Dhan Order Management

I have developed a excel based trading system which using DhanHQ’s Data API as well as trading api. Earlier one I had was very comprehensive and complex and I was managing so many things from excel leading to a very heavy processing platform. I’m planning to simplify it Especially Order management code. I have developed few lines but though of seeking your help to fast track my development. Do not modify the following functions I developed, but use it for your logic.

Do not modify following functions

* def connect\_to\_dhan(sandbox=False)
* def place\_buy\_order(dhan,script\_key, order\_qty, trade\_price,correlation\_id)
* def place\_profit\_order(dhan,script\_key, order\_qty, trade\_price,correlation\_id)
* def place\_sl\_order(dhan,script\_key, order\_qty, trade\_price,correlation\_id)
* def place\_cancel\_order(dhan,orderid)

Do not modify following variables

* excel\_file = os.getenv('TradingSystemnNew','PValue\_Pulse.xlsx')
* workbook = xw.Book(excel\_file)
* TradeSheet = workbook.sheets['Trade']
* OrderMgmt = workbook.sheets['TRADEMGMT']
* OrdersSheet = workbook.sheets['Orders']

INPUTS that are coming from Excel.

I have created a Range variable in excel to access them directly from python.

Variables from TradeSheet

TradeSheet.PROFT\_EXIT\_TRIGGER\_PERCENTAGE : This is the percentage of price above from Average Traded price for the traded orders, profit limit order is placed (profit target order). This is the profit trigger price. [ note : Average Traded Price what dhan gives by default is at script level, but need in a different way, I’ll provide details when we compute AVG\_TRADED\_PRICE]

TradeSheet.LOSS\_EXIT\_TRIGGER\_PERCENTAGE : This is the percentage of price below from Average Traded price for the traded orders, SL t order is placed (SL target order) This is the Stop loss trigger price.

TradeSheet.PROFIT\_EXIT\_ACTUAL\_PERCENTAGE: This is the percentage of price above from Average Traded price for the traded orders, profit limit order price.

TradeSheet.LOSS\_EXIT\_ACTUAL\_PERCENTAGE : This is the percentage of price below from Average Traded price for the traded orders, Stop loss order price.

Variables from OrderMgmt

OrderMgmt.SYMBOL\_KEY : this is what we will pass as “security\_id” in all place order functions.

OrderMgmt.LTP : Last traded price of selected symbol.

OrderMgmt.LMT\_PRICE: This is a the potential maximum price we wish to fulfill all out buy orders.

OrderMgmt.SLICE\_ORDER\_QTY : This is the total quantity we will be sending to the “place\_buy\_order” that has dhan.place\_slice\_order method.

OrderMgmt.NON\_SLICE\_ORDER\_QTY: This is the total quantity we will be sending to the “place\_buy\_order” that has dhan.place\_order method.

OrderMgmt .SELL\_SLICE\_ORDER\_QTY : This is the total quantity we will be sending to the “place\_profit\_order” and “place\_sl\_order “ that has dhan.place\_slice\_order method.

OrderMgmt.SELL\_NON\_SLICE\_ORDER\_QTY : This is the total quantity we will be sending to the “place\_profit\_order” and “place\_sl\_order “ that has dhan.place\_order method.

OrderMgmt.INITIATE : This is where the show begins. It is enabled with Data validation dropdown “Trade” and “Stop”. It is defaulted to “Stop” when we start the program.

OrderMgmt.TRADE\_STATUS : This is where we maintain the stage of the trading lifecycle. It has following values “OPEN, INITIATED, INPROGRESS, NEAR\_PROFIT, NEAR\_LOSS”. Defaulted to “OPEN” when program starts.

What I expect?

Moment I select “Trade” in OrderMgmt.INITIATE field, Please Buy order using “**place\_buy\_order**” function. But it can be placed only if the following conditions are met.

1. OrderMgmt.LMT\_PRICE is not blank or it is greater than 0.
2. OrderMgmt.TRADE\_STATUS = “OPEN”

If you look at the place order function I wrote, it calls 2 methods from dhanhq. “dhan.place\_slice\_order” and “dhan.place\_order”, which one to invoke is depends on the values of the parameter “order\_qty” that I pass in the function “place\_buy\_order”.

place\_buy\_order(dhan,script\_key, order\_qty, trade\_price,correlation\_id)

* + script\_key =   OrderMgmt .SYMBOL\_KEY
  + order\_qty = {SLICE\_ORDER\_QTY': OrderMgmt.range('SLICE\_ORDER\_QTY').value,  
    'NON\_SLICE\_ORDER\_QTY': OrderMgmt.range('NON\_SLICE\_ORDER\_QTY').value, }
  + trade\_price = OrderMgmt.range('LMT\_PRICE').value
  + correlation\_id = “ALGOBUYORDER”

The respective method is triggered only if corresponding value is a non-zero positive number. This function returns order status by combining Slice\_order and non\_slice\_order within a single dictionary.

I have developed this logic already.

What should happen next is as follows.

1. Once the “place\_buy\_order” is successful,
2. Update OrderMgmt.TRADE\_STATUS = INITIATED.

Flatten all the orders as a single list called “BUY\_PROSPECT”.

Start tracking the orderStatus for all the orders that are placed from BUY\_ PROSPECT list. To get the order status, use dhan.get\_order\_by\_id(order\_id), I assume you will passing all the order ids from BUY\_PROSPECT list. That function returns a dictionary, we are interested in only orderId and orderStatus.

A order has multiple status as “TRANSIT, PENDING, REJECTED, CANCELLED, PART\_TRADED, TRADED, EXPIRED” . We call any order as “LIVE” only if its orderStatus is “TRADED”, when orderStatus is “TRANSIT or PENDING”, we will call it as “PROSPECT”, the remaining as “INERT”.

Based on the status we will move them to respective list and continue to track as provided below.

If any of the order’s orderStatus is in “TRANSIT or PENDING”, continue to keep them in “BUY\_PROSPECT” list and continue to track their status.

If any of the order’s orderStatus is “REJECTED, CANCELLED or EXPIRED”, we will move them to a list called “BUY\_INERT” and stop tracking the Status.

If any of the order’s orderStatus is “TRADED”, we will move them to a list called “BUY\_LIVE” and stop tracking the Status.

Initiate a empty dataframe “DF\_TRADED\_BUY” with columns “orderId”,” tradedQuantity” & “tradedPrice”

Initiate 2 variables,

total\_traded\_buy\_qty = 0

avg\_buy\_price = 0.0

total\_trade\_cost = 0.0

Update OrderMgmt.TRADE\_STATUS = INPROGRESS

Get the trade details of orders in BUY\_LIVE using dhan method “dhan.get\_trade\_book(order\_id)”. The response json has many values, but we are interested only in following.

* orderId: Order specific identification generated by Dhan
* tradedQuantity : Number of shares executed
* tradedPrice: Price at which trade is executed.
* order\_cost : tradedQuantity\* tradedPrice [computed column]

Append the above values to dataframe “DF\_TRADED\_BUY”.

1. Sum the tradedQuantity from DF\_TRADED\_BUY and store it in total\_traded\_buy\_qty
2. total\_trade\_cost = sum(DF\_TRADED\_BUY. order\_cost)
3. Calcluale average\_buy price using the formula below and update avg\_buy\_price.
   1. avg\_buy\_price = total\_trade\_cost /sum(tradedQuantity)
4. update OrderMgmt AVG\_BUY\_PRICE = avg\_buy\_price

Once BUY\_LIVE becomes non-empty list, 2 process start happening in parallel.

[Here you may think about using async or threading, please DON’T, we will keep the program as simple as possible. all these will run with in “While true:” loop. So, we achieve near Realtime parallel]

Process 1

1. Continue to track the order status of the orders in BUY\_PROSPECT list using dhan.get\_order\_by\_id(order) till It’s status becomes ‘TRADED’ or “REJECTED, CANCELLED or EXPIRED”.
2. Again if any of the order turns out to be ‘TRADED’ move it “BUY\_LIVE” list, update the DF\_TRADED\_BUY data frame and update total\_traded\_buy\_qty , avg\_buy\_price, total\_trade\_cost. This will continue till all the orders in BUY\_PROSPECT is removed and it becomes empty or OrderMgmt.TRADE\_STATUS is not INPROGRESS

Process 2

1. Keep comparing the OrderMgmt.LTP with following variables and take actions as provided below

* OrderMgmt.LTP >= OrderMgmt.TRIGGER\_PROFIT call def place\_profit\_order(dhan,script\_key, order\_qty, trade\_price,correlation\_id)
  + script\_key =   OrderMgmt .SYMBOL\_KEY
  + order\_qty = {SELL\_SLICE\_ORDER\_QTY': OrderMgmt.range(' SELL\_SLICE\_ORDER\_QTY').value,  
    'SELL\_NON\_SLICE\_ORDER\_QTY': OrderMgmt.range(' SELL\_NON\_SLICE\_ORDER\_QTY').value, }
  + trade\_price = OrderMgmt.range(' PROFIT\_TARGET').value
  + correlation\_id = “ALGOSELLPROFITORDER”
* Update OrderMgmt.TRADE\_STATUS = “NEAR\_PROFIT”
* Cancel all orders present in BUY\_PROSPECT list, once you get the confirmation of cancelled orders empty the BUY\_PROSPECT list. Use “place\_cancel\_order(dhan,orderid)” to send the cancel request.
* Create 3 lists similar to buy order lists.
  + SELL\_PROFIT\_PROSPECT
  + SELL\_PROFIT\_LIVE
  + SELL\_PROFIT\_INERT
* By default append all orders that returned by “place\_profit\_order” to SELL\_PROFIT\_PROSPECT.
* Start getting status of all orders from SELL\_PROFIT\_PROSPECT using dhan method “dhan,get\_order\_by\_id(self, order\_id)”.
* All the orders whose orderStatus is “TRADED”, move to SELL\_PROFIT\_LIVE
* All the orders whose orderStatus is in “REJECTED, CANCELLED or EXPIRED” move to SELL\_PROFIT\_INERT
* All the orders whose orderStatus is in “TRANSIT or PENDING”, move to SELL\_PROFIT\_PROSPECT
* Continue to get orderStatus till all the orders in SELL\_PROFIT\_PROSPECT is executed and SELL\_PROFIT\_PROSPECT is emptied.
* During this loop if LTP becomes less than OrderMgmt AVG\_BUY\_PRICE, and if OrderMgmt.TRADE\_STATUS = “NEAR\_PROFIT” and SELL\_PROFIT\_PROSPECT is not empty, call