>_app_seats.json` file.

Commands

```
>> python tools/cmp_files.py <user> <year>     #tag = <user>
>> python tools/cmp_files.py <user> <year> -t new
```

3 Edition and Report

`myAccounting.py` is the main operating app for your personal accounting. Seats are edited on its user interface, however since most of the seats are downloaded from the bank account in excel sheets, a specific tool has been made available.

excel.py tool used to take bank accounts excel files to export their transactions into json format so that main program can load them.

myReporting.py is available to display data compared to the previous year, and accounts evolution along the years. It's useful when to understand the year.

Commands

```
>> python tools/excel.py <user>
>> python myAccounting.py <user>
>> python myReporting.py <user>
```

3.1 Conceptual Model

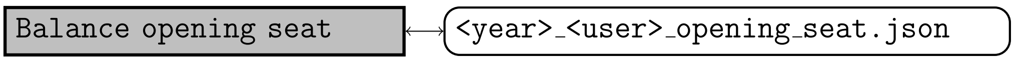
Yet simple, but important to bear in mind:

Transactions[1] are held in Journals. BookEntries in Ledgers, and Accounts in the Accounts Book. In data base terms, Journals, Ledgers and Accounts book are tables.

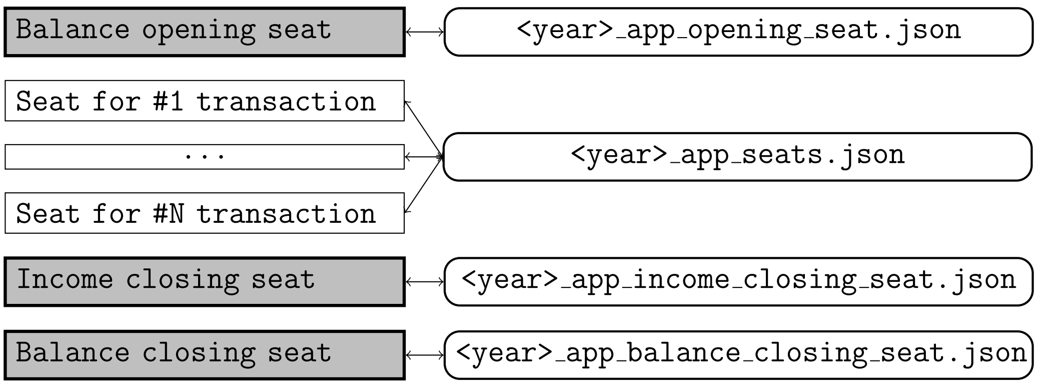
Besides, a transaction references several book entries; and each book entry references an account. Therefore, a transaction is build by aggregating a number of records in the data base.

Each record holds an identifier that makes it unique. However, when it comes to accounts, besides its unique data base id, they also have a unique code, which we set up in the profile file, and propagates to the `accounts.json` file, and from there to the `accounts` table in the data base.

The first seat is created when starting our accounting. This first seat that holds our opening balance is saved in a file, and from here by using the main app loaded into the data base.



The normal accounting year has three mandatory seats, which are created by the main application upon user request, and are stored in different files. However, they must be loaded by the user to have an effect.



[1]Seats

Why are seats stored both in data base and files? The application has been designed to store data both in the data base and in data files for safety reasons mainly. Yet it isn't the only reason. As it is an accounting learning tool, it allows to you to evolve your accounting profile, or opening seat, and to `remake` the data base and opening and closing seats consistently.

4 Administration use cases

In order to make operation easy, it's been created `mev_session.py` file, which open the database and creates basic variables like db, user, accounts, tree, etc, which are useful when having to fix issues by hand.

4.1 How to import mev_session

```
>>> import mev_session as mev
>>> dir(mev)
```

4.2 How to search an account

Import session as in 4.1 then follow one of the alternatives shown below:

```
#1
>>> acc = next(filter(lambda x: x.code == '64', mev.accounts))
>>> len(acc.entries)
>>> for entry in acc.entries: print(entry)
#2
>>> from dbase import  Account,Transaction, BookEntry
>>> for item in mev.db.query(Account): print(item)
>>> print(db.query(Account).filter_by(code=645).one())
#3
>>> from sqlalchemy import select
>>> stmt = select(Account).where(Account.code.in_([64,641]))
>>> for item in mev.db.scalars(stmt): print(item)
#4
>>> stmt = select(BookEntry).join(Account).where(Account.code==622)
>>> print(len(db.scalars(stmt).all()))
>>> for item in db.scalars(stmt): print(item)
```

4.3 How to search transactions

Import session as in 4.1 then follow one of the alternatives shown below:

```
#0
>>> mev.transactions
#1
>>> trans = next(filter(lambda x:x.id == 3902, mev.transactions))
>>> trans
>>> trans.entries
#2
>>> from sqlalchemy import select
>>> from dbase import Transaction, BookEntry
>>> stmt = select(Transaction).where(Transaction.id == 3902)
>>> item = db.scalars(stmt).one()
>>> print(item)
>>> for entry in item.entries: print(entry)
#3
>>> stmt = select(BookEntry).join(Transaction).where(Transaction.id==3902)
>>> for item in db.scalars(stmt): print(item)
#4
>>> items = db.query(BookEntry).join(Transaction).filter_by(id=3902)
>>> for item in items : print(item)
```

4.4 How to search book entries

Import session as in 4.1 then follow one of the alternatives below:

```
#0
>>> mev.entries
#1
>>> entry = next(filter(lambda x:x.id == 4460, mev.entries))
>>> entry.account
>>> entry.transaction
#2
>>> from sqlalchemy import select
>>> from dbase import BookEntry
>>> for item in db.scalars(select(BookEntry)):print(item)
>>> for item in db.query(BookEntry): print(item)
```

4.5 How to modify an account description

Edit the account description in file `<user>_profile.json`, then follow

```
# update accounts.json
>> python tools/config_user.py <user> refresh

# update db with new info in accounts.json
>> python tools/db_tools.py <user> setup
```

4.6 How to modify a transaction description

Import session as in 4.1, find the transaction as in 4.3 then follow:

```
#modify
>>> trans.description = 'Balance Opening Statement'
>>> mev.db.commit()
#verify
>>> item = db.get(Transaction,1)
>>> item
Transaction(1 | 2015-11-15 | #entries=5 | Balance Opening Statement)
# save to files
>> python tools/db_tool.py <user> save!
```

4.7 How to create a new account

When a new account is created, a new entry is created in the Account table.
Follow these steps:

1. Edit the `<user>_profile.json` file
   - (a) Open the file, which is located in the folder `configfiles` with a text editor
   - (b) Add a new line, and write the new entry in the **accounts** section `{"type":"DEBIT","content":"NOMINAL","code":"663","name":"Tax-House"}`
   - (c) Save the file
2. Run `>> python tools/config_user.py <user> refresh` to regenerate the application file: `accounts.json`
3. Run `>> python tools/db_tool <user> setup` to update data base account table with the records.

4.8 How to modify the code in a nominal account

Let's take an example: on the profile we have the following entry:
`{"type":"DEBIT","content":"NOMINAL","code":"621","name":"ExpH-Water"}`
where we want to transform the code `621` to become `6211`

Follow these steps:

1. Let's make a backup of the data base before modifying it by running `>> python tools/db_tool <user> backup`, that makes a backup copy.
2. Let's edit the `<user>_profile.json` file
   (a) Open the file located in the folder `configfiles` with a text editor.
   (b) Modify the account's code in the line where it was declared.
   `{"type":"DEBIT","content":"NOMINAL","code":"6211","name":"ExpH-Water"}`
   (c) Save the file.
3. Let's modify account code by following these steps:

   Import session as in 4.1, search the account with code=621 as in 4.2, then follow:

```
>>> account
Account(33 | Type.DEBIT | Content.NOMINAL | 621 | ExpH-Water | None )
>>> account.code = 6211
>>> mev.db.commit()
>>> quit()
# save db to files
>> python tools/db_tool.py <user> save!
# refresh accounts.json
>> python tools/config_user.py <user> refresh!
# remake data base from files
python tools/remake.py <user>
```

4.9 How to modify the code in a real account

The process described in section 4.8 is valid but taking into account a important difference. If the account whose code is going to be modified is used in any of the seats declared in the user opening file: `<user>_opening_seat.json`, there's need to modify this seat before running `>> python tools/remake.py <user>`