finac

Jun 13, 2023

Contents

CHAPTER 1

Function aliases

The following functions are aliases for *Core functions*:

```
import finac as f

f.tr(...) # alias for finac.core.transaction_create
f.mv(...) # alias for finac.core.transaction_move
f.rm(...) # alias for finac.core.transaction_delete
f.complete(...) # alias for finac.core.transaction_complete
f.apply(...) # alias for finac.core.transaction_apply
f.stmt(...) # alias for finac.core.account_statement_summary
f.balance(...) # alias for finac.core.account_balance
f.balance_range(...) # alias for finac.core.account_balance_range, in
                     # opposite to the original function, returns dates as
                     # datetime object by default
f.df(...) # alias for finac.df.df
```

CHAPTER 2

Special interactive functions

finac.**format_money**(*amnt*, *precision=2*)
Format output for money values

Finac doesn't use system locale, in the interactive mode all numbers are formatted with this function. Override it to set the number format you wish

finac.**ls**(*account=None, asset=None, tp=None, passive=None, start=None, end=None, tag=None, pending=True, hide_empty=False, order_by=['tp', 'asset', 'account', 'balance'], group_by=None, base=None*)
Primary interactive function. Prints account statement if account code is specified, otherwise prints summary for all accounts

Account code may contain '%' symbol as a wildcard.

Parameters

- **account** – account code
- **asset** – filter by asset code
- **tp** – filter by account type (or types)
- **passive** – list passive, active or all (if None) accounts
- **start** – start date (for statement), default: first day of current month
- **end** – end date (or balance date for summary)
- **tag** – filter transactions by tag (for statement)
- **pending** – include pending transactions
- **hide_empty** – hide empty accounts (for summary)
- **order_by** – column ordering (ordering by base is not supported)
- **base** – specify base asset

finac.**lsa**(*asset=None*, *start=None*, *end=None*)
Print list of assets or asset rates for the specified one

Currency filter can be specified either as code, or as pair “code/code”

If asset == ‘*’ - print rates table

Parameters

- **asset** – asset code
- **start** – start date (for rates), default: first day of current month
- **end** – end date (for rates)

CHAPTER 3

DataFrame functions

Convert Finac data to Pandas DataFrame

finac.df.**df**(*fn*, **args*, ***kwargs*)

Get Finac DB data as Pandas DataFrame

Converts Finac data to Pandas DataFrame. Requires pandas Python module.

- rate - asset_rate
- asset - asset_list
- account - account_list
- statement - account_statement
- balance_range - account_balance_range
- rate_range - asset rate_range

Parameters

- **fn** – rate, asset, account, statement or balance
- **arguments** (*other*) – passed to called function as-is

Returns formatted Pandas dataframe

Raises ValueError – if invalid function has been specified

CHAPTER 4

Plot functions

Wrappers around *matplotlib.pyplot*

finac.plot.**account_pie**(*tp=None*, *asset=None*, *mb=0*, *base=None*, *passive=None*, *group_by=None*, *shadow=True*, *autopct='%1.1f%%'*, ***kwargs*)

Plot pie chart of the account balances

Parameters

- **tp** – account types to include
- **mb** – min balace (or section goes to "other")
- **base** – base asset to recalc amounts (default: usd)
- **shadow** –
- **autopct** –
- ****kwargs** – passed as-is to matplotlib.pyplot.pie

finac.plot.**account_plot**(*account=None*, *tp=None*, *start=None*, *end=None*, *step=1*, *base=None*, ***kwargs*)

Plot account balance chart for the specified time range

Either account code or account types must be specified

Parameters

- **account** – account code
- **tp** – account type (or types)
- **start** – start date/time, default: first day of current month
- **end** – end date/time, if not specified, current time is used
- **step** – chart step in days
- **base** – base currency
- ****kwargs** – passed as-is to matplotlib.pyplot.plot

CHAPTER 5

Core functions

Core functions provide Finac API when embedded. Core module should not be imported directly, please import main Finac module instead:

```
import finac as f

f.<function(...)>
```

exception finac.core.**OverdraftError**
Raised when transaction is trying to break account max overdraft

JRPC code: -32003

exception finac.core.**OverlimitError**
Raised when transaction is trying to break account max balance

JRPC code: -32004

exception finac.core.**RateNotFound**
Raised when accessed asset rate is not found

JRPC code: -32002

exception finac.core.**ResourceAlreadyExists**
Raised when trying to create already existing resource

JRPC code: -32005

exception finac.core.**ResourceNotFound**
Raised when accessed resource is not found

JRPC code: -32001

finac.core.**account_balance**(*account=None*, *asset=None*, *tp=None*, *base=None*, *date=None*, *_natural=False*, *_time_ms=False*)
Get account balance

Parameters

- **account** – account code

- **asset** – account asset filter
- **tp** – account type/types, value, list or values, separated with |
- **base** – base asset (if not specified, config.base_asset is used)
- **date** – get balance for specified date/time

finac.core.**account_balance_range**(*start*, *account=None*, *asset=None*, *tp=None*, *end=None*, *step=1*, *return_timestamp=False*, *base=None*, *_time_ms=False*)

Get list of account balances for the specified time range

step argument usage:

4, 4d - 4 days 2h - 2 hours 5a - split time range into 5 parts

Parameters

- **account** – account code
- **asset** – account asset filter
- **tp** – account type/types
- **start** – start date/time, required
- **end** – end date/time, if not specified, current time is used
- **step** – time step
- **return_timestamp** – return dates as timestamps if True, otherwise as datetime objects. Default is False

Returns tuple with time series list and corresponding balance list

finac.core.**account_create**(*account*, *asset*, *tp='current'*, *note=None*, *passive=None*, *max_overdraft=None*, *max_balance=None*)

Parameters

- **asset** – asset code
- **account** – account code
- **note** – account notes
- **passive** – if True, account is considered as passive
- **tp** – account type (credit, current, saving, cash etc.)
- **max_overdraft** – maximum allowed overdraft (set to negative to force account to have minimal positive balance), default is None (unlimited)
- **max_balance** – max allowed account balance, default is None (unlimited)

Accounts of type 'tax', 'supplier' and 'finagent' are passive by default

finac.core.**account_credit**(*account=None, asset=None, date=None, tp=None, order_by=['tp', 'account', 'asset'], hide_empty=False*)

Get credit operations for the account

Parameters

- **account** – filter by account code
- **asset** – filter by asset code

- **date** – get balance for specified date/time
- **tp** – FIlter by account type
- **sort** – field or list of sorting fields
- **hide_empty** – don't return zero balances

Returns generator object

finac.core.**account_debit**(*account=None, asset=None, date=None, tp=None, order_by=['tp', 'account', 'asset'], hide_empty=False*)

Get debit operations for the account

Parameters

- **account** – filter by account code
- **asset** – filter by asset code
- **date** – get balance for specified date/time
- **tp** – FIlter by account type
- **sort** – field or list of sorting fields
- **hide_empty** – don't return zero balances

Returns generator object

finac.core.**account_delete**(*account*, *lock_token=None*)

Delete account

finac.core.**account_info**(*account=None*)

Get dict with account info

If no account is specified, generator object with info for all accounts is returned

finac.core.**account_list**(*asset=None, tp=None, passive=None, code=None, date=None, base=None, order_by=['tp', 'asset', 'account', 'balance'], group_by=None, hide_empty=False, _time_ms=False*)

List accounts and their balances

Parameters

- **asset** – filter by asset
- **tp** – filter by account type (or types), value, list or values, separated with |
- **passive** – list passive, active or all (if None) accounts
- **code** – filter by acocunt code (may contain '%' as a wildcards)
- **date** – get balances for the specified date
- **base** – convert account balances to base currency
- **order_by** – list ordering
- **group_by** – 'asset' or 'type'
- **hide_empty** – hide accounts with zero balance, default is False

finac.core.**account_list_summary**(*asset=None, tp=None, passive=None, code=None, date=None, order_by=['tp', 'asset', 'account', 'balance'], group_by=None, hide_empty=False, base=None, _time_ms=False, _rsingle=False*)

List accounts and their balances plus return a total sum

Parameters

- **asset** – filter by asset
- **tp** – filter by account type (or types)
- **passive** – list passive, active or all (if None) accounts
- **code** – filter by acocunt code (may contain '%' as a wildcards)
- **date** – get balances for the specified date
- **order_by** – list ordering
- **group_by** – 'asset' or 'type'
- **hide_empty** – hide accounts with zero balance, default is False
- **base** – base asset (if not specified, config.base_asset is used)

Returns

list of accounts or assets: list of assets or account_types: list of accoun types

total: total sum in base asset

Return type accounts

finac.core.**account_lock**(*account*, *token*)

Lock account

Account locking works similarly to threading.RLock(), but instead of thread ID, token is used.

If token is provided and match the current lock token, lock counter will be increased and lock is passed

When locked, all account transaction operation are freezed until unlocked (unless current lock token is provided for the operation)

Returns specified lock token or new lock token if no token provided

finac.core.**account_statement**(*account*, *start=None*, *end=None*, *tag=None*, *pending=True*, *datefmt=False*, *_time_ms=False*)

Parameters

- **account** – account code
- **start** – statement start date/time
- **end** – statement end date/time
- **tag** – filter transactions by tag
- **pending** – include pending transactions
- **datefmt** – format date according to configuration

Returns generator object

finac.core.**account_statement_summary**(*account*, *start=None*, *end=None*, *tag=None*, *pending=True*, *datefmt=False*)

Parameters

- **account** – account code
- **start** – statement start date/time
- **end** – statement end date/time
- **tag** – filter transactions by tag

- **pending** – include pending transactions
- **datefmt** – format date according to configuration

Returns debit: debit turnover credit: credit turonver net: net debit statement: list of transactions

Return type dict with fields

finac.core.**account_unlock**(*account*, *token*)
Unlock account

Note that if you call account_lock, you must always unlock account, otherwise it will be locked until process restart

finac.core.**account_update**(*account*, ***kwargs*)
Update account parameters

Parameters, allowed to be updated: code, note, tp, max_balance, max_overdraft

finac.core.**archive_transactions**(*account=None*, *tp=None*, *due_date=None*, *keep_deleted=False*, *lock_token=None*, *_open_dbt=True*, *_db=None*)
Archive account transactions

Combines account transactions to due_date into a single service transaction. After archivation process is finished, cleanup() method should be called to remove archived transactions from the database.

Only completed transactions are archived.

WARNING: backing up database is always recommended before performing archiving procedure. If copy of archived transactions is required, it should be performed manually.

Parameters

- **account** – account to archive transactions on due_date: archivation date
- **tp** – or account type (types)
- **(default** – now) keep_deleted: keep deleted transactions (default: False)

finac.core.**asset_create**(*asset*, *precision=2*)
Create asset

Parameters

- **asset** – asset code (e.g. “CAD”, “AUD”)
- **precision** – precision (digits after comma) for statements and exchange operations. Default is 2 digits

finac.core.**asset_delete**(*asset*)
Delete asset

Warning: all accounts linked to this asset will be deleted as well

finac.core.**asset_delete_rate**(*asset_from*, *asset_to=None*, *date=None*)
Delete currrency rate

finac.core.**asset_list**()
List assets

finac.core.**asset_list_rates**(*asset=None*, *start=None*, *end=None*, *datefmt=False*, *_time_ms=False*)
List asset rates

Asset can be specified either as code, or as pair “code/code”

If asset is not specified, "end" is used as date to get rates for all assets

finac.core.**asset_precision**(*asset*)
Get precision (digits after comma) for the asset Note: asset precision is cached, so process restart required if changed

finac.core.**asset_rate_range**(*start*, *asset_from=None*, *asset_to=None*, *end=None*, *step=1*, *asset=None*, *return_timestamp=False*, *_time_ms=False*)
Get list of asset rates for the specified time range

step argument usage:

4, 4d - 4 days 2h - 2 hours 5a - split time range into 5 parts

Returns tuple with time series list and corresponding asset rate

finac.core.**asset_set_rate**(*asset_from*, *asset_to=None*, *value=None*, *date=None*)
Set asset rate

Parameters

- **asset_from** – asset from code
- **asset_to** – asset to code
- **value** – exchange rate value
- **date** – date/time exchange rate is set on (default: now)

Function can be also called as e.g. asset_set_rate('EUR/USD', value=1.1)

finac.core.**asset_update**(*asset*, ***kwargs*)
Update asset parameters

Parameters, allowed to be updated: code, precision

Note that asset precision is cached and requires process restart if changed

finac.core.**cleanup**()
Cleanup database

finac.core.**config_set**(*prop*, *value*)
Set configuration property on-the-fly (config.insecure = True required)

Recommended to use for testing only

Parameters

- **prop** – property name
- **value** – property value

finac.core.**exec_query**(*q*, *_time_ms=False*)
Execute FinacQL query statement

Parameters **q** – query to execute

Returns List of dicts is always returned

Raises

- RuntimeError – unsupported statement / function called
- other – passed from called function as-is

finac.core.**format_amount**(*i*, *asset*, *passive=False*)
Format amount for values and exchange operations. Default: apply asset precision

finac.core.**init**(*db=None*, ***kwargs*)
Initialize finac database and configuration

Parameters

- **db** – SQLAlchemy DB URI or sqlite file name
- **db_pool_size** – DB pool size (default: 10)
- **thread_pool_size** – thread pool size for internal processes (default: 30)
- **keep_integrity** – finac should keep database integrity (lock accounts, watch overdrafts, overlimits etc. Default is True
- **lazy_exchange** – allow direct exchange operations betwen accounts. Default: True
- **rate_allow_reverse** – allow reverse rates for lazy exchange (e.g. if "EUR/USD" pair exists but no USD/EUR, use 1 / "EUR/USD"
- **rate_allow_cross** – if exchange rate is not found, allow finac to look for the nearest cross-asset rate
- **rate_cache_size** – set rate cache size (default: 1024)
- **rate_cache_ttl** – set rate cache ttl (default: 5 sec)
- **full_transaction_update** – allow updating transaction date and amount
- **base_asset** – default base asset. Default is "USD"
- **date_format** – default date format in statements
- **multiplier** – use data multiplier
- **restrict_deletion** – 1 - forbid purge, 2 - forbid delete functions
- **redis_host** – Redis host
- **redis_port** – Redis port (default: 6379)
- **redis_db** – Redis database (default: 0)
- **redis_timeout** – Redis server timeout
- **redis_blocking_timeout** – Redis lock acquisition timeout
- **custom_account_types** – custom account types dict

Note: if Redis server is specified, Finac will use it for integrity locking (if enabled). In this case, lock tokens become Redis lock objects.

finac.core.**preload**()
Preload static data

finac.core.**purge**()
Purge deleted resources

finac.core.**transaction_apply**(*fname*)
Apply transaction yaml file

File format example:

transactions:

- account: acc1 amount: 500 tag: test
- dt: acc2 ct: acc1 amount: 200 tag: moving

If "account" is specified, function transaction_create is called, otherwise transaction_move. All arguments are passed to the functions as-is

Returns list of transaction IDs

finac.core.**transaction_complete**(*transaction_ids*, *completion_date=None*, *lock_token=None*)

Parameters

- **transaction_ids** – single or list/tuple of transaction ID
- **completion_date** – completion date (default: now)

finac.core.**transaction_copy**(*transaction_ids*, *date=None*, *completion_date=None*, *mark_completed=None*, *amount=None*)

Copy transaction :param transaction_ids: one or list/tuple of transaction id :param date: transaction date :param completion_date: transaction completion date :param mark_completed: if no completion_date is specified, set completion

date equal to creation. Default is True

Parameters **amount** – new amount, if old transaction has chain_transact_id will be exception

Returns list with id/ids new transaction

finac.core.**transaction_create**(*account*, *amount=None*, *tag=None*, *note=None*, *date=None*, *completion_date=None*, *mark_completed=True*, *target=None*, *lock_token=None*)

Create new simple transaction on account

Parameters

- **account** – account code
- **amount** – tranasction amount (>0 for debit, <0 for credit)
- **tag** – transaction tag
- **note** – transaction note
- **date** – transaction date
- **completion_date** – transaction completion date
- **mark_completed** – if no completion_date is specified, set completion date equal to creation. Default is True
- **target** – if no amount but target is specified, calculate transaction amount to make final balance equal to target

finac.core.**transaction_delete**(*transaction_ids*)

Delete (mark deleted) transaction

finac.core.**transaction_info**(*transaction_id*)

Get dict with transaction info

finac.core.**transaction_move**(*dt=None*, *ct=None*, *amount=0*, *tag=None*, *note=None*, *date=None*, *completion_date=None*, *mark_completed=True*, *target_ct=None*, *target_dt=None*, *rate=None*, *xdt=True*, *credit_lock_token=None*, *debit_lock_token=None*)

Create new standard (double-entry bookkeeping) transaction

Parameters

- **ct** – source (credit) account code

- **dt** – target (debit) account code
- **amount** – transaction amount (always >0)
- **tag** – transaction tag
- **note** – transaction note
- **date** – transaction creation date (default: now)
- **completion_date** – transaction completion date (default: now)
- **mark_completed** – mark transaction completed (set completion date)
- **target_ct** – target credit account balance
- **target_dt** – target debit account balance
- **rate** – exchange rate (lazy exchange should be on)
- **xdt** – for lazy exchange: True (default): amount is debited and calculate rate for credit False: amount is credited and calculate rate for debit

Returns tuple of two transactions

Return type transaction id, if lazy exchange performed

finac.core.**transaction_purge**(*_lock=True*)

Purge deleted transactions

finac.core.**transaction_update**(*transaction_id*, ***kwargs*)

Update transaction parameters

Parameters, allowed to be updated: tag, note

CHAPTER 6

Account types

Finac account types are hard-coded and can not be changed. Type codes are reserved for the future.

6.1 Financial assets

Finac considers accounts with types “credit”, “cash”, “current” and “saving” are primary financial assets and includes them in various listings by default.

- **credit** credit accounts
- **cash** cash (cash desk) accounts
- **current** current bank accounts
- **saving** saving accounts and deposits

6.2 Special accounts

- **transit** transit accounts
- **escrow** escrow accounts
- **holding** holding accounts
- **virtual** virtual accounts
- **temp** temporary accounts
- **exchange** virtual exchange accounts (not used by lazy exchange operations)

6.3 Customers, counterparties and company assets

Finac considers accounts with the below types are primary financial assets and includes them in various listings by default.

- **gs** goods and services
- **supplier** supplier accounts, *passive by default*
- **customer** customer accounts
- **finagent** financial agent accounts, *passive by default*

6.4 Investment accounts

- **stock** stocks
- **bond** bonds
- **fund** mutual and other funds
- **metal** precious metals
- **reality** real estate objects

6.5 Taxes

All tax accounts have the same type: **tax**. Taxes are included in listings by default, all tax accounts are passive (unless changed).

6.6 Custom accounts

ID ranges 800-899 (included in account lists by default) and 1800-1899 are reserved for custom account types.

> **Warning:** Custom account type codes should be constant.

To use custom account types, call *finac.init* with *custom_account_types option*:

```
CUSTOM_ACCOUNT_TYPES = [
    {
        'name': 'mytype1',
        'code': 800
    },
    {
        'name': 'mytype2',
        'code': 801
        'passive': True
    }
]

import finac as f
```

(continues on next page)

(continued from previous page)

```
f.init( # general options,
        custom_account_types=CUSTOM_ACCOUNT_TYPES)
```

CHAPTER 7

Finac Server and HTTP API

You may run Finac as the server with HTTP API. Running in server mode is highly recommended if you work with remote database.

7.1 Finac server

To run Finac server, create a WSGI app, in example put the following code inside file, named **server.py**:

```
import finac as f
import finac.api as api

f.init('mysql+pymysql://someuser:somepassword@dbhost/dbname')
# optional API key
api.key = 'secret'
application = api.app

if __name__ == '__main__':
    application.run(host='0.0.0.0', debug=True)
```

The code above can be launched directly for the debugging purposes, for production it's recommended to use WSGI server, e.g. **gunicorn**:

```
gunicorn -b 0.0.0.0:80 server:application
```

7.2 Finac client

To use Finac library as client for Finac server, provide API params for **init**() function:

```
import finac as f

```

(continues on next page)

(continued from previous page)

```
f.init(api_uri='http://finac-host:80/jrpc',
   # optional API key
   api_key='secret',
   # API timeout in seconds, default: 5
   api_timeout=10,
   # cache rates for 600 seconds
   rate_ttl=600)
# preload static data to avoid unnecessary future requests
f.preload()
```

7.3 Calling API functions directly

Finac uses JSON RPC 2.0 protocol. Default API URI is

    http(s)://host:port/jrpc

You may call any *core function*, single or batch.

Error codes:

- **-32699** Internal error
- **-32601** Method not found
- **-32602** Invalid method param
- **-32603** Invalid value provided
- **-32000** Access denied (if API key is set)
- **-32001** ResourceNotFound exception
- **-32002** RateNotFound exception
- **-32003** OverdraftError exception
- **-32004** OverlimitError exception
- **-32005** ResourceAlreadyExists exception

CHAPTER 8

Queries

8.1 Syntax

Finac has a simple query language to access core functions.

Currently only function call statements are supported:

```
SELECT <function>([args, kwargs])

/* e.g. */

SELECT account_balance("myaccount")
```

Supported core functions:

- get_version
- asset_list
- asset_list_rates
- asset_rate^
- asset_rate_range^
- account_info
- account_statement
- account_list
- account_balance^
- account_balance_range^

Functions marked with "^" support data column assignment with "AS":

```
SELECT account_balance("myaccount") AS myacc
```

8.2 Executing queries

8.2.1 Interactive

In the interactive mode, query can be executed as:

```
f.query('select account_list()')
```

The function outputs query result to stdout.

8.2.2 Embedded

If the application want to execute Finac query, it should call method

```
f.exec_query('select account_list()')
```

The function always returns list of dicts, where list items are result rows and dict keys are result columns.

8.2.3 API

Calling queries via *API* is possible either via JSON RPC, or via special URI **/query**.

The URI can be requested either via GET (with param q=<query>) or via POST (with list of queries in JSON payload).

Finac API key should be put into *X-Auth-Key* request header variable.

The response format is:

```
{
    'ok': true,
    'result': query_result, // (list of dicts)
    'rows': number_of_rows, // integer
    'time': time_spent_in_seconds // float
}
```

For GET, errors are returned as HTTP status:

- **400** bad request (e.g. invalid query format / params)
- **404** resource not found
- **409** resource already exists, over limit / overdraft error
- **500** all other errors

For POST, list of responses is returned. If certain query failed with an error, its response contains *error* field only.

CHAPTER 9

Finac - financial accounting for humans

Finac is a library and function set for Jupyter/ipython, which provides the double-entry bookkeeping database.

Finac is simple, open and free. It can work with SQLite or any database supported by SQLAlchemy (tested: SQLite, MySQL, PostgreSQL).

Finac can be used either in the interactive mode with Jupyter, Spyder-IDE, ipython or other similar environment or Finac library can be embedded into 3rd party projects. The library can be used in accounting applications and is useful for fin-tech services.

Finac supports multiple currencies, simple transactions, double-entry bookkeeping transactions, watches overdrafts, balance limits and has got many useful features, which make accounting simple and fun.

9.1 Install

```
pip3 install finac
```

Sources: https://github.com/alttch/finac

Documentation: https://finac.readthedocs.io/

9.2 Updating

CHAPTER 10

from 0.4.10

```
ALTER TABLE transact ADD service bool;
UPDATE transact SET service=true WHERE d_created<'1970-01-03';
ALTER TABLE transact ADD FOREIGN KEY(chain_transact_id)
  REFERENCES transact(id) ON DELETE SET null;
```

CHAPTER 11

from 0.3.x

Starting from 0.4, Finac uses DateTime columns for:

- asset_rate.d
- transact.d
- transact.d_created
- transact.deleted

Depending to the database type, it's REQUIRED to convert these columns to either DATETIME (SQLite, for MySQL DATETIME(6) recommended) or TIMESTAMPTZ (PostgreSQL, with timezone).

11.1 How to use in interactive mode

Finac database contains 3 entity types:

- **asset** currency, ISIN, stock code etc., currencies "USD" and "EUR" are created automatically. Finac does not separate assets into currencies, property and other. This allows creating applications for various areas using the single library.
- **account** bank account, counterparty account, tax account, special account etc. Everything is accounts :)
- **transaction** movements from (credit) / to (debit) and between accounts

Assets have got **rates** - the value of one asset, relative to other.

Transactions can be simple (no counterparty) or classic double-entry bookkeeping (between debit and credit account).

```
import finac as f
# init finac,
f.init('/tmp/test.db')
# create a couple of accounts
f.account_create('acc1', 'USD')
f.account_create('acc2', 'USD')
```

(continues on next page)

(continued from previous page)

```
f.account_create('depo', 'USD', 'saving')
# import initial balance with a simple transaction
f.tr('acc1', 10000, tag='import')
# move some assets to other accounts
f.mv(dt='acc2', ct='acc1', amount=2000)
f.mv(dt='depo', ct='acc1', amount=3000)
```

```
# display statement for acc1
f.ls('acc1')
```

```
id     amount  cparty  tag     note  created              completed
-------------------------------------------------------------------------------
7   10 000.00          import        2019-10-26 03:04:02  2019-10-26 03:04:02
8   -2 000.00  ACC2                  2019-10-26 03:04:02  2019-10-26 03:04:02
9   -3 000.00  DEPO                  2019-10-26 03:04:02  2019-10-26 03:04:02
-------------------------------------------------------------------------------
Debit turnover: 10 000.00, credit turnover: 5 000.00

Net profit/loss: 5 000.00 USD
```

```
# display summary for all accounts
f.ls()
```

```
account  type      asset     balance  balance USD
-------------------------------------------------
ACC1     current    USD     5 000.00     5 000.00
ACC2     current    USD     2 000.00     2 000.00
DEPO     saving     USD     3 000.00     3 000.00
-------------------------------------------------
Total: 10 000.00 USD
```

```
# display summary only for current accounts
f.ls(tp='current')
```

```
account  type     asset     balance   balance USD
-------------------------------------------------
ACC1     current    USD     5 000.00     5 000.00
ACC2     current    USD     2 000.00     2 000.00
-------------------------------------------------
Total: 7 000.00 USD
```

```
# display assets pie chart, (wrapper for matplotlib.pyplot, requires Jupyter,
# Spyder-IDE or a similar interactive environment)
f.pie()
```

Note: when addressing currencies and accounts both in interactive and API mode, account and asset codes should be used as object identifiers. **All codes are case-insensitive**.

Inside database Finac uses numeric IDs to connect objects, so the codes can be changed without any problems.

11.2 Special features

11.2.1 Lazy exchange

Finac can automatically move assets between accounts having different currencies if exchange rate is set or specified in the transaction details:

```
# create EUR account
f.account_create('acc5', 'eur')
# set exchange rate (in real life you would probably use cron job)
f.asset_set_rate('eur/usd', value=1.1)
f.mv(dt='acc5', ct='acc1', amount=100)
```

hoorah, account acc5 have got 100 EUR! And exchange rate was 1.1. Check it:

```
>>> f.ls('acc1')
```

```
id     amount  cparty  tag     note  created              completed
-------------------------------------------------------------------------------
..............
..............
14    -110.00                       2019-10-26 03:15:41  2019-10-26 03:15:41
-------------------------------------------------------------------------------
```

```
>>> f.ls('acc5')
```

```
id  amount  cparty  tag  note  created              completed
-------------------------------------------------------------------------
15  100.00                    2019-10-26 03:15:41  2019-10-26 03:15:41
-------------------------------------------------------------------------
Debit turnover: 100.00, credit turnover: 0.00

Net profit/loss: 100.00 EUR
```

As shown, there is no a counterparty account in the lazy exchange. This feature is useful for personal accounting and special applications, but for professional accounting, create counterparty exchange accounts should be created and buy-sell transactions should be performed between them.

11.2.2 Targets

Targets is a feature I have created Finac for. Consider there are account balances in a bank and in the accounting. They differ in some amount and this need to be recorded in the accounting with a single transaction.

But the problem is: there is a lot of transactions which should be sum up. Or the difference between bank balance and accounting must be calculated manually. Pretty common, eh? Don't do this, Finac has got targets.

Specifying targets instead of amount asks Finac to calculate transaction amount by itself.

After the previous operation, there is *4,890.00* USD on "acc1" and consider all except $1000 should be moved to "acc2". Let us do it:

```
>>> f.mv(dt='acc2', ct='acc1', target_ct=1000)
```

```
id     amount  cparty  tag     note  created              completed
-------------------------------------------------------------------------------
......
......
16  -3 890.00  ACC2                  2019-10-26 03:25:56  2019-10-26 03:25:56
-------------------------------------------------------------------------------
Debit turnover: 10 000.00, credit turnover: 9 000.00

Net profit/loss: 1 000.00 USD
```

The transaction amount is automatically calculated. Lazy people are happy :)

If the debit account balance target should be specified, *target_dt* function argument can be used. Note: calculated transaction amount must be always greater than zero (if credit account target higher than its current balance is specified, *ValueError* is raised)

For simple transactions (*f.tr(...)*), use *target=*.

11.2.3 Transaction templates

Example: there is a repeating payment orders in a bank, which pay office utility bills every 5th day of month, plus automatically move $100 to a saving account. To fill this into accounting, YAML transaction template can be used:

```
transactions:
  - account: acc1
    amount: 200
    tag: electricity
    note: energy company deposit
  - account: acc1
    amount: 800
    tag: rent
    note: office rent
  - dt: depo
    ct: acc1
    amount: 200
    tag: savings
    note: rainy day savings
```

then create a cron job which calls *f.transaction_apply("/path/to/file.yml")* and that is it.

Actually, transaction templates are useful for any repeating operations. The same arguments, as for the core functions, can be specified.

11.2.4 Number formatting

Finac does not use system locale. If amounts and targets are inputted as strings, they can be specified in any format and Finac tries converting strings into float numeric automatically. The following values for amounts and targets are valid and are automatically parsed:

- 1 000,00 = 1000.0
- 1,000.00 = 1000.0
- 1.000,00 = 1000.0
- 1,000.00 = 1000.0
- 10,0 = 10.0

- 10.0 = 10.0

11.2.5 Passive accounts

If account is passive, its assets are decremented from totals. To create passive account, *passive* argument must be used:

```
f.account_create('passive1', 'usd', passive=True)
```

Accounts of types “tax”, “supplier” and “finagent” are passive by default.

11.2.6 Data multiplier

Depending on data, it may be useful to store numeric values in the database as integers instead of floats. Finac library has got a built-in data multiplier feature. To enable it, set *multiplier=N* in *finac.init()* method, e.g. *multiplier=1000*. This makes Finac to store integers into tables and use the max precision of 3 digits after comma.

Note: table fields must be manually converted to numeric/integer types. In production databases the field values must be also manually multiplied.

Full list of tables and fields, required to be converted, is available in the dict *finac.core.multiply_fields*.

Note: the multiplier can be used only with integer and numeric(X) field types, as core conversion functions always return rounded value.

11.3 How to embed Finac library into own project

See Finac documentation for core function API details.

11.4 Client-server mode and HTTP API

See Finac documentation for server mode and HTTP API details.

11.5 Enterprise server and support

Want to integrate Finac into an own enterprise app or service? Need a support? Check Finac Enterprise Server.

Python Module Index

f

Index