Package 'bitmexr'

October 12, 2022

Type Package

Title R Client for BitMEX

Version 0.3.2

Description A client for cryptocurrency exchange BitMEX <https://www.bitmex.com/> including the ability to obtain historic trade data and place, edit and cancel orders. BitMEX's Testnet and live API are both supported.

License MIT + file LICENSE

URL https://github.com/hfshr/bitmexr/, https://hfshr.github.io/bitmexr/

BugReports https://github.com/hfshr/bitmexr/issues/

Imports attempt, curl, digest, dplyr, httr, jsonlite, lubridate, magrittr, progress, purrr, rlang, stringr, utils

Suggests covr, ggplot2, httptest, knitr, rmarkdown, testthat, tidyquant

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.2.1

NeedsCompilation no

Author Harry Fisher [cre, aut, cph]

Maintainer Harry Fisher <harryfisher21@gmail.com>

Repository CRAN

Date/Publication 2022-09-30 17:30:02 UTC

R topics documented:

`available_symbols` *Available symbols*

Description

Available symbols

Usage

```
available_symbols()
```

Value

A character vector of currently available symbols to be used as the `symbol` value in functions within the package.

Examples

```
## Not run:
available_symbols()

## End(Not run)
```

`bitmexr` *bitmexr: R Client for the BitMEX Exchange*

Description

bitmexr provides tools to access the API for the BitMEX cryptocurrency derivatives exchange https://www.bitmex.com/.

See Also

- https://www.bitmex.com/app/apiOverview
- https://www.bitmex.com/api/explorer/

`bucket_trades` *Bucketed trade data*

Description

`bucket_trades()` retrieves open high low close (OHLC) data for the specified symbol/time frame.

Usage

```
bucket_trades(
  binSize = "1m",
  partial = "false",
  symbol = "XBTUSD",
  count = 1000,
  reverse = "true",
  filter = NULL,
  columns = NULL,
  start = NULL,
  startTime = NULL,
  endTime = NULL,
  use_auth = FALSE
)
```

Arguments

`binSize`	character string. The time interval to bucket by, must be one of: `"1m"`, `"5m"`, `"1h"` or `"1d"`.
`partial`	character string. Either `"true"` of `"false"`. If `"true"`, will send in-progress (incomplete) bins for the current time period.
`symbol`	a character string for the instrument symbol. Use `available_symbols()` to see available symbols.

count	an optional integer to specify the number of rows to return. Maximum of 1000 (the default) per request.
reverse	an optional character string. Either "true" of "false". If "true", result will be ordered with starting with the newest (defaults to "true").
filter	an optional character string for table filtering. Send JSON key/value pairs, such as "{'key':'value'}". See examples.
columns	an optional character vector of column names to return. If NULL, all columns will be returned.
start	an optional integer. Can be used to specify the starting point for results.
startTime	an optional character string. Starting date for results in the format "yyyy-mm-dd" or "yyyy-mm-dd hh-mm-ss".
endTime	an optional character string. Ending date for results in the format "yyyy-mm-dd" or "yyyy-mm-dd hh-mm-ss".
use_auth	logical. Use TRUE to enable authentication with API key.

Details

The API will only return 1000 rows per call. If the desired time frame requires more than one API call, consider using map_bucket_trades().

Value

bucket_trades() returns a data.frame containing:

- timestamp: POSIXct. Date and time of trade.
- symbol: character. Instrument ticker.
- open: numeric. Opening price for the bucket.
- high: numeric. Highest price in the bucket.
- low: numeric. Lowest price in the bucket.
- close: numeric. Closing price of the bucket.
- trades: numeric. Number of trades executed within the bucket.
- volume: numeric. Volume in USD.
- vwap: numeric. Volume weighted average price.
- lastSize: numeric. Size of the last trade executed.
- turnover: numeric. How many satoshi were exchanged.
- homeNotional: numeric. BTC value of the bucket.
- foreignNotional: numeric. USD value of the bucket.

References

urlhttps://www.bitmex.com/api/explorer/#!/Trade/Trade_getBucketed

Examples

```
## Not run:

# Return most recent data for symbol `"ETHUSD"` for 1 hour buckets

bucket_trades(
  binSize = "1h",
  symbol = "ETHUSD",
  count = 10
)

## End(Not run)
```

`cancel_all_orders` *Cancel all orders*

Description

Cancel all orders that have been placed for a specific symbol, or use a filter to select specific orders.

Usage

```
cancel_all_orders(symbol = NULL, filter = NULL, text = NULL)
```

Arguments

`symbol`	string. Optional symbol. If provided, only cancels orders for that symbol.
`filter`	string. Optional filter for cancellation. Use to only cancel some orders, e.g. '"side": "Buy"'.
`text`	string. Optional cancellation annotation. e.g. 'Spread Exceeded'.

Value

Returns a `data.frame` with information about the orders that were cancelled. See https://www.bitmex.com/api/explorer/#!/Order/Order_cancelAll for more information.

Examples

```
## Not run:
# cancel all "Buy" orders
cancel_all_orders(filter = '{"side": "Buy"}')

## End(Not run)
```

cancel_order *Cancel order*

Description

Cancel an order that has been placed.

Usage

```
cancel_order(orderID = NULL, clOrdID = NULL, text = NULL)
```

Arguments

orderID	string. Order ID.
clOrdID	string. Optional client ID set when placing an order.
text	string. Optional cancellation annotation. e.g. 'Spread Exceeded'.

Value

Returns a data.frame with details about the order that was cancelled. See https://www.bitmex.com/api/explorer/#!/Order/Order_cancel for more information.

Examples

```
## Not run:
# Cancel an order
cancel_order(clOrdID = "myorderid")

## End(Not run)
```

edit_order *Edit an order*

Description

Edit an order that has been placed.

Usage

```
edit_order(
  orderID = NULL,
  origClOrdID = NULL,
  clOrdID = NULL,
  orderQty = NULL,
  leavesQty = NULL,
  price = NULL,
  stopPx = NULL,
  pegOffsetValue = NULL,
  text = NULL
)
```

Arguments

orderID	string. Order ID.
origClOrdID	string. The original client order ID
clOrdID	string. Optional new client order ID.
orderQty	double. Order quantity in units of the instrument (i.e. contracts).
leavesQty	string. Optional leaves quantity in units of the instrument (i.e. contracts). Useful for amending partially filled orders.
price	double. Optional limit price for 'Limit', 'StopLimit', and 'LimitIfTouched' orders.
stopPx	double. Optional trigger price for 'Stop', 'StopLimit', 'MarketIfTouched', and 'LimitIfTouched' orders. Use a price below the current price for stop-sell orders and buy-if-touched orders. Use execInst of 'MarkPrice' or 'LastPrice' to define the current price used for triggering.
pegOffsetValue	string. Optional trailing offset from the current price for 'Stop', 'StopLimit', 'MarketIfTouched', and 'LimitIfTouched' orders; use a negative offset for stop-sell orders and buy-if-touched orders. Optional offset from the peg price for 'Pegged' orders.
text	string. Optional amend annotation. e.g. 'Adjust skew'.

Value

A data.frame with information about the amended order. See https://www.bitmex.com/api/explorer/#!/Order/Order_amend for more information.

Examples

```
## Not run:

# place an order

place_order(symbol = "XBTUSD", price = 5000, orderQty = 100, clOrdID = "myorderid")

# edit the order
```

```
edit_order(origClOrID = "myorderid", orderQty = 200)

## End(Not run)
```

`get_bitmex` *GET requests*

Description

Use `get_bitmex()` to send GET requests. For private endpoints, authentication is required.

Usage

```
get_bitmex(path, args = NULL, use_auth = FALSE)
```

Arguments

`path`	string. End point for the api.
`args`	A named list containing valid parameters for the given API endpoint.
`use_auth`	logical. Use `TRUE` to access private endpoints if authentication has been set up

Value

Returns a `data.frame` containing the response from the request.

References

https://www.bitmex.com/api/explorer/

Examples

```
## Not run:

# Access a public endpoint
chat <- get_bitmex(path = "/chat", args = list(channelID = 1, reverse = "true"))

# Access private endpoint using `use_auth` = `TRUE`.
user <- get_bitmex(path = "/execution", args = list(symbol = "XBTUSD"), use_auth = TRUE)

## End(Not run)
```

map_bucket_trades *Bucket trade data over an extended period*

Description

map_bucket_trades() uses purrr::map_dfr to execute multiple API calls. This is useful when the data you want to return exceeds the maximum 1000 row response limit, but do not want to have to manually call bucket_trades() repeatedly.

Usage

```
map_bucket_trades(
  start_date = "2015-09-25 13:00:00",
  end_date = now(tzone = "UTC"),
  binSize = "1d",
  symbol = "XBTUSD",
  partial = "false",
  filter = NULL,
  use_auth = FALSE,
  verbose = FALSE
)
```

Arguments

start_date	character string. Starting date for results in the format "yyyy-mm-dd" or "yyyy-mm-dd hh-mm-ss".
end_date	character string. Ending date for results in the format "yyyy-mm-dd" or "yyyy-mm-dd hh-mm-ss".
binSize	character string. The time interval to bucket by, must be one of: "1m", "5m", "1h" or "1d".
symbol	a character string for the instrument symbol. Use available_symbols() to see available symbols.
partial	character string. Either "true" or "false". If "true", will send in-progress (incomplete) bins for the current time period.
filter	an optional character string for table filtering. Send JSON key/value pairs, such as "{'key':'value'}". See examples in trades().
use_auth	logical. Use TRUE to enable authentication with API key.
verbose	logical. If TRUE, will print information to the console. Useful for long running requests.

Details

map_bucket_trades() takes a start and end date, and creates a sequence of start dates which are passed in to the 'startTime" parameter in bucket_trades().

The length of time between each start time in each API call is determined by the binSize. For example, `"1d"` is chosen as the binSize the length of time between start dates will be 1000 days. If `"1h"` is chosen, it will be 1000 hours etc.

The function will print the number of API calls being sent and provides a progress bar in the console

Public API requests are limited to 30 per minute. Consequently, `map_bucket_trades()` uses `purrr::slowly` to restrict how often the function is called.

Value

`map_bucket_trades` returns a `data.frame` containing:

- timestamp: POSIXct. Date and time of trade.
- symbol: character. Instrument ticker.
- open: numeric. Opening price for the bucket.
- high: numeric. Highest price in the bucket.
- low: numeric. Lowest price in the bucket.
- close: numeric. Closing price of the bucket.
- trades: numeric. Number of trades executed within the bucket.
- volume: numeric. Volume in USD.
- vwap: numeric. Volume weighted average price.
- lastSize: numeric. Size of the last trade executed.
- turnover: numeric. How many satoshi were exchanged.
- homeNotional: numeric. BTC value of the bucket.
- foreignNotional: numeric. USD value of the bucket.

References

https://www.bitmex.com/api/explorer/#!/Trade/Trade_getBucketed

Examples

```
## Not run:
# Get hourly bucketed trade data between 2020-01-01 and 2020-02-01

map_bucket_trades(
  start_date = "2020-01-01",
  end_date = "2020-02-01",
  binSize = "1h"
)

## End(Not run)
```

map_trades *Trade data over an extended period*

Description

The map variant of `trades()` uses a repeat loop to continually request trade data between two time points. The function will stop when the `start_date` is greater than `end_date`. Given the large number of trades executed per day, a warning message with a choice to continue is presented when inputting a date range spanning more than one day.

Usage

```
map_trades(
  symbol = "XBTUSD",
  start_date = "2019-01-01 12:00:00",
  end_date = "2019-01-01 12:15:00",
  filter = NULL,
  use_auth = FALSE,
  verbose = FALSE
)
```

Arguments

`symbol`	a character string for the instrument symbol. Use `available_symbols()` to see available symbols.
`start_date`	character string. Starting date for results in the format `"yyyy-mm-dd"` or `"yyyy-mm-dd hh-mm-ss"`.
`end_date`	character string. Ending date for results in the format `"yyyy-mm-dd"` or `"yyyy-mm-dd hh-mm-ss"`.
`filter`	an optional character string for table filtering. Send JSON key/value pairs, such as `"{'key':'value'}"`. See examples in `trades()`.
`use_auth`	logical. Use `TRUE` to enable authentication with API key.
`verbose`	logical. If `TRUE`, will print information to the console. Useful for long running requests.

Details

Warning! Due to the extremely large number of trades executed on the exchange, using this function over an extended of time frame will result in an extremely long running process. For example, during 2019 the exchange averaged approximately 630000 trades per day, with a maximum of 2114878 trades being executed in a single day. Obtaining the trade data for this day alone would take over an hour, and the use of `map_bucket_trades()` with a small 'binSize' (e.g., `"1m"`) is preferrable.

Value

`map_trades()` returns a `data.frame` containing:

- timestamp: POSIXct. Date and time of trade.
- symbol: character. The instrument ticker.
- side: character. Whether the trade was buy or sell.
- size: numeric. Size of the trade.
- price: numeric. Price the trade was executed at
- tickDirection: character. Indicates if the trade price was higher, lower or the same as the previous trade price.
- trdMatchID: character. Unique trade ID.
- grossValue: numeric. How many satoshi were exchanged. 1 satoshi = 0.00000001 BTC.
- homeNotional: numeric. BTC value of the trade.
- foreignNotional: numeric. USD value of the trade.

References

https://www.bitmex.com/api/explorer/#!/Trade/Trade_get

Examples

```
## Not run:

# Get all trade data between 2019-05-03 12:00:00 and 2019-05-03 12:15:00

map_trades(
  start_date = "2019-05-03 12:00:00",
  end_date = "2019-05-03 12:15:00",
  symbol = "XBTUSD"
)

## End(Not run)
```

---

place_order *Place an order*

Description

Place an order using the Bitmex API. Requires API key.

Usage

```
place_order(
  symbol = NULL,
  side = NULL,
  orderQty = NULL,
  price = NULL,
  displayQty = NULL,
  stopPx = NULL,
  clOrdID = NULL,
  pegOffsetValue = NULL,
  pegPriceType = NULL,
  ordType = NULL,
  timeInForce = NULL,
  execInst = NULL,
  text = NULL
)
```

Arguments

`symbol`	string. Instrument symbol. e.g. 'XBTUSD'.
`side`	string. Order side. Valid options: Buy, Sell. Defaults to 'Buy' unless `orderQty`is negative.
`orderQty`	double. Order quantity in units of the instrument (i.e. contracts).
`price`	double. Optional limit price for 'Limit', 'StopLimit', and 'LimitIfTouched' orders.
`displayQty`	double. Optional quantity to display in the book. Use 0 for a fully hidden order.
`stopPx`	double. Optional trigger price for 'Stop', 'StopLimit', 'MarketIfTouched', and 'LimitIfTouched' orders. Use a price below the current price for stop-sell orders and buy-if-touched orders. Use `execInst` of 'MarkPrice' or 'LastPrice' to define the current price used for triggering.
`clOrdID`	string. Optional Client Order ID. This clOrdID will come back on the order and any related executions.
`pegOffsetValue`	string. Optional trailing offset from the current price for 'Stop', 'StopLimit', 'MarketIfTouched', and 'LimitIfTouched' orders; use a negative offset for stop-sell orders and buy-if-touched orders. Optional offset from the peg price for 'Pegged' orders.
`pegPriceType`	string. Optional peg price type. Valid options: LastPeg, MidPricePeg, MarketPeg, PrimaryPeg, TrailingStopPeg.
`ordType`	string. Order type. Valid options: Market, Limit, Stop, StopLimit, MarketIfTouched, LimitIfTouched, Pegged. Defaults to 'Limit' when `price` is specified. Defaults to 'Stop' when `stopPx` is specified. Defaults to 'StopLimit' when `price` and `stopPx` are specified.
`timeInForce`	string. Time in force. Valid options: Day, GoodTillCancel, ImmediateOrCancel, FillOrKill. Defaults to 'GoodTillCancel' for 'Limit', 'StopLimit', and 'LimitIfTouched' orders.

execInst	string. Optional execution instructions. Valid options: ParticipateDoNotInitiate, AllOrNone, MarkPrice, IndexPrice, LastPrice, Close, ReduceOnly, Fixed. 'AllOrNone' instruction requires displayQty to be 0. 'MarkPrice', 'IndexPrice' or 'LastPrice' instruction valid for 'Stop', 'StopLimit', 'MarketIfTouched', and 'LimitIfTouched' orders.
text	string. Optional order annotation. e.g. 'Take profit'.

Value

A tibble containing information about the trade that has been placed. See https://www.bitmex.com/api/explorer/#!/Order/Order_new for more details.

Examples

```
## Not run:

# place limit order to Buy 10 contracts at a specific price
place_order(symbol = "XBTUSD", price = 6000, orderQty = 10)

## End(Not run)
```

---

post_bitmex *POST requests*

Description

Use post_bitmex() to send POST requests. All POST requests require authentication.

Usage

```
post_bitmex(path, args = NULL)
```

Arguments

path	string. End point for the api.
args	A named list containing valid parameters for the given API endpoint.

Value

Returns a data.frame containing the response from the request.

References

https://www.bitmex.com/api/explorer/

Examples

```
## Not run:
# edit leverage on a position

post_bitmex(
  path = "/position/leverage",
  args = list("symbol" = "XBTUSD", "leverage" = 10)
)

## End(Not run)
```

`tn_bucket_trades` *Bucketed trade data (testnet)*

Description

`tn_bucket_trades()` retrieves open high low close (OHLC) data for the specified symbol/time frame.

Usage

```
tn_bucket_trades(
  binSize = "1m",
  partial = "false",
  symbol = "XBTUSD",
  count = 1000,
  reverse = "true",
  filter = NULL,
  columns = NULL,
  start = NULL,
  startTime = NULL,
  endTime = NULL,
  use_auth = FALSE
)
```

Arguments

`binSize`	character string. The time interval to bucket by, must be one of: `"1m"`, `"5m"`, `"1h"` or `"1d"`.
`partial`	character string. Either `"true"` of `"false"`. If `"true"`, will send in-progress (incomplete) bins for the current time period.
`symbol`	a character string for the instrument symbol. Use `available_symbols()` to see available symbols.
`count`	an optional integer to specify the number of rows to return. Maximum of 1000 (the default) per request.

reverse	an optional character string. Either "true" of "false". If "true", result will be ordered with starting with the newest (defaults to "true").
filter	an optional character string for table filtering. Send JSON key/value pairs, such as "{'key':'value'}". See examples.
columns	an optional character vector of column names to return. If NULL, all columns will be returned.
start	an optional integer. Can be used to specify the starting point for results.
startTime	an optional character string. Starting date for results in the format "yyyy-mm-dd" or "yyyy-mm-dd hh-mm-ss".
endTime	an optional character string. Ending date for results in the format "yyyy-mm-dd" or "yyyy-mm-dd hh-mm-ss".
use_auth	logical. Use TRUE to enable authentication with API key.

Details

The API will only return 1000 rows per call. If the desired time frame requires more than one API call, consider using tn_map_bucket_trades().

Examples

```
## Not run:

# Return most recent data for symbol `"ETHUSD"` for 1 hour buckets

tn_bucket_trades(
  binSize = "1h",
  symbol = "ETHUSD",
  count = 10
)

## End(Not run)
```

---

tn_cancel_all_orders *Cancel all orders (testnet)*

Description

Cancel all orders that have been placed using testnet API for a specific symbol, or use a filter to select specific orders.

Usage

```
tn_cancel_all_orders(symbol = NULL, filter = NULL, text = NULL)
```

Arguments

symbol	string. Optional symbol. If provided, only cancels orders for that symbol.
filter	string. Optional filter for cancellation. Use to only cancel some orders, e.g. '"side": "Buy"'.
text	string. Optional cancellation annotation. e.g. 'Spread Exceeded'.

Value

Returns a data.frame with information about the orders that were cancelled. See https://www.bitmex.com/api/explorer/#!/Order/Order_cancelAll for more information.

Examples

```
## Not run:
# cancel all "Buy" orders
tn_cancel_all_orders(filter = '{"side": "Buy"}')

## End(Not run)
```

---

tn_cancel_order *Cancel order (testnet)*

Description

Cancel an order that has been placed using the testnet API.

Usage

```
tn_cancel_order(orderID = NULL, clOrdID = NULL, text = NULL)
```

Arguments

orderID	string. Order ID.
clOrdID	string. Optional client ID set when placing an order.
text	string. Optional cancellation annotation. e.g. 'Spread Exceeded'.

Value

Returns a data.frame with details about the order that was cancelled. See https://www.bitmex.com/api/explorer/#!/Order/Order_cancel for more information.

Examples

```
## Not run:
# Cancel an order
tn_cancel_order(clOrdID = "myorderid")

## End(Not run)
```

`tn_edit_order` *Edit an order (testnet)*

Description

Edit an order that has been placed with the testnet API.

Usage

```
tn_edit_order(
  orderID = NULL,
  origClOrdID = NULL,
  clOrdID = NULL,
  orderQty = NULL,
  leavesQty = NULL,
  price = NULL,
  stopPx = NULL,
  pegOffsetValue = NULL,
  text = NULL
)
```

Arguments

`orderID`	string. Order ID.
`origClOrdID`	string. The original client order ID
`clOrdID`	string. Optional new client order ID.
`orderQty`	double. Order quantity in units of the instrument (i.e. contracts).
`leavesQty`	string. Optional leaves quantity in units of the instrument (i.e. contracts). Useful for amending partially filled orders.
`price`	double. Optional limit price for 'Limit', 'StopLimit', and 'LimitIfTouched' orders.
`stopPx`	double. Optional trigger price for 'Stop', 'StopLimit', 'MarketIfTouched', and 'LimitIfTouched' orders. Use a price below the current price for stop-sell orders and buy-if-touched orders. Use `execInst` of 'MarkPrice' or 'LastPrice' to define the current price used for triggering.

pegOffsetValue	string. Optional trailing offset from the current price for 'Stop', 'StopLimit', 'MarketIfTouched', and 'LimitIfTouched' orders; use a negative offset for stop-sell orders and buy-if-touched orders. Optional offset from the peg price for 'Pegged' orders.
text	string. Optional amend annotation. e.g. 'Adjust skew'.

Value

A data.frame with information about the amended order. See https://www.bitmex.com/api/explorer/#!/Order/Order_amend for more information.

Examples

```
## Not run:

# place an order

tn_place_order(symbol = "XBTUSD", price = 5000, orderQty = 100, clOrdID = "myorderid")

# edit the order

tn_edit_order(origClOrID = "myorderid", orderQty = 200)

## End(Not run)
```

---

tn_get_bitmex *GET requests (testnet)*

Description

Use tn_get_bitmex() to send GET requests to the testnet API. For private endpoints, authentication is required.

Usage

```
tn_get_bitmex(path, args = NULL, use_auth = FALSE)
```

Arguments

path	string. End point for the api.
args	A named list containing valid parameters for the given API endpoint.
use_auth	logical. Use TRUE to access private endpoints if authentication has been set up.

Value

Returns a data.frame containing the response from the request.

References

https://www.bitmex.com/api/explorer/

Examples

```
## Not run:
# Access a public endpoint
chat <- tn_get_bitmex(path = "/chat", args = list(channelID = 1, reverse = "true"))

# Access private endpoint using `use_auth` = `TRUE`.

user <- tn_get_bitmex(path = "/execution", args = list(symbol = "XBTUSD"), use_auth = TRUE)

## End(Not run)
```

---

tn_map_bucket_trades *Bucket trade data over an extended period (testnet)*

Description

`tn_map_bucket_trades()` uses `purrr::map_dfr` to execute multiple API calls. This is useful when the data you want to return exceeds the maximum 1000 row response limit, but do not want to have to manually call `tn_bucket_trades()` repeatedly.

Usage

```
tn_map_bucket_trades(
  start_date = "2015-09-25 13:00:00",
  end_date = now(tzone = "UTC"),
  binSize = "1d",
  symbol = "XBTUSD",
  partial = "false",
  filter = NULL,
  use_auth = FALSE,
  verbose = FALSE
)
```

Arguments

`start_date`	character string. Starting date for results in the format `"yyyy-mm-dd"` or `"yyyy-mm-dd hh-mm-ss"`.
`end_date`	character string. Ending date for results in the format `"yyyy-mm-dd"` or `"yyyy-mm-dd hh-mm-ss"`.
`binSize`	character string. The time interval to bucket by, must be one of: `"1m"`, `"5m"`, `"1h"` or `"1d"`.

symbol	a character string for the instrument symbol. Use available_symbols() to see available symbols.
partial	character string. Either "true" or "false". If "true", will send in-progress (incomplete) bins for the current time period.
filter	an optional character string for table filtering. Send JSON key/value pairs, such as "{'key':'value'}". See examples in trades().
use_auth	logical. Use TRUE to enable authentication with API key.
verbose	logical. If TRUE, will print information to the console. Useful for long running requests.

References

https://testnet.bitmex.com/api/explorer/#!/Trade/Trade_getBucketed

See Also

map_bucket_trades() for more information.

Examples

```
## Not run:
# Get hourly bucketed trade data between 2020-01-01 and 2020-02-01

tn_map_bucket_trades(
  start_date = "2020-01-01",
  end_date = "2020-02-01",
  binSize = "1h"
)

## End(Not run)
```

---

tn_map_trades *Trade data over an extended period (testnet)*

Description

The map variant of tn_trades() uses a repeat loop to continually request trade data between two time points. The function will stop when the start_date is greater than end_date.

Usage

```
tn_map_trades(
  symbol = "XBTUSD",
  start_date = "2019-01-01 12:00:00",
  end_date = "2019-01-01 12:15:00",
  filter = NULL,
  use_auth = FALSE,
  verbose = FALSE
)
```

Arguments

symbol	a character string for the instrument symbol. Use available_symbols() to see available symbols.
start_date	character string. Starting date for results in the format "yyyy-mm-dd" or "yyyy-mm-dd hh-mm-ss".
end_date	character string. Ending date for results in the format "yyyy-mm-dd" or "yyyy-mm-dd hh-mm-ss".
filter	an optional character string for table filtering. Send JSON key/value pairs, such as "{'key':'value'}". See examples in trades().
use_auth	logical. Use TRUE to enable authentication with API key.
verbose	logical. If TRUE, will print information to the console. Useful for long running requests.

Examples

```
## Not run:

# Get all trade data between 2019-05-03 12:00:00 and 2019-05-03 12:15:00

tn_map_trades(
  start_date = "2019-05-03 12:00:00",
  end_date = "2019-05-03 12:15:00",
  symbol = "XBTUSD"
)

## End(Not run)
```

---

tn_place_order *Place an order (testnet)*

Description

Place an order using the Bitmex testnet API. Requires testnet API key.

Usage

```
tn_place_order(
  symbol = NULL,
  side = NULL,
  orderQty = NULL,
  price = NULL,
  displayQty = NULL,
  stopPx = NULL,
  clOrdID = NULL,
  pegOffsetValue = NULL,
```

```
  pegPriceType = NULL,
  ordType = NULL,
  timeInForce = NULL,
  execInst = NULL,
  text = NULL
)
```

Arguments

`symbol`	string. Instrument symbol. e.g. 'XBTUSD'.
`side`	string. Order side. Valid options: Buy, Sell. Defaults to 'Buy' unless `orderQty`is negative.
`orderQty`	double. Order quantity in units of the instrument (i.e. contracts).
`price`	double. Optional limit price for 'Limit', 'StopLimit', and 'LimitIfTouched' orders.
`displayQty`	double. Optional quantity to display in the book. Use 0 for a fully hidden order.
`stopPx`	double. Optional trigger price for 'Stop', 'StopLimit', 'MarketIfTouched', and 'LimitIfTouched' orders. Use a price below the current price for stop-sell orders and buy-if-touched orders. Use `execInst` of 'MarkPrice' or 'LastPrice' to define the current price used for triggering.
`clOrdID`	string. Optional Client Order ID. This clOrdID will come back on the order and any related executions.
`pegOffsetValue`	string. Optional trailing offset from the current price for 'Stop', 'StopLimit', 'MarketIfTouched', and 'LimitIfTouched' orders; use a negative offset for stop-sell orders and buy-if-touched orders. Optional offset from the peg price for 'Pegged' orders.
`pegPriceType`	string. Optional peg price type. Valid options: LastPeg, MidPricePeg, MarketPeg, PrimaryPeg, TrailingStopPeg.
`ordType`	string. Order type. Valid options: Market, Limit, Stop, StopLimit, MarketIfTouched, LimitIfTouched, Pegged. Defaults to 'Limit' when `price` is specified. Defaults to 'Stop' when `stopPx` is specified. Defaults to 'StopLimit' when `price` and `stopPx` are specified.
`timeInForce`	string. Time in force. Valid options: Day, GoodTillCancel, ImmediateOrCancel, FillOrKill. Defaults to 'GoodTillCancel' for 'Limit', 'StopLimit', and 'LimitIfTouched' orders.
`execInst`	string. Optional execution instructions. Valid options: ParticipateDoNotInitiate, AllOrNone, MarkPrice, IndexPrice, LastPrice, Close, ReduceOnly, Fixed. 'AllOrNone' instruction requires `displayQty` to be 0. 'MarkPrice', 'IndexPrice' or 'LastPrice' instruction valid for 'Stop', 'StopLimit', 'MarketIfTouched', and 'LimitIfTouched' orders.
`text`	string. Optional order annotation. e.g. 'Take profit'.

Value

Returns a `tibble` containing information about the trade that has been placed. See https://testnet.bitmex.com/api/explorer/#!/Order/Order_new for more details.

Examples

```
## Not run:
# place limit order to Buy at specific price
tn_place_order(symbol = "XBTUSD", price = 6000, orderQty = 10)

## End(Not run)
```

`tn_post_bitmex` *POST requests (testnet)*

Description

Use `tn_post_bitmex()` to send POST requests to the testnet API. All POST requests require authentication.

Usage

```
tn_post_bitmex(path, args = NULL)
```

Arguments

`path`	string. End point for the api.
`args`	A named list containing valid parameters for the given API endpoint.

Value

Returns a `data.frame` containing the response from the request.

References

https://www.bitmex.com/api/explorer/

Examples

```
## Not run:
# edit leverage on a position

tn_post_bitmex(
  path = "/position/leverage",
  args = list("symbol" = "XBTUSD", "leverage" = 10)
)

## End(Not run)
```

tn_trades *Individual trade data (testnet)*

Description

tn_trades() retrieves data regarding individual trades that have been executed on the testnet exchange.

Usage

```
tn_trades(
  symbol = "XBTUSD",
  count = 1000,
  reverse = "true",
  filter = NULL,
  columns = NULL,
  start = NULL,
  startTime = NULL,
  endTime = NULL,
  use_auth = FALSE
)
```

Arguments

symbol	a character string for the instrument symbol. Use available_symbols() to see available symbols.
count	an optional integer to specify the number of rows to return. Maximum of 1000 (the default) per request.
reverse	an optional character string. Either "true" of "false". If "true", result will be ordered with starting with the newest (defaults to "true").
filter	an optional character string for table filtering. Send JSON key/value pairs, such as "{'key':'value'}". See examples.
columns	an optional character vector of column names to return. If NULL, all columns will be returned.
start	an optional integer. Can be used to specify the starting point for results.
startTime	an optional character string. Starting date for results in the format "yyyy-mm-dd" or "yyyy-mm-dd hh-mm-ss".
endTime	an optional character string. Ending date for results in the format "yyyy-mm-dd" or "yyyy-mm-dd hh-mm-ss".
use_auth	logical. Use TRUE to enable authentication with API key.

References

https://testnet.bitmex.com/api/explorer/#!/Trade/Trade_get

Examples

```
## Not run:
# Return 1000 most recent trades for symbol "XBTUSD".
tn_trades(symbol = "XBTUSD")

# Use filter for very specific values: Return trade data executed at 12:15.
tn_trades(
  symbol = "XBTUSD",
  filter = "{'timestamp.minute':'12:15'}"
)

# Also possible to combine more than one filter.
tn_trades(
  symbol = "XBTUSD",
  filter = "{'timestamp.minute':'12:15', 'size':10000}"
)

## End(Not run)
```

---

trades *Individual trade data*

Description

`trades()` retrieves data regarding individual trades that have been executed on the exchange.

Usage

```
trades(
  symbol = "XBTUSD",
  count = 1000,
  reverse = "true",
  filter = NULL,
  columns = NULL,
  start = NULL,
  startTime = NULL,
  endTime = NULL,
  use_auth = FALSE
)
```

Arguments

`symbol`	a character string for the instrument symbol. Use `available_symbols()` to see available symbols.
`count`	an optional integer to specify the number of rows to return. Maximum of 1000 (the default) per request.

reverse	an optional character string. Either "true" of "false". If "true", result will be ordered with starting with the newest (defaults to "true").
filter	an optional character string for table filtering. Send JSON key/value pairs, such as "{'key':'value'}". See examples.
columns	an optional character vector of column names to return. If NULL, all columns will be returned.
start	an optional integer. Can be used to specify the starting point for results.
startTime	an optional character string. Starting date for results in the format "yyyy-mm-dd" or "yyyy-mm-dd hh-mm-ss".
endTime	an optional character string. Ending date for results in the format "yyyy-mm-dd" or "yyyy-mm-dd hh-mm-ss".
use_auth	logical. Use TRUE to enable authentication with API key.

Value

trades() returns a data.frame containing:

- timestamp: POSIXct. Date and time of trade.
- symbol: character. The instrument ticker.
- side: character. Whether the trade was buy or sell.
- size: numeric. Size of the trade.
- price: numeric. Price the trade was executed at
- tickDirection: character. Indicates if the trade price was higher, lower or the same as the previous trade price.
- trdMatchID: character. Unique trade ID.
- grossValue: numeric. How many satoshi were exchanged. 1 satoshi = 0.00000001 BTC.
- homeNotional: numeric. BTC value of the trade.
- foreignNotional: numeric. USD value of the trade.

References

https://www.bitmex.com/api/explorer/#!/Trade/Trade_get

Examples

```
## Not run:
# Return 1000 most recent trades for symbol "XBTUSD".
trades(symbol = "XBTUSD", count = 10)

# Use filter for very specific values: Return trade data executed at 12:15.
trades(
  symbol = "XBTUSD",
  filter = "{'timestamp.minute':'12:15'}",
  count = 10
)
```

```
# Also possible to combine more than one filter.
trades(
  symbol = "XBTUSD",
  filter = "{'timestamp.minute':'12:15', 'size':10000}",
  count = 10
)

## End(Not run)
```

`valid_dates` *Start date of data availability for available symbols*

Description

Pass in a symbol from `available_symbols()` or no symbol to return dates for all available symbols

Usage

```
valid_dates(symbol = NULL)
```

Arguments

`symbol`	character string of the instrument symbol to find start date for.

Value

A data.frame containing the symbol and date from which data is available

Examples

```
## Not run:
valid_dates("XBTUSD")

## End(Not run)
```

Index