Package 'rKraken'

August 4, 2026

Title 'Kraken API'

Version 1.0.0

Description
The 'Kraken API' <https://docs.kraken.com/api/docs/rest-api/get-server-time> allows clients to access their brokerage accounts, request market data, and place crypto orders.

License GPL-3

Language en-US

Encoding UTF-8

Depends R (>= 4.1.0)

Imports digest, base64enc, jsonlite, httr2, uuid, lubridate

VignetteBuilder knitr

Suggests testthat (>= 3.0.0), knitr, rmarkdown

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Jason Guevara [aut, cre]

Maintainer Jason Guevara <Jason.guevara.yt@gmail.com>

Repository CRAN

Date/Publication 2026-08-04 09:40:08 UTC

Contents

.get_kraken_signature 3
.get_nonce 4
.kraken_request 4
.rk_env 5
.rk_read_tokens 6
rk_acc_xfer 6
rk_add_order 7
rk_add_order_batch 10

rk_alloc_earn_funds . . . 12
rk_amend_order . . . 13
rk_build_orders . . . 15
rk_cancel_all_orders . . . 17
rk_cancel_all_orders_after . . . 18
rk_cancel_order . . . 19
rk_cancel_order_batch . . . 20
rk_cancel_withdrawal . . . 20
rk_create_sub_acc . . . 21
rk_dealloc_earn_funds . . . 22
rk_delete_export_rpt . . . 23
rk_download_data_export . . . 23
rk_edit_order . . . 24
rk_get_account_balance . . . 26
rk_get_alloc_status . . . 27
rk_get_api_info . . . 28
rk_get_assets . . . 28
rk_get_asset_pairs . . . 29
rk_get_closed_orders . . . 30
rk_get_credit_lines . . . 31
rk_get_dealloc_status . . . 32
rk_get_deposit_address . . . 33
rk_get_deposit_methods . . . 34
rk_get_deposit_status . . . 35
rk_get_earn_strats . . . 36
rk_get_export_status . . . 37
rk_get_extended_balance . . . 38
rk_get_grouped_order_book . . . 39
rk_get_level3_order_book . . . 40
rk_get_ohlc_data . . . 40
rk_get_open_margin_pos . . . 41
rk_get_open_orders . . . 42
rk_get_order_amends . . . 43
rk_get_order_book . . . 44
rk_get_order_id . . . 45
rk_get_recent_spreads . . . 45
rk_get_recent_trades . . . 46
rk_get_server_time . . . 47
rk_get_system_status . . . 47
rk_get_ticker_info . . . 48
rk_get_trades_hist . . . 48
rk_get_trade_balance . . . 50
rk_get_trade_volume . . . 50
rk_get_withdrawal_addresses . . . 51
rk_get_withdrawal_information . . . 52
rk_get_withdrawal_methods . . . 53
rk_ledgers_info . . . 54
rk_list_earn_alloc . . . 55

rk_post_trade_data 57
rk_pre_trade_data 57
rk_query_ledgers 58
rk_query_orders 59
rk_query_trades_info 60
rk_req_export_rpt 61
rk_req_wallet_xfer 62
rk_withdraw 63
rk_withdrawal_status 64

Index **66**

`.get_kraken_signature` *Kraken Signature (Internal)*

Description

Kraken Signature (Internal)

Usage

```
.get_kraken_signature(path, data = "", nonce = "", secret)
```

Arguments

`path`	= endpoint to use
`data`	= data httr2::request
`nonce`	= nonce value
`secret`	= api secret

Value

Encoding For API httr2::requests

Examples

```
## Not run:
  # For Internal Use
  signature <- .get_kraken_signature(path = path,
                                     data = paste0(query_str, body_str),
                                     nonce = nonce,
                                     secret = api_secret
                                     )

## End(Not run)
```

.get_nonce *Get Nonce (Internal)*

Description

Get Nonce (Internal)

Usage

```
.get_nonce()
```

Value

Create Nonce to Pass Into httr2::requests

Examples

```
## Not run:
  # For Internal Use
  nonce <- .get_nonce()

## End(Not run)
```

.kraken_request *Create API httr2::request (Internal)*

Description

Create API httr2::request (Internal)

Usage

```
.kraken_request(
  method = "GET",
  path = "",
  query = NULL,
  body = NULL,
  api_key = "",
  api_secret = "",
  base_url = "https://api.kraken.com"
)
```

Arguments

`method`	= httr2::request method
`path`	= endpoint
`query`	= query httr2::request
`body`	= body httr2::request
`api_key`	= kraken api key
`api_secret`	= kraken api secret
`base_url`	= kraken base url

Value

Helps create the API httr2::request

Examples

```
## Not run:
  # For Internal Use
  req <- .kraken_request(method='GET')

## End(Not run)
```

.rk_env *temporary working environment*

Description

temporary working environment

Usage

```
.rk_env
```

Value

An auto generated `environment` to store our tokens

Examples

```
## Not run:
  .rk_env <- new.env(parent = emptyenv())

## End(Not run)
```

.rk_read_tokens *httr2::request token file (Internal)*

Description

httr2::request token file (Internal)

Usage

```
.rk_read_tokens()
```

Value

httr2::requests your token file 'rk_tokens.rds' from working directory & assigns a working `environment` if it exists

Examples

```
## Not run:
  # For Internal Use (assigns tokens inside of the 'rp' environment)
  .rk_read_tokens()

## End(Not run)
```

rk_acc_xfer *Account Transfer*

Description

Transfer funds to and from master and subaccounts. Note: AccountTransfer must be called using an API key from the master account.

Usage

```
rk_acc_xfer(.asset, asset_class = "currency", .amount, .from, .to)
```

Arguments

`.asset`	= Asset being transferred
`asset_class`	= Specify the asset class of the asset being transferred. Possible values: `currency`, `tokenized_asset` Default value: currency
`.amount`	= Amount of asset to transfer
`.from`	= Public account ID of the source account (Example ABCD 1234 EFGH 5678)
`.to`	= Public account ID of the destination account (Example ABCD 1234 EFGH 5678)

Value

Returns API response as a `list` of funds transferred between accounts.

Examples

```
## Not run:
 # XFER XTZ
 rk_resp = rk_acc_xfer(.asset = 'XTZ', .amount = "0.000001",
 .from = "AA08 N84G FVOT ZYAA", .to = "AA02 N84G AHGV XD4A")

## End(Not run)
```

`rk_add_order` *Add Order*

Description

Note: See the AssetPairs endpoint for details on the available trading pairs, their price and quantity precisions, order minimums, available leverage, etc. API Key Permissions Required: Orders and trades - Create & modify orders

Usage

```
rk_add_order(
  userref = NULL,
  cl_ord_id = NULL,
  ordertype.,
  type.,
  volume.,
  displayvol = NULL,
  pair.,
  asset_class = NULL,
  price = NULL,
  price2 = NULL,
  trigger = NULL,
  leverage = NULL,
  reduce_only = NULL,
  stptype = NULL,
  oflags = NULL,
  timeinforce = NULL,
  starttm = NULL,
  expiretm = NULL,
  close_ordertype = NULL,
  close_price = NULL,
  close_price2 = NULL,
  deadline = NULL,
  validate = NULL,
  broker = NULL
)
```

Arguments

userref	= This is an optional non-unique, numeric identifier which can associated with a number of orders by the client. T This field is mutually exclusive with cl_ord_id parameter. userref is an optional user-specified integer id that can be associated with any number of orders. Many clients choose a userref corresponding to a unique integer id generated by their systems (e.g. a timestamp). However, because we don't enforce uniqueness on our side, it can also be used to easily group orders by pair, side, strategy, etc. This allows clients to more readily cancel or query information about orders in a particular group, with fewer API calls by using userref instead of our txid, where supported.
cl_ord_id	= Adds an alphanumeric client order identifier which uniquely identifies an open order for each client. This field is mutually exclusive with userref parameter. The cl_ord_id parameter can be one of the following formats: • Long UUID: 6d1b345e-2821-40e2-ad83-4ecb18a06876 32 hex characters separated with 4 dashes. • Short UUID: da8e4ad59b78481c93e589746b0cf91f 32 hex characters with no dashes. • Free text: arb-20240509-00010 Free format ascii text up to 18 characters.
ordertype.	= The execution model of the order. Possible values: `market`, `limit`, `iceberg`, `stop-loss`, `take-profit`, `stop-loss-limit`, `take-profit-limit`, `trailing-stop`, `trailing-stop-limit`, `settle-position`
type.	= Order direction (buy/sell). Possible values: `buy`, `sell`
volume.	= Order quantity in terms of the base asset
displayvol	= For iceberg orders only, it defines the quantity to show in the book while the rest of order quantity remains hidden. Minimum value is 1 / 15 of volume.
pair.	= Asset pair id or altname. Example: XBTUSD
asset_class	= This parameter is required on httr2::requests for non-crypto pairs, i.e. use tokenized_asset for xstocks. Possible values: `tokenized_asset`
price	= Price: • Limit price for limit and iceberg orders • Trigger price for stop-loss, stop-loss-limit, take-profit, take-profit-limit, trailing-stop and trailing-stop-limit orders Notes: • Relative Prices: Either price or price2 can be preceded by +, -, or # to specify the order price as an offset relative to the last traded price. + adds the amount to, and - subtracts the amount from the last traded price. # will either add or subtract the amount to the last traded price, depending on the direction and order type used. Prices can also be suffixed with a % to signify the relative amount as a percentage, rather than an absolute price difference. • Trailing Stops: Must use a relative price for this field, namely the + prefix, from which the direction will be automatic based on if the original order is a buy or sell (no need to use - or #). The % suffix also works for these order types to use a relative percentage price. Example: 40000.0
price2	= Secondary Price: • Limit price for stop-loss-limit, take-profit-limit and trailing-stop-limit orders Note: • Trailing Stops: Must use a relative price for this field, namely one of the + or - prefixes. This will provide the offset from the trigger price to the limit price, i.e. +0 would set the limit price equal to the trigger price. The % suffix also works for this field to use a relative percentage limit price.

`trigger` = Price signal used to trigger stop-loss, stop-loss-limit, take-profit, take-profit-limit, trailing-stop and trailing-stop-limit orders Notes: • This trigger type will also be used for any associated conditional close orders. • To keep triggers serviceable, the last price will be used as fallback reference price during connectivity issues with external index feeds. Possible values: `index`, `last` Default value: last

`leverage` = Amount of leverage desired (default: none). Example: 5

`reduce_only` = If true, order will only reduce a currently open position, not increase it or open a new position. Default value: false

`stptype` = Self Trade Prevention (STP) is a protection feature to prevent users from inadvertently or deliberately trading against themselves. To prevent a self-match, one of the following STP modes can be used to define which order(s) will be expired: • cancel-newest: arriving order will be canceled • cancel-oldest: resting order will be canceled • cancel-both: both arriving and resting orders will be canceled Possible values: `cancel-newest`, `cancel-oldest`, `cancel-both` Default value: cancel-newest

`oflags` = Comma delimited list of order flags • post post-only order (available when ordertype = limit) • fcib prefer fee in base currency (default if selling) • fciq prefer fee in quote currency (default if buying, mutually exclusive with fcib) • nompp (DEPRECATED) — disabling Market Price Protection for market orders is no longer supported. If supplied, the flag is accepted but ignored. • viqc order volume expressed in quote currency. This option is supported only for buy market orders. Also not available on margin orders. Example: post

`timeinforce` = Time-in-force of the order to specify how long it should remain in the order book before being cancelled. GTC (Good-'til-cancelled) is default if the parameter is omitted. IOC (immediate-or-cancel) will immediately execute the amount possible and cancel any remaining balance rather than resting in the book. FOK (fill-or-kill) will execute the order in full immediately or cancel it entirely without any partial fill. GTD (good-'til-date), if specified, must coincide with a desired expiretm. Possible values: `GTC`, `IOC`, `FOK`, `GTD` Default value: GTC

`starttm` = Scheduled start time, can be specified as an absolute timestamp or as a number of seconds in the future: 0 now (default) unix timestamp of start time '+' = schedule start time x seconds from now Note that URL encoding of the + character changes it to a space, so please use %2b followed by the number of seconds instead of +

`expiretm` = Expiry time on GTD orders can be set up to one month in future, it is specified as an absolute timestamp or as a number of seconds from now: 0 no expiration (default) unix timestamp of expiration time '+' = expire x seconds from now, minimum 5 seconds Note that URL encoding of the + character changes it to a space, so please use %2b followed by the number of seconds instead of +

`close_ordertype` = Conditional close order type Note: Conditional close orders are triggered by execution of the primary order in the same quantity and opposite direction, but once triggered are independent orders that may reduce or increase net position Possible values: `limit`, `iceberg`, `stop-loss`, `take-profit`, `stop-loss-limit`, `take-profit-limit`, `trailing-stop`, `trailing-stop-limit`

close_price	= Conditional close order price. Example: 50000.0
close_price2	= Conditional close order price2
deadline	= RFC3339 timestamp (e.g. 2021-04-01T00:18:45Z) after which the matching engine should reject the new order httr2::request, in presence of latency or order queueing: min now() + 2 seconds, max now() + 60 seconds.
validate	= If set to true the order will be validated only, it will not trade in the matching engine. Default value: false
broker	= Broker IIBAN (Partner's Kraken IIBAN)

Value

Returns `list` of newly placed order

Examples

```
## Not run:
 # Place single-limit order - (to place live order change or remove the validate param)
 ord_id = rk_get_order_id()
 ord1 = rk_add_order(cl_ord_id = ord_id, ordertype. = "limit",
  type. = 'buy', volume. = '0.00005',
  pair. = 'BTC/USD', price = '5000', validate = 'true')

 # get open order
 oo = rk_get_open_orders(cl_ord_id = ord_id)

 # take-profit-limit order
 ord_id = rk_get_order_id()
   ord2 = rk_add_order(cl_ord_id = ord_id, ordertype. = "take-profit-limit",
    type. = 'buy', volume. = '0.00005',
    pair. = 'BTC/USD', price = '50000', price2 = '100000',
    validate = 'true')

 # place limit order -100 points from market price
  ord_id = rk_get_order_id()
  tbl = rk_add_order(cl_ord_id = ord_id, ordertype. = "limit", type. = 'buy',
                     volume. = '0.00005', pair. = 'BTC/USD', price = '-100',
                     validate='true')


## End(Not run)
```

`rk_add_order_batch` *Add Order Batch*

Description

Sends a collection of orders (minimum of 2 and maximum 15):

Validation is performed on the whole batch prior to submission to the engine. If an order fails validation, the whole batch will be rejected. On submission to the engine, if an order fails pre-match checks (i.e. funding), then the individual order will be rejected and remainder of the batch will be processed. All orders in batch are limited to a single pair. Note: See the AssetPairs endpoint for details on the available trading pairs, their price and quantity precisions, order minimums, available leverage, etc.

API Key Permissions Required: Orders and trades - Create & modify orders and Orders and trades - Cancel & close orders

Usage

```
rk_add_order_batch(
  orders_list,
  pair,
  asset_class = NULL,
  deadline = NULL,
  validate = "false",
  broker = NULL
)
```

Arguments

`orders_list`	= array (use rk_build_orders)
`pair`	= Asset pair id or altname
`asset_class`	= This parameter is required on httr2::requests for non-crypto pairs, i.e. use tokenized_asset for xstocks. Possible values: `tokenized_asset`
`deadline`	= RFC3339 timestamp (e.g. 2021-04-01T00:18:45Z) after which the matching engine should reject the new order httr2::request, in presence of latency or order queueing. min now() + 2 seconds, max now() + 60 seconds.
`validate`	= Validate inputs only. Do not submit order. Default value: false
`broker`	= Broker IIBAN (Partner's Kraken IIBAN)

Value

Returns API response as a `list` of all batch orders

Examples

```
## Not run:
# create 4 different orders to be placed simultaneously
# 1st order is a market buy order
# 2nd/3rd order is a limit buy order
# 4th order is a take-profit-limit order
ord1 = rk_get_order_id()
ord2 = rk_get_order_id()
```

```
ord3 = rk_get_order_id()
ord4 = rk_get_order_id()
all_ords <- rk_build_orders(
  cl_ord_id  = c(ord1, ord2, ord3, ord4),
  ordertype. = c("market", "limit", "limit", "take-profit-limit"),
  type.      = c("buy","buy","buy", "sell"),
  volume.    = c("0.00005","0.00005","0.00005","0.00010"),
  price      = c(NA, '50000', '60000',"100000"),
  trigger    = c(NA, NA, NA,'last'),
  price2     = c(NA, NA, NA, '99950'),
  starttm    = c('+10','0','0','0'),
  expiretm   = c(NA, "+60", as.integer(Sys.time()+minutes(5)), "0")
)

 # add batch order to Kraken
resp <- rk_add_order_batch(
  orders   = all_ords,
  pair     = "BTC/USD",
  validate = 'true'
)



# Creates 2 limit orders to buy 0.00005 BTC at -10 & -50 dollars from current market price
# adds condition to place 1st order 10 seconds from now and the 2nd at +20 seconds
ord1 = rk_get_order_id()
ord2 = rk_get_order_id()
all_ords <- rk_build_orders(
  cl_ord_id  = c(ord1, ord2),
  ordertype. = c("limit", "limit"),
  type.      = c("buy","buy"),
  volume.    = c("0.00005","0.00005"),
  price      = c('-10', '-50'),
  starttm    = c('+10','+20')
)

# add batch order to Kraken
resp <- rk_add_order_batch(
  orders   = all_ords,
  pair     = "BTC/USD",
  validate = 'false'
)

## End(Not run)
```

`rk_alloc_earn_funds` *Allocate Earn Funds*

Description

Allocate funds to the Strategy.

Requires the Earn Funds API key permission. The amount must always be defined.

This method is asynchronous. A couple of preflight checks are performed synchronously on behalf of the method before it is dispatched further. The client is required to poll the result using the /0/private/Earn/AllocateStatus endpoint.

There can be only one (de)allocation httr2::request in progress for given user and strategy at any time. While the operation is in progress:

pending attribute in /Earn/Allocations response for the strategy indicates that funds are being allocated, pending attribute in /Earn/AllocateStatus response will be true. Following specific errors within Earnings class can be returned by this method:

Minimum allocation: EEarnings:Below min:(De)allocation operation amount less than minimum Allocation in progress: EEarnings:Busy:Another (de)allocation for the same strategy is in progress Service temporarily unavailable: EEarnings:Busy. Try again in a few minutes. User tier verification: EEarnings:Permission denied:The user's tier is not high enough Strategy not found: EGeneral:Invalid arguments:Invalid strategy ID

Usage

```
rk_alloc_earn_funds(.amount, .strategy_id)
```

Arguments

`.amount`	= The amount to allocate.
`.strategy_id`	= A unique identifier of the chosen earn strategy, as returned from /0/private/Earn/Strategies.

Value

Returns API response as a `list` of allocating funds to a strategy

Examples

```
## Not run:
 # Allocate Funds to Strategy
 rk_resp = rk_alloc_earn_funds(.amount = "0.000001", .strategy_id = "ESZ4QWD-E7FCW-7PBERA")

## End(Not run)
```

`rk_amend_order` *Amend Order*

Description

The amend httr2::request enables clients to modify the order parameters in-place without the need to cancel the existing order and create a new one. The order identifiers assigned by Kraken and/or client will stay the same. Queue priority in the order book will be maintained where possible. If an amend httr2::request will reduce the order quantity below the existing filled quantity, the remaining quantity will be cancelled. For more detail, see amend transaction guide. API Key Permissions Required: Orders and trades - Create & modify orders or Orders and trades - Cancel & close orders

Usage

```
rk_amend_order(
  txid = NULL,
  cl_ord_id = NULL,
  order_qty,
  display_qty = NULL,
  limit_price = NULL,
  trigger_price = NULL,
  pair,
  post_only = "false",
  deadline = NULL
)
```

Arguments

`txid`	= The Kraken identifier for the order to be amended. Either txid or cl_ord_id is required.
`cl_ord_id`	= The client identifier for the order to be amended. Either txid or cl_ord_id is required.
`order_qty`	= The new order quantity in terms of the base asset.
`display_qty`	= For iceberg orders only, it defines the new quantity to show in the book while the rest of order quantity remains hidden. Minimum value is 1 / 15 of remaining order quantity.
`limit_price`	= The new limit price restriction on the order (for order types that support limit price only). The relative pricing can be set by using the +, - prefixes and/or % suffix. '+' adds the amount from the reference price, i.e. market rises 50 USD "+50". '-' subtracts the amount from the reference price, i.e. market drops 100 USD "-100".
`trigger_price`	= The new trigger price to activate the order (for triggered order types only). The relative pricing can be set by using the +, - prefixes and/or % suffix. '+' adds the amount from the reference price, i.e. market rises 50 USD "+50". '-' subtracts the amount from the reference price, i.e. market drops 100 USD "-100".
`pair`	= The pair is required on amends for non-crypto pairs, i.e. provide the pair symbol for xstocks.
`post_only`	= An optional flag for limit_price amends. If true, the limit price change will be rejected if the order cannot be posted passively in the book. Default value: false
`deadline`	= RFC3339 timestamp (e.g. 2021-04-01T00:18:45Z) after which the matching engine should reject the new order httr2::request, in presence of latency or order queueing. min now() + 2 seconds, max now() + 60 seconds.

Value

Returns `list` of newly amended order

Examples

```
## Not run:
 # Modify order to limit Market price - 100 points
 resp = rk_amend_order(cl_ord_id = ord_id, pair = "BTC/USD",
  limit_price = "-100", order_qty = '0.00005')


## End(Not run)
```

`rk_build_orders` *Build Order Batch*

Description

Template to build batch orders.

Usage

```
rk_build_orders(
  userref = NULL,
  cl_ord_id = NULL,
  ordertype.,
  type.,
  volume.,
  displayvol = NULL,
  price = NULL,
  price2 = NULL,
  trigger = NULL,
  leverage = NULL,
  reduce_only = "false",
  stptype = "cancel-newest",
  oflags = NULL,
  timeinforce = "GTC",
  starttm = NULL,
  expiretm = NULL
)
```

Arguments

`userref`	= User reference id userref is an optional user-specified integer id that can be associated with any number of orders. Many clients choose a userref corresponding to a unique integer id generated by their systems (e.g. a timestamp). However, because we don't enforce uniqueness on our side, it can also be used to easily group orders by pair, side, strategy, etc. This allows clients to more readily cancel or query information about orders in a particular group, with fewer API calls by using userref instead of our txid, where supported.

`cl_ord_id` = Adds an alphanumeric client order identifier which uniquely identifies an open order for each client. This field is mutually exclusive with userref parameter.

`ordertype.` = Order type Possible values: `market`, `limit`, `iceberg`, `stop-loss`, `take-profit`, `stop-loss-limit`, `take-profit-limit`, `trailing-stop`, `trailing-stop-limit`, `settle-position`

`type.` = Order direction (buy/sell). Possible values: `buy`, `sell`

`volume.` = Order quantity in terms of the base asset Note: Volume can be specified as 0 for closing margin orders to automatically fill the requisite quantity.

`displayvol` = For iceberg orders only, it defines the quantity to show in the book while the rest of order quantity remains hidden. Minimum value is 1 / 15 of volume.

`price` = Price: Limit price for limit and iceberg orders Trigger price for stop-loss, stop-loss-limit, take-profit, and take-profit-limit orders Notes: Relative Prices: Either `price` or `price2` can be preceded by `+`, `-`, or `#` to specify the order price as an offset relative to the last traded price. `+` adds the amount to, and `-` subtracts the amount from the last traded price. `#` will either add or subtract the amount to the last traded price, depending on the direction and order type used. Prices can also be suffixed with a % to signify the relative amount as a percentage, rather than an absolute price difference.

`price2` = Secondary Price: Limit price for stop-loss-limit and take-profit-limit

`trigger` = Price signal used to trigger stop-loss, stop-loss-limit, take-profit, and take-profit-limit orders Notes: To keep triggers serviceable, the last price will be used as fallback reference price during connectivity issues with external index feeds. Possible values: `index`, `last` Default value: last

`leverage` = Amount of leverage desired (default: none)

`reduce_only` = If true, order will only reduce a currently open position, not increase it or open a new position. Default value: false

`stptype` = Self trade prevention behaviour definition: cancel-newest - if self trade is triggered, arriving order will be canceled cancel-oldest - if self trade is triggered, resting order will be canceled cancel-both - if self trade is triggered, both arriving and resting orders will be canceled Possible values: `cancel-newest`, `cancel-oldest`, `cancel-both` Default value: cancel-newest

`oflags` = oflags (string) Comma delimited list of order flags • post post-only order (available when ordertype = limit) • fcib prefer fee in base currency (default if selling) • fciq prefer fee in quote currency (default if buying, mutually exclusive with fcib) • nompp (DEPRECATED) — disabling Market Price Protection for market orders is no longer supported. If supplied, the flag is accepted but ignored. • viqc order volume expressed in quote currency. This option is supported only for buy market orders. Also not available on margin orders. Example: post

`timeinforce` = Time-in-force of the order to specify how long it should remain in the order book before being cancelled. GTC (Good-'til-cancelled) is default if the parameter is omitted. IOC (immediate-or-cancel) will immediately execute the amount possible and cancel any remaining balance rather than resting in the book. FOK (fill-or-kill) will execute the order in full immediately or cancel it entirely without any partial fill. GTD (good-'til-date), if specified, must coincide with a desired expiretm. Possible values: `GTC`, `IOC`, `FOK`, `GTD` Default value: GTC

starttm = Scheduled start time, can be specified as an absolute timestamp or as a number of seconds in the future: 0 now (default)

- schedule start time seconds from now unix timestamp of start time

expiretm = Expiry time on GTD orders can be set up to one month in future, it is specified as an absolute timestamp or as a number of seconds from now: 0 no expiration (default)

- = expire seconds from now, minimum 5 seconds unix timestamp of expiration time

Value

Returns `list` of all batch orders

Examples

```
## Not run:
 # Creates 3 orders: 1 market order, 2 limit orders all to buy 0.00005 BTC
  rk_build_orders(
                ordertype. = c("market", "limit", "limit"),
                type.      = c("buy", "buy", "buy"),
                volume.    = c("0.00005", "0.00005", "0.00005"),
                price      = c(NA, "50000", "60000")
                )


## End(Not run)
```

`rk_cancel_all_orders` *Cancel All Orders*

Description

Cancel all open orders API Key Permissions Required: Orders and trades - Create & modify orders or Orders and trades - Cancel & close orders

Usage

```
rk_cancel_all_orders()
```

Value

Returns response `list` of all canceled orders

Examples

```
## Not run:
 # Cancel all open orders
 resp = rk_cancel_all_orders()


## End(Not run)
```

`rk_cancel_all_orders_after` *Cancel All Orders After X*

Description

CancelAllOrdersAfter provides a "Dead Man's Switch" mechanism to protect the client from network malfunction, extreme latency or unexpected matching engine downtime. The client can send a httr2::request with a timeout (in seconds), that will start a countdown timer which will cancel all client orders when the timer expires. The client has to keep sending new httr2::requests to push back the trigger time, or deactivate the mechanism by specifying a timeout of 0. If the timer expires, all orders are cancelled and then the timer remains disabled until the client provides a new (non-zero) timeout.

The recommended use is to make a call every 15 to 30 seconds, providing a timeout of 60 seconds. This allows the client to keep the orders in place in case of a brief disconnection or transient delay, while keeping them safe in case of a network breakdown. It is also recommended to disable the timer ahead of regularly scheduled trading engine maintenance (if the timer is enabled, all orders will be cancelled when the trading engine comes back from downtime - planned or otherwise). API Key Permissions Required: Orders and trades - Create & modify orders or Orders and trades - Cancel & close orders

Usage

```
rk_cancel_all_orders_after(timeout)
```

Arguments

`timeout`	= Duration (in seconds) to set/extend the timer, it should be less than 86400 seconds.

Value

Returns response `list` of all canceled orders

Examples

```
## Not run:
 # Cancel all open orders
 resp = rk_cancel_all_orders_after(timeout = 60)


## End(Not run)
```

`rk_cancel_order` *Cancel Order*

Description

Cancel a particular open order (or set of open orders) by txid, userref or cl_ord_id API Key Permissions Required: Orders and trades - Create & modify orders or Orders and trades - Cancel & close orders

Usage

```
rk_cancel_order(txid = NULL, cl_ord_id = NULL)
```

Arguments

`txid`	= Kraken order identifier (txid) or user reference (userref)
`cl_ord_id`	= An alphanumeric client order identifier which uniquely identifies an open order for each client.

Value

Returns response `list` of canceled order

Examples

```
## Not run:
 # Cancel order using cl_ord_id
 resp = rk_cancel_order(cl_ord_id = ord_id)


## End(Not run)
```

rk_cancel_order_batch *Cancel Order Batch*

Description

Cancel multiple open orders by txid, userref or cl_ord_id(maximum 50 total unique IDs/references) API Key Permissions Required: Orders and trades - Create & modify orders or Orders and trades - Cancel & close orders

Usage

```
rk_cancel_order_batch(orders_list = NULL, cl_ord_ids = NULL)
```

Arguments

orders_list	= Open order transaction IDs (txid) or user references (userref), up to a maximum of 50 total unique IDs/references.
cl_ord_ids	= An alphanumeric client order identifier which uniquely identifies an open order for each client. Up to a maximum of 50 total unique IDs/references.

Value

Returns API response as a `list` of all canceled batch orders

Examples

```
## Not run:
 # Cancel batch orders
   cl_ord_ids = list(ord1, ord2)
   rk_cancel_order_batch(orders_list = cl_ord_ids)


## End(Not run)
```

rk_cancel_withdrawal *httr2::request Withdrawal Cancellation*

Description

Cancel a recently httr2::requested withdrawal, if it has not already been successfully processed. API Key Permissions Required: Funds permissions - Withdraw, unless withdrawal is a WalletTransfer, then no permissions are required.

Usage

```
rk_cancel_withdrawal(.asset, .refid)
```

Arguments

.asset	= Asset being withdrawn
.refid	= Withdrawal reference ID

Value

Returns API response as a list of withdrawal cancellation

Examples

```
## Not run:
 # Cancel Withdrawal
 rk_resp = rk_cancel_withdrawal(.asset = "BTC", .refid = 'FTjQv5a-beekqjvUCUEAFKc5mHbboW')

## End(Not run)
```

---

rk_create_sub_acc *httr2::request Wallet Transfer*

Description

Create a trading subaccount. Note: CreateSubaccount must be called using an API key from the master account. API Key Permissions Required: Funds permissions - Withdraw

Usage

```
rk_create_sub_acc(.username, .email)
```

Arguments

.username	= Username for the subaccount
.email	= Email address for the subaccount

Value

Returns API response as a list of whether subaccount creation was successful or not.

Examples

```
## Not run:
 # Don't Run if sub account not needed
 rk_resp = rk_create_sub_acc(.username = 'bravo', .email = 'bravo@example.com')

## End(Not run)
```

`rk_dealloc_earn_funds` *Deallocate Earn Funds*

Description

Deallocate funds from a strategy.

Requires the Earn Funds API key permission. The amount must always be defined.

This method is asynchronous. A couple of preflight checks are performed synchronously on behalf of the method before it is dispatched further. If the method returns HTTP 202 code, the client is required to poll the result using the /Earn/DeallocateStatus endpoint.

There can be only one (de)allocation httr2::request in progress for given user and strategy. While the operation is in progress:

pending attribute in Allocations response for the strategy will hold the amount that is being deallocated (negative amount) pending attribute in DeallocateStatus response will be true. Following specific errors within Earnings class can be returned by this method:

Minimum allocation: EEarnings:Below min:(De)allocation operation amount less than minimum allowed Allocation in progress: EEarnings:Busy:Another (de)allocation for the same strategy is in progress Strategy not found: EGeneral:Invalid arguments:Invalid strategy ID

Usage

```
rk_dealloc_earn_funds(.amount, .strategy_id)
```

Arguments

`.amount`	= The amount to deallocate. This field is required.
`.strategy_id`	= A unique identifier per earn strategy.

Value

Returns API response as a `list` of deallocating funds to a strategy

Examples

```
## Not run:
 # Allocate Funds to Strategy
 rk_resp = rk_dealloc_earn_funds(.amount = "0.0001", .strategy_id = "ESZ4QWD-E7FCW-7PBERA")

## End(Not run)
```

rk_delete_export_rpt *Delete Export Report*

Description

Delete exported trades/ledgers report API Key Permissions Required: Data - Export data

Usage

```
rk_delete_export_rpt(.id, .type)
```

Arguments

.id	= ID of report to delete or cancel
.type	= delete can only be used for reports that have already been processed. Use cancel for queued or processing reports. Possible values: cancel, delete

Value

deletes data export httr2::request

Examples

```
## Not run:
 # Download Zip folder using the report ID
rk_resp = rk_delete_export_rpt(.id = "CTER", .type = "delete")

## End(Not run)
```

rk_download_data_export *Retrieve Data Export*

Description

Retrieve a processed data export API Key Permissions Required: Data - Export data

Usage

```
rk_download_data_export(.id, output_path = NULL)
```

Arguments

.id	= Report ID to retrieve
output_path	= path to save file

Value

Downloads ZIP folder into working directory

Examples

```
## Not run:
 # Download Zip folder using the report ID
 rk_download_data_export(.id = "CTER")

## End(Not run)
```

`rk_edit_order` *Edit Order*

Description

Sends a httr2::request to edit the order parameters of a live order. When an order has been successfully modified, the original order will be cancelled and a new order will be created with the adjusted parameters a new txid will be returned in the response. API Key Permissions Required: Orders and trades - Create & modify orders and Orders and trades - Cancel & close orders

Usage

```
rk_edit_order(
  userref = "",
  txid,
  volume = "",
  displayvol = "",
  pair = "",
  asset_class = "",
  price = "",
  price2 = "",
  oflags = "",
  deadline = "",
  cancel_response = "",
  validate = ""
)
```

Arguments

`userref`	= User reference id. userref is an optional user-specified integer id associated with edit httr2::request. Note: userref from parent order will not be retained on the new order after edit.
`txid`	= Original Order ID or User Reference Id (userref) which is user-specified integer id used with the original order. If userref is not unique and was used with multiple order, edit httr2::request is denied with an error.
`volume`	= Order quantity in terms of the base asset.

`displayvol` = For iceberg orders only, it defines the quantity to show in the book while the rest of order quantity remains hidden. Minimum value is 1 / 15 of volume.

`pair` = Asset pair id or altname

`asset_class` = This parameter is required on httr2::requests for non-crypto pairs, i.e. use tokenized_asset for xstocks. Possible values: `tokenized_asset`

`price` = Price: Limit price for limit and iceberg orders Trigger price for stop-loss, stop-loss-limit, take-profit, take-profit-limit, trailing-stop and trailing-stop-limit orders Notes: Relative Prices: Either `price` or `price2` can be preceded by +, -, or # to specify the order price as an offset relative to the last traded price. + adds the amount to, and - subtracts the amount from the last traded price. # will either add or subtract the amount to the last traded price, depending on the direction and order type used. Prices can also be suffixed with a % to signify the relative amount as a percentage, rather than an absolute price difference. Trailing Stops: Must use a relative price for this field, namely the + prefix, from which the direction will be automatic based on if the original order is a buy or sell (no need to use - or #). The % suffix also works for these order types to use a relative percentage price.

`price2` = Secondary Price: Limit price for stop-loss-limit, take-profit-limit and trailing-stop-limit orders Note: Trailing Stops: Must use a relative price for this field, namely one of the + or - prefixes. This will provide the offset from the trigger price to the limit price, i.e. +0 would set the limit price equal to the trigger price. The % suffix also works for this field to use a relative percentage limit price.

`oflags` = Comma delimited list of order flags. Only these flags can be changed:

- post post-only order (available when ordertype = limit). All the flags from the parent order are retained except post-only. post-only needs to be explicitly mentioned on edit httr2::request.

`deadline` = RFC3339 timestamp (e.g. 2021-04-01T00:18:45Z) after which the matching engine should reject the new order httr2::request, in presence of latency or order queueing. min now() + 2 seconds, max now() + 60 seconds.

`cancel_response` = Used to interpret if client wants to receive pending replace, before the order is completely replaced

`validate` = Validate inputs only. Do not submit order. Default value: false

Value

Returns API response as a `list` of all edited orders

Examples

```
## Not run:
 # Modify Order
   rk_edit_order(txid = "XXXXXX-XXXXX-HEDSA", volume = '0.00005', pair = "BTC/USD")


## End(Not run)
```

rk_get_account_balance *Get Account Balance*

Description

Retrieve all cash balances, net of pending withdrawals. Note on Staking/Earn assets: We have begun to migrate assets from our legacy Staking system over to a new Earn system. As such, the following assets may appear in your balances and ledger. Please see our Support article for more details. Note that these assets are "read-only", to interact with your balances in them please use the base asset (e.g. USDT to transact with your USDT and USDT.F balances). Symbol Extensions:

.B: balances in new yield-bearing products, similar to .S (staked) and .M (opt-in rewards) balances .F: balances earning automatically in Kraken Rewards .T: tokenized assets. API Key Permissions Required: Funds permissions - Query

Usage

```
rk_get_account_balance(rebase_multiplier = "rebased")
```

Arguments

rebase_multiplier	= Optional parameter for viewing xstocks data. rebased: Display in terms of underlying equity. base: Display in terms of SPV tokens. Possible values: `rebased`, `base` Default value: rebased

Value

Retrieve all cash balances, net of pending withdrawals as a `data.frame`

Examples

```
## Not run:
  # No params required
  rk_tbl1 = rk_get_account_balance()

## End(Not run)
```

`rk_get_alloc_status` *Get Allocation Status*

Description

Get the status of the last allocation httr2::request.

Requires either the Earn Funds or Query Funds API key permission.

(De)allocation operations are asynchronous and this endpoint allows client to retrieve the status of the last dispatched operation. There can be only one (de)allocation httr2::request in progress for given user and strategy.

The pending attribute in the response indicates if the previously dispatched operation is still in progress (true) or has successfully completed (false). If the dispatched httr2::request failed with an error, then HTTP error is returned to the client as if it belonged to the original httr2::request.

Following specific errors within Earnings class can be returned by this method:

Insufficient funds: EEarnings:Insufficient funds:Insufficient funds to complete the (de)allocation httr2::request User cap exceeded: EEarnings:Above max:The allocation exceeds user limit for the strategy Total cap exceeded: EEarnings:Above max:The allocation exceeds the total strategy limit Minimum allocation: EEarnings:Below min:(De)allocation operation amount less than minimum

Usage

```
rk_get_alloc_status(.strategy_id)
```

Arguments

`.strategy_id`	= A unique identifier per earn strategy.

Value

Returns API response as a `list` of allocation status

Examples

```
## Not run:
 # Get Earn Allocation Status
 rk_resp = rk_get_alloc_status(.strategy_id = "ESZ4QWD-E7FCW-7PBERA")

## End(Not run)
```

rk_get_api_info *Get API Key Info*

Description

Retrieve information about the API key that is used to make the httr2::request, including its name, permissions, restrictions, and usage timestamps. API Key Permissions Required: None

Usage

```
rk_get_api_info(otp = NULL)
```

Arguments

otp	= Two-factor authentication password (required only if 2FA is configured for the API key)

Value

Returns list of API key details

Examples

```
## Not run:
 # List API Key info
 rk_resp = rk_get_api_info()

## End(Not run)
```

rk_get_assets *Get Kraken List of Assets*

Description

Get Kraken List of Assets

Usage

```
rk_get_assets(asset = "", aclass = "currency")
```

Arguments

asset	= Comma delimited list of assets to get info on (optional, default all available assets) Example: XBT,ETH
aclass	= Possible values: currency, tokenized_asset Filters the asset class to retrieve (optional) currency = spot currency pairs. tokenized_asset = xstocks. Default value: currency

Value

Returns Kraken List of Assets as `list`

Examples

```
## Not run:
  # Get multiple Crypto Assets
  rk_assets = rk_get_assets(c("BTC","ETH","SOL"))
  # Get Tokenized Assets
  rk_assets2= rk_get_assets(c("NVDAx","AAPLx","MSTRx"), aclass = "tokenized_asset")

## End(Not run)
```

`rk_get_asset_pairs` *Get Tradeable Asset Pairs*

Description

Get Tradeable Asset Pairs

Usage

```
rk_get_asset_pairs(
  pair,
  aclass_base = "currency",
  info = "info",
  country_code = "US",
  execution_venue = "international"
)
```

Arguments

`pair`	= Asset pairs to get data for Example: BTC/USD,ETH/BTC
`aclass_base`	= Possible values: `currency`, `tokenized_asset` Filters the asset class to retrieve (optional) currency = spot currency pairs. tokenized_asset = tokenized asset pairs, i.e. xstocks. Default value: currency
`info`	= Possible values: `info`, `leverage`, `fees`, `margin` Info to retrieve (optional) info = all info leverage = leverage info fees = fees schedule margin = margin info Default value: info
`country_code`	= Filter for response to only include pairs available in the provided country/region. Example: GB
`execution_venue`	= Possible values: `international`, `bitnomial_exchange` Comma-separated list of execution venues to filter by (optional) international = International exchange bitnomial_exchange = Bitnomial exchange Default value: international

Value

Returns Kraken `list` of Tradeable Assets

Examples

```
## Not run:
  # Get multiple Crypto Assets
  rk_trade_assets = rk_get_asset_pairs(pair=c("ETH/USD", "BTC/USD"))

## End(Not run)
```

`rk_get_closed_orders` *Get Closed Orders*

Description

Retrieve information about orders that have been closed (filled or cancelled). 50 results are returned at a time, the most recent by default.

Note: If an order's tx ID is given for start or end time, the order's opening time (opentm) is used API Key Permissions Required: Orders and trades - Query closed orders & trades

Usage

```
rk_get_closed_orders(
  trades = NULL,
  userref = NULL,
  cl_ord_id = NULL,
  start = NULL,
  end = NULL,
  ofs = NULL,
  closetime = NULL,
  consolidate_taker = NULL,
  without_count = NULL,
  rebase_multiplier = NULL
)
```

Arguments

`trades`	= Whether or not to include trades related to position in output . Default value: false
`userref`	= Restrict results to given user reference
`cl_ord_id`	= Restrict results to given client order id
`start`	= Starting unix timestamp or order tx ID of results (exclusive)
`end`	= Ending unix timestamp or order tx ID of results (inclusive)
`ofs`	= Result offset for pagination

`closetime` = Which time to use to search Possible values: `open`, `close`, `both` Default value: both

`consolidate_taker` = Whether or not to consolidate trades by individual taker trades

`without_count` = Whether or not to include page count in result (true is much faster for users with many closed orders) Default value: false

`rebase_multiplier` = Optional parameter for viewing xstocks data. rebased: Display in terms of underlying equity. base: Display in terms of SPV tokens. Possible values: `rebased`, `base` Default value: rebased

Value

Returns a `list` of closed orders

Examples

```
## Not run:
  # get closed orders from 1st
  tbl = rk_get_closed_orders(ofs = 1)
  df = as.data.frame(do.call(rbind,tbl$result$closed))

  # get closed orders with specific time
 tbl2 = rk_get_closed_orders(start = as.numeric(Sys.time()-days(10)), end=as.numeric(Sys.time()))
  df2 = as.data.frame(do.call(rbind,tbl2$result$closed))

## End(Not run)
```

---

rk_get_credit_lines *Get Credit Lines*

Description

Retrieve all credit line details for VIPs with this functionality. API Key Permissions Required: Funds permissions - Query

Usage

```
rk_get_credit_lines(rebase_multiplier = "rebased")
```

Arguments

`rebase_multiplier` = Optional parameter for viewing xstocks data. rebased: Display in terms of underlying equity. base: Display in terms of SPV tokens. Possible values: `rebased`, `base` Default value: rebased

Value

Get credit lines as a `data.frame`

Examples

```
## Not run:
  # No params required
  rk_tbl1 = rk_get_credit_lines()

## End(Not run)
```

`rk_get_dealloc_status` *Get Deallocation Status*

Description

Get the status of the last deallocation httr2::request.

Requires either the Earn Funds or Query Funds API key permission.

(De)allocation operations are asynchronous and this endpoint allows client to retrieve the status of the last dispatched operation. There can be only one (de)allocation httr2::request in progress for given user and strategy.

The pending attribute in the response indicates if the previously dispatched operation is still in progress (true) or has successfully completed (false). If the dispatched httr2::request failed with an error, then HTTP error is returned to the client as if it belonged to the original httr2::request.

Following specific errors within Earnings class can be returned by this method:

Insufficient funds: EEarnings:Insufficient funds:Insufficient funds to complete the (de)allocation httr2::request Minimum allocation: EEarnings:Below min:(De)allocation operation amount less than minimum

Usage

```
rk_get_dealloc_status(.strategy_id)
```

Arguments

`.strategy_id` = A unique identifier per earn strategy.

Value

Returns API response as a `list` of deallocation status

Examples

```
## Not run:
 # Get Earn Allocation Status
 rk_resp = rk_get_dealloc_status(.strategy_id = "ESZ4QWD-E7FCW-7PBERA")

## End(Not run)
```

rk_get_deposit_address *Get Deposit Addresses*

Description

Retrieve (or generate a new) deposit addresses for a particular asset and method. API Key Permissions Required: Funds permissions - Query

Usage

```
rk_get_deposit_address(
  .asset,
  aclass = "currency",
  .method,
  new = "false",
  amount = NULL
)
```

Arguments

.asset	= Asset being deposited
aclass	= Asset class being deposited. Possible values: currency, tokenized_asset Default value: currency
.method	= Name of the deposit method
new	= Whether or not to generate a new address. Default value: false
amount	= Amount you wish to deposit (only required for method=Bitcoin Lightning) oneOf: string, integer, number

Value

Returns API response as a list of deposit address

Examples

```
## Not run:
 # Get deposit address
 rk_dep = rk_get_deposit_address(.asset = "USD", .method = "Plaid US")

## End(Not run)
```

rk_get_deposit_methods *Get Deposit Methods*

Description

Retrieve methods available for depositing a particular asset. API Key Permissions Required: Funds permissions - Query and Funds permissions - Deposit

Usage

```
rk_get_deposit_methods(
  .asset,
  aclass = "currency",
  rebase_multiplier = "rebased"
)
```

Arguments

`.asset`	= Asset being deposited
`aclass`	= Asset class being deposited. Possible values: `currency`, `tokenized_asset`. Default value: currency
`rebase_multiplier`	= Optional parameter for viewing xstocks data. rebased: Display in terms of underlying equity. base: Display in terms of SPV tokens. Possible values: `rebased`, `base` Default value: rebased

Value

Returns API response as a `list` of deposit methods

Examples

```
## Not run:
 # Get deposit limits for USD
 rk_resp = rk_get_deposit_methods(.asset = "USD")

## End(Not run)
```

`rk_get_deposit_status` *Get Status of Recent Deposits*

Description

Retrieve information about recent deposits. Results are sorted by recency, use the cursor parameter to iterate through list of deposits (page size equal to value of limit) from newest to oldest. API Key Permissions Required: Funds permissions - Query

Usage

```
rk_get_deposit_status(
  asset,
  aclass = "currency",
  method = NULL,
  start = NULL,
  end = NULL,
  cursor = NULL,
  limit = 25,
  rebase_multiplier = "rebased"
)
```

Arguments

`asset`	= Filter for specific asset being deposited
`aclass`	= Filter for specific asset class being deposited. Possible values: `currency`, `tokenized_asset`. Default value: currency
`method`	= Filter for specific name of deposit method
`start`	= Start timestamp, deposits created strictly before will not be included in the response
`end`	= End timestamp, deposits created strictly after will be not be included in the response
`cursor`	= true/false to enable/disable paginated response (boolean) or cursor for next page of results (string) anyOf `MOD1`, `MOD2`
`limit`	= Number of results to include per page. Default value: 25
`rebase_multiplier`	= Optional parameter for viewing xstocks data. rebased: Display in terms of underlying equity. base: Display in terms of SPV tokens. Possible values: `rebased`, `base` Default value: rebased

Value

Returns API response as a `list` of deposit status

Examples

```
## Not run:
 # Get deposit status for USD
 rk_resp= rk_get_deposit_status(asset = "USD", method = NULL)

## End(Not run)
```

`rk_get_earn_strats` *List Earn Strategies*

Description

List earn strategies along with their parameters.

Requires a valid API key but not specific permission is required.

Returns only strategies that are available to the user based on geographic region.

When the user does not meet the tier restriction, can_allocate will be false and allocation_restriction_info indicates Tier as the restriction reason. Earn products generally require Intermediate tier. Get your account verified to access earn.

A note about lock_type: instant: can be deallocated without an unbonding period. This is called flexible in the UI. bonded: has an unbonding period. Deallocation will not happen until this period has passed. flex: "Kraken rewards". This is earning on your spot balances where eligible. It's turned on account wide from the UI and you cannot manually allocate to these strategies. Paging isn't yet implemented, so the endpoint always returns all data in the first page.

Usage

```
rk_get_earn_strats(
  ascending = NULL,
  asset = NULL,
  cursor = NULL,
  limit = NULL,
  lock_type = NULL
)
```

Arguments

`ascending`	= true to sort ascending, false (the default) for descending.
`asset`	= Filter strategies by asset name
`cursor`	= None to start at beginning/end, otherwise next page ID
`limit`	= How many items to return per page. Note that the limit may be cap'd to lower value in the application code.
`lock_type`	= Filter strategies by lock type. Possible values: `flex`, `bonded`, `timed`, `instant`

Value

Returns API response as a `list` of earn strategies

Examples

```
## Not run:
 # list earn strategies
 rk_resp = rk_get_earn_strats()
 # View 1st strategy
 do.call(rbind, rk_resp$result$items[[1]]) |> t
 # combine all
 data.table::rbindlist((lapply(as.list(1:length(rk_resp$result$items)),
 function(i) do.call(rbind, rk_resp$result$items[[i]]) |> t |> data.frame)),
  use.names = T, fill = T)

## End(Not run)
```

---

rk_get_export_status *Get Export Report Status*

Description

Get status of httr2::requested data exports. API Key Permissions Required: Data - Export data

Usage

```
rk_get_export_status(.report)
```

Arguments

`.report`	= Type of data to export. Possible values: `trades`, `ledgers`

Value

Returns a status `list` for recent export reports

Examples

```
## Not run:
 # Get Export Report Status
 rk_resp = rk_get_export_status(.report = 'trades')
 do.call(rbind, rk_resp$result)

## End(Not run)
```

`rk_get_extended_balance` *Get Extended Balance*

Description

Retrieve all extended account balances, including credits and held amounts. Balance available for trading is calculated as: available balance = balance + credit - credit_used - hold_trade Note that held amounts only include spot non margin orders. Note on Staking/Earn assets: We have begun to migrate assets from our legacy Staking system over to a new Earn system. As such, the following assets may appear in your balances and ledger. Please see our Support article for more details. Note that these assets are "read-only", to interact with your balances in them please use the base asset (e.g. USDT to transact with your USDT and USDT.F balances).

Symbol Extensions:

.B: balances in new yield-bearing products, similar to .S (staked) and .M (opt-in rewards) balances .F: balances earning automatically in Kraken Rewards .T: tokenized assets. API Key Permissions Required: Funds permissions - Query

Usage

```
rk_get_extended_balance(rebase_multiplier = "rebased")
```

Arguments

`rebase_multiplier`
: = Optional parameter for viewing xstocks data. rebased: Display in terms of underlying equity. base: Display in terms of SPV tokens. Possible values: `rebased`, `base` Default value: rebased

Value

Get Extended Balances as a `data.frame`

Examples

```
## Not run:
  # No params required
  rk_tbl1 = rk_get_extended_balance()

## End(Not run)
```

rk_get_grouped_order_book *Get Grouped Order Book*

Description

The GroupedBook endpoint aggregates the volume in the order book over a specified tick range. It provides a summary of liquidity deep into the book, useful for user interface display. Bids and asks between grouped price levels are accumulated to the nearest passive level (asks rounded up, bids down).

Usage

```
rk_get_grouped_order_book(pair, depth = 10, grouping = 1)
```

Arguments

pair	= Asset pair to get order book for. Example: XBTUSD
depth	= Number of price levels to return per side (bids/asks). Use 0 to return the full book. Possible values: 10, 25, 100, 250, 1000 Default value: 10
grouping	= Specifies how many tick levels should be within each price level. Bids and asks between grouped price levels are accumulated to the nearest passive level (asks rounded up, bids down). Possible values: 1, 5, 10, 25, 50, 100, 250, 500, 1000 Default value: 1

Value

Retrieve grouped order book as a data.frame

Examples

```
## Not run:
  # Get Grouped order book for BTC
  rk_book_grp2 = rk_get_grouped_order_book(pair = "BTC/USD", depth = 25)

  # Get Grouped Order Book for ETH
  rk_book_grp2 = rk_get_grouped_order_book(pair = "ETH/USD", depth = 10)

## End(Not run)
```

rk_get_level3_order_book *Get Kraken level-3 data*

Description

Retrieve Level3 order book data, which provides individual order information at each price level. This includes order IDs and timestamps for each order in the book. The Level3 endpoint requires authentication.

Usage

```
rk_get_level3_order_book(pair, depth = 100)
```

Arguments

pair	= Asset pair to get order book for. Example: XBTUSD
depth	= Number of price levels to return per side (bids/asks). Use 0 to return the full book. Possible values: 0, 10, 25, 100, 250, 1000 Default value: 100

Value

Retrieve full order book info for the selected depth & return a data.frame

Examples

```
## Not run:
  # get the depth for the top-10 only
  rk_lvl3 = rk_get_level3_order_book(pair = "ETH/USD", depth = 10)

  # Get ALL the level-3 data for BTC
  rk_lvl3 = rk_get_level3_order_book(pair = "BTC/USD", depth = 0)

## End(Not run)
```

rk_get_ohlc_data *Get OHLC Data*

Description

Get OHLC Data

Usage

```
rk_get_ohlc_data(pair, interval = 1, since = NULL, asset_class = "")
```

Arguments

pair	= Asset pair to get data for. Example: XBTUSD
interval	= Possible values: 1, 5, 15, 30, 60, 240, 1440, 10080, 21600 Time frame interval in minutes Default value: 1
since	= Return OHLC entries since the given timestamp (intended for incremental updates). Example: 1688671200
asset_class	= Possible values: tokenized_asset. This parameter is required on httr2::requests for non-crypto pairs, i.e. use tokenized_asset for xstocks.

Value

Retrieve OHLC market data & return as a data.frame

Examples

```
## Not run:
  # Gets 1-HR OHLCV Bars for BTC/USD Since Start Time
  rk_ohlcv1 = rk_get_ohlc_data(pair = "XBTUSD",
  interval = 60, since = as.numeric(as.POSIXct("2026-05-01 01:00:00")))

  # Retrive 15-MIN Bars for ETH/USD
  rk_ohlcv2 = rk_get_ohlc_data(pair = "ETH/USD", interval = 15)

  # Tokenized Asset Bars Not Available in US?
  rk_ohlcv3 = get_ohlc_data(pair = "AAPLx", interval = 60,
   asset_class = 'tokenized_asset')

  # Convert to XTS if needed
  require("xts")
  XTS = xts(rk_ohlcv2[,c(2:ncol(rk_ohlcv2))],
  order.by = as.POSIXct(rk_ohlcv2$time, tz = "UTC"))
  index(XTS) = with_tz(index(XTS), tzone = "America/Los_Angeles")

## End(Not run)
```

---

rk_get_open_margin_pos

Get Open Positions

Description

Get information about open margin positions. API Key Permissions Required: Orders and trades - Query open orders & trades

Usage

```
rk_get_open_margin_pos(
  txid = NULL,
  docalcs = "false",
  consolidation = "market",
  rebase_multiplier = "rebased"
)
```

Arguments

`txid`	= Comma delimited list of transaction IDs to query info about (20 maximum)
`docalcs`	= Whether to include P&L calculations. Default value: false
`consolidation`	= Consolidate positions by market/pair. Possible values: `market`
`rebase_multiplier`	= Optional parameter for viewing xstocks data. rebased: Display in terms of underlying equity. base: Display in terms of SPV tokens. Possible values: `rebased`, `base` Default value: rebased

Value

Returns open positions as a `list`

Examples

```
## Not run:
  # Query Order by passing in ID
 rk_resp = rk_get_open_margin_pos(txid = 'XXXXX-XXXXX-2PNSFD')

## End(Not run)
```

---

rk_get_open_orders *Get Open Orders*

Description

Retrieve information about currently open orders. API Key Permissions Required: Orders and trades - Query open orders & trades

Usage

```
rk_get_open_orders(
  trades = "false",
  userref = NULL,
  cl_ord_id = NULL,
  rebase_multiplier = ""
)
```

Arguments

trades	= Whether or not to include trades related to position in output Default value: false
userref	= Restrict results to given user reference
cl_ord_id	= Restrict results to given client order id
rebase_multiplier	= Optional parameter for viewing xstocks data. rebased: Display in terms of underlying equity. base: Display in terms of SPV tokens. Possible values: rebased, base Default value: rebased

Value

Returns a list of open orders

Examples

```
## Not run:
  # Get open orders
  rk_tbl1 = rk_get_open_orders()

## End(Not run)
```

---

rk_get_order_amends *Get Order Amends*

Description

Retrieves an audit trail of amend transactions on the specified order. The list is ordered by ascending amend timestamp. API Key Permissions Required: Orders and trades - Query open orders & trades or Orders and trades Query closed orders & trades, depending on status of order.

Usage

```
rk_get_order_amends(order_id, rebase_multiplier = NULL)
```

Arguments

order_id	= The Kraken order identifier for the amended order.
rebase_multiplier	= Optional parameter for viewing xstocks data. rebased: Display in terms of underlying equity. base: Display in terms of SPV tokens. Possible values: rebased, base Default value: rebased

Value

Returns a list of order httr2::requested

Examples

```
## Not run:
  # Query Order by passing in the order_id
  tbl = rk_get_order_amends(order_id = 'OHFIO4-XXXXX-XXXXX')

## End(Not run)
```

`rk_get_order_book` *Get Kraken Order Book*

Description

Get Kraken Order Book

Usage

```
rk_get_order_book(pair, count = 100, asset_class = "")
```

Arguments

`pair`	= Asset pair to get data for. Example: XBTUSD
`count`	= Possible values: >= 1 and <= 500
`asset_class`	= Possible values: `tokenized_asset`. This parameter is required on httr2::requests for non-crypto pairs, i.e. use tokenized_asset for xstocks.

Value

Retrieve order book info for asset and return as a `data.frame`

Examples

```
## Not run:
  # Gets Info for BTC/USD
  rk_order_book = rk_get_order_book(pair = "BTC/USD", count = 5)

## End(Not run)
```

rk_get_order_id *Get Kraken Order ID*

Description

Get Kraken Order ID

Usage

```
rk_get_order_id()
```

Value

Generates UUID Order ID

Examples

```
## Not run:
  # For Internal Use
  my_id <- rk_get_order_id()

## End(Not run)
```

rk_get_recent_spreads *Get Recent Spreads*

Description

Get Recent Spreads

Usage

```
rk_get_recent_spreads(pair, since = NULL, asset_class = "")
```

Arguments

pair	= Asset pair to get data for. Example: XBTUSD
since	= Returns spread data since given timestamp. Optional, intended for incremental updates within available dataset (does not contain all historical spreads). Example: 1678219570
asset_class	= Possible values: tokenized_asset This parameter is required on httr2::requests for non-crypto pairs, i.e. use tokenized_asset for xstocks.

Value

Returns the last ~200 top-of-book spreads for a given pair as a data.frame

Examples

```
## Not run:
  # Get recent spreads for BTC
  rk_tbl1 = rk_get_recent_spreads(pair = "BTC/USD")

  # Get Recent Spreads for ETH from specific time
 rk_tbl2 = rk_get_recent_spreads(pair = "ETH/USD", since = as.numeric(Sys.time()-minutes(5)))

## End(Not run)
```

`rk_get_recent_trades` *Get Recent Trades*

Description

Get Recent Trades

Usage

```
rk_get_recent_trades(
  pair,
  since = as.numeric(as.POSIXct(Sys.time() - lubridate::hours(1))),
  count = 1000,
  asset_class = ""
)
```

Arguments

`pair`	= Asset pair to get data for. Example: XBTUSD
`since`	= Return trade data since given timestamp. Example: 1616663618
`count`	= Possible values: >= 1 and <= 1000. Return specific number of trades, up to 1000 Default value: 1000
`asset_class`	= Possible values: `tokenized_asset` This parameter is required on httr2::requests for non-crypto pairs, i.e. use tokenized_asset for xstocks.

Value

Returns the last 1000 trades by default as a `data.frame`

Examples

```
## Not run:
  # Get Recent trades for BTC
  rk_recent_tr = rk_get_recent_trades(pair = "BTC/USD")

  # Get Recent trades for ETH
  rk_recent_tr = rk_get_recent_trades(pair = "ETH/USD", count = 10)

## End(Not run)
```

rk_get_server_time *Get Kraken Server Time*

Description

Get Kraken Server Time

Usage

```
rk_get_server_time()
```

Value

Returns `list` of Kraken Server Time

Examples

```
## Not run:
  # No parameters
  rk_time <- rk_get_server_time()

## End(Not run)
```

rk_get_system_status *Get Kraken System Status*

Description

Get Kraken System Status

Usage

```
rk_get_system_status()
```

Value

Returns Kraken System Status as a `list`

Examples

```
## Not run:
  # No parameters
  rk_status <- rk_get_system_status()

## End(Not run)
```

rk_get_ticker_info *Get Kraken Ticker Info*

Description

Get Kraken Ticker Info

Usage

```
rk_get_ticker_info(pair, asset_class = "forex")
```

Arguments

pair	= Asset pair to get data for (optional, default: all tradeable exchange pairs) Example: XBTUSD
asset_class	= Possible values: tokenized_asset, forex This parameter is required on httr2::requests for tokenized pairs, i.e. xstocks. If asset_class is provided without the pair parameter, all pairs for that asset class will be returned. Default value: forex

Value

Returns a list pertaining to the symbol httr2::requested

Examples

```
## Not run:
  # Gets Info for BTC/USD
  rk_asset_info = rk_get_ticker_info(pair="XBTUSD")

## End(Not run)
```

rk_get_trades_hist *Get Trades History*

Description

Retrieve information about trades/fills. 50 results are returned at a time, the most recent by default. Unless otherwise stated, costs, fees, prices, and volumes are specified with the precision for the asset pair (pair_decimals and lot_decimals), not the individual assets' precision (decimals). API Key Permissions Required: Orders and trades - Query closed orders & trades

Usage

```
rk_get_trades_hist(
  type = "all",
  trades = NULL,
  start = NULL,
  end = NULL,
  ofs = NULL,
  without_count = NULL,
  consolidate_taker = NULL,
  ledgers = NULL,
  rebase_multiplier = "rebased"
)
```

Arguments

type	= Type of trade. Possible values: all, any position, closed position, closing position, no position Default value: all
trades	= Whether or not to include trades related to position in output. Default value: false
start	= Starting unix timestamp or trade tx ID of results (exclusive)
end	= Ending unix timestamp or trade tx ID of results (inclusive)
ofs	= Result offset for pagination
without_count	= if true, does not retrieve count of ledger entries. httr2::request can be noticeably faster for users with many ledger entries as this avoids an extra database query. Default value: false
consolidate_taker	= Whether or not to consolidate trades by individual taker trades. Default value: true
ledgers	= Whether or not to include related ledger ids for given trade. **Note that setting this to true will slow httr2::request performance Default value: false
rebase_multiplier	= Optional parameter for viewing xstocks data. rebased: Display in terms of underlying equity. base: Display in terms of SPV tokens. Possible values: rebased, base Default value: rebased

Value

Returns a list of trade history

Examples

```
## Not run:
  # Query Order by passing in the order_id
 tbl = rk_get_trades_hist(start = as.numeric(Sys.time()-days(30)),
                          end = as.numeric(Sys.time()),
                          ledgers = 'true')
```

```
 df = as.data.frame(do.call(rbind, tbl[["result"]][["trades"]]))

## End(Not run)
```

`rk_get_trade_balance` *Get Trade Balance*

Description

Retrieve a summary of collateral balances, margin position valuations, equity and margin level. API Key Permissions Required: Orders and trades - Query open orders & trades

Usage

```
rk_get_trade_balance(asset = "ZUSD", rebase_multiplier = "rebased")
```

Arguments

`asset`	= Base asset used to determine balance Default value: ZUSD
`rebase_multiplier`	= Optional parameter for viewing xstocks data. rebased: Display in terms of underlying equity. base: Display in terms of SPV tokens. Possible values: `rebased`, `base` Default value: rebased

Value

Returns Account Balances as a `data.frame`

Examples

```
## Not run:
  # get account trade balance
  rk_tbl1 = rk_get_trade_balance(asset = "USD")

## End(Not run)
```

`rk_get_trade_volume` *Get Trade Volume*

Description

Returns 30 day USD trading volume and resulting fee schedule for any asset pair(s) provided. Fees will not be included if pair is not specified as Kraken fees differ by asset pair. Note: If an asset pair is on a maker/taker fee schedule, the taker side is given in fees and maker side in fees_maker. For pairs not on maker/taker, they will only be given in fees.

API Key Permissions Required: Funds permissions - Query

Usage

```
rk_get_trade_volume(pair = NULL, rebase_multiplier = "rebased")
```

Arguments

pair	= Comma delimited list of asset pairs to get fee info on (optional, but required if any fee info is desired)
rebase_multiplier	= Optional parameter for viewing xstocks data. rebased: Display in terms of underlying equity. base: Display in terms of SPV tokens. Possible values: rebased, base Default value: rebased

Value

Returns account volume info as a list

Examples

```
## Not run:
 # Get Trade Volume for BTC
 rk_resp = rk_get_trade_volume(pair = 'BTC/USD')

## End(Not run)
```

---

rk_get_withdrawal_addresses

Get Withdrawal Addresses

Description

Retrieve a list of withdrawal addresses available for the user. API Key Permissions Required: Funds permissions - Query and Funds permissions - Withdraw

Usage

```
rk_get_withdrawal_addresses(
  asset,
  aclass = "currency",
  method = NULL,
  key = NULL,
  verified = NULL
)
```

Arguments

asset	= Filter addresses for specific asset
aclass	= Filter addresses for specific asset class. Possible values: currency, tokenized_asset. Default value: currency
method	= Filter addresses for specific method
key	= Find address for by withdrawal key name, as set up on your account
verified	= Filter by verification status of the withdrawal address. Withdrawal addresses successfully completing email confirmation will have a verification status of true.

Value

Returns API response as a list of withdrawal addresses

Examples

```
## Not run:
 # Get withdrawal addresses
 rk_resp= rk_get_withdrawal_addresses(asset = "USD")

## End(Not run)
```

---

rk_get_withdrawal_information

Get Withdrawal Information

Description

Retrieve fee information about potential withdrawals for a particular asset, key and amount. API Key Permissions Required: Funds permissions - Query and Funds permissions - Withdraw

Usage

```
rk_get_withdrawal_information(.asset, .key, .amount)
```

Arguments

.asset	= Asset being withdrawn
.key	= Withdrawal key name, as set up on your account
.amount	= Amount to be withdrawn

Value

Returns API response as a list of withdrawal information

Examples

```
## Not run:
 # Get withdrawal information
 rk_resp= rk_get_withdrawal_information(.asset = "USD", .key = 'my_key', .amount = '1.50')

## End(Not run)
```

`rk_get_withdrawal_methods` *Get Withdrawal Methods*

Description

Retrieve a list of withdrawal methods available for the user. API Key Permissions Required: Funds permissions - Query and Funds permissions - Withdraw

Usage

```
rk_get_withdrawal_methods(
  asset,
  aclass = "currency",
  network = NULL,
  rebase_multiplier = "rebased"
)
```

Arguments

`asset`	= Filter methods for specific asset
`aclass`	= Filter methods for specific asset class. Possible values: `currency`, `tokenized_asset`. Default value: currency
`network`	= Filter methods for specific network
`rebase_multiplier`	= Optional parameter for viewing xstocks data. rebased: Display in terms of underlying equity. base: Display in terms of SPV tokens. Possible values: `rebased`, `base` Default value: rebased

Value

Returns API response as a `list` of withdrawal methods

Examples

```
## Not run:
 # Get withdrawal methods
 rk_resp= rk_get_withdrawal_methods(asset = "USD")

## End(Not run)
```

`rk_ledgers_info` *Get Ledgers Info*

Description

Retrieve information about ledger entries. 50 results are returned at a time, the most recent by default. Note on Staking/Earn assets: We have begun to migrate assets from our legacy Staking system over to a new Earn system. As such, the following assets may appear in your balances and ledger. Please see our Support article for more details. Note that these assets are "read-only", to interact with your balances in them please use the base asset (e.g. USDT to transact with your USDT and USDT.F balances).

.B, which represents balances in new yield-bearing products, similar to .S (staked) and .M (opt-in rewards) balances .F, which represents balances earning automatically in Kraken Rewards API Key Permissions Required: Data - Query ledger entries

Usage

```
rk_ledgers_info(
  asset = "all",
  aclass = "currency",
  type = "all",
  start = NULL,
  end = NULL,
  ofs = NULL,
  without_count = NULL,
  rebase_multiplier = "rebased"
)
```

Arguments

`asset`	= Filter output by asset or comma delimited list of assets. Default value: all
`aclass`	= Filter output by asset class. Default value: currency
`type`	= Type of ledger to retrieve. Possible values: `all`, `trade`, `deposit`, `withdrawal`, `transfer`, `margin`, `adjustment`, `rollover`, `credit`, `settled`, `staking`, `dividend`, `sale`, `nft_rebate` Default value: all
`start`	= Starting unix timestamp or ledger ID of results (exclusive)
`end`	= Ending unix timestamp or ledger ID of results (inclusive)
`ofs`	= Result offset for pagination
`without_count`	= If true, does not retrieve count of ledger entries. httr2::request can be noticeably faster for users with many ledger entries as this avoids an extra database query. Default value: false
`rebase_multiplier`	= Optional parameter for viewing xstocks data. rebased: Display in terms of underlying equity. base: Display in terms of SPV tokens. Possible values: `rebased`, `base` Default value: rebased

Value

Returns ledger info as a `list`

Examples

```
## Not run:
   # simple httr2::request for first 50 transactions
   rk_resp = rk_ledgers_info(type = 'all')
   as.data.frame(do.call(rbind, rk_resp$result$ledger))


  # Assign in increments of 50 since that is the limit for multiple transactions
  nums = c(0, 50, 100, 150, 200)

  # loop to get all pages
  all_pgs = lapply(as.list(nums), function(ii){
            tbl2 = rk_ledgers_info(type = 'all', ofs = ii)
            df2 = as.data.frame(as.data.frame(data.table::rbindlist(tbl2$result$ledger,
            use.names = T, fill = T)))
            df2$time <- force_tz(as.POSIXct(as.numeric(df2$time), "UTC"),
             tzone = "America/Los_Angeles")
            df2
            })

 # combine results
 all_pgs = as.data.frame(do.call(rbind,all_pgs))

 # remove any duplicates
 all_pgs = all_pgs[!duplicated(all_pgs),]
 # write out as CSV
 tmp_file <- file.path(tempdir(), "kraken.csv")
 write.table(all_pgs, tmp_file, sep=",", row.names = FALSE)


## End(Not run)
```

`rk_list_earn_alloc` *List Earn Allocations*

Description

List all allocations for the user.

Requires the Query Funds API key permission.

By default all allocations are returned, even for strategies that have been used in the past and have zero balance now. That is so that the user can see how much was earned with given strategy in the past. hide_zero_allocations parameter can be used to remove zero balance entries from the output. Paging hasn't been implemented for this method as we don't expect the result for a particular user to be overwhelmingly large.

All amounts in the output can be denominated in a currency of user's choice (the converted_asset parameter).

Information about when the next reward will be paid to the client is also provided in the output.

Allocated funds can be in up to 4 states:

1. bonding
2. allocated
3. exit_queue (ETH only)
4. unbonding

Any funds in total not in bonding/unbonding are simply allocated and earning rewards. Depending on the strategy funds in the other 3 states can also be earning rewards. Consult the output of /Earn/Strategies to know whether bonding/unbonding earn rewards. ETH in exit_queue still earns rewards.

Note that for ETH, when the funds are in the exit_queue state, the expires time given is the time when the funds will have finished unbonding, not when they go from exit queue to unbonding.

(Un)bonding time estimate can be inaccurate right after having (de)allocated the funds. Wait 1-2 minutes after (de)allocating to get an accurate result.

Usage

```
rk_list_earn_alloc(
  ascending = NULL,
  converted_asset = NULL,
  hide_zero_allocations = NULL
)
```

Arguments

`ascending`	= true to sort ascending, false (the default) for descending.
`converted_asset`	= A secondary currency to express the value of your allocations (the default is USD).
`hide_zero_allocations`	= Omit entries for strategies that were used in the past but now they don't hold any allocation (the default is false)

Value

Returns API response as a `list` of earn allocations

Examples

```
## Not run:
 # List earn allocations
 rk_resp = rk_list_earn_alloc()


## End(Not run)
```

rk_post_trade_data *Post-Trade Data*

Description

Returns a list of trades on the spot exchange. If no filter parameters are specified, the last 1000 trades for all pairs are received.

Usage

```
rk_post_trade_data(symbol, from_ts, to_ts, count = 1000)
```

Arguments

symbol	= Filter the results to the currency pair.
from_ts	= Filter the results to include the trades after this timestamp.
to_ts	= Filter the results to include the trades before or at this timestamp.
count	= possible values: >= 1 and <= 1000. Default value: 1000

Value

Returns API response as a `list` of the latest spot trades

Examples

```
## Not run:
 # Make httr2::request for BTC
 rk_resp = rk_post_trade_data(symbol = 'BTC/USD',
 from_ts = '2026-05-23T00:34:56.123456789Z',
 to_ts = '2026-05-23T00:39:56.123456789Z')


## End(Not run)
```

rk_pre_trade_data *Pre-Trade Data*

Description

Returns the price levels in the order book with aggregated order quantities at each price level. The top 10 levels are returned for each trading pair.

Usage

```
rk_pre_trade_data(symbol)
```

Arguments

symbol	= Possible values: >= 3 characters and <= 32 characters. A list of symbols for the currency pairs. Example: BTC/USD

Value

Returns API response as a `list` of the top price levels of the aggregated order book.

Examples

```
## Not run:
 # Make httr2::request for BTC
 rk_resp = rk_pre_trade_data(symbol = "BTC/USD")


## End(Not run)
```

---

rk_query_ledgers *Query Ledgers*

Description

Retrieve information about specific ledger entries. Note on Staking/Earn assets: We have begun to migrate assets from our legacy Staking system over to a new Earn system. As such, the following assets may appear in your balances and ledger. Please see our Support article for more details. Note that these assets are "read-only", to interact with your balances in them please use the base asset (e.g. USDT to transact with your USDT and USDT.F balances).

.B, which represents balances in new yield-bearing products, similar to .S (staked) and .M (opt-in rewards) balances .F, which represents balances earning automatically in Kraken Rewards API Key Permissions Required: Data - Query ledger entries

Usage

```
rk_query_ledgers(
  id.,
  aclass = "currency",
  trades = "false",
  rebase_multiplier = "rebased"
)
```

Arguments

id.	= Comma delimited list of ledger IDs to query info about (20 maximum)
aclass	= Filter output by asset class. Default value: currency
trades	= Whether or not to include trades related to position in output. Default value: false

rebase_multiplier
: = Optional parameter for viewing xstocks data. rebased: Display in terms of underlying equity. base: Display in terms of SPV tokens. Possible values: rebased, base Default value: rebased

Value

Returns ledger query as a list

Examples

```
## Not run:
 # Query Order by passing in ID which is required
 rk_resp = rk_query_ledgers(id. = 'XXXXXX-XXXXX-IN4GPZ')

 # for multiple
 tbl2 = rk_query_ledgers(id. = c('XXXXXX-XXXXX-IN4GPZ', 'XXXXXX-XXXXX-RS5JXS'))
 do.call(rbind, tbl2$result)

## End(Not run)
```

---

rk_query_orders *Query Orders Info*

Description

Retrieve information about specific orders. API Key Permissions Required: Orders and trades - Query open orders & trades or Orders and trades Query closed orders & trades, depending on status of order

Usage

```
rk_query_orders(
  trades = NULL,
  userref = NULL,
  txid = NULL,
  consolidate_taker = NULL,
  rebase_multiplier = NULL
)
```

Arguments

trades	= Whether or not to include trades related to position in output
userref	= Restrict results to given user reference
txid	= The Kraken order identifier. To query multiple orders, use comma delimited list of up to 50 ids.

consolidate_taker
: = Whether or not to consolidate trades by individual taker trades Default value: true

rebase_multiplier
: = Optional parameter for viewing xstocks data. rebased: Display in terms of underlying equity. base: Display in terms of SPV tokens. Possible values: `rebased`, `base` Default value: rebased

Value

Returns a `list` of order httr2::requested

Examples

```
## Not run:
  # Query Order by passing in the txid
  tbl = rk_query_orders(txid = 'ORB6DS-XXXXX-XXXXXX')
  df = as.data.frame(rbind(unlist(tbl$result)))

## End(Not run)
```

---

rk_query_trades_info *Query Trades Info*

Description

Retrieve information about specific trades/fills. API Key Permissions Required: Orders and trades - Query closed orders & trades

Usage

```
rk_query_trades_info(txid, trades = NULL, rebase_multiplier = "rebased")
```

Arguments

txid
: = Comma delimited list of transaction IDs to query info about (20 maximum)

trades
: = Whether or not to include trades related to position in output. Default value: false

rebase_multiplier
: = Optional parameter for viewing xstocks data. rebased: Display in terms of underlying equity. base: Display in terms of SPV tokens. Possible values: `rebased`, `base` Default value: rebased

Value

Returns specific trade info as a `list`

Examples

```
## Not run:
  # Query Order by passing in ID
 tbl = rk_query_trades_info(txid = 'XXXXXX-XXXXX-SMEC5M')

 df = as.data.frame(do.call(rbind, tbl[["result"]]))

## End(Not run)
```

`rk_req_export_rpt` *httr2::request Export Report*

Description

httr2::request export of trades or ledgers. API Key Permissions Required: Data - Export data

Usage

```
rk_req_export_rpt(
  .report,
  format = "CSV",
  .description,
  fields = "all",
  starttm = NULL,
  endtm = NULL
)
```

Arguments

`.report`	= Type of data to export. Possible values: `trades, ledgers`
`format`	= File format to export. Possible values: `CSV, TSV`. Default value: CSV
`.description`	= Description for the export
`fields`	= Comma-delimited list of fields to include trades: ordertxid, time, ordertype, price, cost, fee, vol, margin, misc, ledgers ledgers: refid, time, type, subtype, aclass, asset, amount, fee, balance, wallet Default value: all
`starttm`	= UNIX timestamp for report start time (default 1st of the current month)
`endtm`	= UNIX timestamp for report end time (default now)

Value

Returns result ID as a `list`

Examples

```
## Not run:
 # Generate export report for trades
 rk_resp = rk_req_export_rpt(.report = 'trades', .description = "my trades")

## End(Not run)
```

`rk_req_wallet_xfer` *httr2::request Wallet Transfer*

Description

Transfer from a Kraken spot wallet to a Kraken Futures wallet. Note that a transfer in the other direction must be httr2::requested via the Kraken Futures API endpoint for withdrawals to Spot wallets. API Key Permissions Required: Funds permissions - Query

Usage

```
rk_req_wallet_xfer(.asset, .from, .to, .amount)
```

Arguments

`.asset`	= Asset to transfer (asset ID or altname). Example 'XBT'
`.from`	= Source wallet. Possible values: `Spot Wallet`
`.to`	= Destination wallet. Possible values: `Futures Wallet`
`.amount`	= Amount to transfer

Value

Returns API response as a `list` of wallet transfer httr2::request

Examples

```
## Not run:
 # Don't Run if you have ETH
 rk_resp = rk_req_wallet_xfer(.asset = "ETH", .from = "Spot Wallet",
  .to = "Futures Wallet", .amount = "0.00001")

## End(Not run)
```

rk_withdraw *Withdraw Funds*

Description

Make a withdrawal httr2::request. API Key Permissions Required: Funds permissions - Withdraw

Usage

```
rk_withdraw(
  .asset,
  aclass = "currency",
  .key,
  address = NULL,
  .amount,
  max_fee = NULL,
  rebase_multiplier = "rebased"
)
```

Arguments

.asset	= Asset being withdrawn
aclass	= Specify the asset class of the asset being withdrawn. Possible values: currency, tokenized_asset. Default value: currency
.key	= Withdrawal key name, as set up on your account
address	= Optional, crypto address that can be used to confirm address matches key (will return Invalid withdrawal address error if different)
.amount	= Amount to be withdrawn
max_fee	= Optional, if the processed withdrawal fee is higher than max_fee, withdrawal will fail with EFunding:Max fee exceeded
rebase_multiplier	= Optional parameter for viewing xstocks data. rebased: Display in terms of underlying equity. base: Display in terms of SPV tokens. Possible values: rebased, base Default value: rebased

Value

Returns API response as a list generated when withdrawing funds

Examples

```
## Not run:
 # Withdraw 0.00005 BTC
 rk_resp = rk_withdraw(.asset = 'BTC', .key = 'BTC Test Wallet', .amount = "0.00005")

## End(Not run)
```

`rk_withdrawal_status` *Get Status of Recent Withdrawals*

Description

Retrieve information about recent withdrawals. Results are sorted by recency, use the cursor parameter to iterate through list of withdrawals (page size equal to value of limit) from newest to oldest. API Key Permissions Required: Funds permissions - Withdraw or Data - Query ledger entries

Usage

```
rk_withdrawal_status(
  asset,
  aclass = "currency",
  method = NULL,
  start = NULL,
  end = NULL,
  cursor = NULL,
  limit = 500,
  rebase_multiplier = "rebased"
)
```

Arguments

`asset`	= Filter for specific asset being withdrawn
`aclass`	= Filter for specific asset class being withdrawn. Possible values: `currency`, `tokenized_asset`. Default value: currency
`method`	= Filter for specific name of withdrawal method
`start`	= Start timestamp, withdrawals created strictly before will not be included in the response
`end`	= End timestamp, withdrawals created strictly after will be not be included in the response
`cursor`	= true/false to enable/disable paginated response (boolean) or cursor for next page of results (string), default false anyOf: `MOD1`, `MOD2`
`limit`	= Number of results to include per page. Default value: 500
`rebase_multiplier`	= Optional parameter for viewing xstocks data. rebased: Display in terms of underlying equity. base: Display in terms of SPV tokens. Possible values: `rebased`, `base` Default value: rebased

Value

Returns API response as a `list` of withdrawal status

Examples

```
## Not run:
 # Withdraw status for USD
 rk_resp = rk_withdrawal_status(asset = "USD")

## End(Not run)
```

Index

.get_kraken_signature, 3
.get_nonce, 4
.kraken_request, 4
.rk_env, 5
.rk_read_tokens, 6

rk_acc_xfer, 6
rk_add_order, 7
rk_add_order_batch, 10
rk_alloc_earn_funds, 12
rk_amend_order, 13
rk_build_orders, 15
rk_cancel_all_orders, 17
rk_cancel_all_orders_after, 18
rk_cancel_order, 19
rk_cancel_order_batch, 20
rk_cancel_withdrawal, 20
rk_create_sub_acc, 21
rk_dealloc_earn_funds, 22
rk_delete_export_rpt, 23
rk_download_data_export, 23
rk_edit_order, 24
rk_get_account_balance, 26
rk_get_alloc_status, 27
rk_get_api_info, 28
rk_get_asset_pairs, 29
rk_get_assets, 28
rk_get_closed_orders, 30
rk_get_credit_lines, 31
rk_get_dealloc_status, 32
rk_get_deposit_address, 33
rk_get_deposit_methods, 34
rk_get_deposit_status, 35
rk_get_earn_strats, 36
rk_get_export_status, 37
rk_get_extended_balance, 38
rk_get_grouped_order_book, 39
rk_get_level3_order_book, 40
rk_get_ohlc_data, 40
rk_get_open_margin_pos, 41
rk_get_open_orders, 42
rk_get_order_amends, 43
rk_get_order_book, 44
rk_get_order_id, 45
rk_get_recent_spreads, 45
rk_get_recent_trades, 46
rk_get_server_time, 47
rk_get_system_status, 47
rk_get_ticker_info, 48
rk_get_trade_balance, 50
rk_get_trade_volume, 50
rk_get_trades_hist, 48
rk_get_withdrawal_addresses, 51
rk_get_withdrawal_information, 52
rk_get_withdrawal_methods, 53
rk_ledgers_info, 54
rk_list_earn_alloc, 55
rk_post_trade_data, 57
rk_pre_trade_data, 57
rk_query_ledgers, 58
rk_query_orders, 59
rk_query_trades_info, 60
rk_req_export_rpt, 61
rk_req_wallet_xfer, 62
rk_withdraw, 63
rk_withdrawal_status, 64