ccGains Documentation

Release 1.0.0 beta

Jürgen Probst

Jan 22, 2019

Contents:

The ccGains (cryptocurrency gains) package provides a python library for calculating capital gains made by trading cryptocurrencies or foreign currencies.

Some of its features are:

- calculates the capital gains using the first-in/first out (FIFO) principle,
- creates capital gains reports as CSV, HTML or PDF (instantly ready to print out for the tax office),
- can create a more detailed capital gains report outlining the calculation and used bags,
- differs between short and long term gains (amounts held for less or more than a year),
- treats amounts held and traded on different exchanges separately,
- treats exchange fees and transaction fees directly resulting from trading properly as losses,
- provides methods to import your trading history from various exchanges,
- loads historic cryptocurrency prices from CSV files and/or
- loads historic prices from APIs provided by exchanges,
- caches historic price data on disk for quicker and offline retrieval and less traffic to exchanges,
- for highest accuracy, uses the decimal data type for all amounts
- supports saving and loading the state of your portfolio as JSON file for use in ccGains calculations in following years

CHAPTER 1

ccgains package

1.1 Module contents

1.2 ccgains.bags module

class ccgains.bags.**Bag**(*id*, *dtime*, *currency*, *amount*, *cost_currency*, *cost*, *price=None*)

Create a bag which holds an *amount* of *currency*.

Parameters

- **(integer)** (*id*) – A unique number for each bag. Usually the first created bag receives an id of 1, which increases for every bag created.
- **dtime** – The datetime when the currency was purchased.
- **currency** – The currency this bag holds, the currency that was bought.
- **amount** – The amount of currency that was bought. This is the amount that is available, i.e. fees are already substracted.
- **cost_currency** – The base currency which was paid for the money in this bag. The base value of this bag is recorded in this currency.
- **cost** – The amount of *cost_currency* paid for the money in this bag. This covers all expenses, so fees are included.
- **price** – (optional, default: None): If *price* is given, *cost* will be ignored and calculated from *price*: *cost* = *amount* * *price*.

is_empty()

spend(*amount*)

Spend some amount out of this bag. This updates the current amount and the base value, but leaves the price constant.

Returns

the tuple (spent_amount, bcost, remainder), where

- *spent_amount* is the amount taken out of the bag, in units of *self.currency*;
- *bcost* is the base cost of the spent amount, in units of *self.cost_currency*;
- *remainder* is the leftover of *amount* after the spent amount is substracted.

class ccgains.bags.**BagQueue**(*base_currency*, *relation*, *mode='FIFO'*, *json_dump='./precrash.json'*)

Create a BagQueue object.

This is the class that processes trades, handles all bags (i.e. creating and spending them) and calculates the capital gains. After calculation, the member BagQueue.report provides methods to create reports.

Parameters

- **base_currency** – The base currency (string, e.g. "EUR"). All bags' base cost (the money spent for them at buying time) will be recorded in units of this currency and finally the capital gains will be calculated for this currency.
- **relation** – A CurrencyRelation object which serves exchange rates between all currencies involved in trades which will later be added to this BagQueue. If solely trades involving base_currency will be processed, a CurrencyRelation object is not necessary and can be *None*. In this case, if a trade between non-base currencies is encountered, an exception will be raised.
- **mode** – (string) Inventory accounting method to use. The following methods are supported: - "FIFO": First In First Out - "LIFO": Last In First Out - "LPFO": Lowest Price First Out
- **json_dump** – (filename) If specified, the state of the BagQueue will be saved as JSON formatted file with this file name just before an error is raised due to missing or conflicting data. If the error is fixed, the state can be loaded from this file and the calculation might be able to continue from that point.

buy_with_base_currency(*dtime*, *amount*, *currency*, *cost*, *exchange*)

Create a new bag with *amount* money in *currency*.

Creation time of the bag is the datetime *dtime*. The *cost* is paid in base currency, so no money is taken out of another bag. Any fees for the transaction should already have been substracted from *amount*, but included in *cost*.

deposit(*dtime*, *currency*, *amount*, *fee*, *exchange*)

Deposit *amount* monetary units of *currency* into an exchange for a *fee* (also given in *currency*), making it available for trading. The fee is included in amount. The deposit happened at datetime *dtime*.

The pair of methods *withdraw* and *deposit* is used for transfers of the same currency from one exhange to another.

If the amount is more than the amount withdrawn before (minus fees), a warning will be printed and a bag created with a base cost of 0.

See also *withdraw* about the handling of fees.

Note that, currently, the fees for this deposit, if any, will be taken from the oldest funds on the exchange after the deposit, which are not necessarily the deposited funds.

load(*filepath_or_buffer*)

Restore a previously saved state of a BagQueue and its list of bags from a JSON formatted file.

Everything from the current BagQueue object will be overwritten with the file's contents.

Parameters **filepath_or_buffer** – The filename of the JSON formatted file or a general buffer with a *read()* method streaming the JSON formatted string.

pay(*dtime*, *currency*, *amount*, *exchange*, *fee_ratio=0*, *custom_rate=None*, *report_info=None*)
Spend an amount of funds.

The money is taken out of the bag on the exchange with the proper currency according to self.mode first, then from the next fitting bags in line each time a bag is emptied. The bags' prices are not changed, but their current amount and base value (original cost) are decreased proportionally.

This transaction and any profits or losses made will be logged and added to the capital gains report data.

If *amount* is higher than the available total amount, ValueError is raised.

Parameters

- **dtime** – (datetime) The date and time when the payment occured.
- **amount** – (number, decimal or parsable string) The amount being spent, including fees.
- **currency** – (string) The currency being spent.
- **exchange** – (string) The unique name of the exchange/wallet where the funds that are being spent are taken from.
- **fee_ratio** – (number, decimal or parsable string), 0 <= fee_ratio <= 1; The ratio of *amount* that are fees. Default: 0.
- **custom_rate** – (number, decimal or parsable string), Default: None; Provide a custom rate for conversion of *currency* to the base currency. Usually (i.e. if this is None), the rate is fetched from the CurrencyRelation object provided when this BagQueue object was created. In some cases, one should rather provide a rate, for example when base currency was bought with this payment, meaning a more specific rate for this trade can be provided than relying on the averaged historic data used otherwise.
- **report_info** – dict, default None; Additional information that will be added to the capital gains report data. Currently, the only keys looked for are: 'kind', 'buy_currency' and 'buy_ratio'. For each one of them omitted in the dict, these default values will be used:

  {'kind': 'payment', 'buy_currency': '', 'buy_ratio': 0},

  This is also the default dict used when *report_info* is *None*.

  – 'kind' is the type of transaction, i.e. 'sale', 'withdrawal fee', 'payment' etc.;

  – 'buy_currency' is the currency bought in this trade;

  – 'buy_ratio' is the amount of 'buy_currency' bought with one unit of *currency*, i.e. *bought_amount / spent_amount*; only used if 'buy_currency' is not empty.

Returns

the tuple *(short_term_profit, total_proceeds)*, with each value given in units of the base currency, where:

- *short_term_profit*

  is the taxable short term profit (or loss if negative) made in this sale. This only takes into account the part of *amount* which was acquired less than a year prior to *dtime* (or whatever time period is used by *is_short_term*). The short term profit equals the proceeds made by liquidating this amount for its price at *dtime* minus its original cost, with the fees already substracted from these proceeds.

- *total_proceeds*

are the total proceeds returned from this sale, i.e. it includes the full *amount* (held for any amount of time) at its price at *dtime*, with fees already substracted. This value equals the base cost of any new currency purchased with this sale.

pick_bag(*exchange*, *currency*, *start_index=None*)
Pick bag from bag queue according to self.mode

Parameters

- **exchange** – (string) The unique name of the exchange/wallet where the funds that are being spent are taken from.
- **currency** – (string) The currency being spent.
- **start_index** – (int) List index to start from

process_trade(*trade*)
Process the trade or transaction documented in a Trade object.

The trade must be newer than the last processed trade, otherwise a ValueError is raised.

Pay attention to the definitions in Trade.__init__, especially that *buy_amount* is given without transaction fees, while *sell_amount* includes them.

save(*filepath_or_buffer*)
Save the current state of this BagQueue and its list of bags to a JSON formatted file, so that it can later be restored with *self.load*.

As an external utility, self.relation will not be included in this string.

Parameters **filepath_or_buffer** – The destination file's name, which will be overwritten if existing, or a general buffer with a *write()* method.

sort_bags(*exchange*)
Sort bags according to self.mode

Parameters **exchange** – (string) The unique name of the exchange/wallet where the bags being sorted are.

to_data_frame()
Put all bags from all exchanges in one big pandas.DataFrame.

to_json(***kwargs*)
Return a JSON formatted string representation of the current state of this BagQueue and its list of bags.

As an external utility, self.relation will not be included in this string.

Parameters **kwargs** – Keyword arguments that will be forwarded to *json.dumps*.

Returns JSON formatted string

withdraw(*dtime*, *currency*, *amount*, *fee*, *exchange*)
Withdraw *amount* monetary units of *currency* from an exchange for a *fee* (also given in *currency*). The fee is included in amount. The withdrawal happened at datetime *dtime*.

The pair of methods *withdraw* and *deposit* is used for transfers of the same currency from one exhange to another.

If the amount is more than the total available, a ValueError will be raised.

—

Losses made by fees (which must be directly resulting from and connected to short term trading activity!) are substracted from the total taxable profit (recorded in base currency).

This approach can be logically justified by looking at what happens to the amount of fiat money that leaves a bank account solely for trading with cryptocurrencies, which in turn are sold entirely for fiat money before the end of the year. If nothing else was bought with the cryptocurrencies in between, the difference between the amount of fiat before and after trading is exactly the taxable profit. For simplicity, say we buy some Bitcoin at one exchange for X fiat money (i.e. X fiat money is leaving the bank account), then transfer it to another exchange (paying withdrawal and/or deposit fees) where we sell it again for fiat, e.g.:

- buy 1 BTC @ 1000 EUR at exchangeA; now we own 1 BTC with base value 1000 EUR
- transfer 1 BTC to exchangeB for 0.1 BTC fees; now we own 0.9 BTC with base value 900 EUR, 100 EUR for the fees are counted as loss
- Example 1: We sell 0.9 BTC at a better price than before: we get exactly 1000 EUR. The immediate profit is 100 EUR (1000 EUR proceeds minus 900 EUR base value), but minus the 100 EUR fee loss from earlier we have exactly a taxable profit of 0 EUR, which makes sense considering we started with 1000 EUR and now still have only 1000 EUR.
- Example 2: We sell 0.9 BTC at a much better price: we get exactly 2000 EUR. The immediate profit is 1100 EUR (2000 EUR proceeds minus 900 EUR base value), but minus the 100 EUR fee loss from earlier we have exactly a taxable profit of 1000 EUR, which also makes sense since we started with 1000 EUR and now have 2000 EUR.

Note: The exact way how withdrawal, deposit and in general transaction fees are handled should be made user-configurable in future.

exception ccgains.bags.**CurrencyTypeException**

ccgains.bags.**is_short_term**(*adate*, *tdate*)

Return whether a transaction/trade done on *tdate* employing currency acquired on *adate* is a short term activity, i.e. the profits and/or losses made with it are taxable.

Currently, this simply returns whether the difference between the two dates is less than one year, as is the rule in some countries, e.g. Germany and the U.S.A.

TODO: This needs to be made user-configurable in future, to adapt to laws in different countries.

1.3 ccgains.historic_data module

class ccgains.historic_data.**HistoricData**(*unit*)

Bases: object

Create a HistoricData object with no data. The unit must be a string given in the form 'currency_one/currency_two', e.g. 'EUR/BTC'.

Only use this constructor if you want to manually set the data, otherwise use one of the subclasses *HistoricDataCSV* or *HistoricDataAPI*.

To manually set data, self.data must be a pandas time series with a fixed frequency.

get_price(*dtime*)

Return the price at datetime *dtime*

prepare_request(*dtime*)

Return a Pandas DataFrame which contains the data at the requested datetime *dtime*.

class ccgains.historic_data.**HistoricDataAPI**(*cache_folder*, *unit*, *interval='H'*)

Bases: *ccgains.historic_data.HistoricData*

Initialize a HistoricData object which tranparently fetch data on request (*get_price*) from the public Poloniex API: https://poloniex.com/public?command=returnTradeHistory

For faster loading times on future calls, a HDF5 file is created from the requested data and used transparently the next time a request for the same day and pair is made. These HDF5 files are saved in *cache_folder*.

The *unit* must be a string given in the form 'currency_one/currency_two', e.g. 'EUR/BTC'.

The data will be resampled by calculating the weighted price for interval steps specified by *interval*. See: http://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases for possible values.

prepare_request(*dtime*)
Return a Pandas DataFrame which contains the data for the requested datetime *dtime*.

class ccgains.historic_data.**HistoricDataAPIBinance**(*cache_folder*, *unit*, *interval='H'*)
Bases: *ccgains.historic_data.HistoricData*

Initialize a HistoricData object which will transparently fetch data on request (*get_price*) from the public Binance API: https://api.binance.com/api/v1/klines

For faster loading times on future calls, a HDF5 file is created from the requested data and used transparently the next time a request for the same day and pair is made. These HDF5 files are saved in *cache_folder*.

The *unit* must be a string in the form 'currency_one/currency_two', e.g. 'NEO/BTC'.

The data will be resampled by calculating the weighted price for interval steps specified by *interval*. See: http://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases for possible values.

prepare_request(*dtime*)
Return a pandas DataFrame which contains the data for the requested datetime *dtime*.

class ccgains.historic_data.**HistoricDataAPICoinbase**(*cache_folder*, *unit*, *interval='H'*)
Bases: *ccgains.historic_data.HistoricData*

Initialize a HistoricData object which transparently fetches data on request (*get_price*) from the public Coinbase API: 'https://api.pro.coinbase.com/products/:SYMBOL:/candles'

For faster loading times on future calls, a HDF5 file is created from the requested data and used transparently the next time a request for the same day and pair is made. These HDF5 files are saved in *cache_folder*.

The *unit* must be a string given in the form 'currency_one/currency_two', e.g. 'BTC/USD'.

The data will be resampled by calculating the weighted price for interval steps specified by *interval*. See: http://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases for possible values.

prepare_request(*dtime*)
Return a pandas DataFrame which contains the data for the requested datetime *dtime*.

class ccgains.historic_data.**HistoricDataCSV**(*file_name*, *unit*, *interval='H'*)
Bases: *ccgains.historic_data.HistoricData*

Initialize a HistoricData object with data loaded from a csv file. The unit must be a string given in the form 'currency_one/currency_two', e.g. 'EUR/BTC'. The csv must consist of three columns: first a unix timestamp, second the rate given in *unit*, third the amount traded. (Such a csv can be downloaded from bitcoincharts.com)

The file may also be compressed and will be deflated on-the-fly; allowed extensions are: '.gz', '.bz2', '.zip' or '.xz'.

The data will be resampled by calculating the weighted price for interval steps specified by *interval*. See: http://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases for possible values.

For faster loading times, a HDF5 file is created from the csv the first time it is loaded and used transparently the next time an HistoricData object is created with the same csv. If the csv file is newer than the HDF5 file, the latter will be updated.

ccgains.historic_data.**resample_weighted_average**(*df*, *freq*, *data_col*, *weight_col*, *include_weights=False*)

Resample a DataFrame with a DatetimeIndex. Return weighted averages of groups.

Parameters

- **df** – The pandas.DataFrame to be resampled
- **freq** – The new frequency of the resampled time series
- **data_col** – Column to take the average of
- **weight_col** – Column with the weights
- **include_weights** – (default False) If True, include the summed weights in result

Returns

if include_weights:

pandas.DataFrame with two columns:

- data_col: the weighted averages,
- weight_col: the summed weights

else:

pandas.Series with weighted averages

Source: ErnestScribbler, https://stackoverflow.com/a/44683506

1.4 ccgains.relations module

class ccgains.relations.**CurrencyPair**(*base*, *quote*)

Create new instance of CurrencyPair(base, quote)

reversed()

Swap base(from) for quote(to) currencies

class ccgains.relations.**CurrencyRelation**(**args*)

Create a CurrencyRelation object. This object allows exchanging values between currencies, using historical exchange rates at specific times.

Parameters **args** – Any number of HistoricData objects. If multiple HistoricData objects are given with the same unit, only the last one will be used. More/updated HistoricData objects can be supplied later with *add_historic_data* method.

add_historic_data(*hist_data*)

Add an HistoricData object. If a HistoricData object with the same unit has already been added, it will be updated.

get_rate(*dtime*, *from_currency*, *to_currency*)

Return the rate for conversion of *from_currency* to *to_currency* at the datetime *dtime*.

If a direct relation of the currency pair has not been added with *add_historic_data* before, an indirect route using multiple added pairs is tried. If this also fails, a KeyError is raised.

update_available_pairs(*update_pair=None*)

Update internal list of pairs with available historical rate.

Parameters **update_pair** – tuple (from_currency, to_currency); If supplied, updates existing recipes with pair supplied. Default (None) will force rebuild of pairs list from scratch based on supplied historical data sets.

class ccgains.relations.**Recipe**

A conversion recipe made up of *num_steps* steps which are given in *recipe_steps*, showing how to convert from *recipe_steps[0].base* to *recipe_steps[-1].quote*

Create new instance of Recipe(num_steps, recipe_steps)

reversed()

A reversed recipe has RecipeSteps in the reversed order, and the opposite value for *reciprocal* for each step

class ccgains.relations.**RecipeStep**

One step in a conversion recipe, indicating base (from) currency, quote (to) currency, and whether the reciprocal of the price should be used for this step.

Create new instance of RecipeStep(base, quote, reciprocal)

as_recipe()

Create a Recipe (with one step) from this RecipeStep

reversed()

Return a copy of this recipe step with reciprocal set opposite

1.5 ccgains.trades module

class ccgains.trades.**Trade**(*kind*, *dtime*, *buy_currency*, *buy_amount*, *sell_currency*, *sell_amount*, *fee_currency=''*, *fee_amount=0*, *exchange=''*, *mark=''*, *comment=''*, *default_timezone=None*)

This class holds details about a single transaction, like a trade between two currencies or a withdrawal of a single currency.

Create a Trade object.

All parameters may be strings, the numerical values will be converted to decimal.Decimal values, *dtime* to a datetime.

Parameters

- **kind** – a string denoting the kind of transaction, which may be e.g. "trade", "withdrawal", "deposit". Not currently used, so it can be any comment.
- **dtime** – a string, number or datetime object: The date and time of the transaction. A string will be parsed with pandas.Timestamp; a number will be interpreted as the elapsed seconds since the epoch (unix timestamp), in UTC timezone.
- **buy_amount** – The amount of *buy_currency* bought. This value excludes any transaction fees, i.e. it is the amount that is fully available after the transaction.
- **sell_amount** – The amount of *sell_currency* sold. This value includes fees that may have been paid for the transaction, i.e. it is the total amount that left the account for the transaction.

buy_amount and *sell_amount* may be given in any order if exactly one of the two values is negative, which will then be identified as the sell amount. In that case, *buy_currency* and *sell_currency* will be swapped accordingly, so the currency will always stay with the amount. It's an error if both values are negative.

Parameters

- **fee_amount** – The fees paid, given in *fee_currency*. May have any sign, the absolute value will be taken as fee amount regardless.
- **default_timezone** – This parameter is ignored if there is timezone data included in *dtime*, or if *dtime* is a number (unix timestamp), in which case the timezone will always be UTC. Otherwise, if default_timezone=None (default), the time data in *dtime* will be interpreted as time in the local timezone according to the locale setting; or it must be a tzinfo subclass (from dateutil.tz or pytz), which will be added to *dtime*.

to_csv_line(*delimiter=', ', endl='\n'*)

class ccgains.trades.**TradeHistory**

The TradeHistory class is a container for a sorted list of *Trade* objects, but most importantly it provides methods for importing transactions exported from various exchanges, programs and web applications.

TradeHistory() creates a TradeHistory object.

self.tlist is a sorted list of trades available after some trades have been imported.

add_missing_transaction_fees(*raise_on_error=True*)

Some exchanges do not include withdrawal fees in their exported csv files. This will try to add these missing fees by comparing withdrawn amounts with amounts deposited on other exchanges shortly after withdrawal. Call this only after all transactions from every involved exchange and wallet were imported.

This uses a really simple algorithm, so it is not guaranteed to work in every case. Basically, it finds the first deposit following each withdrawal and compares the withdrawn amount with the deposited amount. The difference (withdrawn - deposited) is then assigned as the fee for the withdrawal, if this fee is greater than zero. This might not work if there are withdrawals in tight succession whose deposits register in a different order than the withdrawals.

If *raise_on_error* is True (which is the default), a ValueError will be raised if a pair is found that cannot possibly match (higher deposit than withdrawal), otherwise only a warning is logged and the withdrawal skipped (which will be tried to be matched with the next deposit) while the deposit is tried to be matched with another withdrawal that came before it.

append_binance_csv(*file_name*, *which_data='trades'*, *delimiter=', '*, *skiprows=1*, *default_timezone=<Mock name='mock.tz.tzutc()' id='140455774214408'>*)

Import trades or transfers from a csv file from Binance and add them to this TradeHistory.

Afterwards, all trades will be sorted by date and time.

Parameters

- **which_data** – (string) Must be one of *"trades"*, *"withdrawals"*, *"deposits"*, or *"distributions"*. Binance separates generated CSV histories into these four categories; specify which is being imported here.
- **default_timezone** – This parameter is ignored if there is timezone data in the csv string; by default Binance does not. Otherwise, if None, the time data in the csv will be interpreted as the time in the local timezone according to the locale setting; or it must be a tzinfo subclass (from dateutil.tz or pytz); The default is UTC time, which is what Binance exports at the time of writing, but it may change in the future

append_bisq_csv(*trade_file_name*, *transactions_file_name*, *delimiter=', '*, *skiprows=1*, *default_timezone=None*)

Import trades from the csv files exported from Bisq (former Bitsquare) and add them to this TradeHistory.

Afterwards, all trades will be sorted by date and time.

From the Bisq program, two kinds of csv files can be exported: One with the trading history and one with the transaction history. Because of how Bisq works, these two histories are intertwined and in order to properly connect the fees to trades, both files must be imported together.

Parameters

- **trade_file_name** – The csv file name with the trading history. In case you only made transactions and no trades, this may be an empty string: “”
- **transaction_file_name** – The csv file name with the transaction history
- **default_timezone** – This parameter is ignored if there is timezone data in the csv string. Otherwise, if None, the time data in the csv will be interpreted as time in the local timezone according to the locale setting; or it must be a tzinfo subclass (from dateutil.tz or pytz); The default is None, i.e. the local timezone, which is what Bitsquare exports at time of writing this, but it might change in future.

append_bitcoin_de_csv(*file_name*, *delimiter=';'*, *skiprows=1*, *default_timezone=None*)

Import trades from a csv file exported from Bitcoin.de and add them to this TradeHistory.

Afterwards, all trades will be sorted by date and time.

Parameters **default_timezone** – This parameter is ignored if there is timezone data in the csv string. Otherwise, if None, the time data in the csv will be interpreted as time in the local timezone according to the locale setting; or it must be a tzinfo subclass (from dateutil.tz or pytz); The default is None, i.e. the local timezone, which is what Bitcoin.de exports at time of writing, but it might change in future.

append_bitsquare_csv(*trade_file_name*, *transactions_file_name*, *delimiter=','*, *skiprows=1*, *default_timezone=None*)

Import trades from the csv files exported from Bisq (former Bitsquare) and add them to this TradeHistory.

Afterwards, all trades will be sorted by date and time.

From the Bisq program, two kinds of csv files can be exported: One with the trading history and one with the transaction history. Because of how Bisq works, these two histories are intertwined and in order to properly connect the fees to trades, both files must be imported together.

Parameters

- **trade_file_name** – The csv file name with the trading history. In case you only made transactions and no trades, this may be an empty string: “”
- **transaction_file_name** – The csv file name with the transaction history
- **default_timezone** – This parameter is ignored if there is timezone data in the csv string. Otherwise, if None, the time data in the csv will be interpreted as time in the local timezone according to the locale setting; or it must be a tzinfo subclass (from dateutil.tz or pytz); The default is None, i.e. the local timezone, which is what Bitsquare exports at time of writing this, but it might change in future.

append_bittrex_csv(*file_name*, *which_data='trades'*, *skiprows=1*, *delimiter=','*, *default_timezone=None*)

Import trades from a csv file exported from Bittrex.com and add them to this TradeHistory.

Afterward, all trades will be sorted by date and time.

Parameters

- **which_data** – (string) Must be one of *“trades”* or *“transfers”*. Bittrex only exports trade history, but displays transfer history in a table that can be pasted into a csv file manually. This parser assumes the same column layout as is shown on the Bittrex transfer history page.”
- **default_timezone** – This parameter is ignored if there is timezone data in the csv string. Otherwise, if None, the time data in the csv will be interpreted as time in the local timezone according to the locale setting; or it must be a tzinfo subclass (from dateutil.tz

or pytz); The default is None, as Bittrex (at the time of writing) outputs local time (at the time of purchase) with transaction history

append_ccgains_csv(*file_name*, *delimiter=', '*, *skiprows=1*, *default_timezone=None*)
Import trades from a csv file exported from *ccgains.TradeHistory.export_to_csv()* and add them to this TradeHistory.

Afterwards, all trades will be sorted by date and time.

Parameters **default_timezone** – This parameter is ignored if there is timezone data in the csv string. Otherwise, if None (default) the time data in the csv will be interpreted as time in the local timezone according to the locale setting; or it must be a tzinfo subclass (from dateutil.tz or pytz)

append_coinbase_csv(*file_name*, *currency=None*, *skiprows=4*, *delimiter=', '*, *default_timezone=None*)
Import trades from a csv file exported from Coinbase.com for all wallets (Tools > History > Download History) and adds them to this TradeHistory.

Afterwards, all trades will be sorted by date and time

Parameters

- **currency** – (string) The quote currency used for transactions (e.g. USD/EUR). If not provided, will attempt to determine currency from the csv file, but this may not always be accurate.
- **default_timezone** – This parameter is ignored if there is timezone data in the csv string. Otherwise, if None, the time data in the csv will be interpreted as time in the local timezone according to the locale setting; or it must be a tzinfo subclass (from dateutil.tz or pytz); The default is None, as Coinbase (at the time of writing) outputs local time (at the time of purchase) with transaction history

append_csv(*file_name*, *param_locs=range(0, 11)*, *delimiter=', '*, *skiprows=1*, *default_timezone=None*)
Import trades from a csv file and add them to this TradeHistory.

Afterwards, all trades will be sorted by date and time.

Parameters

- **param_locs** – (list or dict): Locations of Trade's parameters in csv-file. Each entry denotes the column number where a *Trade*-parameter can be found in the csv (Columns are counted starting with 0). If the value is not in the csv, use -1 to use an empty value, a string for a constant value to fill the parameter with, or a function of one parameter (which will be called for each row with a list of the splitted strings in the row as parameter). Note that buy and sell values may be given in reverse order if one of them is negative.
- **default_timezone** – This parameter is ignored if there is timezone data in the csv string. Otherwise, if None (default) the time data in the csv will be interpreted as time in the local timezone according to the locale setting; or it must be a tzinfo subclass (from dateutil.tz or pytz)

append_electrum_csv(*file_name*, *skiprows=1*, *default_timezone=None*)
Import trades from a csv file exported from the Electrum Wallet and add them to this TradeHistory.

It wolrks with exported files from the original Electrum Wallet (BTC) as well as for the Electrum Litecoin Wallet (LTC), as the format is exactly the same.

Afterwards, all trades will be sorted by date and time.

Parameters **default_timezone** – This parameter is ignored if there is timezone data in the csv string. Otherwise, if None, the time data in the csv will be interpreted as time in the local

timezone according to the locale setting; or it must be a tzinfo subclass (from dateutil.tz or pytz); The default is None, i.e. the local timezone, which is what Bitcoin.de exports at time of writing, but it might change in future.

append_poloniex_csv(*file_name*, *which_data='trades'*, *condense_trades=False*, *delimiter=','*, *skiprows=1*, *default_timezone=<Mock name='mock.tz.tzutc()' id='140455774214408'>*)

Import trades from a csv file exported from Poloniex.com and add them to this TradeHistory.

Afterwards, all trades will be sorted by date and time.

Parameters

- **which_data** – (string) Must be one of *"trades"*, *"withdrawals"* or *"deposits"*. Poloniex only allows exporting the three categories 'trading history', 'withdrawal history' and 'deposit history' in separate csv files. Specify which type is loaded here. Default is 'trades'.
- **condense_trades** – (bool) Merge consecutive trades with identical order number? The time of the last merged trade will be used for the resulting trade. Only has an effect if *which_data == 'trades'*.
- **default_timezone** – This parameter is ignored if there is timezone data in the csv string. Otherwise, if None, the time data in the csv will be interpreted as time in the local timezone according to the locale setting; or it must be a tzinfo subclass (from dateutil.tz or pytz); The default is UTC time, which is what Poloniex exports at time of writing, but it might change in future.

append_trezor_csv(*file_name*, *currency*, *skiprows=1*, *default_timezone=None*)

Import trades from a csv file exported from the Trezor wallet and add them to this TradeHistory.

Afterwards, all trades will be sorted by date and time.

Parameters

- **default_timezone** – This parameter is ignored if there is timezone data in the csv string. Otherwise, if None, the time data in the csv will be interpreted as time in the local timezone according to the locale setting; or it must be a tzinfo subclass (from dateutil.tz or pytz); The default is None, i.e. the local timezone, which is what Bitcoin.de exports at time of writing, but it might change in future.
- **currency** – The currency corresponding to the file to be imported. The Trezor wallet exports the information of each wallet separately, but the information of the currency is not supplied. Therefore, the user has to supply the crypto currency accordingly when importing the csv file.

export_to_csv(*path_or_buf=None*, *year=None*, *convert_timezone=True*, ***kwargs*)

Write the list of trades to a csv file.

The csv table will contain the columns: 'kind', 'dtime', 'buy_currency', 'buy_amount', 'sell_currency', 'sell_amount', 'fee_currency', 'fee_amount', 'exchange', 'mark' and 'comment'.

Parameters

- **path_or_buf** – File path (string) or file handle, default None; If None is provided the result is returned as a string.
- **year** – None or 4-digit integer, default: None; Leave *None* to export all trades or choose a specific year to export.
- **convert_timezone** – string, pytz.timezone, dateutil.tz.tzfile, True or False; All dates will be converted to this timezone. The default value, True, will lead to a conversion to the locale timezone according to the system's locale setting. False keeps all dates at UTC time. Otherwise, specify a parameter that will be forwarded to *pandas.Timestamp.tz_convert()*.

export_to_pdf(*file_name*, *year=None*, *convert_timezone=True*, *font_size=11*, *template_file='generic_landscape_table.html'*, *caption='Digital currency trades %(year)s'*, *intro='<h4>Listing of all transactions between %(fromdate)s and %(todate)s</h4>'*, *drop_columns=None*, *custom_column_names=None*, *custom_formatters=None*, *locale=None*)

Export the trade history to a pdf file.

Parameters

- **file_name** – string; Destination file name.
- **year** – None or 4-digit integer, default: None; Leave *None* to export all trades or choose a specific year to export.
- **convert_timezone** – string, pytz.timezone, dateutil.tz.tzfile, True or False; All dates will be converted to this timezone. The default value, True, will lead to a conversion to the locale timezone according to the system's locale setting. False keeps all dates at UTC time. Otherwise, specify a parameter that will be forwarded to *pandas.Timestamp.tz_convert()*.
- **template_file** – file name of html template inside package folder: *ccgains/templates*. Default: 'generic_landscape_table.html'
- **drop_columns** – None or list of strings; Column names specified here (as returned from *to_data_frame*) will be omitted from output.
- **custom_column_names** – None or list of strings; If None (default), the column names of the DataFrame returned from *to_data_frame()* will be used. To rename them, supply a list of length 11-len(*drop_columns*).
- **custom_formatters** – None or dict of one-parameter functions; If None (default), a set of default formatters for each column will be used, using babel.numbers and babel.dates. Individual formatting functions can be supplied with the (renamed) column names as keys. The result of each function must be a unicode string.
- **locale** – None or locale identifier, e.g. 'de_DE' or 'en_US'; The locale used for formatting numeric and date values with babel. If None (default), the locale will be taken from the *LC_NUMERIC* or *LC_TIME* environment variables on your system, for numeric or date values, respectively.

to_data_frame(*year=None*, *convert_timezone=True*)

Put all trades in one big pandas.DataFrame.

Parameters

- **year** – None or 4-digit integer, default: None; Leave *None* to export all trades or choose a specific year to export.
- **convert_timezone** – string, pytz.timezone, dateutil.tz.tzfile, True or False; All dates will be converted to this timezone. The default value, True, will lead to a conversion to the locale timezone according to the system's locale setting. False keeps all dates at UTC time. Otherwise, specify a parameter that will be forwarded to *pandas.Timestamp.tz_convert()*.

to_html(*year=None*, *convert_timezone=True*, *font_size=11*, *template_file='generic_landscape_table.html'*, *caption='Digital currency trades %(year)s'*, *intro='<h4>Listing of all transactions between %(fromdate)s and %(todate)s</h4>'*, *merge_currencies=True*, *drop_columns=None*, *custom_column_names=None*, *custom_formatters=None*, *locale=None*)

Return the trade history as HTML-formatted string.

Parameters

- **year** – None or 4-digit integer, default: None; Leave *None* to export all trades or choose a specific year to export.
- **convert_timezone** – string, pytz.timezone, dateutil.tz.tzfile, True or False; All dates will be converted to this timezone. The default value, True, will lead to a conversion to the locale timezone according to the system's locale setting. False keeps all dates at UTC time. Otherwise, specify a parameter that will be forwarded to *pandas.Timestamp.tz_convert()*.
- **template_file** – file name of html template inside package folder: *ccgains/templates*. Default: 'generic_landscape_table.html'
- **merge_currencies** – Boolean, default True; If True, the three currency columns (e.g. 'buy_currency') will be dropped, with the currency names added to the amount columns (e.g. added to 'buy_amount').
- **drop_columns** – None or list of strings; Column names specified here (as returned from *to_data_frame*) will be omitted from output. If *merge_currencies* is True, don't specify the currency columns here, only the amount column that you want removed.
- **custom_column_names** – None or list of strings; If None (default), the column names of the DataFrame returned from *to_data_frame()* will be used. To rename them, supply a list of proper length (that is, *11 - len(drop_columns)* if *merge_currencies* is False or *8 - len(drop_columns)* otherwise).
- **custom_formatters** – None or dict of one-parameter functions; If None (default), a set of default formatters for each column will be used, using babel.numbers and babel.dates. Individual formatting functions can be supplied with the (renamed) column names as keys. The result of each function must be a unicode string.
- **locale** – None or locale identifier, e.g. 'de_DE' or 'en_US'; The locale used for formatting numeric and date values with babel. If None (default), the locale will be taken from the *LC_NUMERIC* or *LC_TIME* environment variables on your system, for numeric or date values, respectively.

Returns HTML-formatted string

update_ticker_names(*changes=None*)

Update the names of a ticker previously imported into this TradeHistory.

Coins occasionally change ticker symbols, but older history files may not include the change, and instead still refer to the coin by its old name, although pricing history has changed all data to the new name. This method allows for in-place swapping to the new name.

Parameters **changes** – (dict{string: string}) A dictionary in the form {'old ticker': 'new ticker}. All occurrences of 'old ticker' in this TradeHistory will be updated to 'new ticker' Price, cost, amount data will remain unchanged.

1.6 ccgains.reports module

class ccgains.reports.**CapitalGainsReport**(*data=[]*)

This class facilitates the collecting of data like price, proceeds, profit etc. that accrue when processing payments, sales etc. with foreign or digital currencies. Afterwards, provided methods for creating reports from the gathered data can be used. Capital gains reports created from the gathered data can then be exported to csv, html, pdf etc., using the provided methods.

Create a CaptitalGainsReport object.

Then, with every processed payment, you should add data with *add_payment*.

Parameters **data** – list of PaymentReport objects or list of lists/tuples with entries corresponding to PaymentReport._fields, default: empty list; The internal report data will be initialized with the payment reports in the list.

add_payment(*payment_report*)
Add payment data.

Parameters **payment_report** – PaymentReport object; Contains the data to be collected from a processed payment.

export_extended_report_to_pdf(*file_name*, *year=None*, *date_precision='D'*, *combine=True*, *convert_timezone=True*, *font_size=10*, *template_file='fullreport_en.html'*, *payment_kind_translation=None*, *locale=None*)
Export the extended capital gains report to a pdf file.

Parameters

- **file_name** – string; Destination file name.
- **year** – None or 4-digit integer, default: None; Leave *None* to export all sales or choose a specific year to export.
- **date_precision** – one of 'D', 'H' or 'T' for daily, hourly or minutely, respectively (may also be multiplied, e.g.: '5T' for 5-minutely), default: 'D'; Floors all datetimes to the specified frequency. Does nothing if date_precision is False.
- **combine** – boolean, default True; Combines consecutive transactions which only differ in the 'amount', 'cost', 'proceeds' and 'profit'. Such transactions will be combined by summing up the values in these columns. This is only useful if *date_precision* is set, since otherwise consecutive dates will very seldomly match. Therefore, does nothing if *date_precision* is False.
- **convert_timezone** – string, pytz.timezone, dateutil.tz.tzfile, True or False; All dates (i.e. purchase_date and sell_date entries) will be converted to this timezone. The default value, True, will lead to a conversion to the locale timezone according to the system's locale setting. False keeps all dates at UTC time. Otherwise, specify a parameter that will be forwarded to pandas.Timestamp.tz_convert().
- **template_file** – file name of html template inside package folder: *ccgains/templates*. Default: 'fullreport_en.html'
- **payment_kind_translation** – None (default) or dictionary; This allows for the payment kind (one out of ['sale', 'withdrawal fee', 'deposit fee', 'exchange fee']) to be translated (the dict keys must be the mentioned english strings, the values are the translations used in the output).
- **locale** – None or locale identifier, e.g. 'de_DE' or 'en_US'; The locale used for formatting numeric and date values with babel. If None (default), the locale will be taken from the *LC_NUMERIC* or *LC_TIME* environment variables on your system, for numeric or date values, respectively.

export_report_to_pdf(*file_name*, *year=None*, *date_precision='D'*, *combine=True*, *convert_timezone=True*, *font_size=12*, *template_file='shortreport_en.html'*, *custom_column_names=None*, *custom_formatters=None*, *locale=None*)
Export the capital gains report to a pdf file.

Parameters

- **file_name** – string; Destination file name.

- **year** – None or 4-digit integer, default: None; Leave *None* to export all sales or choose a specific year to export.
- **date_precision** – one of 'D', 'H' or 'T' for daily, hourly or minutely, respectively (may also be multiplied, e.g.: '5T' for 5-minutely), default: 'D'; Floors all datetimes to the specified frequency. Does nothing if date_precision is False.
- **combine** – boolean, default True; Combines consecutive transactions which only differ in the 'amount', 'cost', 'proceeds' and 'profit'. Such transactions will be combined by summing up the values in these columns. This is only useful if *date_precision* is set, since otherwise consecutive dates will very seldomly match. Therefore, does nothing if *date_precision* is False.
- **convert_timezone** – string, pytz.timezone, dateutil.tz.tzfile, True or False; All dates (i.e. purchase_date and sell_date entries) will be converted to this timezone. The default value, True, will lead to a conversion to the locale timezone according to the system's locale setting. False keeps all dates at UTC time. Otherwise, specify a parameter that will be forwarded to pandas.Timestamp.tz_convert().
- **template_file** – file name of html template inside package folder: *ccgains/templates*. Default: 'shortreport_en.html'
- **custom_column_names** – None or list of strings; If None (default), the column names of the DataFrame returned from *get_report_data(extended=False)* will be used. To rename them, supply a list of length 10.
- **custom_formatters** – None or dict of one-parameter functions; If None (default), a set of default formatters for each column will be used, using babel.numbers and babel.dates. Individual formatting functions can be supplied with the (renamed) column names as keys. The result of each function must be a unicode string.
- **locale** – None or locale identifier, e.g. 'de_DE' or 'en_US'; The locale used for formatting numeric and date values with babel. If None (default), the locale will be taken from the *LC_NUMERIC* or *LC_TIME* environment variables on your system, for numeric or date values, respectively.

export_short_report_to_csv(*path_or_buf=None*, *year=None*, *date_precision='D'*, *combine=True*, *convert_timezone=True*, *strip_timezone=True*, *custom_column_names=None*, ***kwargs*)

Write the capital gains table to a csv file.

The csv table will contain the columns: 'kind', 'amount', 'currency', 'purchase_date', 'sell_date', 'exchange', 'short_term', 'cost', 'proceeds' and 'profit'.

Parameters

- **path_or_buf** – File path (string) or file handle, default None; If None is provided the result is returned as a string.
- **year** – None or 4-digit integer, default: None; Leave *None* to export all sales or choose a specific year to export.
- **date_precision** – one of 'D', 'H' or 'T' for daily, hourly or minutely, respectively (may also be multiplied, e.g.: '5T' for 5-minutely), default: 'D'; Floors all datetimes to the specified frequency. Does nothing if date_precision is False.
- **combine** – boolean, default True; Combines consecutive transactions which only differ in the 'amount', 'cost', 'proceeds' and 'profit'. Such transactions will be combined by summing up the values in these columns. This is only useful if *date_precision* is set, since otherwise consecutive dates will very seldomly match. Therefore, does nothing if *date_precision* is False.

- **convert_timezone** – string, pytz.timezone, dateutil.tz.tzfile, True or False; All dates (i.e. purchase_date and sell_date entries) will be converted to this timezone. The default value, True, will lead to a conversion to the locale timezone according to the system's locale setting. False keeps all dates at UTC time. Otherwise, specify a parameter that will be forwarded to pandas.Timestamp.tz_convert().
- **strip_timezone** – boolean, default True; After conversion, the timezone info will be removed from all dates.
- **custom_column_names** – None or list of strings; If None (default), the column names will be: ['kind', 'amount', 'currency', 'purchase_date', 'sell_date', 'exchange', 'short_term', 'cost', 'proceeds', 'profit']. To rename them, supply a list of length 10.

get_extended_report_html(*year=None*, *date_precision='D'*, *combine=True*, *convert_timezone=True*, *font_size=10*, *template_file='fullreport_en.html'*, *payment_kind_translation=None*, *locale=None*)

Return an extended capital gains report as HTML-formatted string.

Parameters

- **year** – None or 4-digit integer, default: None; Leave *None* to export all sales or choose a specific year to export.
- **date_precision** – one of 'D', 'H' or 'T' for daily, hourly or minutely, respectively (may also be multiplied, e.g.: '5T' for 5-minutely), default: 'D'; Floors all datetimes to the specified frequency. Does nothing if date_precision is False.
- **combine** – boolean, default True; Combines consecutive transactions which only differ in the 'amount', 'cost', 'proceeds' and 'profit'. Such transactions will be combined by summing up the values in these columns. This is only useful if *date_precision* is set, since otherwise consecutive dates will very seldomly match. Therefore, does nothing if *date_precision* is False.
- **convert_timezone** – string, pytz.timezone, dateutil.tz.tzfile, True or False; All dates (i.e. purchase_date and sell_date entries) will be converted to this timezone. The default value, True, will lead to a conversion to the locale timezone according to the system's locale setting. False keeps all dates at UTC time. Otherwise, specify a parameter that will be forwarded to pandas.Timestamp.tz_convert().
- **template_file** – file name of html template inside package folder: *ccgains/templates*. Default: 'fullreport_en.html'
- **payment_kind_translation** – None (default) or dictionary; This allows for the payment kind (one out of ['sale', 'withdrawal fee', 'deposit fee', 'exchange fee']) to be translated (the dict keys must be the mentioned english strings, the values are the translations used in the output).
- **locale** – None or locale identifier, e.g. 'de_DE' or 'en_US'; The locale used for formatting numeric and date values with babel. If None (default), the locale will be taken from the *LC_NUMERIC* or *LC_TIME* environment variables on your system, for numeric or date values, respectively.

Returns HTML-formatted string

get_report_data(*year=None*, *date_precision='D'*, *combine=True*, *convert_timezone=True*, *strip_timezone=True*, *extended=False*, *custom_column_names=None*)

Return a pandas.DataFrame listing the capital gains made with the processed trades.

Parameters

- **year** – None or 4-digit integer, default: None; Leave *None* to return all sales or choose a specific year to return.
- **date_precision** – one of 'D', 'H' or 'T' for daily, hourly or minutely, respectively (may also be multiplied, e.g.: '5T' for 5-minutely), default: 'D'; Floors all datetimes to the specified frequency. Does nothing if date_precision is False.
- **combine** – boolean, default True; Combines consecutive transactions which only differ in the amounts 'to_pay', 'bag_amount', 'bag_spent', 'spent_cost', 'proceeds' and 'profit'. Such transactions will be combined by summing up the values in these columns. This is only useful if *date_precision* is set, since otherwise consecutive dates will very seldomly match. Therefore, does nothing if *date_precision* is False.
- **convert_timezone** – string, pytz.timezone, dateutil.tz.tzfile, True or False; All dates (i.e. purchase_date and sell_date entries) will be converted to this timezone. The default value, True, will lead to a conversion to the locale timezone according to the system's locale setting. False keeps all dates at UTC time. Otherwise, specify a parameter that will be forwarded to pandas.Timestamp.tz_convert().
- **strip_timezone** – boolean, default True; After conversion, the timezone info will be removed from all dates.
- **extended** – boolean, default False;

  By default, the returned DataFrame contains the columns:

  ['kind', 'bag_spent', 'currency', 'bag_date', 'sell_date', 'exchange', 'short_term', 'spent_cost', 'proceeds', 'profit'];

  If *extended* is True, these columns will be returned:

  ['kind', 'exchange', 'sell_date', 'currency', 'to_pay','fee_ratio', 'bag_date', 'bag_amount', 'bag_spent', 'cost_currency', 'spent_cost', 'short_term', 'ex_rate', 'proceeds', 'profit', 'buy_currency, buy_ratio']

  Note the reordering of columns in the small dataset.
- **custom_column_names** – None or list of strings; If None (default), the column names will be as described above, depending on *extended*. To rename them, supply a list of proper length, either 10 if not *extended* or 17 otherwise.

Returns A pandas.DataFrame with the requested data.

get_report_html(*year=None*, *date_precision='D'*, *combine=True*, *convert_timezone=True*, *font_size=12*, *template_file='shortreport_en.html'*, *custom_column_names=None*, *custom_formatters=None*, *locale=None*, *extended_data=False*)

Return the capital gains report as HTML-formatted string.

Parameters

- **year** – None or 4-digit integer, default: None; Leave *None* to export all sales or choose a specific year to export.
- **date_precision** – one of 'D', 'H' or 'T' for daily, hourly or minutely, respectively (may also be multiplied, e.g.: '5T' for 5-minutely), default: 'D'; Floors all datetimes to the specified frequency. Does nothing if date_precision is False.
- **combine** – boolean, default True; Combines consecutive transactions which only differ in the 'amount', 'cost', 'proceeds' and 'profit'. Such transactions will be combined by summing up the values in these columns. This is only useful if *date_precision* is set, since otherwise consecutive dates will very seldomly match. Therefore, does nothing if *date_precision* is False.

- **convert_timezone** – string, pytz.timezone, dateutil.tz.tzfile, True or False; All dates (i.e. purchase_date and sell_date entries) will be converted to this timezone. The default value, True, will lead to a conversion to the locale timezone according to the system's locale setting. False keeps all dates at UTC time. Otherwise, specify a parameter that will be forwarded to pandas.Timestamp.tz_convert().
- **template_file** – file name of html template inside package folder: *ccgains/templates*. Default: 'shortreport_en.html'
- **custom_column_names** – None or list of strings; If None (default), the column names of the DataFrame returned from *get_report_data(extended=extended_data)* will be used. To rename them, supply a list with same length than number of columns (depending on *extended_data*).
- **custom_formatters** – None or dict of one-parameter functions; If None (default), a set of default formatters for each column will be used, using babel.numbers and babel.dates. Individual formatting functions can be supplied with the (renamed) column names as keys. The result of each function must be a unicode string.
- **locale** – None or locale identifier, e.g. 'de_DE' or 'en_US'; The locale used for formatting numeric and date values with babel. If None (default), the locale will be taken from the *LC_NUMERIC* or *LC_TIME* environment variables on your system, for numeric or date values, respectively.
- **extended_data** – Boolean, default: False; If the *template_file* makes use of some of the extended data returned from *get_report_data* when called with parameter *extended=True*, this must also be True. See documentation of *get_report_data* for extended data fields.

Returns HTML-formatted string

to_json(***kwargs*)

Convert the collected data to a JSON formatted string.

Parameters **kwargs** – Keyword arguments that will be forwarded to *json.dumps*.

Returns JSON formatted string

class ccgains.reports.**PaymentReport**(*kind*, *exchange*, *sell_date*, *currency*, *to_pay*, *fee_ratio*, *bag_date*, *bag_amount*, *bag_spent*, *cost_currency*, *spent_cost*, *short_term*, *ex_rate*, *proceeds*, *profit*, *buy_currency*, *buy_ratio*)

This is a container for a couple of values that are gathered at every payment, which will be needed for creating a capital gains report.

Create new instance of PaymentReport(kind, exchange, sell_date, currency, to_pay, fee_ratio, bag_date, bag_amount, bag_spent, cost_currency, spent_cost, short_term, ex_rate, proceeds, profit, buy_currency, buy_ratio)

bag_amount

Alias for field number 7

bag_date

Alias for field number 6

bag_spent

Alias for field number 8

buy_currency

Alias for field number 15

buy_ratio

Alias for field number 16

cost_currency
Alias for field number 9

currency
Alias for field number 3

ex_rate
Alias for field number 12

exchange
Alias for field number 1

fee_ratio
Alias for field number 5

kind
Alias for field number 0

proceeds
Alias for field number 13

profit
Alias for field number 14

sell_date
Alias for field number 2

short_term
Alias for field number 11

spent_cost
Alias for field number 10

to_pay
Alias for field number 4

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index