Chat history

You said:

The script below draws indicators in ProRealTime chart. But because it's only an indicator one can't use it in any auto trading code or any alert. What do you suggest to change in the code or write new code so the functionality will be there to give an alert or even auto trading code? When there is an alert the user can have PRT open a position when the alert is triggered.

ChatGPT said:

Understood. Here’s the straight answer: an *indicator* in ProRealTime (ProBuilder) can draw nice visuals, but it can’t place orders. To get alerts and/or auto-trading, you need to (a) expose your indicator’s “signal” in a way the platform can alert on, and/or (b) port the same logic into a **ProOrder strategy** that can actually submit orders. Below is a practical, copy-pasteable path for both.

---

1) Make your indicator “alertable” (no trading, just alerts)

What to change: add boolean signals (0/1) for LONG and SHORT and return them as lines. Then create platform alerts on those lines.

Template (Indicator / ProBuilder):

```
// ==== Indicator: signalized version (for alerts) ====
// Replace the demo logic with your own conditions (the same ones you currently
draw).
// Example condition: close crossing above/below a moving average.

DEFPARAM DrawOnLastBarOnly = False

// --- Inputs (add your own) ---
MAperiod = 50
ma = average[MAperiod](close)

// --- Signals (MAKE THEM NON-REPAINT with [1] references) ---
longCond  = close > ma AND close[1] <= ma[1]
shortCond = close < ma AND close[1] >= ma[1]

// --- One-bar pulses (0/1) you can alert on ---
longPulse  = longCond ? 1 : 0
shortPulse = shortCond ? 1 : 0

// --- Optional: persistent state (stays 1 while condition holds) ---
longState  = close > ma ? 1 : 0
shortState = close < ma ? 1 : 0

// --- Draw visuals if you want ---
IF longCond THEN
   DRAWARROWUP(LOW, 0) COLOURED(0,200,0)
ENDIF
IF shortCond THEN
   DRAWARROWDOWN(HIGH, 0) COLOURED(200,0,0)
ENDIF
```

```
// --- RETURN the lines so you can set alerts on them ---
// Return order matters. Name them clearly in the indicator UI.
RETURN longPulse AS "LongPulse", shortPulse AS "ShortPulse", longState AS
"LongState", shortState AS "ShortState"
```

How to use for alerts:

- Add this indicator to your chart.
- Right-click the *LongPulse* (or *ShortPulse*) line → **Create alert** → "crosses above 0.5" (that's your 0→1 pulse).
- If you prefer "condition stays true," attach the alert to *LongState* crossing above 0.5.
- In ProRealTime, you can have alerts trigger notifications and—if you configure it—*ask PRT to open a position when the alert triggers*. (This is a platform alert → order action mapping.)

Why this works: Indicators can't trade, but PRT can watch any returned indicator line for thresholds/crossings and fire alerts reliably *without repaint* if you anchor conditions to confirmed bars (using `[1]` comparisons as shown).

---

2) Turn the same logic into an auto-trading strategy (ProOrder)

What to change: copy the *logic only* (not the drawing code) into a ProOrder strategy. Build entries/exits, risk, time-windows, and trailing stops. This lets PRT actually place orders.

Template (Strategy / ProOrder):

```
// ==== Strategy: same signals, now tradable ====
// Paste YOUR indicator's logic where indicated (use [1] confirmations to avoid
repainting).

DEFPARAM CumulateOrders = False
DEFPARAM PreLoadBars     = 5000    // ensure enough history for your indicators

// --- Parameters (edit these or expose with user inputs) ---
MAperiod     = 50
tpPoints     = 100                 // take profit in points
slPoints     = 60                  // initial stop in points
tsPoints     = 40                  // trailing stop in points
startTime    = 090000              // trading window start (local to your data feed)
endTime      = 170000              // trading window end

// --- Core calc (replace with your real logic) ---
ma = average[MAperiod](close)
entryLong  = close > ma AND close[1] <= ma[1]
entryShort = close < ma AND close[1] >= ma[1]

// --- Time filter (optional but recommended) ---
inSession = time >= startTime AND time <= endTime

// --- Entries (end-of-bar confirmation by default) ---
IF NOT onmarket AND inSession THEN
   IF entryLong THEN
      BUY 1 CONTRACT AT MARKET
   ELSIF entryShort THEN
      SELLSHORT 1 CONTRACT AT MARKET
   ENDIF
```

```
ENDIF

// --- Risk management (ProOrder syntax) ---
SET TARGET pPROFIT tpPoints
SET STOP pLOSS   slPoints
SET STOP TRAILING tsPoints  // <-- correct syntax

// --- Optional: avoid immediate flip; wait N bars after an entry ---
/*
cooldown = 3
once entryBar = -100000
IF (entryLong OR entryShort) AND NOT onmarket AND inSession THEN
   IF barindex - entryBar >= cooldown THEN
      // place order (as above) and set: entryBar = barindex
   ENDIF
ENDIF
*/

// --- Optional: daily shut-down (flat by endTime) ---
IF time >= endTime AND onmarket THEN
   EXITSHORT AT MARKET
   EXITLONG  AT MARKET
ENDIF

// --- Optional diagnostics for debugging ---
IF entryLong THEN
   PRINT("LongSignal @ ", time, " price=", close)
ENDIF
IF entryShort THEN
   PRINT("ShortSignal @ ", time, " price=", close)
ENDIF
```

Swap in your logic: Replace the demo MA crossover with your exact indicator conditions. Keep the "`... AND close[1] <= ...[1]`" style so signals fire only when a bar actually flips state—this avoids phantom intrabar triggers.

Why do both?

- **Indicator+Alerts** is fast to deploy and keeps your current charting workflow.
- **Strategy (ProOrder)** is required for unattended auto-trading, backtests, and robust risk controls.

---

3) (Optional) Multi-market alerting with ProScreener

If you want alerts across a watchlist, port the same boolean conditions into a **ProScreener** and return the instrument when a signal occurs. Configure the screener to run on your list; PRT can notify when symbols appear.

Template (ProScreener):

```
// ==== ProScreener (find tickers with a fresh signal) ====
MAperiod = 50
ma = average[MAperiod](close)

longCond  = close > ma AND close[1] <= ma[1]
shortCond = close < ma AND close[1] >= ma[1]
```

```
// Return the instrument only when a new signal fires
SCREENER[ longCond OR shortCond ](close AS "Px", volume AS "Vol")
```

---

4) Guardrails you should implement (don’t skip)

1. **No repainting:**
    - Use confirmed bar comparisons (`[1]`), not “instant” intrabar crosses, unless you run on tick/intrabar data intentionally.
    - Avoid centered/triangular MAs or anything that revises history.
2. **One signal per event:**
    - Use pulses (0→1 for one bar) for alerts and entries. Persisting “state=1” is fine for info, but you don’t want dozens of alerts while a state stays true.
3. **Risk first:**
    - Always set both a **stop loss** and either a **target** or **trailing stop**. As you already discovered, use `SET STOP TRAILING`, not the reverse.
4. **Session controls & cooldowns:**
    - Restrict trading hours and add a post-entry cooldown to avoid flip-flopping in chop.
5. **Backtest parity:**
    - Strategy code should be a text-for-text clone of your indicator logic wherever possible to ensure your backtests match what you *think* you’re trading.

---

5) If you want, I’ll convert your script

If you send me your current indicator code, I’ll:

- Add the **alertable** returns (LongPulse/ShortPulse, State lines).
- Produce a **ProOrder** version with your exact logic, parameterized stops/targets, time filter, and optional trailing stop.
- Provide a **ProScreener** for multi-symbol alerts.
- Include comments and defaults aligned with your trading hours (GMT+2) and indices you typically trade.

If you’d rather keep the code private, copy your logic into the three templates above as-is. That will give you alerts today, and a working auto-trader immediately after you set up instrument, size, and stops.

You said:

give me more detail about the parts of the script below. How to use it. ~~~~~~~~~~~~~~ // --- One-bar pulses (0/1) you can alert on --- longPulse = longCond ? 1 : 0 shortPulse = shortCond ? 1 : 0 // --- Optional: persistent state (stays 1 while condition holds) --- longState = close > ma ? 1 : 0 shortState = close < ma ? 1 : 0 ~~~~~~~~~~~~

ChatGPT said:

Got it — those four lines are the *bridge* between your visual-only indicator and something ProRealTime can use for **alerts** or as a **feed into strategies**.

Let’s break them down precisely.

---

1. One-bar pulses

```
longPulse  = longCond ? 1 : 0
shortPulse = shortCond ? 1 : 0
```

What it is:

- These create a **momentary "blip"** (value `1`) *only on the exact bar* where your condition first becomes true, otherwise `0`.
- In your example, `longCond` might be something like:

- `longCond = close > ma AND close[1] <= ma[1]`
- which means "price just crossed above the MA on this bar."

How to use:

- Add the indicator to your chart and ensure these `longPulse` / `shortPulse` values are in the **RETURN** statement so PRT can plot them as separate lines.
- Create an **alert** in PRT on these lines:
    - Right-click the `longPulse` line → *Create Alert* → "Crosses above 0.5" → this triggers only when 0 becomes 1.
    - Same for `shortPulse`.
- Because they last only **one bar**, the alert is clean: no multiple triggers while the condition stays true.

When to use pulses:

- You want an **instant alert** at the moment of signal, not continuous firing while the condition remains true.
- You want to **avoid duplicates** if you leave the chart open for hours.

---

2. Persistent state

```
longState  = close > ma ? 1 : 0
shortState = close < ma ? 1 : 0
```

What it is:

- These remain `1` for **every bar** as long as the condition is true.
- In this example:
    - `longState` = 1 whenever price is above the MA (bullish zone).
    - `shortState` = 1 whenever price is below the MA (bearish zone).

How to use:

- Again, add them to the **RETURN** so they're plotted as lines at value `0` or `1`.
- Create an alert like "`longState` crosses above 0.5" → triggers when the state switches from `0` to `1`.
- Or "`longState` is above 0.5" → triggers continuously (use with care, as it can spam alerts every bar until the state changes).

When to use state:

- You want to know **when the market is in a given condition**, not just at the entry signal.
- Useful for things like:
    - Trend filters (“stay long while in bullish state”).
    - Managing multi-bar trades that depend on condition still being true.

---

3. Putting both together in practice

When you `RETURN` them:

```
RETURN longPulse AS "LongPulse", shortPulse AS "ShortPulse", longState AS
"LongState", shortState AS "ShortState"
```

You’ll see **four horizontal lines** in the indicator:

- Two with occasional spikes (pulses)
- Two with step-like shapes that stay high/low for stretches (states)

Practical Alert Setup Example

1. Add the indicator to chart.
2. In the chart legend, click the line name “LongPulse” → **Create alert**:
    - Condition: *Crosses above 0.5*
    - Action: Play sound, pop-up, send email, or place order (if configured in PRT).
3. Do the same for “ShortPulse”.
4. If you also want to be notified when the bullish/bearish state **begins**:
    - Alert on `LongState` crossing above 0.5 (start of bullish zone).
    - Alert on `ShortState` crossing above 0.5 (start of bearish zone).

💡 Key tip:

- Use **pulses** for **trade entries / one-time alerts**.
- Use **state** for **filtering or “in condition” awareness**.
- Always check that your `longCond` and `shortCond` use `[1]` history references where needed, so you’re not triggering on unconfirmed live bar data (avoids false alerts).

If you want, I can rewrite your existing indicator so that it **outputs both pulses and states** for your exact logic, making it ready for both **alerts** and **auto-trading conditions** without repainting. That way you’d have the chart visuals, plus the alertable lines, plus code you can drop into a ProOrder strategy.

You said:

yes rewrite

ChatGPT said:

Understood. Below are two **ready-to-use** scripts:

1. a **signalized INDICATOR** that exposes clean, non-repainting **pulses** and **states** you can attach alerts to, and
2. a **matching STRATEGY** that uses the exact same logic for auto-trading.

Both are structured so you can drop in your own conditions where indicated.

Indicator (ProBuilder): “SignalBridge”

```
// ===============================
// Indicator: SignalBridge
// Purpose : Turn any visual indicator logic into alertable outputs
// Outputs : LongPulse, ShortPulse, LongState, ShortState
// ===============================

DEFPARAM DrawOnLastBarOnly = False
DEFPARAM PreLoadBars = 5000

// ---------- User parameters (adjust as needed) ----------
MAperiod = 50 // example input used in the demo logic below

// ---------- Your calculations (ADD YOUR OWN HERE) ----------
ma = average[MAperiod](close)

// >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
// REPLACE the two demo conditions below with YOUR conditions.
// IMPORTANT: use [1] to confirm a *change* between prior bar and current bar.
// Example demo: "fresh cross" of price vs MA:
// - longCond  triggers when price just moved from below to above the MA
// - shortCond triggers when price just moved from above to below the MA
// >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
longCond  = close > ma AND close[1] <= ma[1]
shortCond = close < ma AND close[1] >= ma[1]

// ---------- One-bar pulses (0/1) for clean one-time alerts ----------
longPulse  = longCond ? 1 : 0
shortPulse = shortCond ? 1 : 0

// ---------- Persistent state (1 while condition holds) ----------
longState  = close > ma ? 1 : 0
shortState = close < ma ? 1 : 0

// ---------- Optional chart visuals (keep or remove) ----------
IF longCond THEN
   DRAWARROWUP(LOW, 0) COLOURED(0,200,0)
ENDIF
IF shortCond THEN
   DRAWARROWDOWN(HIGH, 0) COLOURED(200,0,0)
ENDIF

// ---------- Return alertable lines (names show in legend) ----------
RETURN longPulse AS "LongPulse", shortPulse AS "ShortPulse", longState AS
"LongState", shortState AS "ShortState"
```

How to use the Indicator

- Add it to your chart.

- In the indicator legend, right-click **LongPulse → Create alert →** condition **“crosses above 0.5”**. (That fires exactly on the bar your signal appears.)
- Do the same for **ShortPulse**.
- If you also want alerts when the market *enters* a regime, attach alerts to **LongState** / **ShortState** “crosses above 0.5”.
- Replace the demo `longCond` / `shortCond` with your actual logic (MACD/DMI/Alligator/etc.). Keep the `[1]` confirmations to avoid repainting.

Strategy (ProOrder): “SignalBridge_ProOrder”

```
// ======================================
// Strategy: SignalBridge_ProOrder
// Purpose : Trade the same logic used in the SignalBridge indicator
// Notes   : Replace the demo conditions with your own (keep [1] where appropriate)
// ======================================

DEFPARAM CumulateOrders = False
DEFPARAM PreLoadBars     = 5000

// ---------- Parameters ----------
MAperiod   = 50
tpPoints   = 100   // take profit (points)
slPoints   = 60    // stop loss (points)
tsPoints   = 40    // trailing stop (points)
startTime  = 090000
endTime    = 170000

// ---------- Core calcs (mirror your indicator logic here) ----------
ma = average[MAperiod](close)

// >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
// Replace with YOUR entry conditions (same as in the indicator).
// Keep the [1]-based confirmation so entries only fire on *fresh* signals.
// >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
entryLong  = close > ma AND close[1] <= ma[1]
entryShort = close < ma AND close[1] >= ma[1]

// ---------- Session filter (optional but recommended) ----------
inSession = time >= startTime AND time <= endTime

// ---------- Entries ----------
IF NOT onmarket AND inSession THEN
   IF entryLong THEN
      BUY 1 CONTRACT AT MARKET
   ELSIF entryShort THEN
      SELLSHORT 1 CONTRACT AT MARKET
   ENDIF
ENDIF

// ---------- Risk management ----------
SET TARGET pPROFIT tpPoints
SET STOP   pLOSS   slPoints
SET STOP TRAILING  tsPoints   // correct ProOrder syntax

// ---------- Optional: flatten by end of session ----------
IF time >= endTime AND onmarket THEN
   EXITSHORT AT MARKET
   EXITLONG  AT MARKET
ENDIF
```

How to use the Strategy

- Paste into a **ProOrder** strategy.
- Replace the demo `entryLong` / `entryShort` with the same conditions used in your indicator.
- Set instrument, size, TP/SL/TS, and trading hours.
- Backtest, then run in **PaperTrading** before Live.

Where to plug in your logic (examples)

If your indicator logic is, say, MACD+Trend alignment:

```
// Example: MACD cross + price above a baseline MA:
fast=12; slow=26; sig=9
macdLine = ExponentialAverage[fast](close) - ExponentialAverage[slow](close)
macdSig  = ExponentialAverage[sig](macdLine)

base = average[100](close)

longCond  = macdLine > macdSig AND macdLine[1] <= macdSig[1] AND close > base
shortCond = macdLine < macdSig AND macdLine[1] >= macdSig[1] AND close < base
```

Or if you require multiple tools (e.g., MACD + DMI + Alligator) to **align**, compute each piece first, then define:

```
// Pseudocode for alignment example:
macdUp    = macdLine > macdSig AND macdLine[1] <= macdSig[1]
macdDown  = macdLine < macdSig AND macdLine[1] >= macdSig[1]

dmiUp     = +DI(14) > -DI(14)
dmiDown   = +DI(14) < -DI(14)

alliUp    = jaw > teeth AND teeth > lips
alliDown  = jaw < teeth AND teeth < lips

longCond  = macdUp  AND dmiUp  AND alliUp
shortCond = macdDown AND dmiDown AND alliDown
```

Keep the **[1]** change-detection where you want a *fresh* event.

Why this structure works

- **Pulses** give you one clean alert per signal, no spam.
- **States** let you filter or monitor regime without entering repeatedly.
- **Indicator ↔ Strategy parity** ensures your chart signals match backtests and live behaviour.
- **[1] confirmations** mitigate repainting from intrabar noise.

If you paste your current indicator logic (or the parts you want to trade/alert on), I'll drop it into both blocks precisely and return a final pair you can run as-is.

You said:

PRT accepts the "EXITSHORT AT MARKET" But gives error on "EXITLONG AT MARKET" with AT underlined as the error

ChatGPT said:

You're right—my bad. **`EXITLONG` isn't a valid ProOrder keyword.**
Use **`SELL`** to close a long, and **`EXITSHORT`** to close a short.

Here's the corrected end-of-session flatten block:

```
// Flatten positions at endTime
IF time >= endTime THEN
   IF longonmarket THEN
      SELL AT MARKET            // closes a LONG
   ELSIF shortonmarket THEN
      EXITSHORT AT MARKET       // closes a SHORT
   ENDIF
ENDIF
```

Quick rules to remember in ProOrder:

- **Open long:** `BUY ...`
- **Close long:** `SELL ...`
- **Open short:** `SELLSHORT ...`
- **Close short:** `EXITSHORT ...`

If you want it more defensive (e.g., also cancel any pending orders before flattening):

```
IF time >= endTime THEN
   CANCEL PENDING ORDERS
   IF longonmarket THEN
      SELL AT MARKET
   ELSIF shortonmarket THEN
      EXITSHORT AT MARKET
   ENDIF
ENDIF
```

I'll update any earlier snippets accordingly.

You said:

No errors & Positions are opening. The way I use it when trading on the chart I only buy/sell when the (FILLCOLOR) appears on the chart """ // === Draw + Set Alerts === IF Valid = 1 THEN DRAWRECTANGLE($RangeIndex[RI], $RangeHigh[RI], TotalBars+2000, $RangeLow[RI]) COLOURED(0,0,0,0) FILLCOLOR(0, 204, 255) //(0,0,0,30) """

ChatGPT said:

Perfect—that FILLCOLOR is just a *visual*. You can't trade off the color itself, but you **can** trade off the exact boolean you used to decide to draw it: `Valid`. Do three things:

1. expose `Valid` as **pulse** and **state** in your **indicator** (so you can set alerts), and
2. **gate** your **strategy** entries so they only fire when the zone appears, and
3. (optional) enforce **one trade per zone** and cancel orders when the zone ends.

Below are drop-in blocks.

A) Indicator: make the rectangle "alertable"

Add this next to your drawing code. It converts your `Valid` (the condition that triggers FILLCOLOR) into a one-bar pulse and a persistent state.

```
// === Zone signals from your rectangle logic ===
// Valid = 1 means "zone active" (same condition you use for FILLCOLOR)

IF Valid = 1 AND Valid[1] = 0 THEN
   ZonePulse = 1          // first bar the zone begins
ELSE
   ZonePulse = 0
ENDIF

ZoneState = Valid         // stays 1 while the zone remains valid

// Keep your rectangle draw (visual only)
IF Valid = 1 THEN
   DRAWRECTANGLE($RangeIndex[RI], $RangeHigh[RI], TotalBars+2000, $RangeLow[RI])
COLOURED(0,0,0,0) FILLCOLOR(0,204,255)
ENDIF

// Return lines so PRT can attach alerts
RETURN ZonePulse AS "ZonePulse", ZoneState AS "ZoneState"
```

How to alert:

- Add the indicator to the chart.
- Create alert on **ZonePulse** → *Crosses above 0.5* (fires once per new zone).
- If you want a regime alert, also add **ZoneState** → *Crosses above 0.5* (fires when the zone turns on).

B) Strategy: only trade when the rectangle appears

You have positions opening already. To **only** trade when the FILLCOLOR zone appears, gate your entries with either the first-bar **pulse** (strict) or the whole-period **state** (looser).

```
// --------- Build the same Valid the indicator uses ----------
/* Paste the exact same Valid calculation here that you use for the rectangle.
   Do NOT rely on the drawing; recompute the logic. */

IF Valid = 1 AND Valid[1] = 0 THEN
   ZonePulse = 1
ELSE
   ZonePulse = 0
ENDIF

ZoneState = Valid

// --------- Choose behavior ----------
UseFirstBarOnly = TRUE     // TRUE: only on the bar the zone starts (matches how you
"trade when it appears")
                           // FALSE: allow entries anytime while zone is active

canTrade = (UseFirstBarOnly AND ZonePulse = 1) OR ((NOT UseFirstBarOnly) AND
ZoneState = 1)

// --------- Your existing entry logic (example names) ----------
entryLong  = { your existing long condition }
```

```
entryShort = { your existing short condition }

// --------- Gate entries with the zone condition ----------
IF NOT onmarket AND inSession AND canTrade THEN
   IF entryLong THEN
      BUY 1 CONTRACT AT MARKET
   ELSIF entryShort THEN
      SELLSHORT 1 CONTRACT AT MARKET
   ENDIF
ENDIF

// --------- Risk management (as you had) ----------
SET TARGET pPROFIT tpPoints
SET STOP   pLOSS   slPoints
SET STOP TRAILING  tsPoints

// --------- Flatten at session end (correct syntax) ----------
IF time >= endTime THEN
   CANCEL PENDING ORDERS
   IF longonmarket THEN
      SELL AT MARKET
   ELSIF shortonmarket THEN
      EXITSHORT AT MARKET
   ENDIF
ENDIF
```

C) (Optional) One trade per zone + cleanups

If you truly want **exactly one** trade per rectangle, track a zone ID and refuse to re-enter until a new zone begins:

```
// Increment zone ID when a new zone starts
ONCE zoneID = 0
ONCE lastTradedZoneID = -1

IF ZonePulse = 1 THEN
   zoneID = zoneID + 1
ENDIF

// Only allow a trade if we haven't traded this zone yet
canTradeThisZone = canTrade AND (zoneID > lastTradedZoneID)

IF NOT onmarket AND inSession AND canTradeThisZone THEN
   IF entryLong THEN
      BUY 1 CONTRACT AT MARKET
      lastTradedZoneID = zoneID
   ELSIF entryShort THEN
      SELLSHORT 1 CONTRACT AT MARKET
      lastTradedZoneID = zoneID
   ENDIF
ENDIF

// If the zone ends, you can optionally cancel any pending orders
IF ZoneState = 0 AND ZoneState[1] = 1 THEN
   CANCEL PENDING ORDERS
ENDIF
```

Practical notes

- You cannot detect **FILLCOLOR** itself in code; you must use the **same boolean (`Valid`)** that decides to draw it.
- Keep the **Valid** computation identical in both indicator and strategy; avoid any centered/forward-looking calcs that can repaint.
- For your use case ("I buy/sell when it appears"), set `UseFirstBarOnly = TRUE` so entries occur only on the **first bar** the rectangle shows.
- If your rectangle implies **direction** (bullish vs bearish zone), add a simple bias filter:

```
• // Example: only buy in bullish zones, only short in bearish zones
• // bullishZone / bearishZone are booleans you compute from your logic
• IF NOT onmarket AND inSession AND ZonePulse = 1 THEN
•    IF bullishZone AND entryLong THEN BUY 1 CONTRACT AT MARKET
•    IF bearishZone AND entryShort THEN SELLSHORT 1 CONTRACT AT MARKET
• ENDIF
• 
```

If you paste the few lines where you compute `Valid` (and how you decide RI), I'll wire it precisely into the indicator + strategy so it behaves *exactly* like your FILLCOLOR rectangle—just with alerts and auto-trading hooks.

You said:

Indicator: make the rectangle "alertable" Get this error when adding indicator: ""The following variable is undefined: RI, TotalBars, Valid""

ChatGPT said:

You're getting that error because `RI`, `TotalBars`, and `Valid` were placeholders from your own script—they're not built-ins. Here's how to fix it *right now* and make the rectangle + alerts work without any undefined variables.

What those names were meant to be

- **`Valid`**: your boolean "zone active" flag (the same thing you used to decide when to `FILLCOLOR`).
- **`RI`** / **`$RangeIndex[RI]`** / **`$RangeHigh[RI]`** / **`$RangeLow[RI]`**: your own arrays/pointers tracking where a zone started and its bounds.
- **`TotalBars`**: not a PRT keyword. Use `barindex` as the current bar number. To "extend into the future," use `barindex + 2000`.

Drop-in, compile-safe indicator (adapter)

This version **doesn't reference** `RI`, `TotalBars`, or your arrays. It derives a rectangle from the *start bar* of the zone and maintains its high/low while the zone stays valid. Replace the demo `Valid` line with your real condition.

```
// ===============================
// Indicator: ZoneAlertAdapter
// Purpose : Make your FILLCOLOR rectangle alertable + tradeable
// ===============================

DEFPARAM DrawOnLastBarOnly = False
DEFPARAM PreLoadBars = 5000

// --- 1) DEFINE YOUR ZONE CONDITION (replace this demo) ---
MAperiod = 50
```

```
ma = average[MAperiod](close)

// DEMO Valid: zone begins when price freshly crosses above MA.
// >>> Replace with your own 'Valid' condition from your script. <<<
Valid = (close > ma AND close[1] <= ma[1]) ? 1 : 0

// --- 2) PULSES & STATE FOR ALERTS ---
ZonePulse = (Valid = 1 AND Valid[1] = 0) ? 1 : 0      // fires on the first bar of a new zone
ZoneState = Valid                                      // stays 1 while zone is active

// --- 3) TRACK THE RECTANGLE BOUNDS ---
ONCE zoneStart = 0
ONCE zoneHigh  = 0.0
ONCE zoneLow   = 0.0

IF Valid = 1 THEN
   IF Valid[1] = 0 THEN
      // Zone just started
      zoneStart = barindex
      zoneHigh  = high
      zoneLow   = low
   ELSE
      // Zone continues: keep envelope if you want the box to grow with extremes
      zoneHigh  = MAX(zoneHigh, high)
      zoneLow   = MIN(zoneLow,  low)
   ENDIF

   // Draw the rectangle from its start to "future"
   DRAWRECTANGLE(zoneStart, zoneHigh, barindex + 2000, zoneLow) COLOURED(0,0,0,0) FILLCOLOR(0,204,255)
ENDIF

// --- 4) RETURN LINES FOR ALERTS ---
RETURN ZonePulse AS "ZonePulse", ZoneState AS "ZoneState"
```

How to use it

1. Paste this as a new indicator, compile.
2. If you don't want the demo MA logic, **replace the `Valid = ...` line** with the exact boolean you use for your rectangle in your current script.
3. Add indicator to chart.
4. Create alerts:
    - On **ZonePulse** → *Crosses above 0.5* (fires once per new zone).
    - Optionally on **ZoneState** → *Crosses above 0.5* (fires when zone turns on).

Notes
• Drawing a rectangle every bar while `Valid=1` is normal in PRT. If you prefer a static rectangle that doesn't "grow," **remove** the `MAX`/`MIN` lines and keep the start bar's `high`/`low`.
• If your zone has its own stored high/low, you can set `zoneHigh`/`zoneLow` from those instead of using `high`/`low`.

If you want to wire it to your original variables

Map your names like this:

- `TotalBars` → `barindex`
- `$RangeIndex[RI]` → `zoneStart`

- `$RangeHigh[RI]` → `zoneHigh`
- `$RangeLow[RI]` → `zoneLow`

Then the draw call becomes:

```
DRAWRECTANGLE(zoneStart, zoneHigh, barindex + 2000, zoneLow) COLOURED(0,0,0,0)
FILLCOLOR(0,204,255)
```

Strategy gating (so you only trade when the box appears)

In your **strategy**, reuse the *same* `Valid` logic and gate entries:

```
// Build the same signals
ZonePulse = (Valid = 1 AND Valid[1] = 0) ? 1 : 0
ZoneState = Valid

UseFirstBarOnly = TRUE  // TRUE: enter only when the zone appears (matches your
manual use)

canTrade = (UseFirstBarOnly AND ZonePulse = 1) OR ((NOT UseFirstBarOnly) AND
ZoneState = 1)

IF NOT onmarket AND inSession AND canTrade THEN
   IF entryLong THEN
      BUY 1 CONTRACT AT MARKET
   ELSIF entryShort THEN
      SELLSHORT 1 CONTRACT AT MARKET
   ENDIF
ENDIF
```

If you paste the few lines where you currently compute `Valid` (and any `$Range*` values), I'll splice them into the adapter so it mirrors your rectangle logic exactly and compiles cleanly.

You said:

Get error on

ChatGPT said:

Thanks—likely causes are (a) the ternary operator `? :` (some PRT installs choke on it), and/or (b) `COLOURED(0,0,0,0)` (PRT expects **three** RGB args, not four). Also I removed any non-built-ins like `RI`/`TotalBars`.

Use this **compile-safe** indicator:

```
// ===============================
// Indicator: ZoneAlertAdapter_v2
// Purpose : Make your FILLCOLOR rectangle alertable (no undefined vars)
// ===============================

DEFPARAM DrawOnLastBarOnly = False
DEFPARAM PreLoadBars = 5000

// --- Demo input (replace with yours) ---
MAperiod = 50
ma = average[MAperiod](close)
```

```
// --- Valid: REPLACE this block with your real zone condition ---
IF close > ma AND close[1] <= ma[1] THEN
   Valid = 1
ELSE
   Valid = 0
ENDIF

// --- Signals for alerts ---
IF Valid = 1 AND Valid[1] = 0 THEN
   ZonePulse = 1     // 1 only on the first bar the zone appears
ELSE
   ZonePulse = 0
ENDIF
ZoneState = Valid    // 1 while the zone stays active

// --- Track rectangle bounds (no external arrays) ---
ONCE zoneStart = 0
ONCE zoneHigh  = 0.0
ONCE zoneLow   = 0.0

IF Valid = 1 THEN
   IF Valid[1] = 0 THEN
      // zone just started
      zoneStart = barindex
      zoneHigh  = high
      zoneLow   = low
   ELSE
      // zone continues; grow the box with extremes (optional)
      IF high > zoneHigh THEN
         zoneHigh = high
      ENDIF
      IF low < zoneLow THEN
         zoneLow = low
      ENDIF
   ENDIF

   // Draw the rectangle from start to "future"
   DRAWRECTANGLE(zoneStart, zoneHigh, barindex + 2000, zoneLow) COLOURED(0,0,0)
FILLCOLOR(0,204,255)
ENDIF

// --- Expose lines for alerts ---
RETURN ZonePulse AS "ZonePulse", ZoneState AS "ZoneState"
```

How to use it

1. Paste as a new **indicator** (ProBuilder), compile.
2. Replace the **Valid** block with your real zone condition (the same boolean you use before calling `FILLCOLOR` in your original code).
3. Add to chart → in the legend, create alerts:
   - **ZonePulse** → "crosses above 0.5" (one alert per new rectangle).
   - **ZoneState** → "crosses above 0.5" (fires when the zone turns on).

If you still get an error, paste the exact message/line. Common pitfalls I've eliminated here:

- No `? :` operator (replaced with IF/ELSE).
- `COLOURED` now has **3** parameters.
- No undefined names (`RI`, `TotalBars`, custom arrays).

When you confirm this compiles, I'll mirror the same `Valid` logic into your **strategy** so entries only happen when the rectangle appears (first bar or entire zone—your choice).

You said:

Get error on "Valid" Get error on "preloadbars" Some detail about "Preloadbars" https://www.prorealcode.com/documentation/preloadbars/

ChatGPT said:

Thanks—two issues:

1. **`PRELOADBARS`**: that directive belongs to **ProBacktest/ProOrder (strategies)**, not (most) indicator contexts. In **indicators**, use `CALCULATEONLASTBARS` (and optionally `DRAWONLASTBARONLY`). See the official ProBuilder manual entries for those two directives, and the PRELOADBARS doc in the ProBacktest manual. ProRealTimeProRealCode
2. **`Valid` undefined**: ProRealTime wants a variable to exist **before** you reference `Valid[1]`. Initialize it once, then compute it each bar, then derive your pulses/states.

Below is a **compile-safe indicator** you can drop in. It avoids `PRELOADBARS`, initializes variables, and keeps the order of operations correct.

```
// ================================
// Indicator: ZoneAlertAdapter_v3
// Purpose : Make your rectangle "alertable" without undefined vars
// ================================

DEFPARAM DRAWONLASTBARONLY = False
DEFPARAM CALCULATEONLASTBARS = 2000   // indicator-friendly history window
:contentReference[oaicite:1]{index=1}

// ---- SAFE INITIALIZATION (prevents 'undefined' when using [1]) ----
ONCE Valid     = 0
ONCE ZonePulse = 0
ONCE ZoneState = 0
ONCE zoneStart = 0
ONCE zoneHigh  = 0.0
ONCE zoneLow   = 0.0

// ---- DEMO INPUT (replace with your logic) ----
MAperiod = 50
ma = average[MAperiod](close)

// ---- YOUR ZONE CONDITION (replace this block) ----
// Example: zone is 'valid' when a fresh cross above MA occurs
IF close > ma AND close[1] <= ma[1] THEN
   Valid = 1
ELSE
   Valid = 0
ENDIF

// ---- PULSE (one bar) & STATE (persistent) ----
IF Valid = 1 AND Valid[1] = 0 THEN
   ZonePulse = 1
ELSE
   ZonePulse = 0
ENDIF
ZoneState = Valid

// ---- TRACK RECTANGLE BOUNDS & DRAW ----
IF Valid = 1 THEN
   IF Valid[1] = 0 THEN
      // zone just started
      zoneStart = barindex
      zoneHigh  = high
      zoneLow   = low
   ELSE
      // grow box with extremes (optional)
      IF high > zoneHigh THEN
         zoneHigh = high
      ENDIF
      IF low < zoneLow THEN
         zoneLow = low
      ENDIF
   ENDIF
   // Both 3-component and 4-component alpha syntax are accepted in v10.3+
   DRAWRECTANGLE(zoneStart, zoneHigh, barindex + 2000, zoneLow) COLOURED(0,0,0)
FILLCOLOR(0,204,255)
ENDIF

// ---- RETURN LINES FOR ALERTS ----
RETURN ZonePulse AS "ZonePulse", ZoneState AS "ZoneState"
```

How to use

- Add the indicator. Create alerts on:
    - **ZonePulse** → “crosses above 0.5” (fires once when the rectangle appears).
    - **ZoneState** → “crosses above 0.5” (fires when the zone turns on).
- Replace the `Valid` block with *your* rectangle condition. Keep the initialization and ordering.

About `PRELOADBARS` vs `CALCULATEONLASTBARS`

- **Strategies (ProBacktest/ProOrder):** use `DEFPARAM PRELOADBARS = N` to ensure enough prior candles are loaded for indicator calculations in a trading system. ProRealCode
- **Indicators (ProBuilder):** use `DEFPARAM CALCULATEONLASTBARS = N` to limit computation to the last *N* bars (performance) and `DEFPARAM DRAWONLASTBARONLY = true/false` to control drawing scope. ProRealTime

If you paste your actual `Valid` logic (and any zone bounds you already compute), I’ll wire it in exactly so the indicator and your auto-trading strategy share identical, non-repainting signals.

You said:

what do you mean by "I’ll wire it in exactly so the indicator and your auto-trading strategy share identical, non-repainting signals."

ChatGPT said:

I meant this, very literally: we take the **exact same boolean logic** that makes your rectangle appear (your `Valid` condition + any supporting calculations) and we **paste that block, unchanged, into both** your indicator and your ProOrder strategy. Then we derive the same `ZonePulse`/`ZoneState` from it in both places. That way:

- the indicator draws (and can alert) on those signals, and
- the strategy trades on the **same bars** the indicator signalled,
- with **no repainting** because signals are confirmed on the prior bar state (`[1]`).

Here’s the pattern to make that airtight.

Single-source-of-truth signal block

Copy this block **verbatim** into both your indicator and your strategy. Replace the “YOUR LOGIC” section with your real rectangle/zone logic (the thing that used to drive `FILLCOLOR`).

```
// ===== BEGIN CORE SIGNAL BLOCK (copy verbatim into indicator & strategy) =====

// 0) Safe init so we can use [1] without “undefined”
ONCE Valid     = 0
ONCE ZonePulse = 0
ONCE ZoneState = 0

// 1) YOUR LOGIC (exactly what decides when the zone is active)
// Example placeholder; replace with your real condition(s).
// Make sure this evaluates on the CLOSE of the bar and uses [1] where needed
// so it only flips when the *previous* bar was different.
IF <your conditions that define the zone> THEN
   Valid = 1
ELSE
   Valid = 0
ENDIF

// 2) Non-repainting signals built from that logic
IF Valid = 1 AND Valid[1] = 0 THEN
   ZonePulse = 1      // one-bar pulse: fires on the first bar a new zone begins
ELSE
   ZonePulse = 0
ENDIF

ZoneState = Valid      // persistent: 1 while the zone remains active

// ===== END CORE SIGNAL BLOCK =====
```

Now, the **indicator** adds drawings/returns around that same block:

```
DEFPARAM DRAWONLASTBARONLY = False
DEFPARAM CALCULATEONLASTBARS = 2000

// (paste the CORE SIGNAL BLOCK here)

// Optional: draw your rectangle using your own bounds
// (or a simple version based on start/high/low)
ONCE zoneStart = 0
ONCE zoneHigh  = 0.0
ONCE zoneLow   = 0.0

IF ZoneState = 1 THEN
   IF ZonePulse = 1 THEN
      zoneStart = barindex
      zoneHigh  = high
      zoneLow   = low
   ELSE
      zoneHigh = MAX(zoneHigh, high)
      zoneLow  = MIN(zoneLow,  low)
   ENDIF
   DRAWRECTANGLE(zoneStart, zoneHigh, barindex + 2000, zoneLow) COLOURED(0,0,0)
FILLCOLOR(0,204,255)
ENDIF

RETURN ZonePulse AS "ZonePulse", ZoneState AS "ZoneState"
```

And the **strategy** trades off the exact same signals:

```
DEFPARAM PRELOADBARS = 5000
DEFPARAM CumulateOrders = False

// (paste the CORE SIGNAL BLOCK here)

UseFirstBarOnly = TRUE     // TRUE: enter only when the zone appears (ZonePulse)
inSession       = time >= 090000 AND time <= 170000

canTrade = (UseFirstBarOnly AND ZonePulse = 1) OR ((NOT UseFirstBarOnly) AND
ZoneState = 1)

// Your entry direction filters/conditions:
entryLong  = <your long condition(s) built on the same inputs>
entryShort = <your short condition(s)>

// Gate entries with the zone
IF NOT onmarket AND inSession AND canTrade THEN
   IF entryLong THEN
      BUY 1 CONTRACT AT MARKET
   ELSIF entryShort THEN
      SELLSHORT 1 CONTRACT AT MARKET
   ENDIF
ENDIF

SET TARGET pPROFIT 100
SET STOP   pLOSS   60
SET STOP TRAILING 40

// Flatten at end of session
IF time >= 170000 THEN
   CANCEL PENDING ORDERS
   IF longonmarket THEN
      SELL AT MARKET
   ELSIF shortonmarket THEN
      EXITSHORT AT MARKET
   ENDIF
ENDIF
```

Why this guarantees “identical, non-repainting”

- **Identical:** the **same lines of code** determine `Valid`, `ZonePulse`, `ZoneState` in both files. No divergence, no “I changed it here but forgot there.”
- **Non-repainting:** you confirm state changes with `[1]` (e.g., `Valid = 1 AND Valid[1] = 0`), so signals only fire when a bar *finishes* and flips state. No intrabar peeking, no future data, no centered/forward-looking averages.
- **Execution parity:** the indicator’s **ZonePulse** spikes exactly on bars where the strategy can place orders (end-of-bar). When you see a pulse on the chart, you should see a backtest trade on that bar too.

Quick QA checklist

- Same inputs/parameters in both (periods, thresholds, time filters).
- No forward-looking functions (don’t reference future bars; avoid centered MAs).
- Use `[1]` for “fresh” events (crosses, start/stop of zones).
- Strategy uses `DEFPARAM PRELOADBARS` high enough for your longest lookback.
- Indicator uses `DEFPARAM CALCULATEONLASTBARS` (optional) for performance; it does **not** affect the logic, only compute span.

- Validate parity: add `PRINT("ZP ", time)` in strategy when `ZonePulse=1` and confirm it matches pulses on the indicator.

If you paste your current **zone/rectangle condition** (the lines that set `Valid` in your indicator), I'll drop them into that CORE SIGNAL BLOCK for you, and you'll have a matched pair you can run immediately.

You said:

I'm bit confused. Write me all separately what should go in to the indicator and what should be in the strategy.

ChatGPT said:

Understood. Here's a clean, **copy-paste** pair — one **Indicator** and one **Strategy** — plus precise instructions on what to edit in each.

Indicator — "ZoneAlert_Indicator"

Purpose: draw your rectangle and expose two alertable lines:

- `ZonePulse` = 1 **only on the first bar** the zone appears (one-bar spike).
- `ZoneState` = 1 **for every bar** while the zone is active.

Edit the **VALIDITY LOGIC** block to match the condition that currently triggers your FILLCOLOR.

```
// ===============================
// Indicator: ZoneAlert_Indicator
// Turns your zone into alertable outputs + rectangle
// ===============================

DEFPARAM DrawOnLastBarOnly = False
DEFPARAM CalculateOnLastBars = 2000   // for performance; adjust/omit if you want
full history

// ---------- SAFE INITIALISATION ----------
ONCE Valid     = 0
ONCE ZonePulse = 0
ONCE ZoneState = 0
ONCE zoneStart = 0
ONCE zoneHigh  = 0.0
ONCE zoneLow   = 0.0

// ---------- [EDIT THIS] VALIDITY LOGIC (your zone condition) ----------
// Demo example below (fresh cross UP of a 50-MA). Replace with your real logic.
MAperiod = 50
ma = average[MAperiod](close)

IF close > ma AND close[1] <= ma[1] THEN
   Valid = 1
ELSE
   Valid = 0
ENDIF

// ---------- SIGNALS ----------
IF Valid = 1 AND Valid[1] = 0 THEN
   ZonePulse = 1
```

```
ELSE
   ZonePulse = 0
ENDIF
ZoneState = Valid

// ---------- RECTANGLE DRAW (simple self-contained version) ----------
IF ZoneState = 1 THEN
   IF ZonePulse = 1 THEN
      // zone just started
      zoneStart = barindex
      zoneHigh  = high
      zoneLow   = low
   ELSE
      // grow with extremes while zone remains active (optional)
      IF high > zoneHigh THEN
         zoneHigh = high
      ENDIF
      IF low < zoneLow THEN
         zoneLow = low
      ENDIF
   ENDIF

   // x1=zoneStart, y1=zoneHigh, x2=future, y2=zoneLow
   DRAWRECTANGLE(zoneStart, zoneHigh, barindex + 2000, zoneLow) COLOURED(0,0,0)
FILLCOLOR(0,204,255)
ENDIF

// ---------- RETURN LINES FOR ALERTS ----------
RETURN ZonePulse AS "ZonePulse", ZoneState AS "ZoneState"
```

How to use the Indicator

1. Paste & compile as a **new indicator**.
2. Replace the **VALIDITY LOGIC** block with the exact condition that used to trigger your `FILLCOLOR`.
3. Add to your chart → create alerts:
   - **ZonePulse** → *Crosses above 0.5* (fires once per new rectangle).
   - **ZoneState** → *Crosses above 0.5* (fires when the zone turns on; optional).

Strategy — "ZoneGate_Strategy"

Purpose: trade **the same logic** used by the indicator. You choose whether to enter only on the **first bar** of a new zone (`ZonePulse`) or **any bar** while the zone is active (`ZoneState`).

Again: edit the **VALIDITY LOGIC** block to **the same code** you used in the indicator.

```
// ======================================
// Strategy: ZoneGate_Strategy
// Trades only when your zone appears (or while active)
// ======================================

DEFPARAM PreLoadBars = 5000
DEFPARAM CumulateOrders = False

// ---------- PARAMETERS ----------
tpPoints   = 100    // take profit (points)
slPoints   = 60     // stop loss (points)
tsPoints   = 40     // trailing stop (points)
```

```
startTime  = 090000 // session start (edit for your market)
endTime    = 170000 // session end

UseFirstBarOnly = TRUE   // TRUE: trade only when zone appears (ZonePulse)
                         // FALSE: allow entries while zone remains active
(ZoneState)

EnableLongs  = 1         // set 0 to disable longs
EnableShorts = 0         // set 1 to also allow shorts (example keeps shorts off by
default)

// ---------- SAFE INITIALISATION ----------
ONCE Valid     = 0
ONCE ZonePulse = 0
ONCE ZoneState = 0

// ---------- [EDIT THIS] VALIDITY LOGIC (SAME AS INDICATOR) ----------
// Demo example (fresh cross UP of a 50-MA). Replace with your real logic.
MAperiod = 50
ma = average[MAperiod](close)

IF close > ma AND close[1] <= ma[1] THEN
   Valid = 1
ELSE
   Valid = 0
ENDIF

// ---------- SIGNALS ----------
IF Valid = 1 AND Valid[1] = 0 THEN
   ZonePulse = 1
ELSE
   ZonePulse = 0
ENDIF
ZoneState = Valid

// ---------- SESSION FILTER ----------
inSession = time >= startTime AND time <= endTime

// ---------- GATING ----------
canTrade = (UseFirstBarOnly AND ZonePulse = 1) OR ((NOT UseFirstBarOnly) AND
ZoneState = 1)

// ---------- DIRECTIONAL ENTRY LOGIC ----------
// For a minimal, “trade-the-zone” prototype:
//  - Longs: enter when allowed by canTrade and longs are enabled.
//  - Shorts: same, if enabled.
IF NOT onmarket AND inSession AND canTrade THEN
   IF EnableLongs = 1 THEN
      BUY 1 CONTRACT AT MARKET
   ELSIF EnableShorts = 1 THEN
      SELLSHORT 1 CONTRACT AT MARKET
   ENDIF
ENDIF

// ---------- RISK MANAGEMENT ----------
SET TARGET pPROFIT tpPoints
SET STOP   pLOSS   slPoints
SET STOP TRAILING  tsPoints

// ---------- FLATTEN AT END OF SESSION ----------
IF time >= endTime THEN
   CANCEL PENDING ORDERS
   IF longonmarket THEN
      SELL AT MARKET            // closes a LONG
   ELSIF shortonmarket THEN
      EXITSHORT AT MARKET       // closes a SHORT
   ENDIF
ENDIF
```

How to use the Strategy

1. Paste & compile as a **ProOrder strategy**.
2. Replace the **VALIDITY LOGIC** block with the **same** code you used in the indicator.
3. Set `UseFirstBarOnly = TRUE` if you only want entries on the very first bar the rectangle appears (this matches your manual approach with FILLCOLOR).
4. Choose directions: `EnableLongs` / `EnableShorts`.
5. Adjust `tpPoints`, `slPoints`, `tsPoints`, `startTime`, `endTime`.
6. Backtest → PaperTrade → Live.

What you must keep identical in both files

- The entire **VALIDITY LOGIC** block.
- The computation of `ZonePulse` and `ZoneState`.

This guarantees the bars where the indicator fires are the same bars the strategy is allowed to trade — no surprises, no repainting.

Optional add-ons (strategy)

- **One trade per zone:** track a zone counter and block re-entries until a new zone starts.
- **Bias (bullish vs bearish) zones:** if your zone can be tagged bullish/bearish, gate longs/shorts accordingly instead of `EnableLongs/EnableShorts`.
- **Cooldown:** require N bars between trades to prevent flip-flops in chop.

If you share the lines that define your current zone (the logic behind your FILLCOLOR), I'll drop them straight into both blocks and return a final pair that mirrors your chart exactly.

You said:

1.) I dont see any "VALIDITY LOGIC" in the "ZoneGate_Strategy" 2.) Error on the "TRUE" in : UseFirstBarOnly = TRUE 3.) Error on the "PENDING" in: CANCEL PENDING ORDERS

ChatGPT said:

Good catches. Three fixes up front:

1. I've made the **VALIDITY LOGIC** block impossible to miss — it's now wrapped with big "EDIT HERE" markers.
2. Use **1/0** (integers) instead of TRUE/FALSE.
3. The correct cancel syntax is **`CANCEL PENDINGORDERS`** (no space).

Below are clean, compile-safe versions.

Indicator — ZoneAlert_Indicator (final)

```
// ===============================
// Indicator: ZoneAlert_Indicator
// Exposes ZonePulse (one bar) + ZoneState (persistent) and draws the rectangle
// ===============================

DEFPARAM DrawOnLastBarOnly = False
DEFPARAM CalculateOnLastBars = 2000

// ---- SAFE INIT ----
ONCE Valid     = 0
ONCE ZonePulse = 0
ONCE ZoneState = 0
ONCE zoneStart = 0
ONCE zoneHigh  = 0.0
ONCE zoneLow   = 0.0

// ==============================================
// =============== EDIT HERE: VALIDITY LOGIC ======
// Replace the demo with YOUR exact condition that
// previously triggered FILLCOLOR in your indicator.
// ----------------------------------------------
// Demo: fresh cross UP of a 50-MA
MAperiod = 50
ma = average[MAperiod](close)

IF close > ma AND close[1] <= ma[1] THEN
   Valid = 1
ELSE
   Valid = 0
ENDIF
// ============= END VALIDITY LOGIC ===============
// ==============================================

// ---- SIGNALS ----
IF Valid = 1 AND Valid[1] = 0 THEN
   ZonePulse = 1
ELSE
   ZonePulse = 0
ENDIF
ZoneState = Valid

// ---- RECTANGLE DRAW (self-contained bounds) ----
IF ZoneState = 1 THEN
   IF ZonePulse = 1 THEN
      zoneStart = barindex
      zoneHigh  = high
      zoneLow   = low
   ELSE
      IF high > zoneHigh THEN
         zoneHigh = high
      ENDIF
      IF low < zoneLow THEN
         zoneLow = low
      ENDIF
   ENDIF
   // Extend to the right: barindex + 2000
   DRAWRECTANGLE(zoneStart, zoneHigh, barindex + 2000, zoneLow) COLOURED(0,0,0)
FILLCOLOR(0,204,255)
ENDIF

// ---- RETURN LINES FOR ALERTS ----
RETURN ZonePulse AS "ZonePulse", ZoneState AS "ZoneState"
```

Use: add to chart → create alerts on:

- `ZonePulse` → “crosses above 0.5” (one alert when the box appears)
- `ZoneState` → “crosses above 0.5” (alert when zone turns on)

Strategy — ZoneGate_Strategy (final)

```
// ======================================
// Strategy: ZoneGate_Strategy
// Trades only when the zone appears (or while active)
// ======================================

DEFPARAM PreLoadBars = 5000
DEFPARAM CumulateOrders = False

// ---- PARAMETERS ----
tpPoints   = 100
slPoints   = 60
tsPoints   = 40
startTime  = 090000
endTime    = 170000

UseFirstBarOnly = 1     // 1 = enter only when the zone appears (ZonePulse)
                        // 0 = allow entries any bar while zone is active (ZoneState)

EnableLongs  = 1        // 1=allow longs, 0=disable
EnableShorts = 0        // 1=allow shorts, 0=disable

// ---- SAFE INIT ----
ONCE Valid     = 0
ONCE ZonePulse = 0
ONCE ZoneState = 0

// =================================================
// =============== EDIT HERE: VALIDITY LOGIC ======
// Paste the SAME logic you used in the indicator.
// -------------------------------------------------
// Demo: fresh cross UP of a 50-MA
MAperiod = 50
ma = average[MAperiod](close)

IF close > ma AND close[1] <= ma[1] THEN
   Valid = 1
ELSE
   Valid = 0
ENDIF
// ============= END VALIDITY LOGIC ===============
// =================================================

// ---- SIGNALS (must match indicator) ----
IF Valid = 1 AND Valid[1] = 0 THEN
   ZonePulse = 1
ELSE
   ZonePulse = 0
ENDIF
ZoneState = Valid

// ---- SESSION FILTER ----
inSession = time >= startTime AND time <= endTime

// ---- ENTRY GATING ----
```

```
canTrade = (UseFirstBarOnly = 1 AND ZonePulse = 1) OR (UseFirstBarOnly = 0 AND
ZoneState = 1)

// ---- DIRECTIONAL ENTRY ----
IF NOT onmarket AND inSession AND canTrade THEN
   IF EnableLongs = 1 THEN
      BUY 1 CONTRACT AT MARKET
   ELSIF EnableShorts = 1 THEN
      SELLSHORT 1 CONTRACT AT MARKET
   ENDIF
ENDIF

// ---- RISK MANAGEMENT ----
SET TARGET pPROFIT tpPoints
SET STOP   pLOSS   slPoints
SET STOP TRAILING  tsPoints

// ---- FLATTEN AT END OF SESSION ----
IF time >= endTime THEN
   CANCEL PENDINGORDERS
   IF longonmarket THEN
      SELL AT MARKET              // close LONG
   ELSIF shortonmarket THEN
      EXITSHORT AT MARKET         // close SHORT
   ENDIF
ENDIF
```

What you must keep identical

- The **VALIDITY LOGIC** block.
- The **signals** (`ZonePulse`, `ZoneState`).

That guarantees the same bars that show your FILLCOLOR/ZonePulse in the indicator are the bars the strategy is allowed to act on. If you want me to drop your real zone condition in both files, paste that snippet and I'll wire it in.

You said:

error "pendingorders": CANCEL PENDINGORDERS

ChatGPT said:

You're getting that error because **there is no `CANCEL PENDINGORDERS` instruction in ProOrder**. That was on me. In ProRealTime:

- **Pending entry orders (STOP/LIMIT)** expire automatically at the **next bar close** unless you re-issue them in your code. You generally don't "cancel" them manually. ProRealCode
- If you want the platform to **cancel all pending orders and close positions at a time**, use **`DEFPARAM FlatAfter = HHMMSS`**. That parameter explicitly cancels pendings and flattens after the time you set. ProRealTime
- To **remove protective orders you set in code** (targets/stops/trailing stops), set them to **0** (this deactivates them). ProRealTime+1

Here are two clean ways to handle end-of-session flattening **without** the invalid command:

Option A — one-liner policy (preferred)

Put this at the top of your strategy:

```
DEFPARAM FlatAfter = 170000  // set to your session end
```

That’s it. After 17:00:00, ProOrder will **cancel all pending orders and close any open positions** for this system. ProRealTime

Option B — manual block (if you need custom timing/logic)

```
IF time >= endTime THEN
   // 1) deactivate any protection orders you set earlier
   SET TARGET pPROFIT 0
   SET STOP   pLOSS   0
   SET STOP   TRAILING 0   // also valid: SET STOP pTRAILING 0

   // 2) close any open position
   IF longonmarket THEN
      SELL AT MARKET
   ELSIF shortonmarket THEN
      EXITSHORT AT MARKET
   ENDIF

   // 3) do NOT re-place any pending STOP/LIMIT entries after this point.
   //    They expire automatically at next bar if not re-issued.
ENDIF
```

Why it works:

- Step 1 clears targets/stops/trailing stops you created in code by setting them to **0** (the documented way to deactivate). ProRealTime+1
- Step 3 leverages ProOrder’s behavior: **pending entries auto-cancel at bar close unless renewed**. ProRealCode

TL;DR

Delete the `CANCEL PENDINGORDERS` line. Use **`FlatAfter`** for automatic cleanup, or manually close positions and set stops/targets to **0** and simply stop re-issuing any pending orders. This is the supported, compile-safe way to flatten and clear orders in ProOrder.

You said:

No more errors. I'll test it and let you know.

ChatGPT said:

Great. Here’s a tight test plan so you don’t waste time.

What to verify (in order)

1. Indicator-only sanity

- Add **ZoneAlert_Indicator** to your chart.
- Create an alert on **ZonePulse → crosses above 0.5**.
- Visually confirm: each time the rectangle first appears, ZonePulse spikes **once** and the alert fires.

2. Strategy parity

- Load **ZoneGate_Strategy** on the **same instrument/timeframe**.
- Make sure the **VALIDITY LOGIC** block in the strategy is **identical** to the indicator.
- Set `UseFirstBarOnly = 1`.
- Backtest quickly: each ZonePulse (from the indicator) should line up with a strategy entry on the same bar close.

3. Session & flattening

- Confirm `startTime`/`endTime` (GMT+2 for you).
- Prefer `DEFPARAM FlatAfter = endTime` (one line, zero headache). If you didn't add it, your manual block is fine.

4. Risk controls

- Confirm `SET TARGET`, `SET STOP pLOSS`, `SET STOP TRAILING` are actually applied (the backtest detail should show active protections).
- Tune points to match the instrument's point value.

Quick troubleshooting

- No rectangles / no alerts: your **VALIDITY LOGIC** isn't firing on this timeframe/instrument. Test with a simpler condition (e.g., 5-bar breakout) to isolate.
- Multiple trades per rectangle: set `UseFirstBarOnly = 1`. If still noisy, we can add the **one-trade-per-zone** guard I posted earlier.
- Entries not matching indicators: you edited the validity block in one file but not the other, or you're using a function that repaints. Keep `[1]` checks for "fresh" events.

Practical tips

- **PreLoadBars** (strategy): set it larger than your longest lookback. 5000 is a safe default; bump if you use long MAs.
- **CalculateOnLastBars** (indicator): purely for performance; it won't change logic.
- Keep directions explicit (`EnableLongs`/`EnableShorts`) while you validate.

When you report back, include: instrument, timeframe, your final **VALIDITY LOGIC** (the bit that used to trigger FILLCOLOR), and whether you want **one trade per zone**. If anything is off, I'll adjust the code to lock your visual, alert, and trading behavior 1:1.

ChatGPT can make mistakes. Check important info. See Cookie Preferences.