```
// This Pine Script® code is subject to the terms of the Mozilla Public License 2.0 at https://mozilla.org/MPL/2.0/ MPL-2.0

//@version=6

indicator("MFB - Value Reading Session", overlay = true, max_bars_back = 5000, max_lines_count = 500, max_labels_count = 500)


// --- Settings ---

sessionTime = input.session("0930-1600", "Structural Session", tooltip = "The time range of the previous day used for context (Institutional RTH).")

timezoneInput = input.string("America/New_York", "Timezone", tooltip = "Timezone used for the session calculation.")

valueAreaPct = input.float(70.0, "Value Area Percentage", minval = 0, maxval = 100, tooltip = "Percentage of total volume included in the value area.") / 100.0

rowCount = input.int(50, "Row Resolution", minval = 10, maxval = 100, tooltip = "Number of price rows used to build the volume profile.")

histWidth = input.int(40, "Histogram Width (Bars)", minval = 5, maxval = 200, tooltip = "Maximum horizontal width of the volume histogram.")

labelSize = input.string(size.normal, "Label Size", options = [size.small, size.normal, size.large, size.huge], tooltip = "Size used for profile labels.")

showAwarenessInput = input.bool(false, "Show Awareness Box", tooltip = "When enabled, displays the value-reading rules in a chart information box. It is hidden by default.")

showReadingInput = input.bool(true, "Show Value Reading", tooltip = "Controls whether the Bullish, Bearish, or Mixed / Probability Magnet reading appears during the active NY session.")

awarenessBoxPositionInput = input.string(position.middle_right, "Awareness Box Position", options = [position.top_left, position.top_center,
```

position.top_right, position.middle_left, position.middle_center, position.middle_right, position.bottom_left, position.bottom_center, position.bottom_right], tooltip = "Select where the Awareness Box is anchored on the chart.")

awarenessBoxWidthInput = input.float(0.0, "Awareness Box Width %", minval = 0.0, maxval = 80.0, tooltip = "Use 0 for automatic width that fits the text. Enter a percentage above 0 to set a fixed box width.")

awarenessTextSizeInput = input.string(size.normal, "Awareness Text Size", options = [size.small, size.normal, size.large, size.huge], tooltip = "Controls the text size inside the Awareness Box.")

// --- Visibility Settings ---

vaOpacity = input.int(30, "VA Opacity %", minval = 0, maxval = 100, tooltip = "Transparency of histogram rows inside the value area.")

outOpacity = input.int(70, "Outside VA Opacity %", minval = 0, maxval = 100, tooltip = "Transparency of histogram rows outside the value area.")

histThickness = input.int(3, "Histogram Thickness", minval = 1, maxval = 5, tooltip = "Line thickness used for histogram rows.")

// --- Colors ---

COLOR_UP = #26a69a

COLOR_DOWN = #ef5350

COLOR_POC = #9c27b0

COLOR_VA = #6a1b9a

COLOR_MID = #fdd835

COLOR_BG = color.new(#9c27b0, 95)

AWARENESS_DISPLAY_TEXT = "**VALUE READING AWARENESS**\n\nIf price opens below VAL,\nwith POC below VA MID:\nBearish reading at market open.\n\nIf price opens above VAH,\nwith POC above VA MID:\nBullish reading at market open.\n\n- Peace"

// --- Awareness Box ---

var table awarenessTable = na

if barstate.islast and showAwarenessInput

  if na(awarenessTable)

    awarenessTable := table.new(awarenessBoxPositionInput, 1, 1, bgcolor = color.new(chart.bg_color, 0), frame_color = COLOR_MID, frame_width = 2, border_color = COLOR_MID, border_width = 1)

  table.cell(awarenessTable, 0, 0, AWARENESS_DISPLAY_TEXT, width = awarenessBoxWidthInput, text_color = COLOR_MID, text_halign = text.align_left, text_valign = text.align_top, text_size = awarenessTextSizeInput, text_formatting = text.format_bold, bgcolor = color.new(chart.bg_color, 0))

// --- Session Detection ---

t = time(timeframe.period, sessionTime + ":23456", timezoneInput)

inSession = not na(t)

isNewSession = inSession and not inSession[1]

isSessionEnd = inSession[1] and not inSession

// --- Data Collection ---

[ltfO, ltfC, ltfV] = request.security_lower_tf(syminfo.tickerid, "1", [open, close, volume])

```
type BarInfo
   float price
   float volume
   bool isUp

var BarInfo[] currentSessionData = array.new<BarInfo>()
var int currentSessionStart = na
var float currentSessionHigh = 0.0
var float currentSessionLow = 1e10
var float currentSessionOpen = na
var int currentSessionOpenBar = na
var BarInfo[] lastCompletedData = array.new<BarInfo>()
var int lastSessionStartBar = na
var int lastSessionEndBar = na
var float lastSessionHigh = 0.0
var float lastSessionLow = 0.0
var float pocPrice = na
var float vahPrice = na
var float valPrice = na
var float vaMidPrice = na
var float profilePriceStep = na
var float profileMaxVol = na
```

```
var float[] rowUpVol = array.new_float()
var float[] rowDownVol = array.new_float()
var float[] rowTotal = array.new_float()

if isNewSession
   currentSessionData.clear()
   currentSessionStart := bar_index
   currentSessionHigh := high
   currentSessionLow := low
   currentSessionOpen := open
   currentSessionOpenBar := bar_index

if inSession
   currentSessionHigh := math.max(currentSessionHigh, high)
   currentSessionLow := math.min(currentSessionLow, low)
   if array.size(ltfC) > 0
      for i = 0 to array.size(ltfC) - 1
         float lowerClose = array.get(ltfC, i)
         float lowerVolume = array.get(ltfV, i)
         float lowerOpen = array.get(ltfO, i)
         if not na(lowerClose) and not na(lowerVolume)
            currentSessionData.push(BarInfo.new(lowerClose, lowerVolume,
lowerClose >= nz(lowerOpen, lowerClose)))
```

```
// Calculate on EVERY completed session, rather than only on the last chart bar,
// so both the plotted profile and historical alert conditions use the same prices.
if isSessionEnd
    lastCompletedData := currentSessionData.copy()
    lastSessionStartBar := currentSessionStart
    lastSessionEndBar := bar_index
    lastSessionHigh := currentSessionHigh
    lastSessionLow := currentSessionLow
    pocPrice := na
    vahPrice := na
    valPrice := na
    vaMidPrice := na
    profilePriceStep := na
    profileMaxVol := na
    float priceRange = lastSessionHigh - lastSessionLow
    if priceRange > 0 and array.size(lastCompletedData) > 0
        profilePriceStep := priceRange / rowCount
        rowUpVol := array.new_float(rowCount, 0.0)
        rowDownVol := array.new_float(rowCount, 0.0)
        rowTotal := array.new_float(rowCount, 0.0)
        float totalVol = 0.0
```

```
    for profileBar in lastCompletedData
        int rowIndex = math.min(rowCount - 1, math.floor((profileBar.price - lastSessionLow) / profilePriceStep))
        if rowIndex >= 0
            if profileBar.isUp
                rowUpVol.set(rowIndex, rowUpVol.get(rowIndex) + profileBar.volume)
            else
                rowDownVol.set(rowIndex, rowDownVol.get(rowIndex) + profileBar.volume)
            rowTotal.set(rowIndex, rowTotal.get(rowIndex) + profileBar.volume)
            totalVol += profileBar.volume
    profileMaxVol := array.max(rowTotal)
    if profileMaxVol > 0
        int pocIndex = array.indexof(rowTotal, profileMaxVol)
        pocPrice := lastSessionLow + (pocIndex + 0.5) * profilePriceStep
        float targetVolume = totalVol * valueAreaPct
        float valueAreaVolume = profileMaxVol
        int upperIndex = pocIndex
        int lowerIndex = pocIndex
        int expansionSteps = 0
        while valueAreaVolume < targetVolume and (upperIndex < rowCount - 1 or lowerIndex > 0) and expansionSteps < rowCount
            expansionSteps += 1
```

```
        bool canExpandUp = upperIndex < rowCount - 1
        bool canExpandDown = lowerIndex > 0
        float upperVolume = canExpandUp ? rowTotal.get(upperIndex + 1) : 0.0
        float lowerVolume = canExpandDown ? rowTotal.get(lowerIndex - 1) :
0.0
        if canExpandUp and (not canExpandDown or upperVolume >=
lowerVolume)
            upperIndex += 1
            valueAreaVolume += upperVolume
        else if canExpandDown
            lowerIndex -= 1
            valueAreaVolume += lowerVolume
    vahPrice := lastSessionLow + (upperIndex + 1) * profilePriceStep
    valPrice := lastSessionLow + lowerIndex * profilePriceStep
    vaMidPrice := (vahPrice + valPrice) / 2.0


// --- Retrace Alerts ---
// An excursion is armed by a bar CLOSING outside value. Starting on the next
// bar, each distinct level can fire once until price closes inside value and
// then closes outside again, or the opposite excursion starts.
var int retraceSide = 0
var bool returnedInside = false
var bool hitVah = false
var bool hitVal = false
```

```
var bool hitPoc = false
if isSessionEnd
    retraceSide := 0
    returnedInside := false
    hitVah := false
    hitVal := false
    hitPoc := false

bool hasProfile = not na(vahPrice) and not na(valPrice) and not na(pocPrice)
bool canTouch = hasProfile and retraceSide != 0 and barstate.isconfirmed and bar_index > lastSessionEndBar
bool vahTouch = canTouch and not hitVah and low <= vahPrice and high >= vahPrice
bool valTouch = canTouch and not hitVal and low <= valPrice and high >= valPrice
bool pocTouch = canTouch and not hitPoc and low <= pocPrice and high >= pocPrice
bool valueRangeRetrace = vahTouch or valTouch or pocTouch

if vahTouch
    hitVah := true
if valTouch
    hitVal := true
if pocTouch
```

```
    hitPoc := true

if hasProfile and barstate.isconfirmed
    int closingSide = close > vahPrice ? 1 : close < valPrice ? -1 : 0
    if closingSide == 0
        returnedInside := true
    else if closingSide != retraceSide or returnedInside
        retraceSide := closingSide
        returnedInside := false
        hitVah := false
        hitVal := false
        hitPoc := false

// --- Rendering ---
var line[] profileLines = array.new_line()
var label[] profileLabels = array.new_label()
if barstate.islast and hasProfile
    for profileLine in profileLines
        line.delete(profileLine)
    array.clear(profileLines)
    for profileLabel in profileLabels
        label.delete(profileLabel)
    array.clear(profileLabels)
```

```
  for i = 0 to rowCount - 1
    float rowY = lastSessionLow + (i + 0.5) * profilePriceStep
    float rowVolume = rowTotal.get(i)
    if rowVolume > 0
      int totalLength = math.round((rowVolume / profileMaxVol) * histWidth)
      int upLength = math.round((rowUpVol.get(i) / rowVolume) * totalLength)
      int transparency = rowY >= valPrice and rowY <= vahPrice ? vaOpacity :
outOpacity
      profileLines.push(line.new(lastSessionStartBar, rowY,
lastSessionStartBar + upLength, rowY, color = color.new(COLOR_UP,
transparency), width = histThickness))
      profileLines.push(line.new(lastSessionStartBar + upLength, rowY,
lastSessionStartBar + totalLength, rowY, color = color.new(COLOR_DOWN,
transparency), width = histThickness))


  profileLines.push(line.new(lastSessionEndBar, pocPrice, bar_index + 50,
pocPrice, color = COLOR_POC, width = 2, extend = extend.right))
  profileLines.push(line.new(lastSessionEndBar, vahPrice, bar_index + 50,
vahPrice, color = COLOR_VA, style = line.style_dashed, width = 2, extend =
extend.right))
  profileLines.push(line.new(lastSessionEndBar, valPrice, bar_index + 50,
valPrice, color = COLOR_VA, style = line.style_dashed, width = 2, extend =
extend.right))
  profileLines.push(line.new(lastSessionEndBar, vaMidPrice, bar_index + 50,
vaMidPrice, color = COLOR_MID, style = line.style_dotted, width = 2, extend =
extend.right))
```

```
    profileLabels.push(label.new(bar_index + 5, pocPrice, "POC", color =
#00000000, textcolor = COLOR_POC, style = label.style_label_left, size =
labelSize))

    profileLabels.push(label.new(bar_index + 5, vahPrice, "VAH", color =
#00000000, textcolor = COLOR_VA, style = label.style_label_left, size =
labelSize))

    profileLabels.push(label.new(bar_index + 5, valPrice, "VAL", color =
#00000000, textcolor = COLOR_VA, style = label.style_label_left, size =
labelSize))

    profileLabels.push(label.new(bar_index + 5, vaMidPrice, "VA MID", color =
#00000000, textcolor = COLOR_MID, style = label.style_label_left, size =
labelSize))


    if showReadingInput and inSession and not na(currentSessionOpen) and
not na(currentSessionOpenBar)

        bool bullishReading = currentSessionOpen > vahPrice and pocPrice >
vaMidPrice

        bool bearishReading = currentSessionOpen < valPrice and pocPrice <
vaMidPrice

        bool mixedReading = (currentSessionOpen > vahPrice and pocPrice <=
vaMidPrice) or (currentSessionOpen < valPrice and pocPrice >= vaMidPrice)

        if bullishReading or bearishReading or mixedReading

            string readingText = bullishReading ? "BULLISH VALUE
READING\\nOpen > VAH  •  POC > VA MID" : bearishReading ? "BEARISH
VALUE READING\\nOpen < VAL  •  POC < VA MID" : "MIXED / PROBABILITY
MAGNET\\nOpening location and POC disagree"

            string readingTooltip = bullishReading ? "Bullish Value Reading at NY
open.\\n\\nIf price stays above VAH, then Bullish Acceptance." :
```

bearishReading ? "Bearish Value Reading at NY open.\\n\\nIf price stays below VAL, then Bearish Acceptance." : "Mixed Value Reading at NY open.\\n\\nThe opening location and POC disagree.\\n\\nThe POC may act as a probability magnet, but it is not a guaranteed target."

float readingY = bullishReading ? vahPrice + profilePriceStep * 0.5 : bearishReading ? valPrice - profilePriceStep * 0.5 : pocPrice + profilePriceStep * 0.5

labelStyle = bullishReading ? label.style_label_down : bearishReading ? label.style_label_up : label.style_label_left

profileLabels.push(label.new(currentSessionOpenBar, readingY, readingText, color = color.new(chart.bg_color, 15), textcolor = COLOR_MID, style = labelStyle, textalign = text.align_left, tooltip = readingTooltip, size = labelSize))

// --- Background Highlighting ---

bgcolor(inSession ? COLOR_BG : na)

// --- Alerts ---

alertcondition(valueRangeRetrace, title = "Heads up - Value range retrace", message = "{{ticker}} {{interval}}: Price retraced from outside the previous session's value area and touched VAH, VAL or POC. Close: {{close}}")