# Package ‘Rbitcoin’

August 29, 2016

**Type** Package

**Title** R & bitcoin integration

**Version** 0.9.2

**Date** 2014-09-01

**Author** Jan Gorecki

**Maintainer** Jan Gorecki <j.gorecki@wit.edu.pl>

**Depends** R (>= 2.10), data.table

**Imports** RCurl, digest, RJSONIO

**Description** Utilities related to Bitcoin. Unified markets API interface (bitstamp, kraken, btce, bitmarket). Both public and private API calls. Integration of data structures for all markets. Support SSL. Read Rbitcoin documentation (command: ?btc) for more information.

**License** MIT + file LICENSE

**URL** https://github.com/jangorecki/Rbitcoin

**BugReports** https://github.com/jangorecki/Rbitcoin/issues

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2014-09-01 22:59:05

## R topics documented:

## `antiddos` *Anti DDoS*

### Description

Wait if you should before next API call to market (or any other source system) to do not get banned.

### Usage

```
antiddos(market, antispam_interval = 10,
  verbose = getOption("Rbitcoin.verbose", 0))
```

### Arguments

| | |
|---|---|
| `market` | character, a unique name of source system, could be any name `c('kraken','bitstamp','blockchain',` |
| `antispam_interval` | numeric time in seconds between API calls on the particular source system, defeault `10s`. |
| `verbose` | integer. Rbitcoin processing messages, print to console if `verbose > 0`, each subfunction reduce `verbose` by 1. If missing then `getOption("Rbitcoin.verbose",0)` is used, by default `0`. |

### Value

numeric time of wait in seconds.

### Side effect

Environment of name `Rbitcoin.last_api_call` in `.GlobalEnv` which holds the timestamps of last api call per `market` during the R session.

### See Also

`market.api.process`, `wallet_manager`

## Examples

```
## Not run:
# run below code in a batch
wait <- antiddos(market = 'kraken', antispam_interval = 5, verbose = 1)
market.api.process('kraken',c('BTC','EUR'),'ticker')
wait2 <- antiddos(market = 'kraken', antispam_interval = 5, verbose = 1)
market.api.process('kraken',c('BTC','EUR'),'ticker')

## End(Not run)
```

## api.dict *API dictionary*

### Description

This data set contains dictionary (`data.table` object) for `market.api.process` function which perform pre-process API call request, post-process API call results and catch market level errors. Still there is function `market.api.query` that do not require any dictionary and can operate on any currency pairs. Run `data(api.dict); api.dict` to print built-in dictionary. Granularity of data is `c(market, base, quote, action)`. This dictionary can be edited/extended by user for new currency pairs.
Currently supported currency pairs:

- `bitstamp: BTCUSD`
- `btce: BTCUSD, LTCUSD, LTCBTC, NMCBTC`
- `kraken: BTCEUR, LTCEUR, BTCLTC`
- `bitmarket: BTCPLN, LTCPLN`
- `mtgox: BTCUSD`

### Usage

```
data(api.dict)
```

### Note

Do not use `api.dict` from untrusted source or read whole it's code to ensure it is safe! The api dictionary was not fully tested, please follow the examples, if you find any bugs please report.

### Author(s)

Jan Gorecki, 2014-08-13

## `available_wallet` *Available wallet*

### Description

Calculates assets available to trade, not on hold by current open orders.

### Usage

```
available_wallet(wallet, open_orders, verbose = getOption("Rbitcoin.verbose",
  0))
```

### Arguments

| | |
|---|---|
| `wallet` | data.table object returned by `market.api.process` with `action="wallet"` param. |
| `open_orders` | data.table object returned by `market.api.process` with `action="open_orders"` param. |
| `verbose` | integer. Rbitcoin processing messages, print to console if `verbose > 0`, each subfunction reduce `verbose` by 1. If missing then `getOption("Rbitcoin.verbose",0)` is used, by default `0`. |

### Value

data.table object, the same as wallet but with the appropriate amounts after subtracting the open orders amounts.

### See Also

`market.api.process`

### Examples

```
## Not run:
wallet <- market.api.process('kraken',c('BTC','EUR'),'wallet', key = '', secret = '')
Sys.sleep(10)
open_orders <- market.api.process('kraken',c('BTC','EUR'),'open_orders', key = '', secret = '')
aw <- available_wallet(wallet, open_orders, verbose = 1)
print(aw)

## End(Not run)
```

`blockchain.api.process` *Process blockchain.info API*

## Description

Query and process results from blockchain.info.

## Usage

```
blockchain.api.process(..., method, verbose = getOption("Rbitcoin.verbose",
  0))
```

## Arguments

| | |
|---|---|
| `...` | params passed to blockchain.info API, specific for particular method, example `'bitcoin_address'` or `'tx_index'`, for more read `blockchain.api.query`. |
| `method` | character. For details see `blockchain.api.query`, currently supported `'Single Address'` and `'Single Transaction'`. If `method` missing the function will try to guess it based on first param in `...`. |
| `verbose` | integer. Rbitcoin processing messages, print to console if `verbose > 0`, each subfunction reduce `verbose` by 1. If missing then `getOption("Rbitcoin.verbose",0)` is used, by default `0`. |

## Value

data.table object, blockchain api data transformed to table.

## See Also

`blockchain.api.query`

## Examples

```
## Not run:
# Rbitcoin donation address wallet
Rbitcoin_donation_wallet <- blockchain.api.process('15Mb2QcgF3XDMeVn6M7oCG6CQLw4mkedDi')
# some transaction
tx <- blockchain.api.process('e5c4de1c70cb6d60db53410e871e9cab6a0ba75404360bf4cda1b993e58d45f8')

## End(Not run)
```

## blockchain.api.query *Query blockchain.info API*

### Description

Query bitcoin related data from blockchain.info.

### Usage

```
blockchain.api.query(..., method, verbose = getOption("Rbitcoin.verbose", 0))
```

### Arguments

| | |
|---|---|
| `...` | params passed to blockchain.info API, specific for particular method, example `'bitcoin_address'` or `'tx_index'`, for more see references or examples. |
| `method` | character. For details see references, currently supported `'Single Address'` and `'Single Transaction'`. If `method` missing the function will try to guess it based on first param in `...`. |
| `verbose` | integer. Rbitcoin processing messages, print to console if `verbose > 0`, each subfunction reduce `verbose` by 1. If missing then `getOption("Rbitcoin.verbose",0)` is used, by default `0`. |

### Value

result returned by `fromJSON` function applied on the blockchain result, most probably the list.

### References

https://blockchain.info/api/blockchain_api

### See Also

`market.api.query`

### Examples

```
## Not run:
# query bitcoin address information - 'Single Address' method
# Rbitcoin donation address final balance in BTC
blockchain.api.query('15Mb2QcgF3XDMeVn6M7oCG6CQLw4mkedDi',limit=0)[['final_balance']]/100000000
# Rbitcoin donation address full details
blockchain.api.query('15Mb2QcgF3XDMeVn6M7oCG6CQLw4mkedDi',verbose=1)
# some first wallet final balance in BTC
blockchain.api.query('1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa',limit=0)[['final_balance']]/100000000
# some first wallet details (limit to 3 txs, skip two txs)
blockchain.api.query(method = 'Single Address',
                 bitcoin_address = '1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa', limit=3, offset=2)
# query bitcoin transaction information - 'Single Transaction' method
# Some recent transaction of some first wallet
blockchain.api.query('e5c4de1c70cb6d60db53410e871e9cab6a0ba75404360bf4cda1b993e58d45f8')

## End(Not run)
```

---

## market.api.process *Process market API*

---

### Description

Unified processing of API call according to API dictionary api.dict. Limited to markets and currency processing defined in api.dict, in case of currency pairs and methods not availble in dictionary use market.api.query directly. This function perform pre processing of request and post processing of API call results to unified structure across markets. It will result truncation of most (not common across the markets) attributes returned. If you need the full set of data returned by market's API you should use market.api.query.

### Usage

```
market.api.process(market, currency_pair, action, req = list(), ...,
  verbose = getOption("Rbitcoin.verbose", 0),
  on.market.error = expression(stop(e[["message"]], call. = FALSE)),
  on.error = expression(stop(e[["message"]], call. = FALSE)),
  api.dict = NULL, raw.query.res = FALSE)
```

### Arguments

| | |
|---|---|
| market | character, example: 'kraken'. |
| currency_pair | character vector of length 2, ex. c(base = 'BTC', quote = 'EUR'). Order does matter. |
| action | character, defined process to get organized data. |
| req | list with action details (price, amount, tid, oid, etc.) unified across the markets specific per action, see examples. |
| ... | objects to be passed to market.api.query<br>• auth params: key, secret, client_id (last one used on bitstamp), |
| verbose | integer. Rbitcoin processing messages, print to console if verbose > 0, each subfunction reduce verbose by 1. If missing then getOption("Rbitcoin.verbose",0) is used, by default 0. |
| on.market.error | expression to be evaluated on market level error. Rules specified in api.dict. |
| on.error | expression to be evaluated on R level error related to market.api.query. For details read market.api.query. |
| api.dict | data.table user custom API dictionary definition, if not provided function will use default Rbitcoin api.dict. |
| raw.query.res | logical skip post-processing are return results only after fromJSON processing. Useful in case of change results structure from market API. It can always be manually post-processed as a workaround till the Rbitcoin update. |

## Details

To do not spam market's API, use `Sys.sleep(10)` between API calls.

## Value

Returned value depends on the `action` param. All actions will return market, currency pair (except `wallet` and `open_orders` which returns all currencies), R timestamp, market timestamp and below data (in case if market not provide particular data, it will result `NA` value):

- `'ticker'` returns `data.table` with fields: `last`, `vwap`, `volume`, `ask`, `bid`.
- `'wallet'` returns `data.table` with fields: `currency`, `amount`, `fee`.
- `'order_book'` returns `list` with API call level attributes and sub elements `[['asks']]` and `[['bids']]` as `data.table` objects with order book including already calculated cumulative `amount`, `price` and `value`.
- `'open_orders'` returns `data.table` with fields: `oid`, `type`, `price`, `amount`.
- `'place_limit_order'` returns `data.table` with fields: `oid`, `type`, `price`, `amount`.
- `'cancel_order'` returns `data.table` with fields: `oid`.
- `'trades'` returns `list` with API call level attributes and sub element `[['trades']]` as `data.table` (ASC order) with fields: `date`, `price`, `amount`, `tid`, `type`.

## Note

The api dictionary was not fully tested, please follow the examples, if you find any bugs please report. Use only api dictionary `api.dict` from trusted source, in case if you use other `api.dict` it is advised to review pre-process, post-process and catch_market_error functions for markets and currency pairs you are going to use. Market level error handling might not fully work as not all markets returns API call status information.

## See Also

`market.api.query`

## Examples

```
## Not run:
# get ticker from market
market.api.process(market = 'kraken', currency_pair = c('BTC', 'EUR'), action='ticker')
# get ticker from all markets and combine
ticker_all <- rbindlist(list(
  market.api.process(market = 'bitstamp', currency_pair = c('BTC', 'USD'), action='ticker')
  ,market.api.process(market = 'btce', currency_pair = c('LTC', 'USD'), action='ticker')
  ,{Sys.sleep(10);
   market.api.process(market = 'btce', currency_pair = c('LTC', 'BTC'), action='ticker')}
  ,{Sys.sleep(10);
   market.api.process(market = 'btce', currency_pair = c('NMC', 'BTC'), action='ticker')}
 ,market.api.process(market = 'kraken', currency_pair = c('BTC','EUR'), action='ticker')
  ,{Sys.sleep(10);
   market.api.process(market = 'kraken', currency_pair = c('LTC','EUR'), action='ticker')}
  ,{Sys.sleep(10);
   market.api.process(market = 'kraken', currency_pair = c('BTC','LTC'), action='ticker')}
))
print(ticker_all)

# get wallet from market
market.api.process(market = 'kraken', currency_pair = c('BTC', 'EUR'), action = 'wallet',
                   key = '', secret = '')
# get wallet from all markets and combine
wallet_all <- rbindlist(list(
  market.api.process(market = 'bitstamp', currency_pair = c('BTC', 'USD'), action = 'wallet',
                     client_id = '', key = '', secret = ''),
  market.api.process(market = 'btce', currency_pair = c('LTC', 'USD'), action = 'wallet',
                     method = '', key = '', secret = ''),
  market.api.process(market = 'kraken', currency_pair = c('BTC', 'EUR'), action = 'wallet',
                     key = '', secret = '')
))
print(wallet_all)

# get order book from market
market.api.process(market = 'kraken', currency_pair = c('BTC', 'EUR'), action = 'order_book')

# get open orders from market
market.api.process(market = 'kraken', currency_pair = c('BTC', 'EUR'), action = 'open_orders',
                   key = '', secret = '')

# place limit order
market.api.process(market = 'kraken', currency_pair = c('BTC', 'EUR'), action = 'place_limit_order',
               req = list(type = 'sell', amount = 1, price = 8000), # sell 1 btc for 8000 eur
                   key = '', secret = '')

# cancel order
market.api.process(market = 'kraken', currency_pair = c('BTC', 'EUR'), action = 'cancel_order,
                   req = list(oid = 'oid_from_open_orders'),
                   key = '', secret = '')
# get trades
market.api.process(market = 'kraken', currency_pair = c('BTC', 'EUR'), action = 'trades')

## End(Not run)
```

---

## `market.api.query` *Send request to market API*

---

### Description

Route a request to particular market function.

### Usage

```
market.api.query(market, ..., verbose = getOption("Rbitcoin.verbose", 0),
  on.error = expression(stop(e[["message"]], call. = FALSE)))
```

## Arguments

| | |
|---|---|
| `market` | character which identifies market on which we want to send request: bitstamp, btce, kraken, bitmarket. |
| `...` | objects to be passed to API: `url`, `key`, `secret`, `req`, `client_id` (used on bitstamp). |
| `verbose` | integer. Rbitcoin processing messages, print to console if `verbose > 0`, each subfunction reduce `verbose` by 1. If missing then `getOption("Rbitcoin.verbose",0)` is used, by default `0`. |
| `on.error` | expression to be evaluated on R level error of market specific function. It does not catch internal market's error returned as valid object. |

## Details

To do not spam market's API, use `Sys.sleep(10)` between API calls.

## Value

R object created by `fromJSON` decoded result from market's API call.

## Note

It is advised to use this function instead of calling market's function directly. If calling directly one should ensure to send any numeric values in non-exponential notation: `options(scipen=100)`.

## References

API documentation: https://bitbucket.org/nitrous/mtgox-api, https://www.bitstamp.net/api/, https://btc-e.com/api/documentation, https://www.kraken.com/help/api

## See Also

`market.api.process`, `market.api.query.bitstamp`, `market.api.query.btce`, `market.api.query.kraken`, `market.api.query.bitmarket`, `market.api.query.mtgox`

## Examples

```
## Not run:
# ticker
market.api.query(market = 'bitstamp',
                 url = 'https://www.bitstamp.net/api/ticker/')
market.api.query(market = 'btce',
                 url = 'https://btc-e.com/api/2/btc_usd/ticker')
market.api.query(market = 'kraken',
                 url = 'https://api.kraken.com/0/public/Ticker?pair=XXBTZEUR')
market.api.query(market = 'bitmarket',
                 url = 'https://www.bitmarket.pl/json/LTCPLN/ticker.json')
# wallet
market.api.query(market = 'bitstamp',
                 url = 'https://www.bitstamp.net/api/balance/',
                 client_id = '', # bitstamp specific
                 key = '', secret = '')
market.api.query(market = 'btce',
                 url = 'https://btc-e.com/tapi',
                 req = list(method = 'getInfo'),
                 key = '', secret = '')
market.api.query(market = 'kraken',
                 url = 'https://api.kraken.com/0/private/Balance',
                 key = '', secret = '')
market.api.query(market = 'bitmarket',
                 url = 'https://www.bitmarket.pl/api2/',
                 req = list(method = 'info'),
                 key = '', secret = '')
# order book
market.api.query(market = 'kraken',
                 url = 'https://api.kraken.com/0/public/Depth?pair=XXBTZEUR')
# open orders
market.api.query(market = 'kraken',
                 url = 'https://api.kraken.com/0/private/OpenOrders',
                 key = '', secret = '')
# place order
market.api.query(market = 'kraken',
                 url = 'https://api.kraken.com/0/private/AddOrder',
                 key = '', secret = '',
                 req = list(pair = 'XXBTZEUR',
                            type = 'sell',
                            ordertype = 'limit',
                            price = 1200, # 1200 eur
                            volume = 0.1)) # 0.1 btc
# cancel order
market.api.query(market = 'kraken',
                 url = 'https://api.kraken.com/0/private/CancelOrder',
                 key = '', secret = '',
                 req = list(txid = 'id_from_open_orders'))
# trades
market.api.query(market = 'kraken',
                 url = 'https://api.kraken.com/0/public/Trades?pair=XXBTZEUR')

## End(Not run)
```

## `market.api.query.bitmarket` *Send request to bitmarket market API*

### Description

Send request to bitmarket market API.

## Usage

```
market.api.query.bitmarket(url, key, secret, req = list(),
  verbose = getOption("Rbitcoin.verbose", 0))
```

## Arguments

| | |
|---|---|
| `url` | character with url on which query needs to be passed. |
| `key` | character API key used in private API calls. |
| `secret` | character API secret used in private API calls. |
| `req` | list of object passed to API: price and amount of opening order, id of cancelling order, etc. |
| `verbose` | integer. Rbitcoin processing messages, print to console if `verbose > 0`, each subfunction reduce `verbose` by 1. If missing then `getOption("Rbitcoin.verbose",0)` is used, by default `0`. |

## Value

R object created by `fromJSON` decoded result from market's API call.

## Note

Market specific bitmarket `method` param should be provided in `req` object.

## References

https://www.bitmarket.pl/docs.php?file=api_private.html

## See Also

`market.api.query`

## Examples

```
## Not run:
# ticker
market.api.query.bitmarket(url = 'https://www.bitmarket.pl/json/LTCPLN/ticker.json')
# wallet
market.api.query.bitmarket(url = 'https://www.bitmarket.pl/api2/',
                           req = list(method = 'info'),
                           key = '', secret = '')

## End(Not run)
```

## `market.api.query.bitstamp` *Send request to bitstamp market API*

### Description

Send request to bitstamp market API.

### Usage

```
market.api.query.bitstamp(url, client_id, key, secret, req = list(),
  verbose = getOption("Rbitcoin.verbose", 0))
```

### Arguments

| | |
|---|---|
| `url` | character with url on which query needs to be passed. |
| `client_id` | character. Bitstamp market specific parameter used in private API call authorization (check reference for more information). |
| `key` | character API key used in private API calls. |
| `secret` | character API secret used in private API calls. |
| `req` | list of object passed to API: price and amount of opening order, id of cancelling order, etc.. |
| `verbose` | integer. Rbitcoin processing messages, print to console if `verbose > 0`, each subfunction reduce `verbose` by 1. If missing then `getOption("Rbitcoin.verbose",0)` is used, by default `0`. |

### Value

R object created by `fromJSON` decoded result from market's API call. Cancel order is an exception handled by hardcode, as bitstamp will not return json format for that method.

### References

https://www.bitstamp.net/api/

### See Also

`market.api.query`

### Examples

```
## Not run:
# ticker
market.api.query.bitstamp(url = 'https://www.bitstamp.net/api/ticker/')
# wallet
market.api.query.bitstamp(url = 'https://www.bitstamp.net/api/balance/',
                          client_id = '',
                              key = '', secret = '')

## End(Not run)
```

## `market.api.query.btce` *Send request to btce market API*

### Description

Send request to btce market API.

### Usage

```
market.api.query.btce(url, key, secret, req = list(),
  verbose = getOption("Rbitcoin.verbose", 0))
```

### Arguments

| | |
|---|---|
| `url` | character with url on which query needs to be passed. |
| `key` | character API key used in private API calls. |
| `secret` | character API secret used in private API calls. |
| `req` | list of object passed to API: price and amount of opening order, id of cancelling order, etc. See note. |
| `verbose` | integer. Rbitcoin processing messages, print to console if `verbose > 0`, each subfunction reduce `verbose` by 1. If missing then `getOption("Rbitcoin.verbose",0)` is used, by default `0`. |

### Value

R object created by `fromJSON` decoded result from market's API call.

### Note

Market specific btce `method` param should be provided in `req` object.

### References

`https://btc-e.com/api/documentation`

### See Also

`market.api.query`

## Examples

```
## Not run:
# ticker
market.api.query.btce(url = 'https://btc-e.com/api/2/btc_usd/ticker')
# wallet
market.api.query.btce(url = 'https://btc-e.com/tapi',
                      req = list(method = 'getInfo'),
                      key = '', secret = '')

## End(Not run)
```

---

## market.api.query.kraken

*Send request to kraken market API*

---

### Description

Send request to kraken market API.

### Usage

```
market.api.query.kraken(url, key, secret, req = list(),
  verbose = getOption("Rbitcoin.verbose", 0))
```

### Arguments

| | |
|---|---|
| `url` | character with url on which query needs to be passed. |
| `key` | character API key used in private API calls. |
| `secret` | character API secret used in private API calls. |
| `req` | list of object passed to API: price and amount of opening order, id of cancelling order, etc. |
| `verbose` | integer. Rbitcoin processing messages, print to console if `verbose > 0`, each subfunction reduce `verbose` by 1. If missing then `getOption("Rbitcoin.verbose",0)` is used, by default `0`. |

### Value

R object created by `fromJSON` decoded result from market's API call.

### References

`https://www.kraken.com/help/api`

### See Also

`market.api.query`

### Examples

```
## Not run:
# ticker
market.api.query.kraken(url = 'https://api.kraken.com/0/public/Ticker?pair=XBTCZEUR')
# wallet
market.api.query.kraken(url = 'https://api.kraken.com/0/private/Balance',
                        key = '', secret = '')

## End(Not run)
```

---

## market.api.query.mtgox

*Send request to mtgox market API*

---

### Description

Send request to mtgox market API. MtGox is already closed but public API calls are working. Also it's code/dictionary can be reused in future.

### Usage

```
market.api.query.mtgox(url, key, secret, req = list(),
  verbose = getOption("Rbitcoin.verbose", 0))
```

### Arguments

| | |
|---|---|
| `url` | character with url on which query needs to be passed. |
| `key` | character API key used in private API calls. |
| `secret` | character API secret used in private API calls. |
| `req` | list of object passed to API: price and amount of opening order, id of cancelling order, etc. |
| `verbose` | integer. Rbitcoin processing messages, print to console if `verbose > 0`, each subfunction reduce `verbose` by 1. If missing then `getOption("Rbitcoin.verbose",0)` is used, by default `0`. |

### Value

R object created by `fromJSON` decoded result from market's API call.

### References

https://bitbucket.org/nitrous/mtgox-api

### See Also

`market.api.query`

### Examples

```
## Not run:
# ticker
market.api.query.mtgox(url = 'https://data.mtgox.com/api/2/BTCUSD/money/ticker_fast')
# wallet
market.api.query.mtgox(url = 'https://data.mtgox.com/api/2/BTCUSD/money/info',
                       key = '', secret = '')

## End(Not run)
```

---

## `Rbitcoin` *R & bitcoin integration*

---

### Description

Utilities related to Bitcoin and other cryptocurrencies. Core functionalities are:

- `market.api.query` - launch query on market's API (`bitstamp`, `btce`, `kraken`, `bitmarket`). Both public and private API calls supported. All currency pairs supported.
- `market.api.process` - integration of market's processing structures: pre-process of API request, post-process API results, market error catching. Input and output unified structure. Requires API dictionary definition, for details of package built-in dictionary see `api.dict`.
- `blockchain.api.query` - launch query on blockchain.info API json interface.
- `blockchain.api.process` - postprocess blockchain api result, transform to `data.table`.
- `Rbitcoin.plot` - illustrate the data returned by some Rbitcoin functions.
- `wallet_manager` - track the assets amounts and values in multiple wallet sources.

You need to note that imported `digest` package docs states: *Please note that this package is not meant to be deployed for cryptographic purposes for which more comprehensive (and widely tested) libraries such as OpenSSL should be used.* Still `digest` is one of the top downloaded package from CRAN.
To do not get banned by market's API anti-DDoS protection user should use: `Sys.sleep(10)` between the API calls or `antiddos` function.
It is advised to maintain your API keys security level as tight as possible, if you do not need withdraw api method be sure to disable it for api keys.
You can print debug messages of `Rbitcoin` to console using verbose argument in FUNs or `options("Rbitcoin.verbose" =`
Two params `ssl.verify` and `curl.verbose` have been deprecated since `0.8.5`. They can and should be controlled using `options("RCurlOptions")`. SSL verify is by default active.
At the time of writing the most recent market's API version were used:

- bitstamp v2 (public) / ? (private)
- btce v2 (public) / "tapi" (private)
- kraken v0
- bitmarket v2
- mtgox v2 (market already closed)

SSL is by default active, to disable SSL set `RCurlOptions` to `ssl.verify* = FALSE` and `cainfo = NULL`, see examples. In case of SSL error try update certificate CA file (`cacert.pem` in location mentioned below as `cainfo`), see references for CA file source. Alternatively you can always disable SSL.

For others package-level options see examples.

BTC donation: `bitcoin:15Mb2QcgF3XDMeVn6M7oCG6CQLw4mkedDi`

## References

Package discussion thread: `https://bitcointalk.org/index.php?topic=343504`
Example SSL CA file source: `http://curl.haxx.se/docs/caextract.html`

## See Also

`market.api.process`, `blockchain.api.process`, `wallet_manager`, `Rbitcoin.plot`, `api.dict`, `available_wallet`

## Examples

```
## Not run:
# default options used by Rbitcoin

# print Rbitcoin processing to console set "Rbitcoin.verbose" to 1 (or more)
options(Rbitcoin.verbose=0)

# print Rcurl processing to console set RCurlOptions[["verbose"]] to TRUE
options(RCurlOptions=list(ssl.verifypeer = TRUE,
                          ssl.verifyhost = TRUE,
                          cainfo = system.file("CurlSSL","cacert.pem",package="RCurl"),
                          verbose = FALSE))

# currency type dictionary used by wallet_manager
options(Rbitcoin.ct.dict = list(
  crypto = c('BTC','LTC','NMC', ...),
  fiat = c('USD','EUR','GBP', ...)
))

## End(Not run)
```

---

`Rbitcoin.plot` *Plot Rbitcoin objects*

---

## Description

Generic function to plot different objects returned by some Rbitcoin functions. The plot produce basic visualization of the data. The plots will likely be subject to change in future versions.

## Usage

```
Rbitcoin.plot(x, mask = FALSE, ..., export = FALSE,
  export.args = list(format = "svg", filename = NULL),
  verbose = getOption("Rbitcoin.verbose", 0))
```

## Arguments

| | |
|---|---|
| `x` | object to be plotted, result of Rbitcoin function, currently supported: `market.api.process` with `action` in `c("trades","order_book")`, `wallet_manager` with `archive_read = TRUE`. |
| `mask` | logical, default `FALSE`, setting `TRUE` will mask values on wallet manager plot with the ratio of value to the initial value. Use this when you want to share the plot. See examples to mask the bitcoin address. |
| `export` | logical default `FALSE`, if `TRUE` the plot will be exported to file instead of plot to ploting device. |
| `export.args` | list of arguments to be passed to target graphic device function, ex. `svg()` or `png()`, list gives the control over width and height which by default for png are quite small. Element `export.args[['format']]` will be mapped to the function name, by default `svg()`, any others as its `args`. |
| `...` | additional params to be passed to plot function. |
| `verbose` | integer. Rbitcoin processing messages, print to console if `verbose > 0`, each subfunction reduce `verbose` by 1. If missing then `getOption("Rbitcoin.verbose",0)` is used, by default `0`. |

## Value

TRUE

## Export

Element `format` in the `export.args` list defines the export format, default `"svg"`, tested and supported formats are `"svg"` and `"png"`, other might work also. To use custom export filename just pass the filename arg to export.args list. By default `NULL` results timestamped by last `wallet_id` filename. Use custom `export.args[['filename']]` with no file extension while declaring. You may notice the legend is different on exported files. The same legend was not scalling well between export to file and plot to interactive device.

## input trades, order_book

The plot function for `trades`, `order_book` do not process the data, it plot the data as is, so it may result poor visibility due to the data itself (ex. `order_book` containing asks with enormously high price). See examples how this can be handled.

## input wallet manager

To be able to track wallet assets value over time user needs to use `archive_write=TRUE` at least twice in wallet manager processing (with non-NA measures). Using the cryptocurrency which do not have any exchange path to `transfer_currency_pair` and/or `value_currency` will result `NA` as `value`. Error on data downloading from external sources (wallets or exchange rates) will also

result NA. Any wallet processing batch which will contain at least one NA measure will be omitted from plot. If you have some crypto not currenctly supported you may extend dictionary for more currencies or provide its value as manual source to wallet_manager already calculated in common value currency, remember to comment out the previous source which returns the NA measure.
To plot wallet manager data load wallet archive data, see examples.
Plotting function will produce dashboard panel to track different measures of your assets and its value. Use mask if you want to share the results to somebody, it will overwrite value with value ratio. Target value currency is taken from the last execution of wallet_manager.

## See Also

market.api.process, wallet_manager

## Examples

```
## Not run:
# plot trades data from kraken's api
trades <- market.api.process('kraken',c('BTC','EUR'),'trades')
Rbitcoin.plot(trades)
Rbitcoin.plot(trades,export=TRUE,col='blue') #export to file, plot trades line in blue

# plot order book data from kraken's api
order_book <- market.api.process('kraken',c('BTC','EUR'),'order_book')
Rbitcoin.plot(order_book)

# plot order book with filtering margins based on order price
order_book <- market.api.process('bitmarket',c('BTC','PLN'),'order_book')
pct <- 0.75
mid <- ((order_book[["asks"]][1,price] + order_book[["bids"]][1,price]) / 2)
order_book[["asks"]] <- order_book[["asks"]][price <= mid * (1+pct)]
order_book[["bids"]] <- order_book[["bids"]][price >= mid * (1-pct)]
Rbitcoin.plot(order_book)

# plot wallet manager data (from local archive) - for details read ?waller_manager
wallet_dt <- wallet_manager(archive_write=F, archive_read=T) #readRDS("wallet_archive.rds")
Rbitcoin.plot(wallet_dt) # plot in R
Rbitcoin.plot(wallet_dt[value>=100 | is.na(value)]) # filter out low value from plot
Rbitcoin.plot(wallet_dt, export=T) # export to svg
# mask value with ratio value and save to png
Rbitcoin.plot(wallet_dt,mask=T,export=T,
              export.args=list(format="png",
                               width = 2*480,
                               height = 2*480,
                               units = "px",
                               pointsize = 18))
# mask value with ratio and mask bitcoin addresses
Rbitcoin.plot(wallet_dt[,.SD][location_type=="blockchain",location := "*address*"],
              mask=T, export=T)

## End(Not run)
```

## `wallet_manager` *Wallet Manager*

### Description

Downloads wallet balance from multiple sources and calculate value in chosen currency based on actual exchange rates. Function is limited to dictionary `api.dict` plus fiat-fiat exchange rates.

### Usage

```
wallet_manager(market.sources = NULL, blockchain.sources = NULL,
  manual.sources = NULL, min_amount = 1e-04, antispam_interval = 10,
  api.dict = NULL, verbose = getOption("Rbitcoin.verbose", 0),
  value_calc = TRUE, value_currency = "USD", value_currency_type = NULL,
  rate_priority, transfer_currency_pair = c(crypto = "BTC", fiat = "USD"),
  archive_write = FALSE, archive_read = FALSE)
```

### Arguments

| | |
|---|---|
| `market.sources` | list of market sources definition, see examples. Mandatory fields: `market, currency_pair, key, secre` (for bitstamp also `client_id`). |
| `blockchain.sources` | list of blockchain sources definition, see examples. Mandatory field: `address`. |
| `manual.sources` | list of manually provided amounts, see examples. Mandatory fields: `currency, amount`, optional field: `location, location_type`. |
| `min_amount` | numeric used to filter out near zero amounts of source currency, default `0.0001`. |
| `antispam_interval` | numeric time in seconds between API calls on one source system, defeault `10s`. |
| `api.dict` | data.table required when using custom API dictionary, read `market.api.process` for details. |
| `verbose` | integer Rbitcoin processing messages, print to console if `verbose > 0`, each subfunction reduce `verbose` by 1. If missing then `getOption("Rbitcoin.verbose",0)` is used, by default `0`. |
| `value_calc` | logical calculate value, by default `TRUE`, can be turned off by setting to `FALSE`. Process will be slightly faster due to no API calls for exchange rates. |
| `value_currency` | character default `"USD"`, target currency in which measure the current value. |
| `value_currency_type` | character, optional for most currencies, if `value_currency` is an exotic currency you need to define its currency type ('crypto' or 'fiat') in this param or update `getOption("Rbitcoin.ct.dict")` param. |
| `rate_priority` | character vector of market and priorioties for sourcing exchange rates, this param needs to be maintained by user, read Exchange rates note below. Example param value `rate_priority = c('bitstamp','kraken','bitmarket','btce')`. |

| | |
|---|---|
| `transfer_currency_pair` | vector length of 2 of named character, default `c(crypto = "BTC", fiat = "USD")`, read Exchange rates note below. |
| `archive_write` | logical, default `FALSE`, recommended `TRUE`. If `TRUE` wallet manager result will be archived to `"wallet_archive.rds"` file in the working directory, read Wallet archive note below. |
| `archive_read` | logical, default `FALSE`, recommended `FALSE`. If `TRUE` it return full archive of wallets data over time grouped by `wallet_id`. To be used when passing results to `Rbitcoin.plot` function or performing other analysis over time, read notes below. |

## Value

data.table object with wallet information in denormilized structure. Number of columns depends on `value_calc` param, when `FALSE` then columns related to the value will not be returned. When launch with `wallet_read=TRUE` then all historical archived wallet statuses will be returned. Field `wallet_id` is a processing batch id and also the timestamp of single wallet manager processing as integer in Unix time format.

## Wallet archive

To be able to track wallet assets value over time user needs to use `archive_write=TRUE`. It will archive wallet manager result `data.table` to `wallet_archive.rds` file in not encrypted format (not a plain text also), sensitive data like amount and value will be available from R by `readRDS("wallet_archive.rds")`. This can be used to correct/manipulate archived data or union the results of the wallet manager performed on different machines by `readRDS(); rbindlist(); saveRDS()`. Setting `archive_write=FALSE` and `archive_read=TRUE` will skip processing and just load the archive, same as `readRDS()`. You should be aware the archive file will be growing over time, unless you have tons of sources defined or you scheduled `wallet_manager` every hour or less you should not experience any issues because of that. In case of the big size of archived rds file you can move data to database, wrap function into database archiver function and query full archive from database only for for plotting.

## Exchange rates

Exchange rates will be downloaded from different sources. Fiat-fiat rates will be sourced from yahoo finance, if yahoo would not be available then also fiat-fiat rate cannot be calculated. Rates for cryptocurrencies will be downloaded from market's tickers according to `rate_priority` and currency pairs available in `api.dict`. Currency type (crypto or fiat) is already defined in `getOption("Rbitcoin.ct.dict")`, can be edited for support other/new currency.
Markets used for crypto rates are defined by `rate_priority` as vector of market names in order of its priority from highest to lowest. User need to chose own trusted exchange rates providers and keep in mind to update `rate_priority` parameter when necessary. As we recently seen the mtgox after death was still spreading the public API data and any system which sources data from them would be affected, so the control over the source for exchange rates needs to be maintained by user. In case of calculation crypto rate for a currency pair which is not available in `api.dict` then `transfer_currency_pair` will be used to get indirect exchange rate. Example: exchange rate for NMC-GBP will be computed as NMC-BTC-USD-GBP using the default `transfer_currency_pair` and current `api.dict`. The process was not exhaustively tested, you can track all the exchange rates used by setting `options(Rbitcoin.archive_exchange_rate=0)`

for saveRDS(), options(Rbitcoin.archive_exchange_rate=1) for write.table(sep=",", dec=".") or options(Rbitcoin.archive_exchange_rate=2) for write.table(sep=";", dec=","). This option will append the data to exchange_rate_archive rds/csv file in working directory.

## NA measures

In case of missing exchange path (direct and indirect through transfer_currency_pair based on api.dict) between the currency in the wallet and the value_currency the NA will be provided to value for that currency. Any errors while downloading wallet data or exchange rates data will also result NA measure. Be sure to avoid NA measures: for unavailable sources you can provide amounts as manual source, for not supported alt cryptocurrencies precalculate its value to supported currency and provide as manual source. While plotting wallet_manager data any wallet batches which contain at least one NA measure will be omitted from plot.

## Schedule wallet tracking

User may consider to schedule execution of the function with archive_write=TRUE for better wallet assets tracking over time. Schedule can be setup on OS by run prepared R script with wallet_manager function execution. In case of scheduling also plot of wallet manager use archive_read=TRUE and add Rbitcoin.plot function execution.

## Troubleshooting

In case of the issues with this function verify if all of the sources are returning correct data, use blockchain.api.process and market.api.process functions. Possible sources for wallet data: market api, blockchain api, manually provided. Possible sources for exchange rate data: market tickers, yahoo (see references). If all sources works and issue still occurs please report. Additionally you can always use verbose argument to print processing informations.

## References

https://code.google.com/p/yahoo-finance-managed/wiki/csvQuotesDownload

## See Also

Rbitcoin.plot, blockchain.api.process, market.api.process, antiddos

## Examples

```
## Not run:
## define source
# define wallets on markets
market.sources <- list(
  list(market = 'bitstamp', currency_pair = c('BTC', 'USD'),
       client_id = '', key = '', secret = ''),
  list(market = 'btce', currency_pair = c('LTC', 'USD'),
       key = '', secret = ''),
  list(market = 'btce', currency_pair = c('LTC', 'USD'),
       key = '', secret = ''), #multiple accounts on same market possible
  list(market = 'kraken', currency_pair = c('BTC', 'EUR'),
       key = '', secret = '')
)
# define wallets on blockchain
blockchain.sources <- list(
  list(address = ''),
  list(address = '')
)
# define wallets manually
manual.sources <- list(
  list(location = 'while transferring',
       currency = c('BTC','LTC'),
       amount = c(0.08, 0)),
  # manually provided value as workaround for bitstamp api unavailability captcha bug
  list(location = 'bitstamp',
       location_type = 'market'
       currency = c('USD','BTC'),
       amount = c(50,0.012))
)

## launch wallet manager with no value calculation
wallet_dt <- wallet_manager(market.sources,
                            blockchain.sources,
                            manual.sources,
                            value_calc = FALSE)
print(wallet_dt)

## launch wallet manager
wallet_dt <- wallet_manager(
  market.sources = market.sources,
  blockchain.sources = blockchain.sources,
  manual.sources = manual.sources,
  value_currency = 'GBP',
  rate_priority = c('bitstamp','kraken','bitmarket','btce')
  archive_write = TRUE
)
print(wallet_dt)

# export to excel/google spreadsheet
setkey(wallet_dt,wallet_id,currency) #sort
write.table(wallet_dt, "clipboard", sep="\t", row.names=FALSE, na = "")
# now go to excel or google spreadsheet and use "paste" from clipboard

# aggregate measures by currency and type
wallet_dt[,list(amount = sum(amount, na.rm=T),
                value = sum(value, na.rm=T)),
           by = c('wallet_id','currency','value_currency')
           ][order(wallet_id,currency,value_currency)]
# aggregate value by location and type
wallet_dt[,list(value = sum(value, na.rm=T)),
           by = c('wallet_id','location_type','location')
           ][order(wallet_id,location_type,location)]

# send to plot
wallet_dt <- wallet_manager(archive_write=F, archive_read=T)
```

```
Rbitcoin.plot(wallet_dt)

# discard processing batch, by id, from wallet archive (will omit on plot)
dt <- readRDS("wallet_archive.rds")
dt[wallet_id==1390000000,`:=`(amount = NA_real_, value = NA_real_)]
saveRDS(dt, "wallet_archive.rds")

# To track exchange rates used set option Rbitcoin.archive_exchange_rate
options(Rbitcoin.archive_exchange_rate=0)
wallet_dt <- wallet_manager(market.sources,
                            blockchain.sources,
                            manual.sources = manual.sources,
                            rate_priority = c('bitstamp','kraken','bitmarket','btce')
                            archive_write = TRUE)
# all exchange rate data as dt
dt <- readRDS("exchange_rate_archive.rds")
# last exchange rate table as dt
dt <- readRDS("exchange_rate_archive.rds")[value_rate_id==max(value_rate_id)]
# save to csv
write.table(dt, "exchange_rate_archive.csv",
            row.names=FALSE,quote=FALSE,append=FALSE,col.names=TRUE,
            sep=";", dec=",")

## End(Not run)
```

# Index