slight change of topic. I was doing this earlier with an Excel spreadsheet, with two sheets one the rule book that would have different trades under a strategy listed like IC1, IC2, IC3 etc with different entry, exit conditions and legs. the other sheet was a transactional sheet that once entry conditions were met the trade would be taken and moved there to track in real time for exit conditions to meet and exit at PT or sl.

I want to do something similar here the main columns that I want to have on my rule table are:

1. trade name: this is something within a strategy. this file that we created could be considered a strategy. but a strategy can have multiple ICs kind of trades. to differentiate each one of them we can use this column

2. entry conditions: this could be time or market condition based or a combination. sometimes one rule can be configured to create multiple trades as well and sometimes it would not allow that. we can keep multiple logical conditions on this that will be evaluated and combined to decide if entry conditions are met or not.

3. exit conditions: similar to above but it won't be evaluated on the rule table. once an entry is made it'll be evaluated as often as possible to see if PT/sl are met and then exit accordingly. sometimes the rules in this could be complex, like instead of one trade the while strategies with multiple trades pt/sl could be checked.

4. legs: this would store at max 4 legs for an options trade, their quantities and mathematical representation. it could be done by storing leg 1 to 4 separately and then showing an ic as leg 1-leg 2+ leg 3 - leg 4. each legs picking criteria(like leg 2 and leg 4 that are short for an ic could be at higher dental put and call let's say 30 Delta and leg 1 and leg 3 could be 25 width apart from those two). this criteria would be such that at any market condition we would be able to drive these legs. this criteria and this mathematical representation will be tracked on rule table.

their entry price once entered, current price, exit condition evaluated as often as possible, Delta at entry and current Delta for each leg could be part of transaction table.

the Python file we created to test out the ic creation would be a strategy file. each strategy could be coded like that. before an entry is made, each strategy trades could be evaluated for entry conditions and printed into a csv(strategy\_entry). once the conditions are met, they could go into another csv (strategy\_trades). the options legs in the trades/transactions table can be added to a quoue that will be read by my collect\_data.py(https://github.com/meetsang/spx-bot) so that in addition to SPx these options tickers could be tracked for pricing, written to a CSV and referred by a method to evaluate exit conditions as often as possible. I think I have already implemented that part of message quoue and also storing of options pricing separately from SPx, please check my code base at above link.

thoughts on all of this?

Love this direction. You’re essentially proposing a rules-driven “strategy engine” with:

- A declarative rule book (what to trade, when, and how to construct legs)

- A transaction/trade state table (what’s active, how it’s performing, when to exit)

- A pricing/greeks event loop (your existing collect\_data.py) to drive exit evaluation

- Simple queues/CSVs as the glue between components

Below is a concrete, minimal, extensible plan that fits your repo and leverages what you’ve already built.

High-level architecture

- Rule table (CSV or JSON): declarative strategies and trade templates (IC1, IC2…) with entry/exit logic and leg construction rules (e.g., 30Δ shorts, 1-wide for XSP, 5-wide for SPX).

- Strategy evaluator (per strategy Python module): reads rules, computes current signals (entry), resolves legs (strikes/symbols), proposes orders.

- Entry queue (CSV): pending entries (strategy\_entry.csv). Human-readable and auditable.

- Trade table (CSV): live/active trades (strategy\_trades.csv). Includes legs, entry price, greeks at entry, PT/SL definitions, current state.

- Market data engine (your collect\_data.py + streamer): already tracks option symbols and writes prices. It also reads a queue/list of option symbols to track.

- Exit evaluator (poll/async loop): reads live prices from your pricing store, evaluates exit conditions (PT/SL, time-based, composite rules), and emits close orders.

File formats and schemas

1) rules.csv (rule book)

- strategy\_name: e.g., XSP\_IC\_1W

- trade\_name: e.g., IC1, IC2 (unique within strategy)

- underlying: XSP or SPX

- max\_open\_trades: integer (cap concurrent instances for this rule)

- allow\_multiple\_same\_bar: true/false (multiple entries allowed on same eval cycle)

- entry\_time\_window: e.g., 09:35-14:45 ET (optional)

- entry\_conditions: JSON string (logical expression tree or simple DSL)

Example:

{"all":[

{"type":"clock\_between","start":"09:35","end":"14:45","tz":"America/Chicago"},

{"type":"day\_of\_week","in":},

{"type":"price\_above\_sma","window":20,"min\_dist\_pct":0.5}

]}

- exit\_conditions: JSON string (evaluated post-entry)

Example:

{"any":[

{"type":"spread\_pnl\_pct","gte":20}, // take profit 20%

{"type":"spread\_pnl\_pct","lte":-50}, // stop loss -50%

{"type":"dte\_lte","value":0}, // zero DTE (EOD close)

{"type":"vega\_spike","gt":2.0} // optional composite condition

]}

- legs\_spec: JSON string (how to derive legs)

Example (iron condor):

{

"type":"IC",

"short\_call\_delta":0.30,

"short\_put\_delta":-0.30,

"width":1,

"qty":1,

"rounding":"nearest\_strike",

"brackets":{

"credit\_pct\_of\_mid":1.00, // you want 100% of mid

"min\_credit":0.15

}

}

- dte\_target: integer (e.g., 7)

- dte\_tolerance: integer (e.g., 1)

- capital\_tag: label for risk-buckets (optional)

- notes: free text

2) strategy\_entry.csv (proposed entries; evaluated each cycle)

- timestamp

- strategy\_name

- trade\_name

- underlying

- expiration

- legs\_json: JSON array of resolved legs with symbols/strikes/qty/side

Example:

[

{"symbol":"XSP 250815C00640000","side":"SELL","qty":1},

{"symbol":"XSP 250815C00641000","side":"BUY","qty":1},

{"symbol":"XSP 250815P00638000","side":"SELL","qty":1},

{"symbol":"XSP 250815P00637000","side":"BUY","qty":1}

]

- target\_credit: decimal (rounded to $0.05)

- width: numeric

- deltas\_at\_entry\_json: {"short\_call":0.30,"short\_put":-0.30} (optional)

- entry\_conditions\_snapshot: JSON string (what evaluated true)

- state: Proposed

This file is the “staging” area. A separate executor (or your strategy module) can place the order and then move it to strategy\_trades.csv upon fill/ack.

3) strategy\_trades.csv (live trades)

- trade\_id: unique

- timestamp\_opened

- strategy\_name

- trade\_name

- underlying

- expiration

- legs\_json: same structure as above

- entry\_credit

- current\_mid\_credit: updated by exit evaluator (or read from your pricing store)

- pt\_rule: JSON (how PT computed)

- sl\_rule: JSON (how SL computed)

- greeks\_at\_entry\_json: optional

- greeks\_current\_json: optional

- pnl\_pct: updated live

- status: Open/Closing/Closed

- timestamp\_closed

- exit\_reason: PT/SL/Time/Manual

- audit\_log: optional notes

4) symbols\_watch.csv (for collect\_data.py)

- symbol

- source: “strategy” | “manual” | “other”

- active: true/false

- last\_updated

Your collect\_data.py already supports reading in symbols to track and writing live prices. The strategy engine should append the legs’ streamer symbols to symbols\_watch.csv when a trade is opened, and remove when closed.

Evaluation and flow

- Rule evaluation loop (interval N seconds):

1) Load rules.csv.

2) For each rule, check constraints (time window, max\_open\_trades).

3) Evaluate entry\_conditions (you can write a small evaluator that supports “all/any” and a core set of predicates: time window, day\_of\_week, price relation to MA, IV rank, VIX filter, distance from price, etc.). Keep it pluggable: new predicate = new small function.

4) If entry true:

- Resolve strikes via legs\_spec:

- For IC:

- find expiration closest to dte\_target (± tolerance)

- select short\_call at ~+target delta, short\_put at ~-target delta

- define width, compute long legs from short ± width

- fetch option objects/symbols from chain

- compute mid credit via streamer (or leave to order-placement module)

- round to $0.05

- Append to strategy\_entry.csv (Proposed)

- Order executor:

- Reads strategy\_entry.csv

- Places order via tastyware (your existing build\_leg + NewOrder)

- On success, writes row to strategy\_trades.csv with status=Open, entry\_credit, legs, etc.

- Adds all leg streamer symbols to symbols\_watch.csv for collect\_data.py to track

- Marks the strategy\_entry row as Executed/Archived or removes it

- Exit evaluation loop (can run in same process or separate):

- For each Open trade in strategy\_trades.csv:

- Read latest prices for legs from your pricing store (that collect\_data.py updates)

- Compute current spread mid and PnL%

- Evaluate exit\_conditions from the rule (PT/SL/time/composite)

- If exit true, place close order (reverse legs), update status, remove symbols from watch list

Practical implementation notes

- Use JSON-in-CSV for complex fields (entry\_conditions, exit\_conditions, legs\_spec, legs\_json). This keeps CSV simple but flexible.

- Keep predicates minimal at first:

- clock\_between

- day\_of\_week

- price\_above/below (SPX/XSP last)

- dte window

- delta target per leg (use your DXFeed Greeks or chain greeks if available)

- Make strategy modules thin:

- A strategy file simply maps rules to “resolve legs” and “compute target credit,” then outputs to strategy\_entry.csv. Reuse shared utilities for chain access, delta selection, rounding, and order creation.

- Concurrency

- If multiple processes access the CSVs, use file locks (portalocker) or switch to SQLite (lightweight, easy migration later).

- Symbols

- Always store both instrument symbol and streamer\_symbol if tastyware differentiates. You’re already using streamer\_symbol in greeks\_gex.py. Keep consistency.

Minimal evaluators/pseudocode

- Entry conditions evaluator (JSON):

- def eval\_conditions(ctx, cond): supports:

- {"all":[...]} / {"any":[...]}

- {"type":"clock\_between","start":"09:35","end":"14:45","tz":"America/Chicago"}

- {"type":"day\_of\_week","in":[1..5]}

- {"type":"price\_above","underlying":"SPX","buffer\_pct":0.3}

- {"type":"dte\_between","min":5,"max":9}

- etc.

- ctx provides: now, underlying last price, chain, greeks feed accessors (optional)

- Legs resolution (IC):

- Given target deltas and width, find nearest strikes by delta from chain+greeks:

- If you don’t have deltas in chain, use your DXLink Greeks streamer with a quick fetch for the candidate expirations to select by delta, otherwise fallback to distance from spot.

- Build four legs and compute credit mid via your existing streaming mid helper

- Round to $0.05

CSV I/O helpers

- Simple “append row” and “upsert by trade\_id” utilities

- JSON field handling with json.dumps/loads

Suggested first deliverables

- rules.csv with 1-2 example rows (XSP\_IC\_1W: IC1, IC2 with slightly different deltas/OTM)

- strategy\_runner.py:

- load rules.csv

- evaluate entry for each

- resolve legs

- write to strategy\_entry.csv

- order\_executor.py:

- read strategy\_entry.csv

- place orders

- move to strategy\_trades.csv + add symbols to symbols\_watch.csv

- exit\_watcher.py:

- loop over strategy\_trades.csv

- read latest mids from your price store

- evaluate exit conditions

- place close and update status

Example rows

- rules.csv (CSV with JSON fields quoted)

strategy\_name,trade\_name,underlying,max\_open\_trades,allow\_multiple\_same\_bar,entry\_time\_window,entry\_conditions,exit\_conditions,legs\_spec,dte\_target,dte\_tolerance,notes

XSP\_IC\_1W,IC1,XSP,5,false,"09:35-14:45","{""all"":[{""type"":""clock\_between"",""start"":""09:35"",""end"":""14:45"",""tz"":""America/Chicago""},{""type"":""day\_of\_week"",""in"":},{""type"":""dte\_between"",""min"":5,""max"":9}]}", "{""any"":[{""type"":""spread\_pnl\_pct"",""gte"":20},{""type"":""spread\_pnl\_pct"",""lte"":-50},{""type"":""dte\_lte"",""value"":0}]}", "{""type"":""IC"",""short\_call\_delta"":0.30,""short\_put\_delta"":-0.30,""width"":1,""qty"":1,""brackets"":{""credit\_pct\_of\_mid"":1.0,""min\_credit"":0.10}}",7,2,"XSP 1-wide IC at 30/-30 delta"

- strategy\_entry.csv (after evaluation)

timestamp,strategy\_name,trade\_name,underlying,expiration,legs\_json,target\_credit,width,deltas\_at\_entry\_json,entry\_conditions\_snapshot,state

2025-08-10T10:02:35-05:00,XSP\_IC\_1W,IC1,XSP,2025-08-15,"[{""symbol"":""XSP 250815C00640000"",""side"":""SELL"",""qty"":1},{""symbol"":""XSP 250815C00641000"",""side"":""BUY"",""qty"":1},{""symbol"":""XSP 250815P00638000"",""side"":""SELL"",""qty"":1},{""symbol"":""XSP 250815P00637000"",""side"":""BUY"",""qty"":1}]",0.65,1,"{""short\_call"":0.30,""short\_put"":-0.30}","{""clock\_between"":true,""dte\_between"":true}","Proposed"

- strategy\_trades.csv (after placement)

trade\_id,timestamp\_opened,strategy\_name,trade\_name,underlying,expiration,legs\_json,entry\_credit,current\_mid\_credit,pt\_rule,sl\_rule,greeks\_at\_entry\_json,greeks\_current\_json,pnl\_pct,status,timestamp\_closed,exit\_reason,audit\_log

XSP-IC1-20250810-0001,2025-08-10T10:03:00-05:00,XSP\_IC\_1W,IC1,XSP,2025-08-15,"[...]",0.65,0.62,"{""type"":""spread\_pnl\_pct"",""gte"":20}","{""type"":""spread\_pnl\_pct"",""lte"":-50}","{...}","{...}",-4.6,Open,,,Placed via strategy\_runner

How this plugs into your current repo

- Your collect\_data.py already:

- Subscribes to option symbols

- Writes quotes/greeks CSVs

- You mentioned you already have a message queue and separate pricing storage: keep using it. The strategy engine should only add/remove symbols it cares about.

- Add a very small “symbol registry writer” that appends leg streamer symbols to your existing queue file(s) when a trade opens, and removes them when it closes.

Why CSV first

- Easy debugging and auditability

- Git-friendly if you want to snapshot rules

- You can move to SQLite later with identical schemas (columns map cleanly)

Next steps I can help with

- Provide a minimal reference implementation for:

- Rule evaluator with a few core predicates

- IC leg resolver by delta (falling back to distance-from-spot if greeks unavailable that moment)

- CSV writer/reader utilities (append/upsert)

- Simple order executor loop that promotes Proposed entries to Open trades

- Exit loop stub that reads your price store and marks PT/SL

If you share a couple of concrete rule examples (IC1/IC2 with exact deltas/widths/PT/SL), I can draft the initial rules.csv and a tiny evaluator that matches your patterns so you can wire it into your repo quickly.