11. API Documentation

11.1. fxcmpy

class **fxcmpy.fxcmpy**(*access_token=''*, *config_file=''*, *log_file=None*, *log_level=''*, *server='demo'*, *proxy_url=None*, *proxy_port=None*, *proxy_type=None*)

A wrapper class for the FXCM API.

add_to_oco(*order_ids*, *oco_bulk_id=0*)

Add orders to OCO Orders.

Arguments:

order_ids: list of order_ids,
the ids of the orders to add to the OCO Order.

co_bulk_id: integer, default = 0,
the id of the OCO Order, if 0 a new OCO Order will be created.

change_order(*order_id*, *amount*, *rate*, *order_range=0*, *trailing_step=None*)

Change amount, rate, order_range and / or trailling_step of an order.

Arguments:

order_id: int,
the id of the order to change.

amount: int,
the new amount of the order.

rate: float,
the new rate of the order.

order_range: float,
the new range of the order. Only used for 'RangeEntry' orders, for other orders, it is 0 (default).

trailing_step: float,
the new trailing step for the order. Defaults to None.

change_order_stop_limit(*order_id*, *stop=None*, *limit=None*, *is_stop_in_pips=True*, *is_limit_in_pips=True*)

Change an order's stop and / or limit rate. To let an attribute unchanged set the values parameter to None.

Arguments:

order_id: integer,
the id of the order to change.

stop: float or None (Default),
the new stop rate.

limit: float or None (Default),
the new limit rate.

is_stop_in_pips: boolean (Default True),
whether the order's stop rate is in pips.

is_limit_in_pips: boolean (Default True),
whether the order's limit rate is in pips.

change_trade_stop_limit(*trade_id*, *is_stop*, *rate*, *is_in_pips=True*, *trailing_step=0*)

Change an trade's stop or limit rate.

Arguments:

trade_id: integer,
the id of the trade to change.

is_stop: boolean,

defines whether the trade's limit (False) or the stop rate (True) is to be changed.

rate: float,
the new stop or limit rate.

is_in_pips: boolean (Default True),
whether the trade's stop/limit rates are in pips.

trailing_step: float (Default 0),
the trailing step for the stop rate.

close_all(*order_type='AtMarket'*, *time_in_force='GTC'*, *account_id=None*)

Close all positions.

Arguments:

account_id: string,
the order's account id.

order_type: string (default: 'AtMarket'),
the type of order execution, one of 'AtMarket' or 'MarketRange'.

time_in_force: string (default: 'GTC'),
the time in force of the order execution, must be one of 'IOC', 'GTC', 'FOK' or 'DAY'.

account_id: integer (Default None),
the order's account id. If not given, the default account is used.

close_all_for_symbol(*symbol*, *order_type='AtMarket'*, *time_in_force='GTC'*, *account_id=None*)

Close all positions for a given symbol.

Arguments:

account_id: string,
the order's account id.

symbol: string,
the trades symbol as given by get_instruments.

order_type: string (default: 'AtMarket'),
the type of order execution, one of 'AtMarket' or 'MarketRange'.

time_in_force: string (default: 'GTC'),
the time in force of the order execution, must be one of 'IOC', 'GTC', 'FOK' or 'DAY'.

account_id: integer (Default None),
the order's account id. If not given, the default account is used.

close_trade(*trade_id*, *amount*, *order_type='AtMarket'*, *time_in_force='IOC'*, *rate=None*, *at_market=None*)

Close a given trade.

Arguments:

trade_id: integer,
the id of the trade.

amount: integer (default 0),
the trades amount in lots.

order_type: string (default 'AtMarket'),
the order type, must be 'AtMarket' or 'MarketRange'.

time_in_force: string (default 'IOC'),
the time in force of the order execution, must be one of 'IOC', 'GTC', 'FOK' or 'DAY''.

rate: float (default 0),
the trades rate.

at_market: float (default 0),
the markets range.

connect()

Connect to the FXCM server.

create_entry_order(*symbol*, *is_buy*, *amount*, *time_in_force*, *order_type='Entry'*, *limit=0*, *is_in_pips=True*, *rate=0*, *stop=None*, *trailing_step=None*, *trailing_stop_step=None*, *order_range=None*, *expiration=None*, *account_id=None*)

Creates an entry order for a given instrument.

Arguments:

account_id: integer (Default None),
the trading account's id. If None, the default account is used.

symbol: string,
the symbol of the instrument to trade as given by get_instruments().

is_buy: boolean,
True if the trade is a buy, False else.

amount: integer,
the trades amount in lots.

order_type: string,
the order type, must be 'Entry' (default) or 'RangeEntry'.

time_in_force: string,
the time in force of the order execution, must be one of 'GTC', 'DAY', 'IOC', 'FOK' or 'GTD'.

rate: float (default 0),
the trades rate.

is_in_pips: boolean (default True),
whether the trade's stop/limit rates are in pips.

limit: float (default 0),
the trades limit rate.

stop: float or None (default None),
the trades stop rate.

trailing_step: float or None (default None),
the trailing step for the stop rate.

trailing_stop_step: float or None (default None),
the trailing step for the order stop rate.

order_range: float or None (default),
the order's range if order type is 'RangeEntry'.

expiration: datetime.datetime, datetime.date or string (defaut None),
order's expiration date for 'GTD'. If a string, the date is in format 'YYYY-MM-DD hh:mm' or 'YYYY-MM-DD'.

Returns:

The id of the new order.

create_market_buy_order(*symbol*, *amount*, *account_id=None*)

Create an order to buy at market price.

Arguments:

symbol: string,
the symbol of the instrument to trade as given by get_instruments().

amount: integer,
the trades amount in lots.

account_id: integer (Default None),
the trade's account id. If not given, the default account is used.

create_market_sell_order(*symbol*, *amount*, *account_id=None*)

Create an order to sell at market price.

Arguments:

symbol: string,
the symbol of the instrument to trade as given by get_instruments().

amount: integer,
the trades amount in lots.

account_id: integer (Default None),
the trade's account id. If not given, the default account is used.

create_oco_order(*symbol*, *is_buy*, *is_buy2*, *amount*, *is_in_pips*, *time_in_force*, *at_market*, *order_type*, *expiration=None*, *limit=0*, *limit2=0*, *rate=0*, *rate2=0*, *stop=0*, *stop2=0*, *trailing_step=0*, *trailing_step2=0*, *trailing_stop_step=0*, *trailing_stop_step2=0*, *account_id=None*)

Creates an entry order for a given instrument.

Arguments:

account_id: integer (Default None),
the id of the trading account. If None, the default account is used.

symbol: string,
the symbol of the instrument to trade as given by get_instruments().

is_buy: boolean,
True if the first order is a buy, False else.

is_buy2: boolean,
True if the second order is a buy, False else.

amount: integer,
the trades amount in lots.

is_in_pips: boolean (default True),
whether the order's stop/limit rates are in pips.

time_in_force: string,
the time in force of the order execution, must be one of 'GTC', 'DAY' or 'GTD'.

at_market: float (default 0),
the order's markets range.

order_type: string,
the order type, must be 'Entry'.

expiration: string,
the order's expiration date.

limit: float (default 0),
the first order's limit rate.

limit2: float (default 0),
the second order's limit rate.

rate: float (default 0),
the first order's rate.

rate:2 float (default 0),
the second order's rate.

stop: float (default 0),
the first order's stop rate.

stop2: float (default 0),
the second order's stop rate.

trailing_step: float (default 0),
the trailing step for the first order.

trailing_step2: float (default 0),

the trailing step for the second order.

trailing_stop_step: float (default 0),
the trailing step for the first order's stop rate.

trailing_stop_step: float (default 0),
the trailing step for the second order's stop rate.

Returns:

The id of the new order.

delete_order(*order_id*)

Delete an order.

Arguments:

order_id: integer,
the id of the order to delete.

edit_oco(*oco_bulk_id*, *add_order_ids=[]*, *remove_order_ids=[]*)

Add or remove orders to or from OCO Orders.

Arguments:

oco_bulk_id: integer,
the id of the OCO Order.

add_order_ids: list of order_ids,
the id's of the orders to add to OCO Orders.

remove_order_ids: list of order_ids,
the id's of the orders to remove from OCO Orders.

get_account_ids()

Return a list of available account ids.

get_accounts(*kind='dataframe'*)

Return a snapshot of the 'Account' model.

Arguments:

kind: one of 'dataframe' (default) or 'list',
how to return the data, either as list or as a pandas DataFrame.

Returns:

The current data of the 'Account' model.

get_accounts_summary(*kind='dataframe'*)

Return a summary of the 'Account' model.

Arguments:

kind: one of 'dataframe' (default) or 'list',
how to return the data, either as list or as a pandas DataFrame.

Returns:

The summary of the current data of the 'Account' model.

get_all_trade_ids()

Returns a list of all available trade ids.

get_candles(*instrument=''*, *offer_id=None*, *period='H1'*, *number=10*, *start=None*, *end=None*, *with_index=True*, *columns=[]*, *stop=None*)

Return historical data from the fxcm database as pandas.DataFrame.

Arguments:

instrument: string (default ''):

the instrument for which data is requested. For a list of all available instruments for historical data, use get_instruments_for_candles(). If the value is equal to '' (default), offer_id must have a value. If both, instrument and offer_id are given, the value of instrument is used.

offer_id: integer (default None):

the id of the instrument for which data is requested as given in the offer trading table as given by get_offers(). If offer_id is equal to None (default), the parameter instrument must have a value. If both, instrument and offer_id are given, the value of instrument is used.

period: string,

the granularity of the data. Must be one of 'm1', 'm5', 'm15', 'm30', 'H1', 'H2', 'H3', 'H4', 'H6', 'H8', 'D1', 'W1', or 'M1'.

number: integer (default 10),

the number of candles to receive.

start: datetime.datetime, datetime.date or string (defaut None),

the first date to receive data for. If it is a string, the date is in format 'YYYY-MM-DD hh:mm' or 'YYYY-MM-DD'.

end: datetime.datetime, datetime.date or string (default None),

the last date to receive data for. If it is a string, the date is in format 'YYYY-MM-DD hh:mm' or 'YYYY-MM-DD'.

with_index: boolean (default True),

whether the column 'date' should server as index in the resulting pandas.DataFrame.

columns: list (default list()),

a list of column labels the result should include. If empty, all columns are returned.

Available column labels are

'date', 'bidopen', 'bidclose', 'bidhigh', 'bidlow', 'askopen', 'askclose', 'askhigh', 'asklow', 'tickqty'.

Also available is 'asks' as shortcut for all ask related columns and 'bids' for all bid related columns, respectively.

The column 'date' is always included.

Returns:

A pandas DataFrame containing the requested data.

get_closed_position(*position_id*)

Return the closed position with given id.

Arguments:

position_id: (integer),

the id of the position.

Returns:

The fxcmpy_open_position object.

get_closed_positions(*kind='dataframe'*)

Return a snapshot of the 'Closed Position' model.

Arguments:

kind: one of 'dataframe' (default) or 'list',

how to return the data, either as list or as a pandas DataFrame.

Returns:

The current data of the 'Closed Position' model.

get_closed_positions_summary(*kind='dataframe'*)

Return a summary of the 'Closed Position' model.

Arguments:

kind: one of ‘dataframe’ (default) or ‘list’,
how to return the data, either as list or as a pandas DataFrame.

Returns:

The summary of the current data of the ‘Closed Position’ model.

get_closed_trade_ids()
Return a list of all available trade ids of closed positions.

get_default_account()
Return the default account id.

get_instruments()
Return the tradeable instruments of FXCM as a list.

get_instruments_for_candles()
Return a list of all available instruments to receive historical data for.

get_last_price(*symbol*)
Return the last prices of a given subscribed instrument.

Arguments:

symbol: string,
the symbol of the instrument as given by get_instruments().

get_max_prices()
Return the max length of the market price tables.

get_model(*models=[], summary=False*)
Return a snapshot of the the specified model(s).

Arguments:

models: list,
list of the required models, entries must be out of [‘Offer’, ‘Account’, ‘Order’, ‘OpenPosition’, ‘ClosedPosition’,
‘Summary’, ‘Properties’, ‘LeverageProfile’].

Returns:

The current data of the specified model(s) in a json like manner.

get_oco_order(*order_id*)
Returns the oco order object for a given order id.

Arguments:

order_id: (integer),
the id of the oco order.

Returns:

The fxcmpy_oco_order_object.

get_oco_order_ids()
Return a list of the available oco order ids.

get_offers(*kind='dataframe'*)
Return a snapshot of the ‘Offer’ model.

Arguments:

kind: one of ‘dataframe’ (default) or ‘list’,
how to return the data, either as list or as a pandas DataFrame.

Returns:

The current data of the 'Offer' model.

get_open_position(*position_id*)

Return the open position with given id.

Arguments:

position_id: (integer),
the id of the position.

Returns:

The fxcmpy_open_position object.

get_open_positions(*kind='dataframe'*)

Return a snapshot of the 'Open Position' model.

Arguments:

kind: one of 'dataframe' (default) or 'list',
how to return the data, either as list or as a pandas DataFrame.

Returns:

The current data of the 'Open Position' model.

get_open_positions_summary(*kind='dataframe'*)

Return a summary of the 'Open Position' model.

Arguments:

kind: one of 'dataframe' (default) or 'list',
how to return the data, either as list or as a pandas DataFrame.

Returns:

The summary of the current data of the 'Open Position' model.

get_open_trade_ids()

Return a list of all available trade ids of open positions.

get_order(*order_id*)

Returns the order object for a given order id.

Arguments:

order_id: (integer),
the id of the order.

Returns:

The fxcmpy_order object.

get_order_ids()

Return a list of available order ids.

get_orders(*kind='dataframe'*)

Return a snapshot of the 'Order' model.

Arguments:

kind: one of 'dataframe' (default) or 'list',
how to return the data, either as list or as a pandas DataFrame.

Returns:

The current data of the 'Order' model.

get_prices(*symbol*)

Return the prices of a given subscribed instrument.

Arguments:

symbol: string,
the symbol of the instrument as given by get_instruments().

get_subscribed_symbols()

Returns a list of symbols for the subscribed instruments.

get_summary(*kind='dataframe'*)

Return a snapshot of the 'Summary' model.

Arguments:

kind: one of 'dataframe' (default) or 'list',
how to return the data, either as list or as a pandas DataFrame.

Returns:

The current data of the 'Summary' model.

is_connected()

Return True if socket connection is established and False else.

is_subscribed(*symbol*)

Returns True if the instrument is subscribed and False else.

Arguments:

symbol: string,
the symbol of the instrument in question as given by get_instruments().

open_trade(*symbol*, *is_buy*, *amount*, *time_in_force*, *order_type*, *rate=0*, *is_in_pips=True*, *limit=None*, *at_market=0*, *stop=None*, *trailing_step=None*, *account_id=None*)

Opens a trade for a given instrument.

Arguments:

symbol: string,
the symbol of the instrument to trade as given by get_instruments().

is_buy: boolean,
True if the trade is a buy, False else.

amount: integer,
the trades amount in lots.

order_type: string,
the order type, must be 'AtMarket' or 'MarketRange'.

time_in_force: string,
the time in force of the order execution, must be one of 'IOC', 'GTC', 'FOK' or 'DAY'.

rate: float (default 0),
the trades rate.

is_in_pips: boolean (default True),
whether the trades stop/limit rates are in pips.

limit: float (default 0),
the trades limit rate.

at_market: float (default 0),
the markets range.

stop: float or None (default None),

the trades stop rate.

trailing_step: float or None (default None),

the trailing step for the stop rate.

account_id: integer (Default None),

the trade's account id. If not given, the default account is used.

remove_from_oco(*order_ids*)

Remove orders from OCO Orders.

Arguments:

order_ids: list of order_ids,

the ids of the orders to remove from OCO Orders.

set_default_account(*account_id*)

" Set the default account id to account_id.

set_max_prices(*max_prices*)

Set the max lenght of the market price tables.

Arguments:

max_prices, int or None (Default 10000): The max length of the price

tables, if set to None, the price tables are unlimited.

subscribe_data_model(*model=''*, *add_callbacks=()*)

Stream data of a model.

Arguments:

model: string,

the model, must be one of ['Offer', 'Account', 'Order', 'OpenPosition', 'ClosedPosition', 'Summary', 'Properties', 'LeverageProfile'].

add_callbacks: list of callables,

all methods in that list will be called for every incoming dataset of the model. Such a method has to accept two positional arguments, data and dataframe, say. The first should be a json like object with the new data received by the stream and the second should be a Pandas DataFrame with the collected data.

subscribe_instrument(*symbol*)

Subscribe an instrument so that it appears in the offers table.

Arguments:

symbol, string:

the symbol of the instrument to activate.

Returns:

True by success and False else.

subscribe_market_data(*symbol=''*, *add_callbacks=()*)

Stream the prices of an instrument.

Arguments:

symbol: string,

the symbol of the instrument in question as given by get_instruments().

add_callbacks: list of callables,

all methods in that list will be called for every incoming dataset of the instrument. Such a method has to accept two positional arguments, data and dataframe, say. The first should be a json like object with the new price data received by the stream and the second should be a Pandas DataFrame with the collected price data as given by get_prices().

unsubscribe_data_model(*model=''*)

Unsubscribe for the given model.

Arguments:

model: string,
the model, must be one of ['Offer', 'Account', 'Order', 'OpenPosition', 'ClosedPosition', 'Summary', 'Properties', 'LeverageProfile'].

unsubscribe_instrument(*symbol*)
Unsubscribe an instrument so that it does not appears in the offers table.

Arguments:

symbol, string:
the symbol of the instrument to activate.

Returns:
True by success and False else.

unsubscribe_market_data(*symbol=''*)
Unsubscribe for instrument prices of the given symbol.

11.2. fxcmpy_order

class **fxcmpy.fxcmpy_order**(*connection*, *kwargs*)
A class to realize entry orders of the FXCM API.

Caution:

Do not initialize fxcm order object manually, these orders will not registered by the fxcm server, use the create_entry_order() method of the fxcm class instead.

delete()
Delete the order.

get_accountId()
Return the value of attribute accountId.

get_accountName()
Return the value of attribute accountName.

get_amount()
Return the value of attribute amountK.

get_associated_trade()
Returns the potial assoziated trade object.

get_buy()
Return the value of attribute buy.

get_currency()
Return the value of attribute currency.

get_currencyPoint()
Return the value of attribute currencyPoint.

get_expireDate()
Return the value of attribute timeInForce.

get_isBuy()
Return the value of attribute isBuy.

get_isELSOrder()
Return the value of attribute isELSOrder.

get_isEntryOrder()

Return the value of attribute isEntryOrder.

get_isLimitOrder()

Return the value of attribute isLimitOrder.

get_isNetQuantity()

Return the value of attribute isNetQuantity.

get_isStopOrder()

Return the value of attribute isStopOrder.

get_limit()

Return the value of attribute limit.

get_limitPegBaseType()

Return the value of attribute limitPegBaseType.

get_limitRate()

Return the value of attribute limitRate.

get_ocoBulkId()

Return the value of attribute ocoBulkId.

get_orderId()

Return the value of attribute orderId.

get_range()

Return the value of attribute range.

get_sell()

Return the value of attribute sell.

get_status()

Return the value of attribute status.

get_stop()

Return the value of attribute stop.

get_stopMove()

Return the value of attribute stopMove.

get_stopPegBaseType()

Return the value of attribute stopPegBaseType.

get_stopRate()

Return the value of attribute stopRate.

get_time()

Return the value of attribute time.

get_timeInForce()

Return the value of attribute timeInForce.

get_tradeId()

Return the trade id.

get_type()

Return the value of attribute type.

set_amount(*amount*)

Set a value for the attribute amount.

set_limit_rate(*limit_rate*, *is_in_pips=True*)

Set a new value for the limit rate.

Arguments:

limit_rate: float,
the new limit rate.

is_in_pips: bool (default=True),
whether the new rate is in pips or not.

set_range(*order_range*)
Set a value for the attribute range.

set_rate(*rate*)
Set a value for the attribute rate.

set_stop_rate(*stop_rate*, *is_in_pips=True*)
Set a new value for the stop rate.

Arguments:

stop_rate: float,
the new stop rate.

is_in_pips: bool (default=True),
whether the new rate is in pips or not.

set_trailing_step(*trailing_step*)
Set a value for the attribute trailingStep.

11.3. fxcmpy_oco_order

class **fxcmpy.fxcmpy_oco_order**(*bulk_id*, *orders*, *connection*, *logger*)
A class to realize oco orders of the FXCM API.

Caution:

Do not initialize fxcm oco order object manually, these orders will not registered by the fxcm server, use the create_oco_order() method of the fxcm class instead.

add_order(*orders*)
Add orders to the oco order.

Arguments:

orders: list,
list of the orders to add to the oco order.

edit_order(*add_orders*, *remove_orders*)
Add or remove orders to / from the oco order.

Arguments:

add_orders: list,
list of the orders to add to the oco order.

remove_orders: list,
list of the order to remove from the oco order.

get_ocoBulkId()
Return the id.

get_order_ids()
Return all ids of the containing orders.

get_orders()

Return all orders of the oco order.

remove_order(*orders*)

Remove orders from the oco order.

Arguments:

orders: list,

list of the order to remove from the oco order.

11.4. fxcmpy_open_position

class **fxcmpy.fxcmpy_open_position**(*connection*, *kwargs*)

A convenience class for a better handling of open positions.

close(*amount=0*, *order_type='AtMarket'*, *time_in_force='FOK'*, *rate=0*, *at_market=0*)

Close the trade.

Arguments:

amount: integer (default: 0),

the trades amount in lots. If 0, the whole position is closed.

order_type: string (default : *'AtMarket'*),

the order type, must be 'AtMarket' or 'MarketRange'.

time_in_force: string (default: 'FOK'),

the time in force of the order execution, must be one of 'IOC', 'GTC', 'FOK', 'DAY' or 'GTD'.

rate: float (default 0),

the trades rate.

at_market: float (default 0),

the markets range.

get_accountId()

Return the value of the attribute accountId.

get_accountName()

Return the value of the attribute accountName.

get_amount()

Return the value of the attribute amountK.

get_close()

Return the value of the attribute close.

get_com()

Return the value of the attribute com.

get_currency()

Return the value of the attribute currency.

get_currencyPoint()

Return the value of the attribute currencyPoint.

get_grossPL()

Return the value of the attribute grossPL.

get_isBuy()

Return the value of the attribute isBuy.

get_isDisabled()

Return the value of the attribute isDisabled.

get_limit()
Return the value of the attribute limit.

get_open()
Return the value of the attribute open.

get_roll()
Return the value of the attribute roll.

get_stop()
Return the value of the attribute stop.

get_stopMove()
Return the value of the attribute stopMove.

get_time()
Return the value of the attribute time.

get_tradeId()
Return the value of the attribute tradeId.

get_usedMargin()
Return the value of the attribute usedMargin.

get_valueDate()
Return the value of the attribute valueDate.

get_visiblePL()
Return the value of the attribute visiblePL.

11.5. fxcmpy_closed_position

class **fxcmpy.fxcmpy_closed_position**(*connection*, *kwargs*)
A convenience class for a better handling of closed positions.

get_accountName()
Return the value of the attribute accountName.

get_amount()
Return the value of the attribute amountK.

get_close()
Return the value of the attribute close.

get_close_time()
Return the value of the attribute closeTime.

get_com()
Return the value of the attribute com.

get_currency()
Return the value of the attribute currency.

get_currencyPoint()
Return the value of the attribute currencyPoint.

get_grossPL()
Return the value of the attribute grossPL.

get_isBuy()
Return the value of the attribute isBuy.

get_open()

Return the value of the attribute open.

get_open_time()
Return the value of the attribute openTime.

get_roll()
Return the value of the attribute roll.

get_tradeId()
Return the value of the attribute tradeId.

get_valueDate()
Return the value of the attribute valueDate.

get_visiblePL()
Return the value of the attribute visiblePL.

11.6. fxcmpy_tick_data_reader

class fxcmpy.fxcmpy_data_reader.**fxcmpy_tick_data_reader**(*symbol*, *start*, *end*, *verbosity=False*)
fxcm_tick_data_reader, a class to fetch historical tick data provided by FXCM

classmethod **get_available_symbols**()
Return all symbols available

get_data(*start=None*, *end=None*)
Returns the requested data set as pandas DataFrame; DataFrame index is converted to DatetimeIndex object

start: datetime.datetime, datetime.date or string (defaut None),
the first date to receive data for. If it is a string, the date is in format 'YYYY-MM-DD hh:mm' or 'YYYY-MM-DD'. If None, the value of start as given in the constructor is used.

end: datetime.datetime, datetime.date or string (defaut None),
the first date to receive data for. If it is a string, the date is in format 'YYYY-MM-DD hh:mm' or 'YYYY-MM-DD'. If None, the value of end as given in the constructor is used.

get_raw_data()
Returns the raw data set as pandas DataFrame

11.7. fxcmpy_candles_data_reader

class fxcmpy.fxcmpy_data_reader.**fxcmpy_candles_data_reader**(*symbol*, *start*, *end*, *period*, *verbosity=False*)
fxcm_candles_data_reader, a class to fetch historical candles data provided by FXCM

get_available_symbols()
Return all symbols available

get_data(*start=None*, *end=None*)
Returns the requested data set as pandas DataFrame; DataFrame index is converted to DatetimeIndex object

start: datetime.datetime, datetime.date or string (defaut None),
the first date to receive data for. If it is a string, the date is in format 'YYYY-MM-DD hh:mm' or 'YYYY-MM-DD'. If None, the value of start as given in the constructor is used.

end: datetime.datetime, datetime.date or string (defaut None),
the first date to receive data for. If it is a string, the date is in format 'YYYY-MM-DD hh:mm' or 'YYYY-MM-DD'. If None, the value of end as given in the constructor is used.

get_raw_data()
Returns the raw data set as pandas DataFrame