Python interface to GnuCash documents Documentation

Release 1.1.2

sdementen

Oct 24, 2020

CONTENTS

Release 1.1.2

Date Oct 24, 2020

Authors sdementen

Project page https://github.com/sdementen/piecash

CHAPTER

ONE

WHAT'S NEW

1.1 Version 1.1.2 (2020-10-24)

- import requests from functions using it to avoid making it a required dependency (fix #90)
- adapt setup.py to avoid depending on SQLAlchemy-Utils 0.36.8 (fix #91)
- updated gnucash projects page: https://piecash.readthedocs.io/en/latest/doc/github_links.html

1.2 Version 1.1.1 (2020-10-21)

- add a check_exists flag to allow bypassing check existence of DB on opening (fix #91, tx @williamjacksn)

1.3 Version 1.1.0 (2020-10-20)

- fix use of ISO date for ledger export (fix #115 by @MisterY)
- add field is_credit and is_debit to split (fix #105)
- fix get_balance sign when recursing + add natural_sign keyword to specify if sign should be reverse or not
- add support for Gnucash 4.1 (fix #136)
- fix table names not matching in case (fix #137)
- fix test suite to support 3.8
- deprecate python 3.5
- quandl will retrieve API KEY from environment variable QUANDL_API_KEY (if defined)
- yahoo will use exchangeTimezoneName for timezone (vs exchangeTimezoneShortName before), thanks @geoffwright240
- add possibility to export accounts with their short name in ledger (fix #123)

1.4 Version 1.0.0 (2019-04-15)

- drop support of py27 and py34 (fix #53)
- support gnucash 3.0.x format (code + test and book migration)
- set autoflush to False for open_book (was only done for create_book before) (fix #93)
- remove tz info when serialising DateTime to SQL (issue with postgresql doing some TZ conversion)
- add basic support for Jobs

1.5 Version 0.18.0 (2018-04-24)

Mostly refactoring: - refactor common parts of vendor, customer and employee into person - add 'on_book_add' protocol called when object is added to a book - set autoflush to False to prevent weird behavior when using slots (that retrigger a query in the middle of a flush) - refactor slots - align sql schema 100% with 2.6.21 (based on sqlite reference) - support business slots

1.6 Version 0.17.0 (2018-03-16)

- internal refactoring of setup.py
- add optional packages
- move to pipenv
- improve documentation
- fix missing extra blank between account name and amount in ledger export (fix #86)

1.7 Version 0.16.0 (2018-03-04)

- add a documentation section about piecash on android
- fix yahoo finance quote retrieval
- indicate correct reconcile state in ledger output (fix #77)

1.8 Version 0.15.0 (2018-02-21)

- add piecash CLI (refactor of scripts)
- add book.invoices to retrieve all invoices in a book
- expose gnucash rationals as decimals in Entry and Invoice
- fix issue #65 about "template" (scheduled transactions) appearing in ledger export
- fix issue #64 about escaping in double quote mnemonic with non alpha characters
- fix issue #19 allowing to pass the check_same_thread flag for sqlite
- add argument recurse to get_balance (fix #73)

- handle currency conversion in get_balance
- add Commodity.currency_conversion to get a conversion factor between a commodity and a currency

1.9 Version 0.14.1 (2018-02-01)

- fix bug in pc-export

1.10 Version 0.14.0 (2018-02-01)

- fix definition of account get_balance to use quantities (and not values) (@sdementen)
- fix bug when providing a float instead of a Decimal to a numeric value (@gregorias)
- support new format for date for 2.7/2.8 (@MisterY, @sdementen)
- fix bug where transactions based on deleted scheduled transactions cause exceptions (@spookylukey)
- fix bug (#58) where large Decimals where raising an sql exception instead of a ValueError exception (@sdementen)
- add Recurrence to global imports + add documentation to Recurrence (@MisterY)
- add script pc-export to export customers and vendors from a gnucash book (@sdementen)

1.11 Version 0.13.0 (2017-10-08)

- upgrade CI (appveyor and travis) to 2.7/3.4/3.5/3.6
- upgrade dependencies
- df_splits: allow user to specify additional fields to extract (@NigelCleland)
- improve documentation (@Brian-K-Smith)

1.12 Version 0.12.0 (2017-02-15)

- rely on yahoo-finance to retrieve share information and share prices
- use only ISO currency static data (remove support for looking on the web)
- normalise post_date to 11:00AM

1.13 Version 0.11.0 (2016-11-13)

- add support for python 3.5
- add preload method on book to allow preloading all objects at once

1.14 Version 0.10.2 (2015-12-06)

- add children argument to Account constructor
- add a new example (used as answer to http://stackoverflow.com/questions/17055318/create-transaction-in-gnucash-in-response-to-an-email/)
- add a new example showing how to export Split information to pandas DataFrames
- fix an error handling in retrieving currency exchanges in quandl
- fix py3 bugs in dataframe functions
- fix type and source of Pricers to be compatible with GnuCash
- add a Price when entering a commodity Split
- set microsecond to 0 for all datetime
- add pandas for requirements-dev
- add tests for deletion of transaction and for dataframe functions

1.15 Version 0.10.1 (2015-11-29)

- refactor the validation mechanism to work well with autoflush=True
- add support to GLIST in KVP
- add new matching rule for GUID slots
- rename slot 'default_currency' to 'default-currency'
- add tests for single_transaction factory
- update ipython example with pandas dataframes

1.16 Version 0.10.0 (2015-11-18)

- first draft of splits_df and prices_df methods that bring the book data into pandas DataFrames
- add an ipython notebook to show the new dataframes methods
- save default_currency of a book in a slot (when book created by piecash) or use locale to retrieve the default_currency
- improve error handling for quandl queries (currency exchange rates)

1.17 Version 0.9.1 (2015-11-15)

- fix bug with unicode on MySQL

1.18 Version 0.9.0 (2015-11-15)

- ported to SQLAlchemy-1.0
- set autoflush=true on the SA session
- improved coverage above 90% for all modules
- setup coveralls.io and requires.io
- fix bugs discovered by improved testing

1.19 Version 0.8.4 (2015-11-14)

- use AppVeyor for Windows continuous integration and for .exe freezing
- fix bugs in tests suite where files were not properly closed
- add Book.close function to close properly files
- depend on enum-compat instead of directly enum34
- add simple script to import/export prices from a gnucash book

1.20 Version 0.8.3 (2015-11-01)

- fix issue #8 re enum34
- updated sqlalchemy dep to use latest 0.9 series

1.21 Version 0.8.2 (2015-05-09)

- implementing support for creating Customer, Vendor and Employee objects as well as taxtables

1.22 Version 0.8.1 (2015-05-03)

- get 100% coverage on transaction module (except for scheduled transactions)
- account.full_name returns now unicode string

1.23 Version 0.8.0 (2015-05-02)

- get 100% coverage on book and account module
- fix repr and str representations of all objects to be compatible py2 and py3

1.24 Version 0.7.6 (2015-05-01)

- fix version requirement for SA (<0.9.9) and SA-utils

1.25 Version 0.7.5 (2015-03-14)

- improve doc on installation on windows through conda
- add .gitattributes to exclude html from githug language detection algorithm
- update github project list
- refactor sqlite isolation level code
- fix setup.py to avoid sqlalchemy 0.9.9 (buggy version)
- fix requirements.txt to avoid sqlalchemy 0.9.9 (buggy version)

1.26 Version 0.7.4 (2015-03-09)

- remove some remaining print in code

1.27 Version 0.7.3 (2015-03-09)

- fix requirements to include ipython==2.3.1

1.28 Version 0.7.2 (2015-03-09)

- fix bug in doc (was using ledger_str instead of ledger)

1.29 Version 0.7.1 (2015-03-09)

- refactor ledger functionalities
- bug fixing
- read backup functionality (ie backup when opening a book in RW)

1.30 Version 0.7.0 (2015-02-12)

- Merge the GncSession and Book objects
- extract factory function into a factories module

1.31 Version 0.6.2 (2015-02-02)

- add reference to google groups
- disable acquiring lock on file

1.32 Version 0.6.1 (2015-02-01)

- fix: qif scripts was not included in package

1.33 Version 0.6.0 (2015-02-01)

- add a basic QIF exporter script as piecash_toqif
- implemented “Trading accounts”
- improved documentation
- other small api enhancements/changes

1.34 Version 0.5.11 (2015-01-12)

- add a ledger_str method to transaction to output transaction in the ledger-cli format
- add label to Decimal field in sqlalchemy expr
- add backup option when opening sqlite file in RW (enabled by default)
- renamed tx_guid to transaction_guid in Split field
- fix technical bug in validation of transaction

1.35 Version 0.5.10 (2015-01-05)

- add keywords to setup.py

1.36 Version 0.5.8 (2015-01-05)

- add notes to Transaction (via slot)
- removed standalone exe from git/package (as too large)

1.37 Version 0.5.7 (2015-01-04)

- add sign property on account
- raise NotImplementedError when creating an object is not "safe" (ie not __init__ and validators)
- renamed slot_collection to slots in kvp handling
- renamed field of Version + add explicit __init__
- updated test to add explicit __init__ when needed

1.38 Version 0.5.6 (2015-01-04)

- reordering of field definitions to match gnucash order (finished)
- add autoincr

1.39 Version 0.5.5 (2015-01-04)

- reordering of field definitions to match gnucash order (to complete)

1.40 Version 0.5.4 (2015-01-04)

- added back the order table in the declarations

1.41 Version 0.5.3 (2015-01-03)

- add support for schedule_transactions and lots (in terms of access to data, not business logic)
- improved doc

1.42 Version 0.5.2 (2015-01-03)

- reworked documentation
- moved Lot and ScheduledTransaction to transaction module + improved them
- improve slots support
- fixed minor bugs

1.43 Version 0.5.1 (2014-12-30)

- fixed changelog/what's new documentation

1.44 Version 0.5.0 (2014-12-30)

- improve relationship in business model
- fix account.placeholder validation in transaction/splits
- made all relationships dual (with back_populates instead of backref)

1.45 Version 0.4.4 (2014-12-28)

- fix bug in piecash_ledger (remove testing code)
- improve documentation of core objects
- fix dependencies for developers (requests)
- regenerate the github list of projects

1.46 Version 0.4.0 (2014-12-28)

- improve bumpr integration

1.47 Version 0.3.1

- renamed modules in piecash packages
- updated doc

1.48 Version 0.3.0

- ported to python 3.4
- refactored lot of classes
- improved documentation
- added helper functions:
  - `Commodity.create_currency_from_ISO()`
  - `Commodity.create_stock_from_symbol()`
  - `Commodity.update_prices()`
  - `Commodity.create_stock_accounts()`

Contents:

CHAPTER

TWO

DOCUMENTATION

This project provides a simple and pythonic interface to GnuCash files stored in SQL (sqlite3, Pandostgres and MySQL) for Linux and Windows (not tested on Mac OS).

piecash is a pure python package, tested on python 3.6/3.7/3.8, that can be used as an alternative to:

- the official python bindings (as long as no advanced book modifications and/or engine calculations are needed). This is specially useful on Windows where the official python bindings may be tricky to install or if you want to work with python 3.
- XML parsing/reading of XML GnuCash files if you prefer python over XML/XLST manipulations.

piecash is built on the excellent SQLAlchemy library and does not require the installation of GnuCash itself.

piecash allows you to:

- create a GnuCash book from scratch or edit an existing one
- create new accounts, transactions, etc or change (within some limits) existing objects.
- read/browse all objects through an intuitive interface

A simple example of a piecash script:

```
with open_book("example.gnucash") as book:
    # get default currency of book
    print( book.default_currency )  # ==> Commodity<CURRENCY:EUR>

    # iterating over all splits in all books and print the transaction description:
    for acc in book.accounts:
        for sp in acc.splits:
            print(sp.transaction.description)
```

As piecash is essentially a SQLAlchemy layer, it could be potentially reused by any web framework that has a SQLAlchemy interface to develop REST API or classical websites. It can also be used for reporting purposes.

The project has reached beta stage. Knowledge of SQLAlchemy is at this stage not anymore required to use it and/or to contribute to it. Some documentation for developers on the object model of GnuCash as understood by the author is available *here*.

Warning:

1) Always do a backup of your gnucash file/DB before using piecash.

2) Test first your script by opening your file in readonly mode (which is the default mode)

2.1 Installation

To install with pip:

```
$ pip install piecash
```

or to upgrade if piecash is already installed:

```
$ pip install -U piecash
```

piecash comes with 6 extra options (each option depends on extra packages that will be installed only if the option is chosen):

- pandas: install also pandas to use *piecash.core.book.Book.splits_df()* and *piecash.core.book.Book.prices_df()*
- yahoo: to retrieve quotes/prices
- postgres: to support connecting to a book saved on a postgresql database
- mysql: to support connecting to a book saved on a mysql database
- qif: to support export to QIF

For developers, two extra options:

- test: to install what is needed for testing piecash
- dev: to install what is needed for developing piecash (docs, . . .)

To install these options, simply specify them between brackets after the piecash package:

```
$ pip install piecash[pandas,qif,postgres]
```

To install with pipenv:

```
$ pipenv install piecash
```

Otherwise, you can install by unpacking the source distribution from PyPI and then:

```
$ python setup.py install
```

If you are on MS Windows and not so familiar with python, we would suggest you to install the miniconda python distribution from Continuum Analytics available at http://conda.pydata.org/miniconda.html (you can choose whatever version 3.X of python you would like) and then run the following command in the command prompt (cmd.exe):

```
$ conda create -n piecash_venv python=3 pip sqlalchemy
$ activate piecash_venv
$ pip install piecash
```

The first command create a new python environment named "piecash_venv" with python 3.7, pip and sqlalchemy installed.

The second command activates the newly created piecash_venv. Afterwards, you only need to execute this command before using python through the command line.

The third command installs piecash and its dependencies. piecash depends also on sqlalchemy but as the sqlalchemy package requires a compiler if it is installed through pip, we found it easier to install it through conda (this is done in the first command).

If you need to use directly the python interpreter in the newly created "piecash_env", you can find it installed in your user folder under Miniconda3\envs\piecash_venv\python.exe (or Miniconda2\...).

On OS X, this option may also be valuable.

2.2 Quickstart

The simplest workflow to use piecash starts by opening a GnuCash file

```
import piecash

# open a GnuCash Book
book = piecash.open_book("test.gnucash", readonly=True)
```

and then access GnuCash objects through the book, for example to query the stock prices

```
# example 1, print all stock prices in the Book
# display all prices
for price in book.prices:
    print(price)
```

```
<Price 2014-12-22 : 0.702755 EUR/CAD>
<Price 2014-12-19 : 0.695658 EUR/CAD>
<Price 2014-12-18 : 0.689026 EUR/CAD>
<Price 2014-12-17 : 0.69005 EUR/CAD>
<Price 2014-12-16 : 0.693247 EUR/CAD>
<Price 2014-12-22 : 51.15 USD/YHOO>
<Price 2014-12-19 : 50.88 USD/YHOO>
<Price 2014-12-18 : 50.91 USD/YHOO>
<Price 2014-12-17 : 50.12 USD/YHOO>
<Price 2014-12-16 : 48.85 USD/YHOO>
...
```

or to query the accounts:

```
for account in book.accounts:
    print(account)
```

```
Account<[EUR]>
Account<Assets[EUR]>
Account<Assets:Current Assets[EUR]>
Account<Assets:Current Assets:Checking Account[EUR]>
Account<Assets:Current Assets:Savings Account[EUR]>
Account<Assets:Current Assets:Cash in Wallet[EUR]>
Account<Income[EUR]>
Account<Income:Bonus[EUR]>
Account<Income:Gifts Received[EUR]>
...
Account<Expenses[EUR]>
Account<Expenses:Commissions[EUR]>
Account<Expenses:Adjustment[EUR]>
Account<Expenses:Auto[EUR]>
Account<Expenses:Auto:Fees[EUR]>
...
Account<Liabilities[EUR]>
```

(continues on next page)

(continued from previous page)

```
Account<Liabilities:Credit Card[EUR]>
Account<Equity[EUR]>
Account<Equity:Opening Balances[EUR]>
...
```

or to create a new expense account for utilities:

```
# retrieve currency
EUR = book.commodities.get(mnemonic='EUR')

# retrieve parent account
acc_exp = book.accounts.get(fullname="Expenses:Utilities")

# add a new subaccount to this account of type EXPENSE with currency EUR
new_acc = piecash.Account(name="Cable", type="EXPENSE", parent=acc_exp, commodity=EUR)

# save changes (it should raise an exception if we opened the book as readonly)
book.save()
```

Most basic objects used for personal finance are supported (Account, Split, Transaction, Price, ...).

2.3 The piecash command line interface

The *piecash* CLI offers the following features:

```
$ piecash -h
Usage: piecash [OPTIONS] COMMAND [ARGS]...

Options:
  -h, --help  Show this message and exit.

Commands:
  export      Exports GnuCash ENTITIES.
  ledger      Export to ledger-cli format.
  qif         Export to QIF format.
  sql-create  Create an empty book with gnucash
  sql-dump    Dump SQL schema of the gnucash sqlite book
```

To export specific entities out of a GnuCash book:

```
$ piecash export -h
Usage: piecash export [OPTIONS] BOOK [customers|vendors|prices]

  Exports GnuCash ENTITIES.

  This scripts export ENTITIES from the BOOK in a CSV format. When possible,
  it exports in a format that can be used to import the data into GnuCash.

  Remarks:
  - for customers and vendors, the format does not include an header
  - for prices, the format can be used with the `piecash import` command.

Options:
  --output FILENAME  File to which to export the data (default=stdout)
```

(continues on next page)

(continued from previous page)

```
  --inactive          Include inactive entities (for vendors and customers)
  -h, --help          Show this message and exit.
```

To export a GnuCash book to the ledger-cli format:

```
$ piecash ledger -h
Usage: piecash ledger [OPTIONS] BOOK

  Export to ledger-cli format.

  This scripts export a GnuCash BOOK to the ledget-cli format.

Options:
  --locale / --no-locale          Export currency amounts using locale for
                                  currencies format

  --commodity-notes / --no-commodity-notes
                                  Include the commodity_notes for the
                                  commodity (hledger does not support
                                  commodity commodity_notes

  --short-account-names / --no-short-account-names
                                  Use the short name for the accounts instead
                                  of the full hierarchical name.

  --output FILENAME               File to which to export the data
                                  (default=stdout)

  -h, --help                      Show this message and exit.
```

Or in python

```
In [1]: book = open_book(gnucash_books + "simple_sample.gnucash", open_if_lock=True)

In [2]: from piecash import ledger

# printing the ledger-cli (https://www.ledger-cli.org/) representation of the book
In [3]: print(ledger(book))
commodity EUR

account Asset
        check commodity == "EUR"

account Liability
        check commodity == "EUR"

account Income
        check commodity == "EUR"

account Expense
        check commodity == "EUR"

account Equity
        check commodity == "EUR"

account Equity:Opening Balances - EUR
        check commodity == "EUR"
```

(continues on next page)

(continued from previous page)

```
2014-11-30 Opening Balance
        Equity:Opening Balances - EUR             EUR -500.00
        Asset                                     EUR 500.00

2014-12-24 initial load
        Liability                                 EUR -1,000.00
        Asset                                     EUR 1,000.00

2014-12-24 expense 1
        Asset                                     EUR -200.00
        Expense                                   EUR 200.00

2014-12-24 income 1
        Income                                    EUR -150.00
        Asset                                     EUR 150.00

2014-12-24 loan payment
        Asset                                     EUR -130.00 ;   monthly payment
        Expense                                   EUR 30.00 ;   interest
        Liability                                 EUR 100.00 ;   capital



# printing the ledger-cli (https://www.ledger-cli.org/) representation of the book␣
↪using regional settings (locale) for currency output
In [4]: print(ledger(book, locale=True))
commodity €

account Asset
        check commodity == "EUR"

account Liability
        check commodity == "EUR"

account Income
        check commodity == "EUR"

account Expense
        check commodity == "EUR"

account Equity
        check commodity == "EUR"

account Equity:Opening Balances - EUR
        check commodity == "EUR"


2014-11-30 Opening Balance
        Equity:Opening Balances - EUR             -€500.00
        Asset                                     €500.00

2014-12-24 initial load
        Liability                                 -€1,000.00
        Asset                                     €1,000.00
```

(continues on next page)

(continued from previous page)

```
2014-12-24 expense 1
        Asset                                    -€200.00
        Expense                                  €200.00

2014-12-24 income 1
        Income                                   -€150.00
        Asset                                    €150.00

2014-12-24 loan payment
        Asset                                    -€130.00 ;   monthly payment
        Expense                                  €30.00 ;   interest
        Liability                                €100.00 ;   capital
```

For more information on how to use piecash, please refer to the Tutorials on *Using existing objects* and *Creating new objects*, the *Example scripts* or the *package documentation*.

CHAPTER

THREE

TUTORIAL : USING EXISTING OBJECTS

3.1 Opening an existing Book

To open an existing GnuCash document (and get the related *Book*), use the `open_book()` function:

```
import piecash

# for a sqlite3 document
book = piecash.open_book("existing_file.gnucash")

# or through an URI connection string for sqlite3
book = piecash.open_book(uri_conn="sqlite:///existing_file.gnucash")
# or for postgres
book = piecash.open_book(uri_conn="postgres://user:passwd@localhost/existing_gnucash_
↪db")
```

The documents are open as readonly per default. To allow RW access, specify explicitly readonly=False as:

```
book = piecash.open_book("existing_file.gnucash", readonly=False)
```

When opening in full access (readonly=False), piecash will automatically create a backup file named filename.piecash_YYYYMMDD_HHMMSS with the original file. To avoid creating the backup file, specificy do_backup=False as:

```
book = piecash.open_book("existing_file.gnucash", readonly=False, do_backup=False)
```

To force opening the file even through there is a lock on it, use the open_if_lock=True argument:

```
book = piecash.open_book("existing_file.gnucash", open_if_lock=True)
```

3.2 Access to objects

Once a GnuCash book is opened through a *piecash.core.book.Book*, GnuCash objects can be accessed through two different patterns:

The object model

> In this mode, we access elements through their natural relations, starting from the book and jumping from one object to the other:

```
In [1]: book = open_book(gnucash_books + "default_book.gnucash")

In [2]: book.root_account # accessing the root_account
Out[2]: Account<[EUR]>

In [3]: # looping through the children accounts of the root_account
   ...: for acc in book.root_account.children:
   ...:     print(acc)
   ...:
Account<Assets[EUR]>
Account<Liabilities[EUR]>
Account<Income[EUR]>
Account<Expenses[EUR]>
Account<Equity[EUR]>

# accessing children accounts
In [4]:
   ...: root = book.root_account               # select the root_account
   ...: assets = root.children(name="Assets")   # select child account by␣
↪name
   ...: cur_assets = assets.children[0]         # select child account by␣
↪index
   ...: cash = cur_assets.children(type="CASH") # select child account by␣
↪type
   ...: print(cash)
   ...:
Account<Assets:Current Assets:Cash in Wallet[EUR]>

In [5]: # get the commodity of an account
   ...: commo = cash.commodity
   ...: print(commo)
   ...:
Commodity<CURRENCY:EUR>

In [6]: # get first ten accounts linked to the commodity commo
   ...: for acc in commo.accounts[:10]:
   ...:     print(acc)
   ...:
Account<[EUR]>
Account<Assets[EUR]>
Account<Assets:Current Assets[EUR]>
Account<Assets:Current Assets:Checking Account[EUR]>
Account<Assets:Current Assets:Savings Account[EUR]>
Account<Assets:Current Assets:Cash in Wallet[EUR]>
Account<Liabilities[EUR]>
Account<Liabilities:Credit Card[EUR]>
Account<Income[EUR]>
Account<Income:Bonus[EUR]>
```

The “table” access

In this mode, we access elements through collections directly accessible from the book:

```
In [7]: book = open_book(gnucash_books + "default_book.gnucash")

# accessing all accounts
In [8]: book.accounts
Out[8]:
```

(continues on next page)

(continued from previous page)

```
[Account<Assets[EUR]>,
 Account<Assets:Current Assets[EUR]>,
 Account<Assets:Current Assets:Checking Account[EUR]>,
 Account<Assets:Current Assets:Savings Account[EUR]>,
 Account<Assets:Current Assets:Cash in Wallet[EUR]>,
 Account<Liabilities[EUR]>,
 Account<Liabilities:Credit Card[EUR]>,
 Account<Income[EUR]>,
 Account<Income:Bonus[EUR]>,
 Account<Income:Gifts Received[EUR]>,
 Account<Income:Interest Income[EUR]>,
 Account<Income:Interest Income:Checking Interest[EUR]>,
 Account<Income:Interest Income:Other Interest[EUR]>,
 Account<Income:Interest Income:Savings Interest[EUR]>,
 Account<Income:Other Income[EUR]>,
 Account<Income:Salary[EUR]>,
 Account<Expenses[EUR]>,
 Account<Expenses:Adjustment[EUR]>,
 Account<Expenses:Auto[EUR]>,
 Account<Expenses:Auto:Fees[EUR]>,
 Account<Expenses:Auto:Gas[EUR]>,
 Account<Expenses:Auto:Parking[EUR]>,
 Account<Expenses:Auto:Repair and Maintenance[EUR]>,
 Account<Expenses:Bank Service Charge[EUR]>,
 Account<Expenses:Books[EUR]>,
 Account<Expenses:Cable[EUR]>,
 Account<Expenses:Charity[EUR]>,
 Account<Expenses:Clothes[EUR]>,
 Account<Expenses:Computer[EUR]>,
 Account<Expenses:Dining[EUR]>,
 Account<Expenses:Education[EUR]>,
 Account<Expenses:Entertainment[EUR]>,
 Account<Expenses:Entertainment:Music/Movies[EUR]>,
 Account<Expenses:Entertainment:Recreation[EUR]>,
 Account<Expenses:Entertainment:Travel[EUR]>,
 Account<Expenses:Gifts[EUR]>,
 Account<Expenses:Groceries[EUR]>,
 Account<Expenses:Hobbies[EUR]>,
 Account<Expenses:Insurance[EUR]>,
 Account<Expenses:Insurance:Auto Insurance[EUR]>,
 Account<Expenses:Insurance:Health Insurance[EUR]>,
 Account<Expenses:Insurance:Life Insurance[EUR]>,
 Account<Expenses:Laundry/Dry Cleaning[EUR]>,
 Account<Expenses:Medical Expenses[EUR]>,
 Account<Expenses:Miscellaneous[EUR]>,
 Account<Expenses:Online Services[EUR]>,
 Account<Expenses:Phone[EUR]>,
 Account<Expenses:Public Transportation[EUR]>,
 Account<Expenses:Subscriptions[EUR]>,
 Account<Expenses:Supplies[EUR]>,
 Account<Expenses:Taxes[EUR]>,
 Account<Expenses:Taxes:Federal[EUR]>,
 Account<Expenses:Taxes:Medicare[EUR]>,
 Account<Expenses:Taxes:Other Tax[EUR]>,
 Account<Expenses:Taxes:Social Security[EUR]>,
 Account<Expenses:Taxes:State/Province[EUR]>,
 Account<Expenses:Utilities[EUR]>,
```

(continues on next page)

(continued from previous page)

```
 Account<Expenses:Utilities:Electric[EUR]>,
 Account<Expenses:Utilities:Garbage collection[EUR]>,
 Account<Expenses:Utilities:Gas[EUR]>,
 Account<Expenses:Utilities:Water[EUR]>,
 Account<Equity[EUR]>,
 Account<Equity:Opening Balances[EUR]>]

# accessing all commodities
In [9]: book.commodities
Out[9]: [Commodity<CURRENCY:EUR>]

# accessing all transactions
In [10]: book.transactions
Out[10]: []
```

Each of these collections can be either iterated or accessed through some indexation or filter mechanism (return first element of collection satisfying some criteria(s)):

```
# iteration
In [11]: for acc in book.accounts:
   ....:     if acc.type == "ASSET": print(acc)
   ....:
Account<Assets[EUR]>
Account<Assets:Current Assets[EUR]>

# indexation (not very meaningful)
In [12]: book.accounts[10]
Out[12]: Account<Income:Interest Income[EUR]>

# filter by name
In [13]: book.accounts(name="Garbage collection")
Out[13]: Account<Expenses:Utilities:Garbage collection[EUR]>

# filter by type
In [14]: book.accounts(type="EXPENSE")
Out[14]: Account<Expenses[EUR]>

# filter by fullname
In [15]: book.accounts(fullname="Expenses:Taxes:Social Security")
Out[15]: Account<Expenses:Taxes:Social Security[EUR]>

# filter by multiple criteria
In [16]: book.accounts(commodity=book.commodities[0], name="Gas")
Out[16]: Account<Expenses:Auto:Gas[EUR]>
```

The "SQLAlchemy" access (advanced users)

In this mode, we access elements through SQLAlchemy queries on the SQLAlchemy session:

```
# retrieve underlying SQLAlchemy session object
In [1]: session = book.session

# get all account with name >= "T"
In [2]: session.query(Account).filter(Account.name>="T").all()
Out[2]:
[Account<Expenses:Entertainment:Travel[EUR]>,
 Account<Expenses:Taxes[EUR]>,
```

(continues on next page)

(continued from previous page)

```
 Account<Expenses:Utilities[EUR]>,
 Account<Expenses:Utilities:Water[EUR]>,
 Account<>]

# display underlying query
In [3]: str(session.query(Account).filter(Account.name>="T"))
Out[3]: 'SELECT accounts.account_type AS accounts_account_type, accounts.
 ↪commodity_scu AS accounts_commodity_scu, accounts.non_std_scu AS accounts_
 ↪non_std_scu, accounts.placeholder AS accounts_placeholder, accounts.guid␣
 ↪AS accounts_guid, accounts.name AS accounts_name, accounts.commodity_guid␣
 ↪AS accounts_commodity_guid, accounts.parent_guid AS accounts_parent_guid,␣
 ↪accounts.code AS accounts_code, accounts.description AS accounts_
 ↪description, accounts.hidden AS accounts_hidden \nFROM accounts \nWHERE␣
 ↪accounts.name >= ?'
```

3.3 Accounts

Accessing the accounts (*piecash.core.account.Account*):

```
In [1]: book = open_book(gnucash_books + "simple_sample.gnucash", open_if_lock=True)

# accessing the root_account
In [2]: root = book.root_account

In [3]: print(root)
Account<>

# accessing the first children account of a book
In [4]: acc = root.children[0]

In [5]: print(acc)
Account<Asset[EUR]>

# accessing attributes of an account
In [6]: print(f"Account name={acc.name}\n"
   ...:       f"         commodity={acc.commodity.namespace}/{acc.commodity.mnemonic}\n
 ↪"
   ...:       f"         fullname={acc.fullname}\n"
   ...:       f"         type={acc.type}")
   ...:
Account name=Asset
         commodity=CURRENCY/EUR
         fullname=Asset
         type=ASSET

# calculating the balance of the accounts:
In [7]: for acc in root.children:
   ...:     print(f"Account balance for {acc.name}: {acc.get_balance()} (without sign␣
 ↪reversal: {acc.get_balance(natural_sign=False)}")
   ...:
Account balance for Asset: 1320 (without sign reversal: 1320
Account balance for Liability: 900 (without sign reversal: -900
Account balance for Income: 150 (without sign reversal: -150
Account balance for Expense: 230 (without sign reversal: 230
```

(continues on next page)

(continued from previous page)

```
Account balance for Equity: 500 (without sign reversal: -500

# accessing all splits related to an account
In [8]: for sp in acc.splits:
   ...:     print(f"account <{acc.fullname}> is involved in transaction '{sp.
↪transaction.description}'")
   ...:
```

3.4 Commodities and Prices

The list of all commodities in the book can be retrieved via the `commodities` attribute:

```
In [1]: book = open_book(gnucash_books + "book_prices.gnucash", open_if_lock=True)

# all commodities
In [2]: print(book.commodities)
[Commodity<CURRENCY:EUR>, Commodity<NASDAQ:FB>, Commodity<template:template>,␣
↪Commodity<PAR:ENGI.PA>, Commodity<NYQ:KO>, Commodity<CURRENCY:USD>]

In [3]: cdty = book.commodities[0]

# accessing attributes of a commodity
In [4]: print("Commodity namespace={cdty.namespace}\n"
   ...:       "          mnemonic={cdty.mnemonic}\n"
   ...:       "          cusip={cdty.cusip}\n"
   ...:       "          fraction={cdty.fraction}".format(cdty=cdty))
   ...:
Commodity namespace=CURRENCY
          mnemonic=EUR
          cusip=978
          fraction=100
```

The prices (*piecash.core.commodity.Price*) of a commodity can be iterated through the `prices` attribute:

```
# loop on the prices
In [1]: for cdty in book.commodities:
   ...:     for pr in cdty.prices:
   ...:         print("Price date={pr.date}"
   ...:               "      value={pr.value} {pr.currency.mnemonic}/{pr.commodity.
↪mnemonic}".format(pr=pr))
   ...:
Price date=2018-02-10      value=1.730120457024597578303288749E-7 EUR/FB
Price date=2018-02-11      value=54 EUR/FB
Price date=2018-02-13      value=3.808073 EUR/FB
Price date=2018-02-12      value=5 EUR/FB
Price date=2018-02-15      value=3.923077 EUR/FB
```

3.5 Transactions and Splits

The list of all transactions in the book can be retrieved via the `transactions` attribute:

```
In [1]: book = open_book(gnucash_books + "book_schtx.gnucash", open_if_lock=True)

# all transactions (including transactions part of a scheduled transaction␣
↪description)
In [2]: for tr in book.transactions:
   ...:     print(tr)
   ...:
Transaction<[EUR] 'Monthly utility bill' on 2015-01-02>
Transaction<[EUR] 'Insurance' on 2015-01-02>
Transaction<[EUR] 'Monthly utility bill' on 2013-12-31 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2014-02-02 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2014-03-02 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2014-03-31 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2014-04-30 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2014-06-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2014-06-30 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2014-07-31 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2014-08-31 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2014-09-30 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2014-11-02 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2014-11-30 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2014-12-31 (from sch tx)>
Transaction<[EUR] 'Insurance' on 2013-05-31 (from sch tx)>
Transaction<[EUR] 'Insurance' on 2014-05-29 (from sch tx)>
Transaction<[EUR] 'Opening balance' on 2013-01-02>
Transaction<[EUR] 'Monthly utility bill' on 2015-02-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2015-03-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2015-03-31 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2015-04-30 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2015-05-31 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2015-06-30 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2015-08-02 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2015-08-31 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2015-09-30 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2015-11-01 (from sch tx)>
Transaction<[EUR] 'Insurance' on 2015-05-31 (from sch tx)>
Transaction<[EUR] 'test' on 2015-11-17>
Transaction<[EUR] 'salary' on 2013-12-31>
Transaction<[EUR] 'salary' on 2014-12-31>
Transaction<[EUR] 'Monthly utility bill' on 2015-12-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2016-01-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2016-02-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2016-03-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2016-04-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2016-05-02 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2016-06-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2016-07-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2016-08-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2016-09-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2016-10-03 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2016-11-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2016-12-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2017-01-02 (from sch tx)>
```

(continues on next page)

(continued from previous page)

```
Transaction<[EUR] 'Monthly utility bill' on 2017-02-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2017-03-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2017-04-03 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2017-05-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2017-06-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2017-07-03 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2017-08-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2017-09-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2017-10-02 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2017-11-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2017-12-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2018-01-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2018-02-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2018-03-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2018-04-02 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2018-05-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2018-06-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2018-07-02 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2018-08-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2018-09-03 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2018-10-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2018-11-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2018-12-03 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2019-01-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2019-02-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2019-03-01 (from sch tx)>
Transaction<[EUR] 'Monthly utility bill' on 2019-04-01 (from sch tx)>
Transaction<[EUR] 'Insurance' on 2016-06-01 (from sch tx)>
Transaction<[EUR] 'Insurance' on 2017-06-01 (from sch tx)>
Transaction<[EUR] 'Insurance' on 2018-06-01 (from sch tx)>

# selecting first transaction generated from a scheduled transaction
In [3]: tr = [ tr for tr in book.transactions if tr.scheduled_transaction ][0]
```

For a given transaction, the following attributes are accessible:

```
# accessing attributes of a transaction
In [1]: print("Transaction description='{tr.description}'\n"
   ...:       "            currency={tr.currency}\n"
   ...:       "            post_date={tr.post_date}\n"
   ...:       "            enter_date={tr.enter_date}".format(tr=tr))
   ...:
Transaction description='Monthly utility bill'
            currency=Commodity<CURRENCY:EUR>
            post_date=2013-12-31
            enter_date=2015-01-03 08:18:13+00:00

# accessing the splits of the transaction
In [2]: tr.splits
Out[2]:
[Split<Account<Assets:Current Assets:Checking Account[EUR]> -70 EUR>,
 Split<Account<Expenses:Utilities:Electric[EUR]> 30 EUR>,
 Split<Account<Expenses:Utilities:Gas[EUR]> 40 EUR>]

# identifying which split is a credit or a debit
In [3]: for sp in tr.splits:
   ...:     split_type = "credit" if sp.is_credit else "debit"
```

(continues on next page)

(continued from previous page)

```
   ...:     print(f"{sp} is a {split_type}")
   ...:
Split<Account<Assets:Current Assets:Checking Account[EUR]> -70 EUR> is a credit
Split<Account<Expenses:Utilities:Electric[EUR]> 30 EUR> is a debit
Split<Account<Expenses:Utilities:Gas[EUR]> 40 EUR> is a debit

# accessing the scheduled transaction
In [4]: [ sp for sp in tr.scheduled_transaction.template_account.splits]
Out[4]:
[SplitTemplate<Account<Assets:Current Assets:Checking Account[EUR]> credit=70 >,
 SplitTemplate<Account<Expenses:Utilities:Electric[EUR]>  debit=30>,
 SplitTemplate<Account<Expenses:Utilities:Gas[EUR]>  debit=40>]

# closing the book
In [5]: book.close()
```

3.6 Invoices

The list of all invoices in the book can be retrieved via the `invoices` attribute:

```
In [1]: book = open_book(gnucash_books + "invoices.gnucash", open_if_lock=True)

# all invoices
In [2]: for invoice in book.invoices:
   ...:     print(invoice)
   ...:
Invoice<000001>
```

3.7 Other objects

In fact, any object can be retrieved from the session through a generic `get(**kwargs)` method:

```
In [1]: book = open_book(gnucash_books + "invoices.gnucash", open_if_lock=True)

In [2]: from piecash import Account, Commodity, Budget, Vendor

# accessing specific objects through the get method
In [3]: book.get(Account, name="Assets", parent=book.root_account)
Out[3]: Account<Assets[EUR]>

In [4]: book.get(Commodity, namespace="CURRENCY", mnemonic="EUR")
Out[4]: Commodity<CURRENCY:EUR>

In [5]: book.get(Budget, name="my first budget")
Out[5]: Budget<my first budget() for 12 periods following pattern 'month*1 from 2019-
↪04-01 [none]' >

In [6]: book.get(Vendor, name="Looney")
Out[6]: Vendor<000001:Looney>
```

If you know SQLAlchemy, you can get access to the underlying `Session` as `book.session` and execute queries using the piecash classes:

```
In [7]: from piecash import Account, Commodity, Budget, Vendor

# get the SQLAlchemy session
In [8]: session = book.session

# loop through all invoices
In [9]: for invoice in session.query(Invoice).all():
   ...:     print(invoice.notes)
   ...:
```

Note: Easy access to objects from *piecash.business* and *piecash.budget* could be given directly from the session in future versions if deemed useful.

3.8 Working with slots

With regard to slots, GnuCash objects and Frames behave as dictionaries and all values are automatically converted back and forth to python objects:

```
In [1]: import datetime, decimal

In [2]: book = create_book()

# retrieve list of slots
In [3]: print(book.slots)
[]

# set slots
In [4]: book["myintkey"] = 3

In [5]: book["mystrkey"] = "hello"

In [6]: book["myboolkey"] = True

In [7]: book["mydatekey"] = datetime.datetime.today().date()

In [8]: book["mydatetimekey"] = datetime.datetime.today()

In [9]: book["mynumerickey"] = decimal.Decimal("12.34567")

In [10]: book["account"] = book.root_account

# iterate over all slots
In [11]: for k, v in book.iteritems():
   ....:     print("slot={v} has key={k} and value={v.value} of type {t}".format(k=k,
 ↪v=v,t=type(v.value)))
   ....:
slot=<SlotInt myintkey=3> has key=myintkey and value=3 of type <class 'int'>
slot=<SlotString mystrkey='hello'> has key=mystrkey and value=hello of type <class
 ↪'str'>
slot=<SlotInt myboolkey=True> has key=myboolkey and value=True of type <class 'bool'>
slot=<SlotDate mydatekey=datetime.date(2020, 10, 24)> has key=mydatekey and␣
 ↪value=2020-10-24 of type <class 'datetime.date'>
slot=<SlotTime mydatetimekey=datetime.datetime(2020, 10, 24, 19, 14, 9, 274380)> has␣
 ↪key=mydatetimekey and value=2020-10-24 19:14:09.274380 of type <class 'datetime.
 ↪datetime'>
```

(continues on next page)

(continued from previous page)

```
slot=<SlotNumeric mynumerickey=Decimal('12.34567')> has key=mynumerickey and value=12.
↪34567 of type <class 'decimal.Decimal'>
slot=<SlotGUID account=Account<[EUR]>> has key=account and value=Account<[EUR]> of␣
↪type <class 'piecash.core.account.Account'>

# delete a slot
In [12]: del book["myintkey"]

# delete all slots
In [13]: del book[:]

# create a key/value in a slot frames (and create them if they do not exist)
In [14]: book["options/Accounts/Use trading accounts"]="t"

# access a slot in frame in whatever notations
In [15]: s1=book["options/Accounts/Use trading accounts"]

In [16]: s2=book["options"]["Accounts/Use trading accounts"]

In [17]: s3=book["options/Accounts"]["Use trading accounts"]

In [18]: s4=book["options"]["Accounts"]["Use trading accounts"]

In [19]: assert s1==s2==s3==s4
```

Slots of type GUID use the name of the slot to do the conversion back and forth between an object and its guid. For these slots, there is an explicit mapping between slot names and object types.

CHAPTER

FOUR

TUTORIAL : CREATING NEW OBJECTS

4.1 Creating a new Book

piecash can create a new GnuCash document (a *Book*) from scratch through the `create_book()` function.

To create a in-memory sqlite3 document (useful to test piecash for instance), a simple call is enough:

```
In [1]: import piecash

In [2]: book = piecash.create_book()
```

To create a file-based sqlite3 document:

```
In [3]: book = piecash.create_book("example_file.gnucash")

# or equivalently (adding the overwrite=True argument to overwrite the file if it␣
↪already exists)
In [4]: book = piecash.create_book(sqlite_file="example_file.gnucash", overwrite=True)

# or equivalently
In [5]: book = piecash.create_book(uri_conn="sqlite:///example_file.gnucash",␣
↪overwrite=True)
```

and for a postgres document (needs a pacakge installable via "pip install psycopg2"):

```
book = piecash.create_book(uri_conn="postgres://user:passwd@localhost/example_gnucash_
↪db")
```

Note: Per default, the currency of the document is the euro (EUR) but you can specify any other ISO currency through its ISO symbol:

```
In [6]: book = piecash.create_book(sqlite_file="example_file.gnucash",
   ...:                            currency="USD",
   ...:                            overwrite=True)
   ...:
```

If the document already exists, piecash will raise an exception. You can force piecash to overwrite an existing file/database (i.e. delete it and then recreate it) by passing the overwrite=True argument:

```
In [1]: book = piecash.create_book(sqlite_file="example_file.gnucash", overwrite=True)
```

4.2 Creating a new Account

piecash can create new accounts (a *piecash.core.account.Account*):

```
In [1]: from piecash import create_book, Account

In [2]: book = create_book(currency="EUR")

# retrieve the default currency
In [3]: EUR = book.commodities.get(mnemonic="EUR")

# creating a placeholder account
In [4]: acc = Account(name="My account",
   ...:               type="ASSET",
   ...:               parent=book.root_account,
   ...:               commodity=EUR,
   ...:               placeholder=True,)
   ...:

# creating a detailed sub-account
In [5]: subacc = Account(name="My sub account",
   ...:                  type="BANK",
   ...:                  parent=acc,
   ...:                  commodity=EUR,
   ...:                  commodity_scu=1000,
   ...:                  description="my bank account",
   ...:                  code="FR013334...",)
   ...:

In [6]: book.save()

In [7]: book.accounts
Out[7]: [Account<My account[EUR]>, Account<My account:My sub account[EUR]>]
```

4.3 Creating a new Commodity

piecash can create new commodities (a *piecash.core.commodity.Commodity*):

```
In [1]: from piecash import create_book, Commodity, factories

# create a book (in memory) with some currency
In [2]: book = create_book(currency="EUR")

In [3]: print(book.commodities)
[Commodity<CURRENCY:EUR>]

# creating a new ISO currency (if not already available in s.commodities) (warning,␣
↪object should be manually added to session)
In [4]: USD = factories.create_currency_from_ISO("USD")

In [5]: book.add(USD) # add to session

# create a commodity (lookup on yahoo! finance, need web access)
# (warning, object should be manually added to session if book kwarg is not included␣
↪in constructor)
```

(continues on next page)

(continued from previous page)

```
# DOES NOT WORK ANYMORE DUE TO CLOSING OF YAHOO!FINANCE
# apple = factories.create_stock_from_symbol("AAPL", book)
# creating commodities using the constructor
# (warning, object should be manually added to session if book kwarg is not included␣
↪in constructor)
# create a special "reward miles" Commodity using the constructor without book kwarg
In [6]: miles = Commodity(namespace="LOYALTY", mnemonic="Miles", fullname="Reward␣
↪miles", fraction=1000000)

In [7]: book.add(miles) # add to session

# create a special "unicorn hugs" Commodity using the constructor with book kwarg
In [8]: unhugs = Commodity(namespace="KINDNESS", mnemonic="Unhugs", fullname="Unicorn␣
↪hugs", fraction=1, book=book)

In [9]: USD, miles, unhugs
Out[9]: (Commodity<CURRENCY:USD>, Commodity<LOYALTY:Miles>, Commodity<KINDNESS:Unhugs>
↪)
```

Warning: The following (creation of non ISO currencies) is explicitly forbidden by the GnuCash application.

```
# create a bitcoin currency (warning, max 6 digits after comma, current GnuCash␣
↪limitation)
In [1]: XBT = Commodity(namespace="CURRENCY", mnemonic="XBT", fullname="Bitcoin",␣
↪fraction=1000000)

In [2]: book.add(XBT) # add to session

In [3]: XBT
Out[3]: Commodity<CURRENCY:XBT>
```

4.4 Creating a new Transaction

piecash can create new transactions (a *piecash.core.transaction.Transaction*):

```
In [1]: from piecash import create_book, Account, Transaction, Split,␣
↪GncImbalanceError, factories, ledger

# create a book (in memory)
In [2]: book = create_book(currency="EUR")

# get the EUR and create the USD currencies
In [3]: c1 = book.default_currency

In [4]: c2 = factories.create_currency_from_ISO("USD")

# create two accounts
In [5]: a1 = Account("Acc 1", "ASSET", c1, parent=book.root_account)

In [6]: a2 = Account("Acc 2", "ASSET", c2, parent=book.root_account)
```

(continues on next page)

(continued from previous page)

```
# create a transaction from a1 to a2
In [7]: tr = Transaction(currency=c1,
   ...:                  description="transfer",
   ...:                  splits=[
   ...:                      Split(account=a1, value=-100),
   ...:                      Split(account=a2, value=100, quantity=30)
   ...:                  ])
   ...:

In [8]: book.flush()

# ledger() returns a representation of the transaction in the ledger-cli format
In [9]: print(ledger(tr))
2020-10-24 transfer
        Acc 1                                    EUR -100.00
        Acc 2                                    USD 30.00 @@ EUR 100.00


# change the book to use the "trading accounts" options
In [10]: book.use_trading_accounts = True

# add a new transaction identical to the previous
In [11]: tr2 = Transaction(currency=c1,
   ....:                   description="transfer 2",
   ....:                   splits=[
   ....:                       Split(account=a1, value=-100),
   ....:                       Split(account=a2, value=100, quantity=30)
   ....:                   ])
   ....:

In [12]: print(ledger(tr2))
2020-10-24 transfer 2
        Acc 1                                    EUR -100.00
        Acc 2                                    USD 30.00 @@ EUR 100.00


# when flushing, the trading accounts are created
In [13]: book.flush()

In [14]: print(ledger(tr2))
2020-10-24 transfer 2
        Acc 1                                    EUR -100.00
        Acc 2                                    USD 30.00 @@ EUR 100.00


# trying to create an unbalanced transaction trigger an exception
# (there is not automatic creation of an imbalance split)
In [15]: tr3 = Transaction(currency=c1,
   ....:                   description="transfer imb",
   ....:                   splits=[
   ....:                       Split(account=a1, value=-100),
   ....:                       Split(account=a2, value=90, quantity=30)
   ....:                   ])
   ....:

In [16]: print(ledger(tr3))
2020-10-24 transfer imb
```

(continues on next page)

(continued from previous page)

```
        Acc 1                                          EUR -100.00
        Acc 2                                          USD 30.00 @@ EUR 90.00


In [17]: try:
   ....:     book.flush()
   ....: except GncImbalanceError:
   ....:     print("Indeed, there is an imbalance !")
   ....:
```

4.5 Creating new Business objects

piecash can create new 'business' objects (this is a work in progress).

To create a new customer (a *piecash.business.person.Customer*):

```
In [1]: from piecash import create_book, Customer, Address

# create a book (in memory)
In [2]: b = create_book(currency="EUR")

# get the currency
In [3]: eur = b.default_currency

# create a customer
In [4]: c1 = Customer(name="Mickey", currency=eur, address=Address(addr1="Sesame␣
↪street 1", email="mickey@example.com"))

# the customer has not yet an ID
In [5]: c1
Out[5]: Customer<None:Mickey>

# we add it to the book
In [6]: b.add(c1)

# flush the book
In [7]: b.flush()

# the customer gets its ID
In [8]: print(c1)
Customer<000001:Mickey>

# or create a customer directly in a book (by specifying the book argument)
In [9]: c2 = Customer(name="Mickey", currency=eur, address=Address(addr1="Sesame␣
↪street 1", email="mickey@example.com"),
   ...:               book=b)
   ...:

# the customer gets immediately its ID
In [10]: c2
Out[10]: Customer<000002:Mickey>

# the counter of the ID is accessible as
In [11]: b.counter_customer
```

(continues on next page)

(continued from previous page)

```
Out[11]: 2

In [12]: b.save()
```

Similar functions are available to create new vendors (*piecash.business.person.Vendor*) or employees (*piecash.business.person.Employee*).

There is also the possibility to set taxtables for customers or vendors as:

```
In [1]: from piecash import Taxtable, TaxtableEntry

In [2]: from decimal import Decimal

# let us first create an account to which link a tax table entry
In [3]: acc = Account(name="MyTaxAcc", parent=b.root_account, commodity=b.
↪currencies(mnemonic="EUR"), type="ASSET")

# then create a table with on entry (6.5% on previous account
In [4]: tt = Taxtable(name="local taxes", entries=[
   ...:     TaxtableEntry(type="percentage",
   ...:                   amount=Decimal("6.5"),
   ...:                   account=acc),
   ...: ])
   ...:

# and finally attach it to a customer
In [5]: c2.taxtable = tt

In [6]: b.save()

In [7]: print(b.taxtables)
[TaxTable<local taxes:['TaxEntry<6.5 percentage in MyTaxAcc>']>]
```

CHAPTER

FIVE

EXAMPLES OF PROGRAMS WRITTEN WITH PIECASH

You can find examples of programs/scripts (loosely based on the scripts for the official python bindings for gnucash or on questions posted on the mailing list) in the examples subfolder.

5.1 Creating and opening gnucash files

```
from __future__ import print_function
import os
import tempfile

from piecash import open_book, create_book, GnucashException


FILE_1 = os.path.join(tempfile.gettempdir(), "not_there.gnucash")
FILE_2 = os.path.join(tempfile.gettempdir(), "example_file.gnucash")

if os.path.exists(FILE_2):
    os.remove(FILE_2)

# open a file that isn't there, detect the error
try:
    book = open_book(FILE_1)
except GnucashException as backend_exception:
    print("OK", backend_exception)

# create a new file, this requires a file type specification
with create_book(FILE_2) as book:
    pass

# open the new file, try to open it a second time, detect the lock
# using the session as context manager automatically release the lock and close the␣
↪session
with open_book(FILE_2) as book:
    try:
        with open_book(FILE_2) as book_2:
            pass
    except GnucashException as backend_exception:
        print("OK", backend_exception)

os.remove(FILE_2)
```

5.2 Creating an account

```
#!/usr/bin/env python
##  @file
#   @brief Example Script simple sqlite create
#   @ingroup python_bindings_examples

from __future__ import print_function
import os

from piecash import create_book, Account, Commodity, open_book
from piecash.core.factories import create_currency_from_ISO

filename = os.path.abspath('test.blob')
if os.path.exists(filename):
    os.remove(filename)

with create_book(filename) as book:
    a = Account(parent=book.root_account,
                name="wow",
                type="ASSET",
                commodity=create_currency_from_ISO("CAD"))

    book.save()

with open_book(filename) as book:
    print(book.root_account.children)
    print(book.commodities.get(mnemonic="CAD"))

os.remove(filename)
```

5.3 Creating a transaction

```
#!/usr/bin/env python
# # @file
# @brief Creates a basic set of accounts and a couple of transactions
# @ingroup python_bindings_examples
from decimal import Decimal
import os
import tempfile

from piecash import create_book, Account, Transaction, Split, Commodity
from piecash.core.factories import create_currency_from_ISO

FILE_1 = os.path.join(tempfile.gettempdir(), "example.gnucash")

with create_book(FILE_1, overwrite=True) as book:
    root_acct = book.root_account
    cad = create_currency_from_ISO("CAD")
    expenses_acct = Account(parent=root_acct,
                            name="Expenses",
                            type="EXPENSE",
                            commodity=cad)
    savings_acct = Account(parent=root_acct,
```

(continues on next page)

(continued from previous page)

```
                            name="Savings",
                            type="BANK",
                            commodity=cad)
    opening_acct = Account(parent=root_acct,
                            name="Opening Balance",
                            type="EQUITY",
                            commodity=cad)
    num1 = Decimal("4")
    num2 = Decimal("100")
    num3 = Decimal("15")

    # create transaction with core objects in one step
    trans1 = Transaction(currency=cad,
                         description="Groceries",
                         splits=[
                             Split(value=num1, account=expenses_acct),
                             Split(value=-num1, account=savings_acct),
                         ])

    # create transaction with core object in multiple steps
    trans2 = Transaction(currency=cad,
                         description="Opening Savings Balance")

    split3 = Split(value=num2,
                   account=savings_acct,
                   transaction=trans2)

    split4 = Split(value=-num2,
                   account=opening_acct,
                   transaction=trans2)

    # create transaction with factory function
    from piecash.core.factories import single_transaction
    trans3 = single_transaction(None,None,"Pharmacy", num3, savings_acct, expenses_
↪acct)

    book.save()
```

5.4 Simple changes on a newly created book

```
from __future__ import print_function
from piecash import create_book

# create by default an in memory sqlite version
book = create_book(echo=False)

print("Book is saved:", book.is_saved, end=' ')
print(" ==> book description:", book.root_account.description)

print("changing description...")
book.root_account.description = "hello, book"
print("Book is saved:", book.is_saved, end=' ')
```

(continues on next page)

(continued from previous page)

```
print(" ==> book description:", book.root_account.description)

print("saving...")
book.save()

print("Book is saved:", book.is_saved, end=' ')
print(" ==> book description:", book.root_account.description)

print("changing description...")
book.root_account.description = "nevermind, book"
print("Book is saved:", book.is_saved, end=' ')
print(" ==> book description:", book.root_account.description)

print("cancel...")
book.cancel()

print("Book is saved:", book.is_saved, end=' ')
print(" ==> book description:", book.root_account.description)
```

5.5 Create a book with some accounts and add a transaction

```
from piecash import create_book, Account

# create a book with some account tree structure
with create_book("../gnucash_books/simple_book_transaction_creation.gnucash",␣
↪overwrite=True) as mybook:
    mybook.root_account.children = [
        Account(name="Expenses",
                type="EXPENSE",
                commodity=mybook.currencies(mnemonic="USD"),
                placeholder=True,
                children=[
                    Account(name="Some Expense Account",
                            type="EXPENSE",
                            commodity=mybook.currencies(mnemonic="USD")),
                ]),
        Account(name="Assets",
                type="ASSET",
                commodity=mybook.currencies(mnemonic="USD"),
                placeholder=True,
                children=[
                    Account(name="Current Assets",
                            type="BANK",
                            commodity=mybook.currencies(mnemonic="USD"),
                            placeholder=True,
                            children=[
                                Account(name="Checking",
                                        type="BANK",
                                        commodity=mybook.currencies(mnemonic="USD"))
                            ]),
                ]),
    ]
    # save the book
    mybook.save()
```

(continues on next page)

(continued from previous page)

```
from piecash import open_book, Transaction, Split
from datetime import datetime
from decimal import Decimal

# reopen the book and add a transaction
with open_book("../gnucash_books/simple_book_transaction_creation.gnucash",
               open_if_lock=True,
               readonly=False) as mybook:
    today = datetime.now()
    # retrieve the currency from the book
    USD = mybook.currencies(mnemonic="USD")
    # define the amount as Decimal
    amount = Decimal("25.35")
    # retrieve accounts
    to_account = mybook.accounts(fullname="Expenses:Some Expense Account")
    from_account = mybook.accounts(fullname="Assets:Current Assets:Checking")
    # create the transaction with its two splits
    Transaction(
        post_date=today,
        enter_date=today,
        currency=USD,
        description="Transaction Description!",
        splits=[
            Split(account=to_account,
                  value=amount,
                  memo="Split Memo!"),
            Split(account=from_account,
                  value=-amount,
                  memo="Other Split Memo!"),
        ]
    )
    # save the book
    mybook.save()

from piecash import ledger

# check the book by exporting to ledger format
with open_book("../gnucash_books/simple_book_transaction_creation.gnucash",
               open_if_lock=True) as mybook:
    print(ledger(mybook))
```

5.6 Extract Split information as pandas DataFrame

```
from piecash import open_book

# open a book
with open_book("../gnucash_books/simple_sample.gnucash", open_if_lock=True) as mybook:
    # print all splits in account "Asset"
    asset = mybook.accounts(fullname="Asset")
    for split in asset.splits:
        print(split)

    # extract all split information to a pandas DataFrame
```

(continues on next page)

(continued from previous page)

```
    df = mybook.splits_df()

    # print for account "Asset" some information on the splits
    print(df.loc[df["account.fullname"] == "Asset", ["transaction.post_date", "value
↪"]])
```

5.7 Filtered transaction reports

```
from __future__ import print_function
import datetime
import re
import os.path

from piecash import open_book


if __name__=='__main__':
    this_folder = os.path.dirname(os.path.realpath(__file__))
    s = open_book(os.path.join(this_folder, "..", "gnucash_books", "simple_sample.
↪gnucash"), open_if_lock=True)
else:
    s = open_book(os.path.join("gnucash_books", "simple_sample.gnucash"), open_if_
↪lock=True)

# get default currency
print(s.default_currency)

regex_filter = re.compile("^/Rental/")

# retrieve relevant transactions
transactions = [tr for tr in s.transactions  # query all transactions in the book/
↪session and filter them on
                if (regex_filter.search(tr.description)  # description field matching␣
↪regex
                    or any(regex_filter.search(spl.memo) for spl in tr.splits))  # or␣
↪memo field of any split of transaction
                and tr.post_date.date() >= datetime.date(2014, 11, 1)]  # and with␣
↪post_date no later than begin nov.


# output report with simple 'print'
print("Here are the transactions for the search criteria '{}':".format(regex_filter.
↪pattern))
for tr in transactions:
    print("- {:%Y/%m/%d} : {}".format(tr.post_date, tr.description))
    for spl in tr.splits:
        print("\t{amount}  {direction}  {account} : {memo}".format(amount=abs(spl.
↪value),
                                                                   direction="-->" if␣
↪spl.value > 0 else "<--",
                                                                   account=spl.
↪account.fullname,
                                                                   memo=spl.memo))
```

(continues on next page)

(continued from previous page)

```
# same with jinja2 templates
try:
    import jinja2
except ImportError:
    print("\n\t*** Install jinja2 ('pip install jinja2') to test the jinja2 template␣
↪version ***\n")
    jinja2 = None

if jinja2:
    env = jinja2.Environment(trim_blocks=True, lstrip_blocks=True)
    print(env.from_string("""
    Here are the transactions for the search criteria '{{regex.pattern}}':
    {% for tr in transactions %}
    - {{ tr.post_date.strftime("%Y/%m/%d") }} : {{ tr.description }}
      {% for spl in tr.splits %}
        {{ spl.value.__abs__() }} {% if spl.value < 0 %} --> {% else %} <-- {% endif
↪%} {{ spl.account.fullname }} : {{ spl.memo }}
      {% endfor %}
    {% endfor %}
    """).render(transactions=transactions,
                regex=regex_filter))
```

CHAPTER

SIX

PIECASH AND THE OFFICIAL PYTHON BINDINGS

piecash is an alternative to the python bindings that may be bundled with gnucash (http://wiki.gnucash.org/wiki/Python_Bindings).

This page aims to give some elements of comparison between both python interfaces to better understand their relevancy to your needs. Information on the official python bindings may be incomplete (information gathered from mailing lists and wiki).

6.1 Gnucash 3.0.x series

	piecash (>=1.0.0)	official python bindings (gnucash 3.0.n)
book format	gnucash 3.0.n	gnucash 3.0.n
environment	Python 3.6/3.7/3.8/3.9	Python 3
installation	pure python package 'pip install piecash'	compilation (difficult on windows) binaries (available on Linux)
requires GnuCash	no	yes
runs on Android	yes	no
gnucash files	SQL backend only	SQL backend and XML
documentation	yes (read the docs) actively developed	partial
functionalities	creation of new books read/browse objects create objects (basic) update online prices	all functionalities provided by the GnuCash C/C++ engine

6.2 Gnucash 2.6.x series

	piecash (<=0.18.0)	official python bindings (gnucash 2.6.n)
book format	gnucash 2.6.n	gnucash 2.6.n
environment	Python 2.7 & 3.3/3.4/3.5/3.6	Python 2.7
installation	pure python package 'pip install piecash'	compilation (difficult on windows) binaries (available on Linux)
requires GnuCash	no	yes
runs on Android	yes	no
gnucash files	SQL backend only	SQL backend and XML
documentation	yes (read the docs) actively developed	partial
functionalities	creation of new books read/browse objects create objects (basic) update online prices	all functionalities provided by the GnuCash C/C++ engine

CHAPTER

SEVEN

PIECASH ON ANDROID

piecash can successfully run on android which opens interesting opportunities!

7.1 Installing termux

First, you have to install Termux from the Play Store.

You start Termux and:

1. edit your .bash_profile with:

```
export TZ=$(getprop persist.sys.timezone)
export SHELL=$(which bash)
```

2. add the folder ~/storage with access to your android folders (also accessible via USB sync):

```
termux-setup-storage
```

7.2 Installing python and piecash

You start Termux on your android and then:

1. Install python and pipenv:

```
pkg install python
pip install pipenv
```

2. Install piecash for your project:

```
mkdir my-project
cd my-project
pipenv install piecash
```

3. Test piecash:

```
pipenv shell
python
>>> import piecash
```

7.3 Use SSH with your android

You can ssh easily in your android thanks to Termux. For this, on Termux on your android:

1. install openssh:

```
pkg install openssh
```

2. add your public key (id_rsa.pub) in the file *.ssh/authorized_keys* on Termux
3. run the sshd server:

```
sshd
```

On your machine (laptop, . . .):

1. configure your machine to access your android device:

```
Host android
   HostName 192.168.1.4  # <== put the IP address of your android
   User termux
   Port 8022
```

2. log in your android from your machine:

```
ssh android
```

7.4 Use the USB Debugging with your android

To be investigated. . . :

```
# on laptop
adb forward tcp:8022 tcp:8022 && ssh localhost -p 8022

# on android
# On Android 4.2 and higher, the Developer options screen is hidden by default. To␣
↪make it visible, go to Settings > About phone and tap Build number seven times.␣
↪Return to the previous screen to find Developer options at the bottom.
change USB Configuration to "charge only" or "PTP"

# downloading https://developer.android.com/studio/run/win-usb.html
# Click here to download the Google USB Driver ZIP file (Z
# install legacy hardware (in device manager)
# choose the folder of the zip drive and choose ADB interface
```

7.5 References

- https://glow.li/technology/2015/11/06/run-an-ssh-server-on-your-android-with-termux/
- https://termux.com/storage.html
- https://developer.android.com/studio/releases/platform-tools.html
- https://glow.li/technology/2016/9/20/access-termux-via-usb/
- https://github.com/termux/termux-packages/issues/352

CHAPTER

EIGHT

FOR DEVELOPERS

The complete api documentation (apidoc) :

8.1 piecash package

8.1.1 Subpackages

piecash.business package

Submodules

piecash.business.invoice module

piecash.business.person module

class piecash.business.person.**Address**(*name=''*, *addr1=''*, *addr2=''*, *addr3=''*, *addr4=''*, *email=''*, *fax=''*, *phone=''*)

Bases: object

An Address object encapsulates information regarding an address in GnuCash.

name
self explanatory

Type str

addr1
self explanatory

Type str

addr2
self explanatory

Type str

addr3
self explanatory

Type str

addr4
self explanatory

Type str

email
self explanatory

Type str

fax
self explanatory

Type str

phone
self explanatory

Type str

class piecash.business.person.**Person**
Bases: object

A mixin declaring common field for Customer, Vendor and Employee

class piecash.business.person.**Customer**(*name*, *currency*, *id=None*, *notes=''*, *active=1*, *tax_override=0*, *credit=Decimal('0')*, *discount=Decimal('0')*, *taxtable=None*, *address=None*, *shipping_address=None*, *tax_included='USEGLOBAL'*, *book=None*)
Bases: *piecash.business.person.Person*, piecash._declbase.DeclarativeBaseGuid

A GnuCash Customer

name
name of the Customer

Type str

id
autonumber id with 5 digits (initialised to book.counter_customer + 1)

Type str

notes
notes

Type str

active
1 if the customer is active, 0 otherwise

Type int

discount
see Gnucash documentation

Type decimal.Decimal

credit
see Gnucash documentation

Type decimal.Decimal

currency
the currency of the customer

Type *piecash.core.commodity.Commodity*

tax_override
1 if tax override, 0 otherwise

Type int

address
the address of the customer

Type *Address*

shipping_address
the shipping address of the customer

Type *Address*

tax_included
'yes', 'no', 'use global'

Type str

taxtable
tax table of the customer

Type piecash.business.tax.TaxTable

term
bill term of the customer

Type piecash.business.invoice.Billterm

class piecash.business.person.**Employee**(*name*, *currency*, *creditcard_account=None*, *id=None*, *active=1*, *acl=''*, *language=''*, *workday=Decimal('0')*, *rate=Decimal('0')*, *address=None*, *book=None*)

Bases: *piecash.business.person.Person*, piecash._declbase.DeclarativeBaseGuid

A GnuCash Employee

name
name of the Employee

Type str

id
autonumber id with 5 digits (initialised to book.counter_employee + 1)

Type str

language
language

Type str

active
1 if the employee is active, 0 otherwise

Type int

workday
see Gnucash documentation

Type decimal.Decimal

rate
see Gnucash documentation

Type decimal.Decimal

currency
the currency of the employee

Type *piecash.core.commodity.Commodity*

address
the address of the employee

Type *Address*

creditcard_account
credit card account for the employee

Type *piecash.core.account.Account*

on_book_add()
Call when the object is added to a book

class piecash.business.person.**Vendor**(*name*, *currency*, *id=None*, *notes=''*, *active=1*, *tax_override=0*, *taxtable=None*, *credit=Decimal('0')*, *discount=Decimal('0')*, *address=None*, *tax_included='USEGLOBAL'*, *book=None*)
Bases: *piecash.business.person.Person*, piecash._declbase.DeclarativeBaseGuid

A GnuCash Vendor

name
name of the Vendor

Type str

id
autonumber id with 5 digits (initialised to book.counter_vendor + 1)

Type str

notes
notes

Type str

active
1 if the vendor is active, 0 otherwise

Type int

currency
the currency of the vendor

Type *piecash.core.commodity.Commodity*

tax_override
1 if tax override, 0 otherwise

Type int

address
the address of the vendor

Type *Address*

tax_included
'YES', 'NO', 'USEGLOBAL'

Type str

taxtable
tax table of the vendor

Type piecash.business.tax.TaxTable

term
bill term of the vendor

Type piecash.business.invoice.Billterm

piecash.business.tax module

Module contents

piecash.core package

Submodules

piecash.core._commodity_helper module

piecash.core._commodity_helper.**quandl_fx**(*fx_mnemonic*, *base_mnemonic*, *start_date*)
Retrieve exchange rate of commodity fx in function of base.

API KEY will be retrieved from the environment variable QUANDL_API_KEY

piecash.core.account module

class piecash.core.account.**AccountType**(*value*)
Bases: enum.Enum

An enumeration.

class piecash.core.account.**Account**(*name*, *type*, *commodity*, *parent=None*, *description=''*, *commodity_scu=None*, *hidden=0*, *placeholder=0*, *code=''*, *book=None*, *children=None*)
Bases: piecash._declbase.DeclarativeBaseGuid

A GnuCash Account which is specified by its name, type and commodity.

type
type of the Account

Type str

sign
1 for accounts with positive balances, -1 for accounts with negative balances

Type int

code
code of the Account

Type str

commodity
the commodity of the account

Type *piecash.core.commodity.Commodity*

commodity_scu
smallest currency unit for the account

Type int

non_std_scu
1 if the scu of the account is NOT the same as the commodity

Type int

description
description of the account

Type str

name
name of the account

Type str

fullname
full name of the account (including name of parent accounts separated by ':')

Type str

placeholder
1 if the account is a placeholder (should not be involved in transactions)

Type int

hidden
1 if the account is hidden

Type int

is_template
True if the account is a template account (ie commodity=template/template)

Type bool

parent
the parent account of the account (None for the root account of a book)

Type *Account*

children
the list of the children accounts

Type list of *Account*

splits
the list of the splits linked to the account

Type list of *piecash.core.transaction.Split*

lots
the list of lots to which the account is linked

Type list of `piecash.business.Lot`

book
the book if the account is the root account (else None)

Type *piecash.core.book.Book*

budget_amounts
list of budget amounts of the account

Type list of *piecash.budget.BudgetAmount*

scheduled_transaction
scheduled transaction linked to the account

Type *piecash.core.transaction.ScheduledTransaction*

object_to_validate(*change*)
yield the objects to validate when the object is modified (change="new" "deleted" or "dirty").

For instance, if the object is a Split, if it changes, we want to revalidate not the split but its transaction and its lot (if any). split.object_to_validate should yeild both split.transaction and split.lot

validate()
This must be reimplemented for object requiring validation

observe_commodity(*key*, *value*)
Ensure update of commodity_scu when commodity is changed

get_balance(*recurse=True*, *commodity=None*, *natural_sign=True*)
Returns the balance of the account (including its children accounts if recurse=True) expressed in account's commodity/currency. If this is a stock/fund account, it will return the number of shares held. If this is a currency account, it will be in account's currency. In case of recursion, the commodity of children accounts will be transformed to the commodity of the father account using the latest price (if no price is available to convert , it is considered as 0). If natural_sign is True, the sign of the balance is reverted for the account with type {'LIABILITY', 'PAYABLE', 'CREDIT', 'INCOME', 'EQUITY'}

recurse
True if the balance should include children accounts (default to True)
Type bool, optional

commodity
the currency into which to get the balance (default to None, i.e. the currency of the account)
Type *piecash.core.commodity.Commodity*

natural_sign
True if the balance sign is reversed for accounts of type {'LIABILITY', 'PAYABLE', 'CREDIT', 'INCOME', 'EQUITY'} (default to True)
Type bool, optional

Returns the balance of the account

piecash.core.book module

class piecash.core.book.**Book**(*root_account=None*, *root_template=None*)
Bases: piecash._declbase.DeclarativeBaseGuid

A Book represents a GnuCash document. It is created through one of the two factory functions `create_book()` and `open_book()`.

Canonical use is as a context manager like (the book is automatically closed at the end of the with block):

```
with create_book() as book:
    ...
```

Note: If you do not use the context manager, do not forget to close the session explicitly (`book.close()`) to release any lock on the file/DB.

The book puts at disposal several attributes to access the main objects of the GnuCash document:

```
# to get the book and the root_account
ra = book.root_account

# to get the list of accounts, commodities or transactions
for acc in book.accounts:  # or book.commodities or book.transactions
    # do something with acc

# to get a specific element of these lists
EUR = book.commodities(namespace="CURRENCY", mnemonic="EUR")

# to get a list of all objects of some class (even non core classes)
budgets = book.get(Budget)
# or a specific object
budget = book.get(Budget, name="my first budget")
```

You can check a session has changes (new, deleted, changed objects) by getting the `book.is_saved` property. To save or cancel changes, use `book.save()` or `book.cancel()`:

```
# save a session if it is no saved (saving a unchanged session is a no-op)
if not book.is_saved:
    book.save()
```

root_account
the root account of the book

Type *piecash.core.account.Account*

root_template
the root template of the book (usage not yet clear...)

Type *piecash.core.account.Account*

default_currency
the currency of the root account (=default currency of the book)

Type *piecash.core.commodity.Commodity*

uri
connection string of the book (set by the GncSession when accessing the book)

Type str

session
the sqlalchemy session encapsulating the book

Type `sqlalchemy.orm.session.Session`

use_trading_accounts
true if option "Use trading accounts" is enabled

Type bool

use_split_action_field
true if option "Use Split Action Field for Number" is enabled

Type bool

RO_threshold_day
value of Day Threshold for Read-Only Transactions (red line)

Type int

control_mode
list of allowed non-standard operations like : "allow-root-subaccounts"

Type list(str)

counter_customer
counter for *piecash.business.person.Customer* id (link to slot "counters/gncCustomer")

Type int

counter_vendor
counter for *piecash.business.person.Vendor* id (link to slot "counters/gncVendor")

Type int

counter_employee
counter for *piecash.business.person.Employee* id (link to slot "counters/gncEmployee")

Type int

counter_invoice
counter for `piecash.business.invoice.Invoice` id (link to slot "counters/gncInvoice")

Type int

counter_job
counter for `piecash.business.invoice.Job` id (link to slot "counters/gncJob")

Type int

counter_bill
counter for `piecash.business.invoice.Bill` id (link to slot "counters/gncBill")

Type int

counter_exp_voucher
counter for `piecash.business.invoice.Invoice` id (link to slot "counters/gncExpVoucher")

Type int

counter_order
counter for `piecash.business.invoice.Order` id (link to slot "counters/gncOrder")

Type int

business_company_phone
phone number of book company (link to slit "options/Business/Company Phone Number")

Type str

business_company_email
email of book company (link to slit "options/Business/Company Email Address")

Type str

business_company_contact
contact person of book company (link to slit "options/Business/Company Contact Person")

Type str

business_company_ID
ID of book company (link to slit "options/Business/Company ID")

Type str

business_company_name
name of book company (link to slit "options/Business/Company Name")

Type str

business_company_address
address of book company (link to slit "options/Business/Company Address")

Type str

business_company_website
website URL of book company (link to slit "options/Business/Company Website URL")

Type str

property book
Return the gnc book holding the object

validate()
This must be reimplemented for object requiring validation

static track_dirty(*session*, *flush_context*, *instances*)
Record in session._all_changes the objects that have been modified before each flush

trading_account(*cdty*)
Return the trading account related to the commodity. If it does not exist and the option "Use Trading Accounts" is enabled, create it on the fly

add(*obj*)
Add an object to the book (to be used if object not linked in any way to the book)

delete(*obj*)
Delete an object from the book (to remove permanently an object)

save()
Save the changes to the file/DB (=commit transaction)

flush()
Flush the book

cancel()
Cancel all the changes that have not been saved (=rollback transaction)

property is_saved
Save the changes to the file/DB (=commit transaction)

close()
Close a session. Any changes not yet saved are rolled back. Any lock on the file/DB is released.

get(*cls*, ***kwargs*)
Generic getter for a GnuCash object in the *GncSession*. If no kwargs is given, it returns the list of all objects of type cls (uses the sqlalchemy session.query(cls).all()). Otherwise, it gets the unique object which attributes match the kwargs (uses the sqlalchemy session.query(cls).filter_by(**kwargs).one() underneath):

```
# to get the first account with name="Income"
inc_account = session.get(Account, name="Income")

# to get all accounts
accs = session.get(Account)
```

Parameters

- **cls** (*class*) – the class of the object to retrieve (Account, Price, Budget,...)
- **kwargs** (*dict*) – the attributes to filter on

Returns the unique object if it exists, raises exceptions otherwise

Return type object

property transactions
gives easy access to all transactions in the book through a piecash.model_common.CallableList of *piecash.core.transaction.Transaction*

property splits
gives easy access to all splits in the book through a piecash.model_common.CallableList of *piecash.core.transaction.Split*

property accounts
gives easy access to all accounts in the book through a piecash.model_common.CallableList of *piecash.core.account.Account*

property commodities
gives easy access to all commodities in the book through a piecash.model_common.CallableList of *piecash.core.commodity.Commodity*

property invoices
gives easy access to all commodities in the book through a piecash.model_common.CallableList of *piecash.core.commodity.Commodity*

property currencies
gives easy access to all currencies in the book through a piecash.model_common.CallableList of *piecash.core.commodity.Commodity*

property prices
gives easy access to all prices in the book through a piecash.model_common.CallableList of *piecash.core.commodity.Price*

property customers
gives easy access to all commodities in the book through a piecash.model_common.CallableList of piecash.business.people.Customer

property vendors
gives easy access to all commodities in the book through a piecash.model_common.CallableList of piecash.business.people.Vendor

property employees
gives easy access to all commodities in the book through a piecash.model_common.CallableList of piecash.business.people.Employee

property taxtables
gives easy access to all commodities in the book through a piecash.model_common.CallableList of piecash.business.tax.Taxtable

property query
proxy for the query function of the underlying sqlalchemy session

splits_df(*additional_fields=None*)
Return a pandas DataFrame with all splits (piecash.core.commodity.Split) from the book

Parameters list

Returns pandas.DataFrame

prices_df()
Return a pandas DataFrame with all prices (*piecash.core.commodity.Price*) from the book

Returns pandas.DataFrame

piecash.core.commodity module

exception piecash.core.commodity.**GncCommodityError**
Bases: *piecash._common.GnucashException*

exception piecash.core.commodity.**GncPriceError**
Bases: *piecash._common.GnucashException*

class piecash.core.commodity.**Price**(*commodity*, *currency*, *date*, *value*, *type='unknown'*, *source='user:price'*)
Bases: piecash._declbase.DeclarativeBaseGuid

A single Price for a commodity.

commodity
commodity to which the Price relates

Type *Commodity*

currency
currency in which the Price is expressed

Type *Commodity*

date
date object representing the day at which the price is relevant

Type datetime.date

source
source of the price

Type str

type
last, ask, bid, unknown, nav

Type str

value
the price itself

Type decimal.Decimal

object_to_validate(*change*)
yield the objects to validate when the object is modified (change=”new” “deleted” or “dirty”).

For instance, if the object is a Split, if it changes, we want to revalidate not the split but its transaction and its lot (if any). split.object_to_validate should yeild both split.transaction and split.lot

validate()
This must be reimplemented for object requiring validation

class piecash.core.commodity.**Commodity**(*namespace*, *mnemonic*, *fullname*, *fraction=100*, *cusip=''*, *quote_flag=0*, *quote_source=None*, *quote_tz=''*, *book=None*)
Bases: piecash._declbase.DeclarativeBaseGuid

A GnuCash Commodity.

cusip
cusip code

Type str

fraction
minimal unit of the commodity (e.g. 100 for 1/100)

Type int

namespace
CURRENCY for currencies, otherwise any string to group multiple commodities together

Type str

mnemonic
the ISO symbol for a currency or the stock symbol for stocks (used for online quotes)

Type str

quote_flag
1 if piecash/GnuCash quotes will retrieve online quotes for the commodity

Type int

quote_source
the quote source for GnuCash (piecash always use yahoo for stock and quandl for currencies

Type str

quote_tz
the timezone to assign on the online quotes

Type str

base_currency
The base_currency for a commodity:

- if the commodity is a currency, returns the "default currency" of the book (ie the one of the root_account)
- if the commodity is not a currency, returns the currency encoded in the quoted_currency slot

Type *Commodity*

accounts
list of accounts which have the commodity as commodity

Type list of *piecash.core.account.Account*

transactions
list of transactions which have the commodity as currency

Type list of *piecash.core.transaction.Transaction*

prices
iterator on prices related to the commodity (it is a sqlalchemy query underneath)

Type iterator of *Price*

currency_conversion(*currency*)
Return the latest conversion factor to convert self to currency

currency
the currency to which the Price need to be converted
Type *piecash.core.commodity.Commodity*

Returns a Decimal that can be multiplied by an amount expressed in self.commodity to get an amount expressed in currency

Raises *GncConversionError* – not possible to convert self to the currency

update_prices(*start_date=None*)
Retrieve online prices for the commodity:

- for currencies, it will get from quandl the exchange rates between the currency and its base_currency
- for stocks, it will get from yahoo the daily closing prices expressed in its base_currency

Parameters

- **start_date** (datetime.date) – prices will be updated as of the start_date. If None, start_date is today
- **7 days.** (–) –

Note: if prices are already available in the GnuCash file, the function will only retrieve prices as of the max(start_date, last quoted price date)

Todo: add some frequency to retrieve prices only every X (week, month, . . .)

---

object_to_validate(*change*)
yield the objects to validate when the object is modified (change=”new” “deleted” or “dirty”).

For instance, if the object is a Split, if it changes, we want to revalidate not the split but its transaction and its lot (if any). split.object_to_validate should yeild both split.transaction and split.lot

validate()
This must be reimplemented for object requiring validation

piecash.core.currency_ISO module

class piecash.core.currency_ISO.**ISO_type**(*country*, *currency*, *mnemonic*, *cusip*, *fraction*)
Bases: tuple

property country
Alias for field number 0

property currency
Alias for field number 1

property cusip
Alias for field number 3

property fraction
Alias for field number 4

property mnemonic
Alias for field number 2

piecash.core.factories module

piecash.core.factories.**create_stock_accounts**(*cdty*, *broker_account*, *income_account=None*, *income_account_types='D/CL/I'*)

Create the multiple accounts used to track a single stock, ie:

- broker_account/stock.mnemonic

and the following accounts depending on the income_account_types argument

- D = Income/Dividend Income/stock.mnemonic
- CL = Income/Cap Gain (Long)/stock.mnemonic
- CS = Income/Cap Gain (Short)/stock.mnemonic
- I = Income/Interest Income/stock.mnemonic

Parameters

- **broker_account** (*piecash.core.account.Account*) – the broker account where the account holding
- **stock is to be created**(*the*) –
- **income_account** (*piecash.core.account.Account*) – the income account where the accounts holding
- **income related to the stock are to be created**(*the*) –
- **income_account_types** (*str*) – "/" separated codes to drive the creation of income accounts

Returns a tuple with the account under the broker_account where the stock is held and the list of income accounts.

Return type *piecash.core.account.Account*

piecash.core.factories.**create_currency_from_ISO**(*isocode*)

Factory function to create a new currency from its ISO code

Parameters **isocode** (*str*) – the ISO code of the currency (e.g. EUR for the euro)

Returns the currency as a commodity object

Return type Commodity

piecash.core.factories.**create_stock_from_symbol**(*symbol*, *book=None*)

Factory function to create a new stock from its symbol. The ISO code of the quoted currency of the stock is stored in the slot "quoted_currency".

Parameters **symbol** (*str*) – the symbol for the stock (e.g. YHOO for the Yahoo! stock)

Returns the stock as a commodity object

Return type Commodity

Note: The information is gathered from the yahoo-finance package The default currency in which the quote is traded is stored in a slot 'quoted_currency'

Todo: use 'select * from yahoo.finance.sectors' and 'select * from yahoo.finance.industry where id ="sector_id"' to retrieve name of stocks and allow therefore the creation of a stock by giving its "stock name" (or part of it). This could also be used to retrieve all symbols related to the same company

piecash.core.session module

class piecash.core.session.**Version**(*table_name*, *table_version*)
Bases: sqlalchemy.ext.declarative.api.DeclarativeBase

The declarative class for the 'versions' table.

table_version
The version for the table

piecash.core.session.**build_uri**(*sqlite_file=None*, *uri_conn=None*, *db_type=None*, *db_user=None*, *db_password=None*, *db_name=None*, *db_host=None*, *db_port=None*, *check_same_thread=True*)
Create the connection string in function of some choices.

Parameters

- **sqlite_file** (*str*) – a path to an sqlite3 file (only used if uri_conn is None)
- **uri_conn** (*str*) – a sqlalchemy connection string
- **db_type** (*str*) – type of database in ["postgres","mysql"]
- **db_user** (*str*) – username of database
- **db_password** (*str*) – password for the use of database
- **db_name** (*str*) – name of database
- **db_host** (*str*) – host of database
- **db_port** (*int*) – port of database
- **check_same_thread** (*bool*) – sqlite flag that restricts connection use to the thread that created (see False for use in ipython/flask/... but read first https://docs.python.org/3/library/sqlite3.html)

Returns the connection string

Return type str

piecash.core.session.**create_book**(*sqlite_file=None*, *uri_conn=None*, *currency='EUR'*, *overwrite=False*, *keep_foreign_keys=False*, *db_type=None*, *db_user=None*, *db_password=None*, *db_name=None*, *db_host=None*, *db_port=None*, *check_same_thread=True*, ***kwargs*)
Create a new empty GnuCash book. If both sqlite_file and uri_conn are None, then an "in memory" sqlite book is created.

Parameters

- **sqlite_file** (*str*) – a path to an sqlite3 file (only used if uri_conn is None)
- **uri_conn** (*str*) – a sqlalchemy connection string
- **currency** (*str*) – the ISO symbol of the default currency of the book
- **overwrite** (*bool*) – True if book should be deleted and recreated if it exists already

- **keep_foreign_keys** (*bool*) – True if the foreign keys should be kept (may not work at all with GnuCash)
- **db_type** (*str*) – type of database in ["postgres","mysql"]
- **db_user** (*str*) – username of database
- **db_password** (*str*) – password for the use of database
- **db_name** (*str*) – name of database
- **db_host** (*str*) – host of database
- **db_port** (*int*) – port of database
- **check_same_thread** (*bool*) – sqlite flag that restricts connection use to the thread that created (see False for use in ipython/flask/... but read first https://docs.python.org/3/library/sqlite3.html)

Returns the document as a gnucash session

Return type GncSession

Raises ***GnucashException*** – if document already exists and overwrite is False

piecash.core.session.**open_book**(*sqlite_file=None*, *uri_conn=None*, *readonly=True*, *open_if_lock=False*, *do_backup=True*, *db_type=None*, *db_user=None*, *db_password=None*, *db_name=None*, *db_host=None*, *db_port=None*, *check_same_thread=True*, *check_exists=True*, ***kwargs*)

Open an existing GnuCash book

Parameters

- **sqlite_file** (*str*) – a path to an sqlite3 file (only used if uri_conn is None)
- **uri_conn** (*str*) – a sqlalchemy connection string
- **readonly** (*bool*) – open the file as readonly (useful to play with and avoid any unwanted save)
- **open_if_lock** (*bool*) – open the file even if it is locked by another user (using open_if_lock=True with readonly=False is not recommended)
- **do_backup** (*bool*) – do a backup if the file written in RW (i.e. readonly=False) (this only works with the sqlite backend and copy the file with .{:%Y%m%d%H%M%S}.gnucash appended to it)
- **db_type** (*str*) – type of database in ["postgres","mysql"]
- **db_user** (*str*) – username of database
- **db_password** (*str*) – password for the use of database
- **db_name** (*str*) – name of database
- **db_host** (*str*) – host of database
- **db_port** (*str*) – port of database
- **check_same_thread** (*bool*) – sqlite flag that restricts connection use to the thread that created (see False for use in ipython/flask/... but read first https://docs.python.org/3/library/sqlite3.html)
- **check_exists** (*bool*) – check if the database exists before connecting

Returns the document as a gnucash session

Return type GncSession

Raises

- ***GnucashException*** – if the document does not exist
- ***GnucashException*** – if there is a lock on the file and open_if_lock is False

piecash.core.session.**adapt_session**(*session*, *book*, *readonly*)

Change the SA session object to add some features.

Parameters

- **session** – the SA session object that will be modified in place
- **book** – the gnucash singleton book linked to the SA session
- **readonly** – True if the session should not allow commits.

Returns

piecash.core.transaction module

class piecash.core.transaction.**Split**(*account*, *value*, *quantity=None*, *transaction=None*, *memo=''*, *action=''*, *reconcile_date=None*, *reconcile_state='n'*, *lot=None*)

Bases: piecash._declbase.DeclarativeBaseGuid

A GnuCash Split.

Note: A split used in a scheduled transaction has its main attributes in form of slots.

transaction

transaction of the split

Type *piecash.core.transaction.Transaction*

account

account of the split

Type *piecash.core.account.Account*

lot

lot to which the split pertains

Type piecash.business.Lot

memo

memo of the split

Type str

value

amount express in the currency of the transaction of the split

Type decimal.Decimal

quantity

amount express in the commodity of the account of the split

Type decimal.Decimal

reconcile_state
'n', 'c' or 'y'

Type str

reconcile_date
time

Type datetime.datetime

action
describe the type of action behind the split (free form string but with dropdown in the GUI

Type str

object_to_validate(*change*)
yield the objects to validate when the object is modified (change="new" "deleted" or "dirty").

For instance, if the object is a Split, if it changes, we want to revalidate not the split but its transaction and its lot (if any). split.object_to_validate should yeild both split.transaction and split.lot

validate()
This must be reimplemented for object requiring validation

class piecash.core.transaction.**Transaction**(*currency*, *description=''*, *notes=None*, *splits=None*, *enter_date=None*, *post_date=None*, *num=''*)
Bases: piecash._declbase.DeclarativeBaseGuid

A GnuCash Transaction.

currency
currency of the transaction. This attribute is write-once (i.e. one cannot change it after being set)

Type *piecash.core.commodity.Commodity*

description
description of the transaction

Type str

enter_date
datetimetime at which transaction is entered

Type datetime.datetime

post_date
day on which transaction is posted

Type datetime.date

num
user provided transaction number

Type str

splits
list of the splits of the transaction

Type list of *Split*

scheduled_transaction
scheduled transaction behind the transaction

Type *ScheduledTransaction*

notes
notes on the transaction (provided via a slot)

Type str

object_to_validate(*change*)
yield the objects to validate when the object is modified (change="new" "deleted" or "dirty").

For instance, if the object is a Split, if it changes, we want to revalidate not the split but its transaction and its lot (if any). split.object_to_validate should yeild both split.transaction and split.lot

validate()
This must be reimplemented for object requiring validation

calculate_imbalances()
Calculate value and quantity imbalances of a transaction

class piecash.core.transaction.**ScheduledTransaction**(**args*, ***kwargs*)
Bases: piecash._declbase.DeclarativeBaseGuid

A GnuCash Scheduled Transaction.

Attributes adv_creation (int) : days to create in advance (0 if disabled) adv_notify (int) : days to notify in advance (0 if disabled) auto_create (bool) : auto_notify (bool) : enabled (bool) : start_date (datetime.datetime) : date to start the scheduled transaction last_occur (datetime.datetime) : date of last occurence of the schedule transaction end_date (datetime.datetime) : date to end the scheduled transaction (num/rem_occur should be 0) instance_count (int) : name (str) : name of the scheduled transaction num_occur (int) : number of occurences in total (end_date should be null) rem_occur (int) : number of remaining occurences (end_date should be null) template_account (*piecash.core.account.Account*): template account of the transaction

class piecash.core.transaction.**Lot**(*title*, *account*, *notes=''*, *splits=None*, *is_closed=0*)
Bases: piecash._declbase.DeclarativeBaseGuid

A GnuCash Lot. Each lot is linked to an account. Splits in this account can be associated to a Lot. Whenever the balance of the splits goes to 0, the Lot is closed (otherwise it is opened)

is_closed
1 if lot is closed, 0 otherwise

Type int

account
account of the Lot

Type *piecash.core.account.Account*

splits
splits associated to the Lot

Type *piecash.core.transaction.Split*

object_to_validate(*change*)
yield the objects to validate when the object is modified (change="new" "deleted" or "dirty").

For instance, if the object is a Split, if it changes, we want to revalidate not the split but its transaction and its lot (if any). split.object_to_validate should yeild both split.transaction and split.lot

validate()
This must be reimplemented for object requiring validation

Module contents

8.1.2 Submodules

piecash._common module

exception piecash._common.**GnucashException**
Bases: Exception

exception piecash._common.**GncNoActiveSession**
Bases: *piecash._common.GnucashException*

exception piecash._common.**GncValidationError**
Bases: *piecash._common.GnucashException*

exception piecash._common.**GncImbalanceError**
Bases: *piecash._common.GncValidationError*

exception piecash._common.**GncConversionError**
Bases: *piecash._common.GnucashException*

class piecash._common.**Recurrence**(**args*, ***kwargs*)
Bases: sqlalchemy.ext.declarative.api.DeclarativeBase

Recurrence information for scheduled transactions

obj_guid
link to the parent ScheduledTransaction record.

Type str

recurrence_mult
Multiplier for the period type. Describes how many times the period repeats for the next occurrence.

Type int

recurrence_period_type
type or recurrence (monthly, daily).

Type str

recurrence_period_start
the date the recurrence starts.

Type date

recurrence_weekend_adjust
adjustment to be made if the next occurrence falls on weekend / non-working day.

Type str

piecash._common.**hybrid_property_gncnumeric**(*num_col*, *denom_col*)
Return an hybrid_property handling a Decimal represented by a numerator and a denominator column. It assumes the python field related to the sqlcolumn is named as _sqlcolumn.

Returns sqlalchemy.ext.hybrid.hybrid_property

class piecash._common.**CallableList**(**args*)
Bases: list

A simple class (inherited from list) allowing to retrieve a given list element with a filter on an attribute.

It can be used as the collection_class of a sqlalchemy relationship or to wrap any list (see examples in piecash.core.session.GncSession)

get(***kwargs*)
Return the first element of the list that has attributes matching the kwargs dict. The *get* method is an alias for this method.

To be used as:

```
l(mnemonic="EUR", namespace="CURRENCY")
```

piecash._common.**get_system_currency_mnemonic**()
Returns the mnemonic of the locale currency (and EUR if not defined).

At the target, it could also look in Gnucash configuration/registry to see if the user has chosen another default currency.

piecash._declbase module

piecash.budget module

class piecash.budget.**Budget**(**args*, ***kwargs*)
Bases: piecash._declbase.DeclarativeBaseGuid

A GnuCash Budget

name
name of the budget

Type str

description
description of the budget

Type str

amounts
list of amounts per account

Type list of *piecash.budget.BudgetAmount*

class piecash.budget.**BudgetAmount**(**args*, ***kwargs*)
Bases: sqlalchemy.ext.declarative.api.DeclarativeBase

A GnuCash BudgetAmount

amount
the budgeted amount

Type decimal.Decimal

account
the budgeted account

Type *piecash.core.account.Account*

budget
the budget of the amount

Type *Budget*

piecash.kvp module

class piecash.kvp.**KVP_Type**(*value*)
Bases: enum.Enum

An enumeration.

class piecash.kvp.**SlotType**(**args*, ***kwargs*)
Bases: sqlalchemy.sql.type_api.TypeDecorator

Used to customise the DateTime type for sqlite (ie without the separators as in gnucash

impl
alias of sqlalchemy.sql.sqltypes.INTEGER

process_bind_param(*value*, *dialect*)
Receive a bound parameter value to be converted.

Subclasses override this method to return the value that should be passed along to the underlying TypeEngine object, and from there to the DBAPI execute() method.

The operation could be anything desired to perform custom behavior, such as transforming or serializing data. This could also be used as a hook for validating logic.

This operation should be designed with the reverse operation in mind, which would be the process_result_value method of this class.

Parameters

- **value** – Data to operate upon, of any type expected by this method in the subclass. Can be None.
- **dialect** – the Dialect in use.

process_result_value(*value*, *dialect*)
Receive a result-row column value to be converted.

Subclasses should implement this method to operate on data fetched from the database.

Subclasses override this method to return the value that should be passed back to the application, given a value that is already processed by the underlying TypeEngine object, originally from the DBAPI cursor method fetchone() or similar.

The operation could be anything desired to perform custom behavior, such as transforming or serializing data. This could also be used as a hook for validating logic.

Parameters

- **value** – Data to operate upon, of any type expected by this method in the subclass. Can be None.
- **dialect** – the Dialect in use.

This operation should be designed to be reversible by the "process_bind_param" method of this class.

piecash.ledger module

piecash.metadata module

Project metadata

Information describing the project.

piecash.sa_extra module

piecash.sa_extra.**compile_datetime**(*element*, *compiler*, ***kw*)
data type for the date field

note: it went from TEXT(14) in 2.6 to TEXT(19) in 2.8 to accommodate for the new ISO format of date in sqlite

piecash.sa_extra.**mapped_to_slot_property**(*col*, *slot_name*, *slot_transform=<function <lambda>>*)
Assume the attribute in the class as the same name as the table column with "_" prepended

piecash.sa_extra.**pure_slot_property**(*slot_name*, *slot_transform=<function <lambda>>*, *ignore_invalid_slot=False*)
Create a property (class must have slots) that maps to a slot

Parameters

- **slot_name** – name of the slot
- **slot_transform** – transformation to operate before assigning value
- **ignore_invalid_slot** – True if incorrect values (usually due to deleted data) should be converted to None

Returns

piecash.sa_extra.**get_foreign_keys**(*metadata*, *engine*)
Retrieve all foreign keys from metadata bound to an engine :param metadata: :param engine: :return:

class piecash.sa_extra.**ChoiceType**(*choices*, ***kw*)
Bases: sqlalchemy.sql.type_api.TypeDecorator

process_bind_param(*value*, *dialect*)
Receive a bound parameter value to be converted.

Subclasses override this method to return the value that should be passed along to the underlying TypeEngine object, and from there to the DBAPI execute() method.

The operation could be anything desired to perform custom behavior, such as transforming or serializing data. This could also be used as a hook for validating logic.

This operation should be designed with the reverse operation in mind, which would be the process_result_value method of this class.

Parameters

- **value** – Data to operate upon, of any type expected by this method in the subclass. Can be None.
- **dialect** – the Dialect in use.

process_result_value(*value*, *dialect*)
Receive a result-row column value to be converted.

Subclasses should implement this method to operate on data fetched from the database.

Subclasses override this method to return the value that should be passed back to the application, given a value that is already processed by the underlying `TypeEngine` object, originally from the DBAPI cursor method `fetchone()` or similar.

The operation could be anything desired to perform custom behavior, such as transforming or serializing data. This could also be used as a hook for validating logic.

Parameters

- **`value`** – Data to operate upon, of any type expected by this method in the subclass. Can be `None`.
- **`dialect`** – the `Dialect` in use.

This operation should be designed to be reversible by the "process_bind_param" method of this class.

8.1.3 Module contents

Python interface to GnuCash documents

An overall view on the core objects in GnuCash:

8.2 GnuCash SQL Object model and schema

A clear documentation of the SQL schema (tables, columns, relationships) and the implicit semantic (invariants that should be always satisfied, logic to apply in ambiguous/corner cases) is critical for piecash to

a) ensure data integrity (when creating new objects and/or modifying/deleting existing objects)

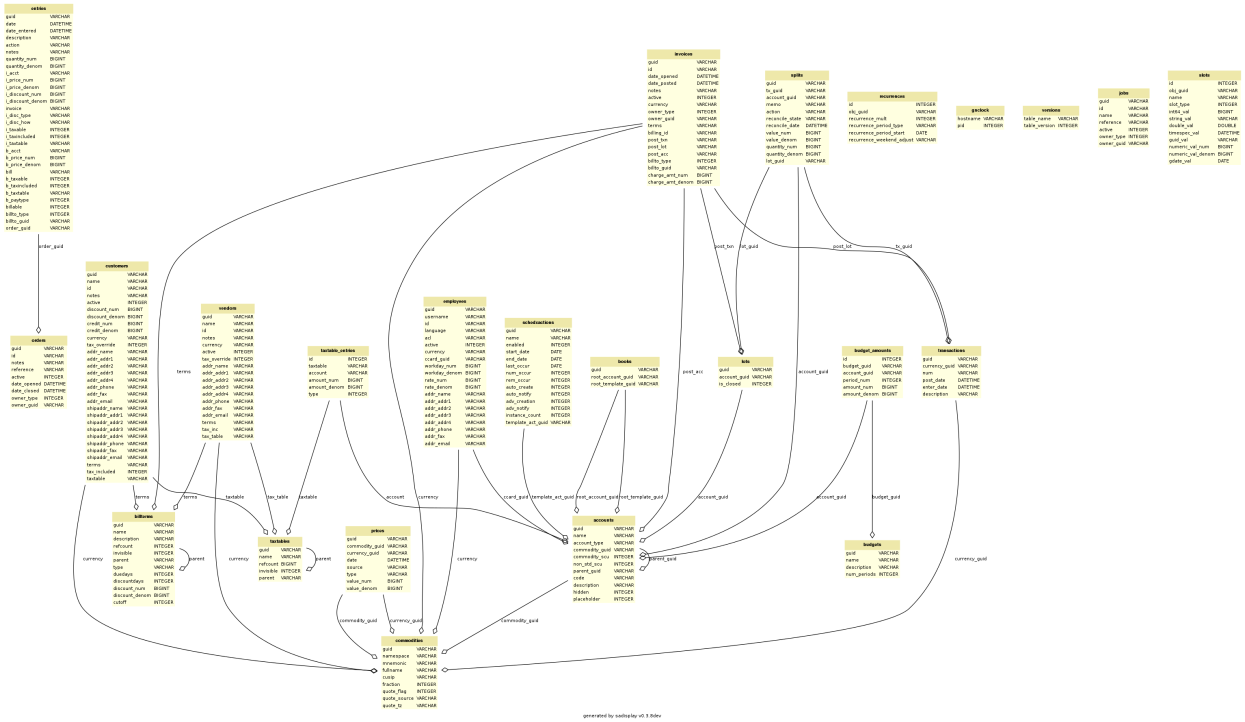
b) ensure compatibility in semantic with the official GnuCash application

Warning: This document explains what the author understands in these domains. It is not the reference documentation, please refer to the official GnuCash documentation for this.

Warning: Disclaimer : piecash primary focus is on reading GnuCash books and creating new *Core objects*. Creating other objects than the core objects, modifying existing objects attributes or relationships and deleting objects can be done through piecash but at the user's own risk (backup your books before doing any of such modifications)

8.2.1 Schema

The following SQL schema has been generated by sadisplay (https://pypi.python.org/pypi/sadisplay) on a GnuCash book generated by piecash on the MySQL backend with the option keep_foreign_keys (the official GnuCash schema does not define foreign keys):



8.2.2 Days, times, dates & datetimes

The use of date and time in GnuCash is somewhat complicated (mainly due to legacy reasons). This chapter described how days and time are encoded in the different tables. For each table.field, the DB type and the PIECASH type are given (DATE = day, DATETIME = day + time) as well as the representation in SQL. All examples are based on a local time in CET (central european time) and for the 11 feb 2018.

prices.date

> DATETIME -> DAY = YYYY-MM-DD 00:00:00 LT expressed as UTC (e.g. 20180210230000) if price entered via the price editor
>
> DATETIME -> DAY = YYYY-MM-DD 10:59:00 UTC (e.g. 20180211105900) if price generated via a transaction
>
> DATETIME -> DAY = ??? (To be completed)) if price retrieved via Finance:Quote

transactions.post_date DATETIME -> DAY = YYYY-MM-DD 10:59:00 UTC (e.g. 20180211105900) In the slots, the date-posted stores the post_date as a day (e.g. 20180211)

transactions.enter_date DATETIME -> DATETIME = YYYY-MM-DD hh:mm:ss UTC (e.g. 20180211123036)

splits.reconcile_date DATETIME -> DAY = 1970-01-01 00:00:00 UTC if not applicable (19700101000000) DATETIME -> DAY = YYYY-MM-DD 23:59:59 LT expressed as UTC (e.g. 20180211225959) In the slots, the reconcile-info/last-date stored and int64 representing the timestamp of the last post_date (e.g. 1518389999 ~ 20180211105900)

schedxactions.start_date, schedxactions.end_date, schedxactions.last_occur DAY -> DAY = YYYY-MM-DD

8.2.3 Core objects

There are 5 core objects in GnuCash : *Book*, *Commodity*, *Account*, *Transaction*, *Split*. An additional object, the *Price*, is strongly linked to the Commodity and is used in reports and for display (for instance, to convert all accounts balance in the default currency). While not as core as the others, it is an essential piece of functionality for anyone using GnuCash to track a stock portfolio value or multi-currency book.

Note: A priori, all these objects are all "create once, never change" objects. Changing some fields of an object may lead to complex renormalisation procedures. Deleting some objects may lead to complex cascade changes/renormalisation procedures. In this respect, it is important to either avoid changes/deletions or to have clear invariants that should stay true at any time.

Book

The Book is the object model representing a GnuCash document. It has a link to the root account, the account at the root of the tree structure.

Fields

root_account (mandatory) The account at the root of the tree structure

root_template (mandatory) Use to attach split from template/scheduled transactions

Invariant

- one (and only one) Book per GnuCash document

Commodity

A Commodity is either a currency (€, $, ...) or a commodity/stock that can be stored in/traded through an Account.

The Commodity object is used in two different (but related) contexts.

a) each Account should specify the Commodity it handles/stores. For usual accounts (Savings, Expenses, etc), the Commodity is a currency. For trading accounts, the Commodity is usually a stock (AMZN, etc). In this role, each commodity (be it a stock or a currency) can have Prices attached to it that give the value of the commodity expressed in a given currency.

b) each Transaction should specify the Currency which is used to balance itself.

Fields

namespace (mandatory) A string representing the group/class of the commodity. All commodities that are currencies should have 'CURRENCY' as namespace. Non currency commodities should have other groups.

mnemonic (mandatory) The symbol/stock sticker of the commodity (relevant for online download of quotes)

fullname The full name for the commodity. Besides the fullname, there is a "calculated property" unique_name equal to "namespace::mnemonic"

cusip unique code for the commodity

fraction The smallest unit that can be accounted for (for a currency, this is equivalent to the scu, the smallest currency unit) This is essentially used for a) display and b) roundings

quote_flag True if Prices for the commodity should be retrieved for the given stock. This is used by the "quote download" functionnality.

quote_source The source for online download of quotes

Invariant

- a currency commodity has namespace=='CURRENCY'
- only currencies referenced by accounts or commodities are stored in the table 'commodities' (the complete list of currencies is available within the GnuCash application)
- a stock commodity has namespace!='CURRENCY'

Account

An account tracks some commodity for some business purpose. Changes in the commodity amounts are modelled through Splits (see Transaction & Splits).

Fields

type (mandatory) the type of the account as string

commodity (mandatory) The commodity that is handled by the account

parent (almost mandatory) the parent account to which the account is attached. All accounts but the root_account should have a parent account.

commodity_scu (mandatory) The smallest currency/commodity unit is similar to the fraction of a commodity. It is the smallest amount of the commodity that is tracked in the account. If it is different than the fraction of the commodity to which the account is linked, the field non_std_scu is set to 1 (otherwise the latter is set to 0).

name self-explanatory

description self-explanatory

placeholder if True/1, the account cannot be involved in transactions through splits (ie it can only be the parent of other accounts). if False/0, the account can have Splits referring to it (as well as be the parent of other accounts). This field, if True, is also stored as a Slot under the key "placeholder" as a string "true".

hidden if True/1, the account will not be displayed in the GnuCash GUI Accounts tab and can be easily excluded from GnuCash GUI Reports. if False/0, the account will be displayed in the GnuCash GUI Accounts tab.

Invariant

- if placeholder, no new splits can be created/changed (like a "freeze")
- only two accounts can have type ROOT (the root_account and the root_template of the book).
- the type of an account is constrained by the type of the parent account
- trading account are used when the option "use trading accounts" is enabled

Transaction & Splits

The transaction represents movement of money between accounts expressed in a given currency (the currency of the transaction). The transaction is modelled through a set of Splits (2 or more). Each Split is linked to an Account and gives the increase/decrease in units of the account commodity (quantity) related to the transaction as well as the equivalent amount in currency (value). For a given transaction, the sum of the split expressed in the currency (value) should be balanced.

Fields for Transaction

currency (mandatory) The currency of the transaction

num (optional) A transaction number (only used for information)

post_date (mandatory) self-explanatory. This field is also stored as a slot under the date-posted key (as a date instead of a time)

enter_date (mandatory) self-explanatory

description (mandatory) self-explanatory

Fields for Split

tx (mandatory) the transaction of the split

account (mandatory) the account to which the split refers to

value (mandatory) the value of the split expressed in the currency of the transaction

quantity (mandatory) the change in quantity of the account expressed in the commodity of the account

reconcile information (Descriptions from official help manual.)

- n - Default status when a transaction is created
- c - Cleared. Status may be assigned either manually or by an import process.
- y - Status assigned solely by the reconciliation process. Places limits optionally requiring confirmation on editing fields in that line of a transaction.
- f - Frozen. Not implemented at this time
- v - Voided. Status is assigned or released manually and applies to every line in the transaction. It hides most of the transaction details but does not delete them. When a transaction is voided a reason entry is required that appears to the right of the description. (Note: There appears to be no way to actually view the reason in the GnuCash GUI at the moment.)

lot reference to the lot (to be investigated)

Invariant

- the sum of the value on all splits in a transaction should = 0 (transaction is balanced). If it is not the case, the GnuCash application create automatically an extra Split entry towards the Account Imbalance-XXX (with XXX the currency of the transaction)
- the value and quantity fields are expressed as numerator / denominator. The denominator of the value should be the same as the fraction of the currency. The denominator of the quantity should be the same as the commodity_scu of the account.
- the currency of a transaction is the currency of the account into which it is created in the GUI
- if "use trading accounts" is enabled then the sum of quantities per commodity should also be balanced. This is done thanks to the automatic creation of splits with trading accounts (of type TRADING)
- the reconcile field in all splits in a transaction that is voided are set to v
- a voided transaction has 4 associated slots with obj_guid equal to the transaction's guid and slot_type 4:
  - name: notes, string_val: Voided transaction
  - name: trans-read-only, string_val: Transaction Voided
  - name: void-reason, string_val: <user-supplied reason string>
  - name: void-time, string_val: date as string in format `YYYY-MM-DD HH:mm:ss.nnnnnn pZZZZ` where n represents milliseconds, p is an optionally present minus sign, and ZZZZ is GMT offset in HHmm format.
- a voided split has 2 nearly identical associated slots with obj_guid equal to the split's guid and slot_type 3:
  - name: void-former-amount, numeric_val_num/numeric_val_denom: the value of the voided split
  - name: void-former-value, numeric_val_num/numeric_val_denom: the value of the voided split

Price

The Price represent the value of a commodity in a given currency at some time.

It is used for exchange rates and stock valuation.

Fields

commodity (mandatory) the commodity related to the Price

currency (mandatory) The currency of the Price

date (mandatory) self-explanatory (expressed in UTC)

value (mandatory) the value in currency of the commodity

Invariant

- the value is expressed as numerator / denominator. The denominator of the value should be the same as the fraction of the currency.

A list of resources used for the project:

8.3 Resources

This page lists resources related to GnuCash, and more specifically, to the use of Python for GnuCash.

8.3.1 GnuCash links

- The official GnuCash page : http://www.gnucash.org/
- The official python bindings : http://wiki.gnucash.org/wiki/Python_Bindings (wiki) and http://svn.gnucash.org/docs/head/python_bindings_page.html (svn)

8.3.2 Web resources

- List (XML) of currencies with their ISO code : http://www.currency-iso.org/dam/downloads/table_a1.xml
- Quandl (for exchange rates) : http://www.quandl.com
- Yahoo! query language : https://developer.yahoo.com/yql/console/

8.3.3 Blogs & discussions

- blog with GnuCash/python links (not 100% correct): http://wideopenstudy.blogspot.be/search/label/GnuCash
- on timezone in GnuCash: http://do-the-right-things.blogspot.be/2013/11/caveats-in-using-gnucash-time-zone.html
- Google search on python in user mailing list: python site:http://lists.gnucash.org/pipermail/gnucash-user" python
- Google search on python in devel mailing list: python site:http://lists.gnucash.org/pipermail/gnucash-devel" python

8.3.4 Github projects related to GnuCash

Projects per language

This page lists all projects found by searching 'gnucash' on github (generated on 2020-10-24 20:47:08) excluding mirrors of the gnucash repository. Projects with a '*' are projects that have not been updated since 12 months.

Language	# of projects	# of projects updated in last 12 months
Python	171	50
Unknown	61	16
Java	42	13

continues on next page

Table 1 – continued from previous page

Language	# of projects	# of projects updated in last 12 months
JavaScript	25	9
Shell	26	6
Dockerfile	11	5
Go	10	4
TypeScript	6	4
C#	11	3
Perl	25	3
Ruby	17	3
Dart	3	2
Jupyter Notebook	5	2
PHP	16	2
Scheme	15	2
Awk	2	1
C++	6	1
CSS	1	1
HTML	4	1
Inno Setup	1	1
PLSQL	1	1
R	2	1
Rust	1	1
Scala	3	1
TSQL	2	1
XSLT	4	1
C	8	0
Diff	1	0
F#	1	0
Gettext Catalog	1	0
Groovy	1	0
Haskell	2	0
Perl6	1	0
Roff	1	0
SQLPL	1	0
Swift	1	0
Tcl	1	0
Visual Basic	1	0

Python

* **accounting-reports by ebridges (last commit on 2019-04-04)** Accounting reports for GNUCash.

* **AceMoney-to-GnuCash by lowvoltage (last commit on 2015-01-11)** A quick and dirty script to convert an AceMoney .XML into a GnuCash .XML

alchemy by zmoog (last commit on 2020-06-05) A very simple Gnucash-inspired web application

* **asset-allocation by MisterY (last commit on 2019-06-03)** Asset Allocation implementation in Python

* **bank_to_qif by engdan77 (last commit on 2019-01-27)** Program for processing XML/XLS bank account exports into QIF-format supported by e.g. GnuCash developed in Python

* **BankCSVtoQif by niknow (last commit on 2019-08-05)** Converts csv files from a bank to qif and replaces descriptions and target accounts according to predefined customizable rules along the way. Intended to work as a

tool for gnucash.

* **beancount2gnucash by wolfm89 (last commit on 2018-12-23)** Convert Beancount ledger files to GnuCash compatible files

* **cookthebooks by colemannugent (last commit on 2017-11-13)** A python3 based, gnucash to ledger converter

* **CSV-pre-processor-for-GnuCash by hughgliderpilot (last commit on 2019-08-16)** Takes transaction CSV download from bank and adds Transfer Account column based on Description

csv2cash by jrwrigh (last commit on 2020-07-13) Python package for importing CSV files to GNUCash

* **csv2mt940 by selva-di (last commit on 2019-09-20)** convert Sparda-West-csv to mt940 for gnucash import

* **django-openbudget by evandavey (last commit on 2012-09-18)** Simple django-based personal budgeting app that sources data from GNUCash sqlite data files

dkb2qif by mzur (last commit on 2020-04-05) Convert a DKB CSV export to QIF

DKB_to_GnuCash by ch3fk0mm7 (last commit on 2019-12-22) Die CSV die man aus dem DKB Online Banking exportiert kann enthält in mehreren Spalten Informationen, die in GnuCash in die Spalte "Beschreibung" sollen, damit die Kategorie der Buchung automatisch erkannt werden kann.

dkcash by quazgar (last commit on 2020-01-24) Direktkreditverwaltung mit gnucash-Backend

* **DnbNor2qif by djiti (last commit on 2011-09-19)** Turning DnbNor CSV files into GNUCash-compatible QIF files

* **dollar by marcotmarcot (last commit on 2018-04-21)** Check if the dollar exchange rate is the same on my Gnucash transactions and UOL

* **dropcopy by juniorbl (last commit on 2019-02-21)** A simple tool for GNOME to copy a given GnuCash file to a local dropbox directory whenever it is saved.

exporterVolksbank_GNUCash by vspaceone (last commit on 2020-01-10) (No description available)

* **finance_convert by RincewindWizzard (last commit on 2017-12-01)** Convert from Paypal and Volksbank to gnucash using csv

* **financial_forecast by skullspace (last commit on 2016-09-06)** A script to take in our Gnucash books and output a CSV with some historical and forecasted data

flux by marcotmarcot (last commit on 2020-07-31) Create a monthly flux report from a gnucash file

* **gcimport by nblock (last commit on 2014-01-12)** Convert various input files (csv, txt) to csv files that can be easily parsed with GnuCash.

* **gcinvoice by ngiger (last commit on 2019-06-04)** GnuCash to Lates (see http://www.smoerz.org/gcinvoice/) + my personal templates

* **generate_prices by barrettTom (last commit on 2019-05-20)** gnucash price database generator script

gnc-fq-helper by yegord (last commit on 2019-10-27) A drop-in replacement for GnuCash's Finance::Quote helper

* **gnc2QuickBooks by jfishe (last commit on 2018-09-28)** Python 2 convert GnuCash to QuickBooks tab delimited import format

* **gnc_budget_scroll by mateuszzz88 (last commit on 2016-09-18)** This is a converter for gnucash bugdet report. It will create html with the same content, but table will be scrollable in such way, that headers (date, account name) are visible

* **gnc_privat24 by gentoo90 (last commit on 2018-11-05)** Imports Privat24 statements to Gnucash book

gnc_tools by armanschwarz (last commit on 2020-08-29) Python tools for validating GnuCash files

gncxml by LiosK (last commit on 2020-06-20) gncxml - extract entries from GnuCash data file to pandas.DataFrame

* gnucash-account2template by EvansMike (last commit on 2015-09-03) Make a GnuCash account template from an exported account

* gnucash-bridge by dbellettini (last commit on 2017-02-18) Expose GnuCash as a microservice

* gnucash-budgerow- by dlex (last commit on 2016-02-05) Predictive budgeter for GnuCash

* gnucash-budget by chrisbrasington (last commit on 2018-05-11) Minimalist budget reporting.

* gnucash-categorizer by seddonym (last commit on 2017-05-22) (No description available)

* gnucash-cfdi by sebastianavina (last commit on 2014-02-16) Proyecto que timbra facturas de gnucash por medio de facturacion moderna.

* gnucash-cli by loftx (last commit on 2019-06-30) (No description available)

* GnuCash-CSV2CSV-for-PowerBi by aidancrane (last commit on 2019-01-01) I use this to convert my GnuCash csv exports for analysis in MS Power BI

* gnucash-csv2html by m13253 (last commit on 2019-08-08) Convert CSV files exported by GnuCash to HTML format

gnucash-docs-rst by codesmythe (last commit on 2020-05-03) GnuCash documentation in RestructuredText as Sphinx project

* gnucash-expense-report by QuLogic (last commit on 2015-01-29) (No description available)

GnuCash-Expenses-Vis by maciek3000 (last commit on 2020-04-29) Visualizations of Expenses created in GnuCash Accounting Software

gnucash-fiximports by sandeepmukherjee (last commit on 2020-03-14) Change target accounts of imported gnucash transactions

gnucash-fiximports by jamessherring (last commit on 2020-08-01) (No description available)

* gnucash-fiximports by HappyPeng2x (last commit on 2017-09-24) Additional developments on the original gnucash-fiximports

* gnucash-gource-viz by C7C8 (last commit on 2019-03-11) Script to convert GNUCash transaction histories into logs that can be visualized by Gource (because why not?)

* gnucash-import by manzato (last commit on 2014-07-15) Imports transactions from a CSV file and places them to the appropriate account depending on a set of rules

gnucash-import-stock by senooken (last commit on 2020-10-24) (No description available)

gnucash-importer by shaform (last commit on 2020-08-29) Utilities to import transactions into GnuCash

* gnucash-importer by gunny26 (last commit on 2017-12-30) import some csv data, categorize bookings automatically with help of some neuronal network categorizer

gnucash-importer by drjeep (last commit on 2020-08-13) (No description available)

* gnucash-importers by rtucker (last commit on 2018-03-02) gnucash importer scripts for Interlock Rochester financial foo

gnucash-imports by dpslwk (last commit on 2020-06-19) Import scripts for Nottingham Hackspace GNUCash

gnucash-input by elChapoSing (last commit on 2020-10-06) personal process for DBS input into gnucash compatible file format

* gnucash-inteligo by lukasszz (last commit on 2018-04-15) Importowanie wyciągów z Inteligo do Gnucash,

* **gnucash-latex by mwellnitz (last commit on 2017-04-28)** Create good looking invoices for gnucash using latex and python

* **gnucash-latex-koma by jappeace (last commit on 2017-04-24)** Create good looking invoices for gnucash using latex/koma and python

* **gnucash-mail-sync by omelkova (last commit on 2018-05-06)** (No description available)

* **gnucash-ofx-brokerage by 7max (last commit on 2012-02-18)** GnuCash OFX importer that handles brokerages, ie stocks, mutual funds, optios

* **gnucash-ofx-export by hoffie (last commit on 2014-07-14)** Selectively export GnuCash transactions into OFX

* **gnucash-portfolio by MisterY (last commit on 2019-04-13)** Tools for managing an investment portfolio in a GnuCash book

* **gnucash-portfolio-cli by MisterY (last commit on 2019-02-04)** Gnucash Portfolio CLI

gnucash-portfolio-webui by MisterY (last commit on 2020-09-04) GnuCash Portfolio Web UI

gnucash-prices by nomis (last commit on 2020-03-08) GnuCash price database management

* **gnucash-pyquotehist by tfree87 (last commit on 2015-06-17)** A simple python script which imports historical price quotes from yahoo into Gnucash via the command line without the need for Perl Finance::QuoteHist. Based on Peter Holtermann's quotes_hist script

* **Gnucash-Python-Example by petarkabashki (last commit on 2015-02-19)** Example python script for importing data into Gnucash

* **gnucash-python-free by tbhartman (last commit on 2014-08-24)** (No description available)

* **GNUCash-Python-Scripts by relyt29 (last commit on 2017-05-31)** various python scripts to add to gnucash functionality

* **gnucash-qif-import by hjacobs (last commit on 2018-12-29)** GnuCash Python helper script to import transactions from QIF text files into GnuCash's own file format

gnucash-reconciler by rmehyde (last commit on 2020-08-02) Tool for comparing Gnucash records to bank records automatically

* **gnucash-reports by MeerkatLabs (last commit on 2019-10-03)** Simple reporting framework for fetching data out of gnucash files for display in a viewer.

gnucash-rest by loftx (last commit on 2020-10-06) A Python based REST framework for the Gnucash accounting application

* **gnucash-rest-docker by loftx (last commit on 2018-09-25)** A dockerfile and associated files to quickly test the Gnucash Rest API

* **gnucash-savings by chrisbrasington (last commit on 2018-08-24)** gnucash-savings projection

* **GNUCash-scripts by spartha80 (last commit on 2019-07-28)** Simple Python scripts to convert Bank statements to QIF format

gnucash-scripts by thomasrebele (last commit on 2019-10-27) (No description available)

gnucash-select by bulletmark (last commit on 2020-08-15) GnuCash File Selector

* **gnucash-society by hendrikvanantwerpen (last commit on 2013-04-13)** Support application for societies based on GnuCash

* **gnucash-stock-portfolio by hjacobs (last commit on 2014-01-15)** GnuCash Python utilities to manage a stock portfolio

gnucash-stock-quotes by DrSkippy (last commit on 2020-08-24) (No description available)

* gnucash-to-beancount by henriquebastos (last commit on 2018-07-22) Gnucash to Beancount Converter.

* gnucash-tools by cnuahs (last commit on 2016-07-26) Command line tools for use with Gnucash (http://gnucash.org/).

* gnucash-tools by yanivmo (last commit on 2016-03-16) GnuCash interoperability scripts

* gnucash-tools by dahnielson (last commit on 2012-05-14) Tools for working with GnuCash

gnucash-tools by iqt4 (last commit on 2020-02-16) (No description available)

* gnucash-toolset by cirrax (last commit on 2019-04-24) Access and manipulate gnucash data.

* gnucash-util by bstpierre (last commit on 2011-01-21) Utility scripts using GnuCash python bindings

* gnucash-utilities by sdementen (last commit on 2017-11-23) Set of python scripts to work with GnuCash books

* gnucash-utils by AndreasHeger (last commit on 2015-04-08) utility scripts for gnucash

gnucash-vis by chrlen (last commit on 2020-05-16) A script to visualize state of accounts from a Gnucash file exported by the client for Android

* gnucash-xml-split by fefe982 (last commit on 2014-08-30) Split GnuCash XML files into files containing transaction in a certain period (e.g. a year)

gnucash2beancount by shaform (last commit on 2020-09-02) (No description available)

* gnucash2googlesheets by erikvanegmond (last commit on 2018-08-06) (No description available)

* gnucash2iif by pawl (last commit on 2013-08-18) Converts a Gnucash general ledger to an IIF file (for quickbooks)

* gnucash_autobudget by rmoehn (last commit on 2017-03-06) Automatically adjust GnuCash transactions for envelope budgeting (discontinued)

* gnucash_balance_report by tbhartman (last commit on 2014-08-24) get balance report from gnucash file

* gnucash_budget by dschwen (last commit on 2017-01-22) Tools to work with a GnuCash database

* gnucash_converter by boszkie (last commit on 2019-05-05) python script to convert rabobank (nl) csv format to gnuCash csv import format

* gnucash_django by RobFisher (last commit on 2013-05-06) GnuCash Web Interface using Django.

gnucash_envelope_assist by MarkOfLark (last commit on 2020-01-02) Scripts that assist in using GnuCash for personal finances under the envelope system

* gnucash_explorer by peap (last commit on 2019-02-13) Another option for exploring your gnucash database

* gnucash_exports by jjuanda (last commit on 2014-01-13) GnuCash export scripts into several DBs/file formats

gnucash_general_journal by dorfsmay (last commit on 2020-01-20) GnuCash csv General Journal

* gnucash_import_from_bank by silvester747 (last commit on 2019-08-03) Convert bank statements in a format GNUCash can import.

gnucash_import_util by shinnkondo (last commit on 2020-05-16) (No description available)

* gnucash_invoice_automator by peanutbutterandcrackers (last commit on 2019-05-05) I haz the power of Libreoffice Calc and python-gnucash on my side

* gnucash_lbb_amazon by elezar (last commit on 2015-12-28) Amazon Credit Card CSV pre-processor for GNUCash

* gnucash_magical_importer by foguinhoperuca (last commit on 2019-01-23) Set of scripts to manage my personal finance with gnucash

* gnucash_ofx by gevious (last commit on 2013-01-10) Convert gnucash xml file to set of OFX files

* **gnucash_quotes by belidzs (last commit on 2019-03-01)** Download stock and currency quotes from Alpha Vantage and save it to GnuCash

* **gnucash_tweaks by jokim (last commit on 2019-07-29)** Simple ad hoc tweaks, because I don't have time to learn to code inside GnuCash

* **gnucashApi by f-angi (last commit on 2017-12-04)** (No description available)

* **GnuCashImporter by sphaero (last commit on 2019-02-25)** Simple console tool to import mt940 file and match transactions to GnuCash accounts

gnucashpricesupdater by danfcosta (last commit on 2020-03-31) Update prices of Brazilian commodities on GnuCash database (SQLite)

* **gnucashreconcile by seddonym (last commit on 2017-04-12)** (No description available)

gnucashreport by partizand (last commit on 2019-11-08) Python library for get reports from GnuCash to xlsx files

* **GnucashReporting by sholly (last commit on 2017-06-27)** Python/Flask backend for gnucash reports

* **gnucashreports by youngchul (last commit on 2011-12-06)** (No description available)

gnucashREST by f-angi (last commit on 2020-05-13) A REST(ful) API for basic GnuCash operations

* **GNUCashTools by s8002sid (last commit on 2019-08-13)** This repository will be used for storing GNUCash tools

* **gnuCashTools by Walms (last commit on 2017-07-17)** Just a few scripts to help manage my budget

* **GnuCashUtils by wlcasper (last commit on 2018-12-31)** GnuCash scripts

* **gnucashxml by jorgenschaefer (last commit on 2017-02-06)** New Maintainer, please use their repository

* **hackerspace-gnucash by cvonkleist (last commit on 2012-08-24)** Gainesville Hackerspace GnuCash scripts, etc.

* **importfindata by gregorias (last commit on 2017-07-01)** Script that updates Polish investment fund quotes in a Gnucash file.

* **ing2gnucash by hjmeijer (last commit on 2013-07-17)** Converts downloaded ING (bank) transaction CSV files to GNUCash importable CSV

* **ing2qif by marijnvriens (last commit on 2014-10-18)** Import ING bank statements and convert them to qif for importing into gnucash

ing2qif2 by tychobismeijer (last commit on 2020-09-29) Convert ING csv to QIF format for GnuCash

* **jeffs-gnucash-utils by n1ywb (last commit on 2018-12-01)** Jeff Laughlin's Python utilities for GnuCash. Includes HTML invoice generator.

* **koert by awesterb (last commit on 2017-07-08)** Toolbox for the inspection of GnuCash (used by the financial control committee of Karpe Noktem).

ledger-explorer by saufrecht (last commit on 2020-10-23) Navigate any[1] hierarchical ledger graphically, all the way down to individual transactions. ([1] as long as it's formatted exactly like a Gnucash CSV export)

* **ledger2gnucash by dotmjs (last commit on 2016-01-19)** Simple python script to convert ledger-cli files to GnuCash

* **mint2gnucash by SAL-e (last commit on 2019-10-06)** Use mint.com together with GnuCash.

* **Mint2GNUCash by alexevans91 (last commit on 2017-03-04)** Converts transaction CSV file from Mint.com to a format that can be imported into GNU Cash.

* **mintcash by hiromu2000 (last commit on 2019-07-05)** Transfer transactions from Mint.com to Gnucash

* **moneyguru-to-gnucash by peppelan (last commit on 2019-04-21)** Data migration from Moneyguru to GnuCash made easy

mwrr by jmtilli (last commit on 2020-08-18) Money-weighted rate of return calculator for GnuCash

* **NokiaCash by sunziping2016 (last commit on 2016-06-01)** A GnuCash-like software on S60v3 developed by python

ofxstatement by kedder (last commit on 2020-10-22) Tool to convert proprietary bank statement to OFX format, suitable for importing to GnuCash or other personal finance applications.

ofxstatement-al_bank by lbschenkel (last commit on 2020-05-23) Arbejdernes Landsbank plugin for ofxstatement

* **ofxstatement-lansforsakringar by lbschenkel (last commit on 2019-03-05)** Länsförsakringar plugin for ofxstatement

* **ofxstatement-sparkasse-freiburg by omarkohl (last commit on 2018-04-04)** ofxstatement plugin for the German bank Sparkasse Freiburg-Nördlicher Breisgau

* **pdf2gc by iqt4 (last commit on 2018-01-06)** Read bank statement and import into Gnucash

piecash by sdementen (last commit on 2020-10-24) Pythonic interface to GnuCash SQL documents

* **pricedb-pull by chrisberkhout (last commit on 2018-05-27)** Pull historical prices for use in GnuCash and Ledger CLI

* **PyBank by dougthor42 (last commit on 2018-01-20)** Personal accounting software. Alternative to the likes of Quicken, iBank, Mint.com, and GnuCash

pygnc by ErwinRieger (last commit on 2020-03-22) My Gnucash extensions for german small businesses using gnucash and aqbanking python-api's (ibr-gnc-module reloaded).

* **pygnclib by tdf (last commit on 2014-05-26)** Pyxb-based read and write support for GnuCash XML files

* **pygnucash by MatzeB (last commit on 2018-05-12)** Python code to read gnucash 2.6 sqlite3 files; features gnucash 2 ledger translator.

pyGnuCash by sebgad (last commit on 2020-10-10) Python Access for GnuCash

pyTry by EpistemikPython (last commit on 2020-09-18) parse Monarch report files and create transactions to write to a Gnucash file

* **qb2gnc by jfishe (last commit on 2018-09-28)** Python 2 convert QuickBooks to GnuCash

* **qif-split by ebridges (last commit on 2018-01-17)** Splits transactions in a QIF file to support budgeting and more granular financial tracking.

qifqif by Kraymer (last commit on 2020-07-15) Enrich your .QIF files with categories.

rabo2ofx by gbonnema (last commit on 2019-12-10) A python script to convert Dutch Rabobank CSV files to OFX files for processing in GnuCash.

* **rabobank-gnucash-converter by boterbloem5 (last commit on 2017-12-09)** (No description available)

* **scripts by cpg314 (last commit on 2019-03-24)** Collection of Python scripts

simple_gnucash_budget_plots by csun (last commit on 2020-05-21) Simple budget plots for GnuCash - discussed in https://www.csun.io/2020/05/17/gnucash-finance.html

* **skr-json by baltpeter (last commit on 2018-07-17)** GnuCash account templates (Standard-Kontenrahmen) JSON

small_scripts by sercxanto (last commit on 2020-10-18) Simple scripts too small for own repo

square_transaction_parser by rwslippey (last commit on 2019-11-30) A simple script to help prepare square transaction csv data for import to accounting software like GNUCash

* **text-messaging-to-gnucash by chrisbrasington (last commit on 2016-01-20)** Create transactions via command-line text-messaging to gnucash sqlite database.

* **Timetracker-to-Gnucash-Invoice by EvansMike (last commit on 2012-03-15)** Takes Anuko Timetracker data and creates an Invoice in GnuCash

* **tws-gnucash by twswn (last commit on 2012-08-05)** (No description available)

UpdateBudgetQtrly by EpistemikPython (last commit on 2020-09-20) gnucash and google functions to update my BudgetQtrly document

* **visa-parser by pguridi (last commit on 2014-01-08)** A parser for the Visa pdf bill from www.visa.com.ar, useful for CSV import in Gnucash.

* **volksbank-csv-to-gnucash-csv-converter by Kaedo (last commit on 2017-11-05)** (No description available)

* **webgnucash by donautech (last commit on 2019-09-26)** Server for web version of GnuCash

* **WestpacGNUCashManager by jakeb1996 (last commit on 2017-08-20)** Export your Westpac transactions in QIF format and prepare them for GnuCash

* **zoysia by honthion (last commit on 2018-12-22)** gnucash python flask

Awk

* **credit-card-statement-reconciler by icyflame (last commit on 2019-07-27)** Scripts to reconcile your credit card statement with your manually maintained accounts from GnuCash

ibank2qif by tomszilagyi (last commit on 2020-02-06) Bank account transactions into GnuCash

C

* **gnc-balcheck by prebbz (last commit on 2018-02-25)** Quickly get the balance of a GnuCash which uses a MySQL backend

* **gnucash-2 by kleopatra999 (last commit on 2011-06-24)** Yet another clone of the gnucash source code

* **gnucash-aqplus by jhs-s (last commit on 2012-06-24)** Usually contains some fixes for aqbanking for GnuCash

* **Gnucash-gnucash by jimmymccord (last commit on 2018-05-19)** (No description available)

* **gnucash-jz-snap by jacobzimmermann (last commit on 2018-08-26)** (No description available)

* **gnucash-svn by kleopatra999 (last commit on 2010-04-08)** another clone of gnucash, but this time not using github's clone, and therefore keeping the svn metadata

* **gnucash_python by davidjo (last commit on 2018-07-20)** gnucash report writing in python

* **tk_gnucash3.3-python by tkerns1965 (last commit on 2018-11-30)** (No description available)

C#

* **bank2qif by piontec (last commit on 2019-08-19)** A companion project for https://www.gnucash.org/. Helps import bank statements to GnuCash.

BudgetApp by Lakendary (last commit on 2020-02-11) Budget web application for GnuCash

* **CS320GnuCashTesting by BrookJacob (last commit on 2018-12-13)** Repository for the testing of GnuCash for CS320

* **FinanceWeb by elohmeier (last commit on 2016-07-09)** C#/.NET OData Adapter for reading GnuCash Databases with MSFT Excel

* **gnucash2ledger-cli by marek-g (last commit on 2014-08-05)** Gnucash (general ledger html report) to ledger-cli converter.

GnuCash2Qif by Jason-Carter (last commit on 2020-02-02) Convert GnuCash Sqlite database to QIF format

* **GnuCashCSLib by kiranvr (last commit on 2018-02-20)** A C# library to read values from GNUCash xml files.

* **GnuCashDotNetAPI by SolidDynamics (last commit on 2019-08-11)** A .NET API for GnuCash using the C API https://wiki.gnucash.org/wiki/Using_the_API

* **GnuCashParser by nikitazu (last commit on 2015-02-19)** .Net parser for GnuCash files format

GnucashPlDataImportGenerator by AdrianS-PL (last commit on 2020-10-20) (No description available)

* **GnuCashSharp by rstarkov (last commit on 2017-07-17)** A library for reading data from GnuCash XML files.

C++

* **gnucash-butchered by iulianu (last commit on 2015-03-28)** My own butchered version of Gnucash

* **gnuCash-price-upload by gavin-blakeman (last commit on 2018-06-24)** Upload .csv files to gnuCash prices

* **gnuCash-pud by gavin-blakeman (last commit on 2018-08-18)** Commodity Price Upload Daemon for gnuCash

* **gnutreemfc by edkirkman (last commit on 2017-11-23)** gnucash using MFC GUI and MySQL backend

investmentManager by gavin-blakeman (last commit on 2020-05-29) Web Based Application to complement gnuCash and provide management and user interface to manage mutual funds

* **UnderBudget by vimofthevine (last commit on 2019-08-05)** Advanced personal budget analysis application that integrates with GnuCash, Quicken, mint.com, etc.

CSS

gnucash_gtk3 by davidschmitt (last commit on 2020-08-22) GTK3 CSS for customizing GnuCash appearance

Dart

dartcash by sandeep84 (last commit on 2020-08-18) Dartlang implementation of GNUCash sqlite format file support.

* **gnucash-flutter by pefdow (last commit on 2018-09-05)** Flutter implementation of gnucash-android

moneybags by sandeep84 (last commit on 2020-08-10) A GNUCash viewer application.

Diff

* **GnuCash-Windows-Fixes by theochino (last commit on 2015-03-19)** Pieces needed to Compile Gnu Cash on a Windows machine ... http://wiki.gnucash.org/wiki/User:Bilkusg

Dockerfile

docker-gnucash by mhitchens (last commit on 2020-06-13) GnuCash running via X11 over SSH

* **docker-gnucash by HodeiG (last commit on 2019-07-11)** docker-gnucash

* **docker-gnucash by Caveja (last commit on 2019-01-03)** Docker container with GnuCash built from source

* **docker-gnucash-novnc by bertlorenz (last commit on 2018-07-25)** (No description available)

* **gnucash-dev-docker by diablodale (last commit on 2019-07-23)** Docker containers for automated OS setup and dev/build environ for gnucash v3+ binaries and docs

gnucash-docker by aitor3ml (last commit on 2020-09-28) dockerized gnucash

gnucash-docker by mtbkapp (last commit on 2020-01-06) Run gnucash in docker with tigervnc and novnc for access from browser.

gnucash-docker by mtbkapp (last commit on 2020-01-06) Run gnucash in docker with tigervnc and novnc for access from browser.

GNUCash-Docker-Build by crossan007 (last commit on 2020-02-09) Docker based build environment for GNUCash

gnucash-docker-for-python by devbar (last commit on 2019-11-21) Container to provide working gnucash backend and python bindings

* **ppa-gnucash-xbt by msvalina (last commit on 2019-09-13)** Unofficial build of GnuCash with Bitcoin support for Ubuntu Bionic

F#

* **gnucash-tools by cantsin (last commit on 2015-11-01)** (No description available)

Gettext Catalog

* **gnucash-el by pgaval (last commit on 2015-03-14)** Automatically exported from code.google.com/p/gnucash-el

Go

bankcsv by lpenz (last commit on 2019-11-25) Tool that I use to convert the CSV from banks to gnucash3-compatible transaction CSV

coin by mkobetic (last commit on 2020-09-10) heavily simplified version of ledger-cli.org with a twist (very much a work in progress)

* **gnc-api-d by vinymeuh (last commit on 2019-07-18)** A read only REST server for GnuCash file

* **gnucash-csv-exporter by andrepnh (last commit on 2018-03-17)** Exports gnucash files to csv

gnucash-graphql by vinymeuh (last commit on 2020-03-05) A GraphQL server for GnuCash files

* **gnucash-parser by xavier268 (last commit on 2019-09-14)** Parse Gnucash files in Go

* **gnucash-viewer by mmbros (last commit on 2017-01-12)** A gnucash file viewer

* **gnucash-viewer-old by mmbros (last commit on 2016-12-18)** (No description available)

* **gocash by remyoudompheng (last commit on 2013-09-21)** gocash is a personal accounting interface similar to gnucash

p24fetch by tuxofil (last commit on 2020-09-26) Fetch transaction log from Privat24 for GnuCash

Groovy

* **Zio-Antunello by masokotanga (last commit on 2011-09-28)** un gnucash online (?)

HTML

* **Gnucash-gnucash-htdocs by jimmymccord (last commit on 2018-05-19)** (No description available)

gnucash-htdocs by Gnucash (last commit on 2020-10-22) GnuCash website.

* **gnucash-jp by omoshetech (last commit on 2016-10-07)** (No description available)

* **gnucashkr.github.io by GnucashKr (last commit on 2017-05-10)** gnucashkr.github.io

Haskell

* **hGnucash by pharaun (last commit on 2017-12-31)** Haskell xml library for parsing the gnucash file format

* **hs-gnucash by knupfer (last commit on 2015-07-09)** Haskell library to work with gnucash

Inno Setup

gnucash-on-windows by Gnucash (last commit on 2020-10-18) Support scripts to build gnucash for Windows using mingw32.

Java

* **accounting by milanogc (last commit on 2016-12-05)** This project is an attempt to create a GnuCash like system, i.e. it adopts the double entry bookkeeping accounting system, for the management of personal finances.

* **android_search-recycler-cardview-learning-gnucash by cc-shifo (last commit on 2017-10-20)** (No description available)

* **androidcash by mbarbon (last commit on 2011-08-09)** Simple Android GnuCash companion

BackupGnuCashLinux by goodvibes2 (last commit on 2020-05-20) Backup GnuCash for Linux (using openjdk + openjfx)

BackupGnuCashMigor by goodvibes2 (last commit on 2020-03-11) Backup GnuCash + Migor (my personal MS Access database)

BackupGnuCashWin by goodvibes2 (last commit on 2020-05-20) Backup GnuCash for Windows (using javafx)

* **barx by pgiu (last commit on 2015-03-05)** Exportador de la información del estado de cuenta de Banco Galicia a CSV/QIF para usar en MoneyManagerEx, GnuCash, etc.

* **barxm by pgiu (last commit on 2015-03-06)** Exportador de la información del estado de cuenta de Banco Galicia a CSV/QIF para usar en MoneyManagerEx, GnuCash, etc.

* **BudgetReportGnuCash by martinlong1978 (last commit on 2011-04-08)** Jasper Budget Report for GnuCash

* **convert-ingcsv-to-gnucash by jonaskoperdraat (last commit on 2016-06-12)** Application to convert csv export from ING to a format GnuCash can import

* **gcchart by jhogan (last commit on 2015-06-14)** An website for reading GnuCash datasources and creating charts written in Java.

* **gnc4a by bwduncan (last commit on 2011-01-16)** GnuCash Companion for Android is an mobile application for devices running Google's Android operating system, which will enable the users of GnuCash to do small things like adding a transaction or creating an invoice or expense voucher on the go.

* **gnc4a by glennji (last commit on 2012-05-06)** Gnucash for Android

* **GncImport by fcuenca (last commit on 2018-07-08)** GnuCash transaction import tool

* **GncXmlLib by fcuenca (last commit on 2015-12-05)** A small library to manipulate GnuCash data in XML format

* **GnuCash-2.6.5-importer by jan438 (last commit on 2015-02-02)** (No description available)

gnucash-android by codinguser (last commit on 2020-07-21) Gnucash for Android mobile companion application.

gnucash-android by BattleCupcake (last commit on 2020-09-25) (No description available)

gnucash-android by yjkang0602 (last commit on 2020-03-31) (No description available)

gnucash-android by yjkang0602 (last commit on 2020-03-31) (No description available)

gnucash-android by nicxleo (last commit on 2020-05-02) (No description available)

* **gnucash-android-example by felipecmuniz (last commit on 2018-05-28)** (No description available)

* **gnucash-merge by pnemonic78 (last commit on 2017-08-16)** Merge two gnucash XML files.

* **gnucash-utils by crankydillo (last commit on 2019-08-31)** (No description available)

gnuCashAdaptors by yrado (last commit on 2020-05-23) Scripts to make import to GnuCash simple

* **GnuCashBudgetReport by bvitale (last commit on 2012-01-21)** A budget report for GnuCash data that is stored in MySQL.

* **gnucashjgnash by leeboardtools (last commit on 2018-01-23)** Plugin for jGnash that converts a (simple) GnuCash database to jGnash

* **gnucashMobile by nhrdl (last commit on 2013-10-19)** (No description available)

* **gnuCashN by nyshthefantastic (last commit on 2017-10-16)** (No description available)

* **gnucashtest by krismess (last commit on 2019-09-26)** Automated test for Gnucash Android app

* **GnuCashToQIF by davidkgerman (last commit on 2011-12-11)** (No description available)

* **GnuCashViewer by jrmcsoftware (last commit on 2014-03-03)** GnuCash Viewer

IngAusOfxFixLinux by goodvibes2 (last commit on 2020-03-11) ING Australia OFX Fix for Linux - Fix OFX file before importing into GnuCash

IngAusOfxFixWin by goodvibes2 (last commit on 2020-03-11) ING Australia OFX Fix for Windows - Fix OFX file before importing into GnuCash

* **javacash by nhrdl (last commit on 2013-08-09)** Yet another implementation of gnucash in Java

javacash by brentwalther (last commit on 2020-10-13) An application for managing money. Inspired by GnuCash.

* **jgc by kevemueller (last commit on 2018-06-11)** Java library to read GnuCash files

jGnuCash2Qif by Jason-Carter (last commit on 2019-10-27) Convert GnuCash Sqlite database to QIF format - the Java version

* **jgnucashlib by tdf (last commit on 2012-07-10)** jGnucashLib - a java access to GnuCash files

Maakboekingen by zwijze (last commit on 2020-10-13) Maakboekingen in gnucash

* **nordea-to-gnucash by mohamedamer (last commit on 2013-12-08)** (No description available)

* **workspace_gnucash by won21kr1 (last commit on 2014-04-16)** (No description available)

JavaScript

* **bcqif by hugozap (last commit on 2015-09-23)** Script simple para generar archivos QIF a partir de archivos .txt exportados desde Bancolombia y poder cargar los registros a programas como GnuCash

cash-cow by Lkxz (last commit on 2020-06-30) A basic double-entry bookeping system, similar to GNUCash, written in Go and React.

cashdash by mbugert (last commit on 2020-03-22) Interactive visualization of GnuCash data based on plotly Dash.

* **cashviz by nunofgl (last commit on 2017-12-28)** Visualizations for GNUCash data.

* **finance_dashboard by manicolosi (last commit on 2014-03-09)** A dashing dashboard to show financial information from GnuCash

* **finquick by dckc (last commit on 2018-10-04)** web app access to gnucash financial data

gnucash-browser by phjardas (last commit on 2020-10-20) Web Interface for GnuCash Ledgers

* **gnucash-django by nylen (last commit on 2015-11-20)** Simple Web frontend for GnuCash, using Django

* **gnucash-ext by matthewbednarski (last commit on 2015-05-30)** (No description available)

gnucash-ppa by chenghlee (last commit on 2020-04-19) Sources for the chenghlee/gnucash PPA

* **gnucash-price-importer by cortopy (last commit on 2018-05-05)** Script for importing historic currency prices into gnucash

* **gnucash-reporter by AaronLenoir (last commit on 2017-03-06)** Visualizes some reports on GnuCash data.

* **gnucash-reporting-view by MeerkatLabs (last commit on 2018-04-24)** Angular JS Based viewer for gnucash-reports

gnucash-sql by wraithgar (last commit on 2020-07-04) Gnucash sqlite thingy

* **gnucash-viewer by drjeep (last commit on 2013-07-19)** Web viewer for Gnucash using Python/Flask

* **gnucash-web by mrkrstphr (last commit on 2013-06-17)** (No description available)

* **GnucashReportingNVD3 by sholly (last commit on 2017-07-04)** NVD3 frontend/playground for gnucash reporting..

monalyzer by Vlad-ku (last commit on 2020-03-13) QIF (GnuCash)

* **profitcash-restful by TheProfitwareGroup (last commit on 2012-09-16)** [DEPRECATED] ProfitCash-RESTful is a RESTful service providing accounts and transaction information based on imported from GnuCash into MongoDB data.

qif-converter by matthijsmelissen (last commit on 2020-09-07) Converts CSV files from ING (Netherlands) and BCEE (Luxembourg) into QIF files. Suitable for GnuCash.

* **remotegnucash by justinhunt1223 (last commit on 2017-04-17)** Remote GnuCash

* **salis by f0x52 (last commit on 2018-04-05)** GNUCash alternative

* **skilap by sergeyksv (last commit on 2017-06-26)** Personal online applications, GnuCash clone and others

vue-gnucash by biker2000on (last commit on 2020-09-08) (No description available)

webgnucash-client by donautech (last commit on 2020-09-10) Client part of Web version of GnuCash

Jupyter Notebook

gnucash-historical-price-data by MichaelSchmidt82 (last commit on 2020-04-10) Add historical stock price data to gnucash ledgers.

* **gnucash-playground by nlzimmerman (last commit on 2016-08-15)** Just playing around with OFX and gnucash files; I am presumably the only person who cares about this sort of things.

* **gnucash_analysis by prattmic (last commit on 2018-12-22)** Analyze GnuCash data with Pandas

GnuCash_tools by PingWIn87 (last commit on 2020-09-23) My toolset for GnuCash

* **py-gnucash by ihkihk (last commit on 2017-05-17)** Python scripts for analysis of gnucash database

PHP

* **buchungen by jungepiraten (last commit on 2016-05-26)** Webinterface für gnucash-Datenbank mit Funktion zum Verifizieren von Buchungen

* **cash-manager by jUnG3 (last commit on 2017-02-18)** (No description available)

* **cashonline-php-server by okovalov (last commit on 2014-10-31)** Backend part for cashonline project (clone of gnucash)

* **gnucash-dashboard by bkrijg (last commit on 2017-12-28)** (No description available)

* **gnucash-eloquent by b3it (last commit on 2016-05-31)** Laravel Eloquent Models for the GnuCash MySQL Backend

* **gnucash-laravel by xstat (last commit on 2015-12-21)** (No description available)

* **gnucash-php by cebe (last commit on 2014-08-04)** A library for reading gnucash XML format in PHP

* **gnucash-reports by xstat (last commit on 2017-07-30)** (No description available)

* **gnucash-tools by cccmzwi (last commit on 2013-12-16)** Convert your onlinebanking-export (CSV) to a neatly pre-categorized QIF-File which can be imported in Quicken or Gnucash

* **gnucash.cakephp by claudineimatos (last commit on 2014-06-25)** (No description available)

* **gnucashreports by pedroabel (last commit on 2013-02-08)** Set of custom reports that I use for my personal finances. To see the reports working on a sample database, check the website http://gnucashreports.comuf.com/ ATTENTION: many problems yet. Many bugs that did not happen in my computer happened in this sample server.

* **php-gnucash by mrkrstphr (last commit on 2013-06-17)** (No description available)

PhpGnuCashMatchTransactions by puggan (last commit on 2019-10-31) Tool for GnuCash databases, to import and match up transactions from banks export-files

* **plans-comptables-francais by Seb35 (last commit on 2019-04-17)** Plans comptables français aux formats CSV et GNUCash

* **ThinkopenAt.Gnucash by kraftb (last commit on 2018-01-12)** A TYPO3 Flow package which allows to interface the Gnucash book keeping application

txs by vvvitaly (last commit on 2019-11-07) Export bills from different sources into Gnucash-like CSV

PLSQL

gnucash-mysql-additions by ohingardail (last commit on 2019-11-02) Custom MySql functions to add useful functions to GnuCash

Perl

* **bsestocks by poomalairaj (last commit on 2017-06-07)** Perl module to fetch live price from Bombay Stock Exchange (BSE) for Gnucash Finance

* **budgetProgram by Beahmer89 (last commit on 2016-10-23)** Converts GNUCash programs xml output to csv file to see monthly/yearly spending habits

Finance–Quote–YahooJapan by LiosK (last commit on 2020-01-26) Finance::Quote::YahooJapan - A Perl module that enables GnuCash to get quotes of Japanese stocks and mutual funds from Yahoo! Finance JAPAN.

* **finance-bomse by abhijit86k (last commit on 2017-11-06)** A Perl module for fetching quotes for Indian stocks, intended for use with gnucash

* **finance-quote-brazil by romuloceccon (last commit on 2018-06-15)** Gnucash/Finance::Quote modules to fetch prices of Brazilian stocks, bonds and mutual funds

* **FinanceQuoteBr by ailtonljr (last commit on 2017-09-08)** Perl Finance Quote modules for Brazilian market. Original files from https://lists.gnucash.org/pipermail/gnucash-br/2016-November/000535.html

* **gc2latex by wertarbyte (last commit on 2011-02-25)** create pretty Gnucash invoices with LaTeX and Perl

* **GnuCash-Branch by LiosK (last commit on 2015-04-26)** GnuCash::Branch - Utilities to implement branch accounting with GnuCash.

* **gnucash-extensions by tomason (last commit on 2017-08-02)** (No description available)

* **gnucash-monthly-budget-report by mhodapp (last commit on 2015-03-01)** perl program to generate monthly budget reports

* **GnuCash-MySQL by theochino (last commit on 2015-03-18)** A MySQL module to Gnu Cash

* **gnucash-perl by goblin (last commit on 2011-01-29)** Perl modules for reading and writing the Gnucash XML file

* **gnucash-quote-sources by tjol (last commit on 2017-04-23)** Some useful methods to fetch market price data for GnuCash

* **GnuCash-SQLite by hoekit (last commit on 2015-05-20)** A perl module to access GnuCash SQLite files.

* **GnuCash-SQLite by gitpan (last commit on 2015-01-08)** Read-only release history for GnuCash-SQLite

* **gnucash-summarizer by Nazrax (last commit on 2016-05-06)** (No description available)

gnucash-xml-to-ledger-dat by icyflame (last commit on 2020-02-11) A script to convert GNUCash's XML file to Ledger's dat file

* **gnucash2mysql by xaprb (last commit on 2014-01-18)** Gnucash to MySQL export script.

* **gnucash2mysql_extras by hmackiernan (last commit on 2018-12-23)** Extra scripts and bits for gnucash2mysql

* **gnuCash_Quote by z-Wind (last commit on 2016-01-06)** modify perlsitelibFinanceQuote.pm

* **MoneyDance-GnuCash-Importer by ets (last commit on 2015-10-31)** Script to convert GnuCash data into a native MoneyDance XML file format

* **Paypal-csv-to-qif-converter by sonologic (last commit on 2011-05-04)** Converts paypal .csv to qif for importing in gnucash

* **perl-gnucash-reader by hoekit (last commit on 2014-06-14)** (No description available)

PerlFinanceQuoteBloomberg **by alex314159 (last commit on 2020-07-01)** Bloomberg module for the Perl Finance::Quote module (used in particular by GnuCash)

* query_gnucash_db **by hmackiernan (last commit on 2016-08-20)** Perl script to query a MySQL db created by the 'gnucash2mysql' script

Perl6

* perl6-gnucash **by eikef (last commit on 2016-08-24)** Use gnucash library from Perl 6

R

gnucashAndR **by mrop (last commit on 2020-07-23)** (No description available)

* ShinyBudgetAnalysis **by paulheider (last commit on 2017-04-06)** A Shiny app (R-based dashboard) that gives insight into your GnuCash budget habits over time.

Roff

* gnucash-docker **by rusodavid (last commit on 2019-04-20)** (No description available)

Ruby

* accounting **by freegeek-pdx (last commit on 2013-11-18)** accounting utilities for xtuple import and allocation in xtuple and gnucash

* arges **by isimluk (last commit on 2016-12-31)** Calc roe from gnucash transaction log

* banks-to-gnucash **by toniprada (last commit on 2019-01-10)** Quaterly bank reports in CSV GnuCash-friendly format directly in your email.

boekhouden-met-gnucash **by mauritslamers (last commit on 2020-05-21)** Een Nederlandse handleiding over het boekhouden met GnuCash

* cnab2ofx **by abinoam (last commit on 2016-01-04)** CNAB240 to OFX conversion script

* dnbnor2qif **by kentdahl (last commit on 2016-09-21)** dnbnor2qif is a simple tool to help integrate data from the DnBNOR online bank monthly transcripts ("kontoutskrift") to a QIF accepting financial program, i.e. GnuCash.

* equity_flow **by fernandors87 (last commit on 2018-07-20)** A personal asset management software

* gnucash-invoice **by ixti (last commit on 2018-07-28)** Easy to use invoice printer for GnuCash.

* gnucash-rb **by vbatts (last commit on 2012-07-11)** Ruby access to Gnucash SQL database

* gnucash-summarizer **by arthurljones (last commit on 2019-01-10)** (No description available)

* gnucash2bmd **by ngiger (last commit on 2017-05-30)** Convert GnuCash CSV files into CSV which can be read by http://www.bmd.com/

* gnucash2ledger **by xaviershay (last commit on 2016-09-28)** Convert GnuCash files to a format supported by the ledger command line application

* gnucash_export **by alibby (last commit on 2009-10-26)** Export gnucash data to sqlite/ csv

* gnucash_getquotes **by hubcity (last commit on 2018-03-14)** (No description available)

rcash **by salex (last commit on 2020-10-21)** Rails Double Entry Accounting app patterned after GNUCash

ruby-gnucash by holtrop (last commit on 2020-02-25) Ruby library for extracting data from GnuCash data files

* **vfwcash by salex (last commit on 2018-09-06)** A Ruby CLI application that produces PDF reports from GnuCash

Rust

stay-the-course by DavidCain (last commit on 2020-09-22) Lazy portfolio rebalancer for GnuCash users

SQLPL

* **gnucash-tools by schoettl (last commit on 2017-07-05)** Collection of tools to work with GnuCash efficiently

Scala

* **gnucash-stuff by crankydillo (last commit on 2011-12-30)** (No description available)

* **GnuCashExtractor by Winbee (last commit on 2015-08-12)** Extract data from gnuCash and copy it into an open document spreadsheet

YNAB4toGnuCashMigrationTool by galbarm (last commit on 2020-02-18) YNAB 4 to GnuCash Migration Tool

Scheme

* **bas-report by spandan888 (last commit on 2017-08-11)** GST India Report / Business Tax Report

* **gc-decl-reports by yawaramin (last commit on 2014-02-09)** GnuCash declarative reports

* **gnucash-account-balance-chart by timabell (last commit on 2009-04-11)** account balance line chart for gnucash

* **gnucash-custom-reports by BenBergman (last commit on 2018-07-11)** (No description available)

* **gnucash-multicolumn by daniel-beet (last commit on 2017-09-18)** Advanced date and text filtering and multicolumn reports

* **gnucash-paypal-invoice-template by charlesmulder (last commit on 2018-06-20)** GnuCash invoice template that resembles a PayPal invoice

* **gnucash-reports by dschwen (last commit on 2016-01-27)** Custom reports for Gnucash

* **gnucash-reports by trailbound (last commit on 2012-08-03)** Custom set of gnucash reports, currently in development.

* **gnucash-reports by cnuahs (last commit on 2015-12-29)** Custom reports for use with GnuCash (http://gnucash.org/).

gnucash-reports by jaminh (last commit on 2020-08-06) Reports for personal finance using gnucash

* **gnucash-reports by wentzel (last commit on 2016-11-14)** Some nice reports for GnuCash

* **gnucash-statement-table by waldeinburg (last commit on 2016-01-13)** GnuCash report with cash flow over time

* **gnucashportable by GordCaswell (last commit on 2016-12-20)** GnuCash packaged in PortableApps.com Format

* **GnuCashReports by wlcasper (last commit on 2018-03-03)** Custom Reports for GnuCash

ibr-gnc-module by ErwinRieger (last commit on 2020-03-28) GnuCash Erweiterungen für deutsche Buchhaltung

Shell

* **AccountsConvertToGnucash by tontako (last commit on 2016-12-23)** Convert Kakeibo(Android Application) CSV exports to QIF format (usable by GnuCash and others)

ansible-role-gnucash by alvistack (last commit on 2020-10-21) Ansible Role for GnuCash Installation

* **archlinux-gnucash-latest by nengxu (last commit on 2015-03-31)** Archlinux AUR package building scripts for latest Gnucash

* **docker_gnucash by tkerns1965 (last commit on 2018-01-19)** (No description available)

* **docker_gnucash_novnc by tkerns1965 (last commit on 2018-01-25)** (No description available)

* **gnucash-build by hanulhan (last commit on 2018-02-20)** (No description available)

* **gnucash-build-script by z-Wind (last commit on 2019-03-08)** for ubuntu

* **gnucash-csv-import by thomasramapuram (last commit on 2016-08-08)** (No description available)

* **gnucash-devel by aur-archive (last commit on 2015-08-15)** (No description available)

* **gnucash-docker by limitedAtonement (last commit on 2017-01-27)** docker image for repeatable gnucash builds.

* **gnucash-docs by aur-archive (last commit on 2015-08-15)** (No description available)

* **gnucash-docs-old by yasuakit (last commit on 2011-09-24)** Manual and User Guide for Gnucash, the open-source accounting program

* **gnucash-hbci by aur-archive (last commit on 2015-08-15)** (No description available)

* **gnucash-on-debian by rayelnigma (last commit on 2018-09-16)** a set of build scripts to build gnucash using Ninja for faster builds

* **gnucash-on-fedora-copr by zhiqinghuang (last commit on 2015-10-29)** A set of build scripts geared towards creating rpm packages for gnucash and gnucash-docs on Fedora's Copr infrastructure. It's currently used to build nightly rpm packages for the maint and master branches intended for testing changes since the last release..

gnucash-on-flatpak by Gnucash (last commit on 2020-10-15) Packaging scripts to generate flatpaks directly from gnucash and gnucash-docs git repositories

gnucash-on-osx by Gnucash (last commit on 2020-10-03) Gtk-OSX moduleset, gtk-mac-bundler bundles, and ancillary files for creating GnuCash OSX Application Bundle.

gnucash-on-osx by jralls (last commit on 2020-07-12) Build GnuCash on OSX without X11

* **gnucash-python by aur-archive (last commit on 2015-08-15)** (No description available)

* **gnucash-svn by aur-archive (last commit on 2015-08-15)** (No description available)

gnucash-util-jp by mikkun (last commit on 2020-08-09) GnuCash

gnucash.AppImage by ecmu (last commit on 2020-04-12) AppImage build for gnucash

* **gnucash.SlackBuild by botzkobg (last commit on 2015-04-02)** SlackBuild script to compile GnuCash

* **LittleBudget by kstripp (last commit on 2012-06-17)** Little Budget Tool for GNUCash

* **nordea2ofx by nsrosenqvist (last commit on 2014-10-24)** A quick implementation of a converter between Nordea's CSV export to OFX so that it can be imported into various applications, such as Homebank or GnuCash. Only supporting Swedish and my edge cases, please consider improving the script and send a pull request for the changes.

* **TW5-GNUCash-Assistant by JulioCantarero (last commit on 2016-03-12)** A custom edition of TiddlyWiki5 designed to collect financial information from your banks and export them in QIF format

Swift

* **cash by cjwirth (last commit on 2017-07-03)** iOS companion app for GnuCash

TSQL

gnucash-reports by fredzica (last commit on 2020-01-12) Custom reports that use data from gnucash's SQL database

* **GNUCashProcs by nicholasceliano (last commit on 2019-10-22)** (No description available)

Tcl

* **pycash by davinirjr (last commit on 2015-01-25)** Some Python utilites that GnuCash users may find useful.

TypeScript

gnucash-global-importer by klodzack (last commit on 2020-09-10) (No description available)

gnucash-graphql by phjardas (last commit on 2020-07-17) GraphQL Wrapper For GnuCash Ledgers

GnuCashImporter by nicholasceliano (last commit on 2020-07-19) (No description available)

v9-geckos-team-07 by chingu-voyages (last commit on 2020-09-05) GeckoCash: A web-based GnuCash clone.

* **web-cash by ashishmondal (last commit on 2017-06-29)** Web version of GnuCash

* **web-cash by ashishmondal (last commit on 2017-06-29)** Web version of GnuCash

Visual Basic

* **OneClickToQif by OneClickToQif (last commit on 2017-04-23)** OneClickToQIF consists of a set of templates and macros, which are used to export data from your spreadsheets to QIF format, as used by programs such as GnuCash, Money and Quicken. You can use the templates as provided, or adapt your own spreadsheet, so your data is automatically exported to QIF with a single click.

XSLT

* **asciidoc-conversion by codesmythe (last commit on 2018-08-27)** Script and tools to convert GnuCash DocBook XML to AsciiDoc

gnucash-docs by Gnucash (last commit on 2020-09-27) Documentation for GnuCash Accounting Program.

* **gnucash-docs by mattig7 (last commit on 2018-02-10)** (No description available)

* **Gnucash-gnucash-docs by jimmymccord (last commit on 2018-05-19)** (No description available)

Unknown

* **accounting-plans by jeblad (last commit on 2018-02-09)** Accounting plans for GnuCash

* **accounts by sjtug (last commit on 2016-09-25)** Accounts of SJTUG in GNUCash Format

accounts_gnucash by Baneeishaque (last commit on 2020-02-29) (No description available)

* **an-gnucash by wanjing (last commit on 2012-07-30)** android app for gnucash

* **ansible-role-gnucash by wtanaka (last commit on 2019-08-26)** Ansible role for installing gnucash

bookee by hesy-mzh (last commit on 2020-08-22) Account data for GnuCash

budget by bhagdave (last commit on 2020-10-24) My Budget from Gnucash

* **chloris by honthion (last commit on 2018-12-17)** gnucash python django

* **Church-Accounting-Using-GnuCash by leggie (last commit on 2017-11-18)** Various excel templates containing vba scripts that makes church accounting using the open source accounting tool GnuCash as described in the blog "http://financeandaccountingforchurches.blogspot.in/2012/12/church-accounting-using-gnucash-1.html". The files in the links mentioned there are hosted in this repository

* **CICtoGNUcash by jbtruffault (last commit on 2016-06-15)** (No description available)

* **commerzbank-csv4gnucash by zanto001 (last commit on 2018-12-05)** Tweak CSV exports from commerzbank for importing into GnuCash.

* **ComptaTest by palric (last commit on 2017-05-05)** Exercices programmation python/panda: importation de fichiers comptas bancaires, formatage et importation dans GnuCash

* **CPA-006-Asap-Cheques by dougransom (last commit on 2017-09-26)** Gnucash Check Configurations to print on Canadian Cheques from ASAP Cheques.

csv2qif by andreaZHrustichelli (last commit on 2020-09-15) Python program to convert csv file into a qif file ready to import in GNUCASH

* **docker-gnucash by mbessler (last commit on 2015-07-15)** Containerized GnuCash

* **docker-gnucash by rainu (last commit on 2017-11-20)** A gnucash docker image

* **docker-gnucash by sgalkin (last commit on 2018-01-31)** gnucash docker image with SpiderOakOne integration

* **docker-mobile-gnucash by au-phiware (last commit on 2016-12-20)** Docker containers for justin-hunt1223/mobilegnucash

GCtranslate by AshokR (last commit on 2020-06-30) GnuCash Translation

* **GnuCash-3part-check by agh1 (last commit on 2012-06-06)** A 3-part check format for GnuCash

gnucash-accounts by ordtrogen (last commit on 2019-12-18) Some Account Hierarchy Templates for GnuCash in Swedish

* **gnucash-android-master by abdullahwale (last commit on 2018-10-19)** (No description available)

* **gnucash-api by jjuanda (last commit on 2013-12-14)** REST APIs for GnuCash files

gnucash-build-debian by willelop (last commit on 2020-09-06) Installs in Debian all the required dependencies for building gnucash

* **gnucash-compose by daveyb (last commit on 2017-03-11)** docker-compose file(s) to bring up local gnucash cluster

* **gnucash-data by anshprat (last commit on 2015-06-02)** data files for my gnucash encrypted with gpg

* **gnucash-docker by KaiLemke (last commit on 2017-12-18)** containerized gnucash 2.6

* **gnucash-docker by dbcesar (last commit on 2018-04-16)** Dockerfile and docker-compose to install and run gnucash 3.0 from a docker container

* **gnucash-docs by cygwinports (last commit on 2018-04-09)** Cygwin gnucash-docs packaging

* **gnucash-fire-tools by KarolOlko (last commit on 2018-10-16)** Slicing gnucash DB for more insights

* **gnucash-guide-asciidoc by codesmythe (last commit on 2018-08-23)** The GnuCash Tutorial and Concept Guide, converted to Asciidoc

* **gnucash-my_chart_of_accounts by 5472qaywsx (last commit on 2019-02-25)** my personal chart of accounts for gnucash

gnucash-pot by fellen (last commit on 2020-03-25) Portable Template of GnuCash

* **gnucash-queries by g2010a (last commit on 2016-10-20)** Queries to extract data from Gnucash's database

* **gnucash-rest by mhitchens (last commit on 2014-05-03)** A Spring Data/Spring REST interface to a gnucash data file

gnucash-stuff by ajablonski (last commit on 2020-08-31) (No description available)

GnuCash-Tutorials by duguqiubailee (last commit on 2020-10-19) (No description available)

* **gnucash-web by djbrown (last commit on 2016-01-01)** (No description available)

GNUCASH.APP-ERICK-CLEWIS by ErickClewisAccountsdatamoneymarketing (last commit on 2020-03-03) USD$ DATABASE OPEN SOURCE DATA

* **gnucash_auto by torchtarget (last commit on 2017-08-18)** A simple transaction importer for Gnucash

* **gnucashdeb by twt2 (last commit on 2019-09-28)** (No description available)

* **gnucashdockeraws by fervincent (last commit on 2019-03-29)** (No description available)

* **GnucashSW by Oldobaba (last commit on 2018-07-04)** (No description available)

gnucashTelegram by f-angi (last commit on 2020-02-14) (No description available)

* **JsCash by rafaelbeckel (last commit on 2019-04-24)** Double entry accounting system inspired by GNUCash

* **kg7je by twt2 (last commit on 2019-09-12)** Stephen Butler's Debian Package Files for GnuCash

* **knab-gnucash-converter by bkrijg (last commit on 2017-05-03)** A python script for converting KNAB bankstatement information files into a suitable gnucash csv import file

* **lepturus by honthion (last commit on 2018-12-17)** gnucashH5 vue

moflow by msobkow (last commit on 2020-10-22) Money Flow is a revisioning of GNU Cash as a web-enabled Spring Tool Suite 4 application set

* **nordea-csv2qif by martinolsen (last commit on 2012-09-09)** Convert Nordea CSV exports to QIF format (usable by GnuCash and others)

org.gnucash.GnuCash by flathub (last commit on 2020-09-27) (No description available)

* **PayPalConvert by rowantree (last commit on 2015-11-04)** Convert PayPal csv files to qif for loading into GnuCash

* **perotis by honthion (last commit on 2018-12-17)** gnucashApp flutter

* **rabo2qif by milovanderlinden (last commit on 2011-01-25)** export mut.txt to qif for gnucash

* **resource-gnucash-build by hgati (last commit on 2018-03-21)** (No description available)

Singularity-GNUCash by rgrandin (last commit on 2019-12-30) Fedora 28 build of GNUCash

* **splitwise-to-gnucash by yfede (last commit on 2018-08-07)** A script to grab transactions from Splitwise and export them in CSV form for import in GnuCash

* **tk_ansible_gnucash_vscode01 by tkerns1965 (last commit on 2019-08-28)** (No description available)

* **Vishal-Ramteke by vishal7788 (last commit on 2019-09-01)** GnuCash Mobile Automation Test

* **Vishal-Ramteke by vishal7788 (last commit on 2019-09-01)** GnuCash Mobile Automation Test

webcash by turlog (last commit on 2020-06-17) WebCash is a proxy service that allows connecting to GNUCash databases with REST API.

8.3.5 Python links

- cross compilation of python executable from Linux to Windows : http://milkator.wordpress.com/2014/07/19/windows-executable-from-python-developing-in-ubuntu/
- SQLAlchemy page: http://www.sqlalchemy.org/

8.3.6 Threads used during the course of development

- sphinx error message: http://stackoverflow.com/questions/15249340/warning-document-isnt-included-in-any-toctree-for-include

8.3.7 Thanks

None of this could be possible without :

- the GnuCash project, its core team of developers and its active community of users
- python and its packages amongst which sqlalchemy
- github, readthedocs and travis-ci for managing code, docs and testing

The todo list:

- write more tests
- review non core objects (`budget`, `business`)
- build a single exe to ease install on windows (following http://milkator.wordpress.com/2014/07/19/windows-executable-from-python-developing-in-ubuntu/)

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

D

E

F

G

H

I

Q

R

S

T

U

V

W