

// 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 - I.B.", overlay = true, max_bars_back = 5000, max_lines_count = 500, max_labels_count = 100)

// --- Constants ---

int ONE_MINUTE_MS = 60000

// --- Inputs ---

rangeSessionInput = input.session("0930-1030", "Initial Balance Window", tooltip = "Current-day time range used to build the fixed-range volume profile.")

timezoneInput = input.string("America/New_York", "Timezone", tooltip = "Timezone used to interpret the Initial Balance Window.")

valueAreaPctInput = input.float(70.0, "Value Area Percentage", minval = 0.0, maxval = 100.0, tooltip = "Percentage of total window volume included in the value area.")

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

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

showDevelopingInput = input.bool(true, "Show Developing Profile", tooltip = "Displays the profile while the Initial Balance Window is still forming.")

```
showIBRangeInput = input.bool(true, "Show IB High / Low", tooltip = "Extends  
the Initial Balance high and low across the chart.")
```

```
showReadingInput = input.bool(true, "Show Current Value Reading", tooltip =  
"Displays a reading after the Initial Balance window when price is outside  
value and the POC agrees or disagrees with the VA midpoint.")
```

```
extendLevelsInput = input.bool(true, "Extend Profile Levels", tooltip = "Extends  
POC, VAH, VAL, and VA MID to the right of the profile.")
```

```
showLabelsInput = input.bool(true, "Show Level Labels", tooltip = "Displays  
labels for the profile levels and Initial Balance range.")
```

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

```
showBackgroundInput = input.bool(false, "Highlight Initial Balance", tooltip =  
"Highlights the selected current-day Initial Balance Window.")
```

```
// --- Style Inputs ---
```

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

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

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

```
// --- Colors ---
```

```
COLOR_UP = #26a69a
```

```
COLOR_DOWN = #ef5350
```

```
COLOR_POC = #9c27b0
```

```
COLOR_VA = #6a1b9a
```

```
COLOR_MID = #fdd835
```

```
COLOR_IB = #5b9cf6
```

```
COLOR_BG = color.new(#5b9cf6, 92)
```

```
// --- Data Types ---
```

```
type ProfileBar
```

```
    float price
```

```
    float volume
```

```
    bool isUp
```

```
// --- Lower-Timeframe Data ---
```

```
// The session filter is applied to each 1-minute bar so a 09:30 window is
```

```
// captured correctly even when the chart itself is on a 1-hour timeframe.
```

```
[ltfTimeArray, ltfDayTimeArray, ltfRangeTimeArray, ltfOpenArray, ltfHighArray,  
ltfLowArray, ltfCloseArray, ltfVolumeArray] =
```

```
request.security_lower_tf(syminfo.tickerid, "1", [time, time("D", "0000-  
2359:1234567", timezoneInput), time("1", rangeSessionInput + ":23456",  
timezoneInput), open, high, low, close, volume])
```

```
// --- Current-Day Collection ---
```

```
var ProfileBar[] profileDataArray = array.new<ProfileBar>()
```

```
var int currentDayKey = na
```

```
var int windowStartBar = na
```

```
var int windowStartTime = na
```

```
var int windowEndTime = na
```

```
var float windowHigh = na
```

```
var float windowLow = na
```

```
var int latestLowerTime = na
```

```
if array.size(ltfTimeArray) > 0
```

```
  for i = 0 to array.size(ltfTimeArray) - 1
```

```
    int lowerTime = array.get(ltfTimeArray, i)
```

```
    int lowerDayTime = array.get(ltfDayTimeArray, i)
```

```
    int lowerRangeTime = array.get(ltfRangeTimeArray, i)
```

```
    int lowerDayKey = na(lowerDayTime) ? year(lowerTime, timezoneInput) *  
10000 + month(lowerTime, timezoneInput) * 100 + dayofmonth(lowerTime,  
timezoneInput) : lowerDayTime
```

```
    latestLowerTime := lowerTime
```

```
if na(currentDayKey) or lowerDayKey != currentDayKey
```

```
  currentDayKey := lowerDayKey
```

```
  profileDataArray.clear()
```

```
  windowStartBar := na
```

```
  windowStartTime := na
```

```
  windowEndTime := na
```

```
  windowHigh := na
```

windowLow := na

if not na(lowerRangeTime)

float lowerOpen = array.get(ltfOpenArray, i)

float lowerHigh = array.get(ltfHighArray, i)

float lowerLow = array.get(ltfLowArray, i)

float lowerClose = array.get(ltfCloseArray, i)

float lowerVolume = array.get(ltfVolumeArray, i)

if not na(lowerClose) and not na(lowerVolume)

if na(windowStartTime)

windowStartBar := bar_index

windowStartTime := lowerRangeTime

windowHigh := lowerHigh

windowLow := lowerLow

else

windowHigh := math.max(windowHigh, lowerHigh)

windowLow := math.min(windowLow, lowerLow)

windowEndTime := lowerRangeTime + ONE_MINUTE_MS

profileDataArray.push(ProfileBar.new(lowerClose, lowerVolume,
lowerClose >= nz(lowerOpen, lowerClose)))

```

// --- Rendering ---

var line[] profileLinesArray = array.new_line()
var label[] profileLabelsArray = array.new_label()

if barstate.islast
    for profileLine in profileLinesArray
        line.delete(profileLine)
    profileLinesArray.clear()

    for profileLabel in profileLabelsArray
        label.delete(profileLabel)
    profileLabelsArray.clear()

    bool rangeComplete = not na(windowEndTime) and not na(latestLowerTime)
    and latestLowerTime >= windowEndTime

    bool canRender = array.size(profileDataArray) > 0 and not na(windowHigh)
    and not na(windowLow) and windowHigh > windowLow and
    (showDevelopingInput or rangeComplete)

    if canRender
        float priceRange = windowHigh - windowLow
        float priceStep = priceRange / rowCountInput
        float[] rowUpVolumeArray = array.new_float(rowCountInput, 0.0)
        float[] rowDownVolumeArray = array.new_float(rowCountInput, 0.0)

```

```

float[] rowTotalVolumeArray = array.new_float(rowCountInput, 0.0)

float totalVolume = 0.0

for profileBar in profileDataArray

    int rowIndex = int(math.min(rowCountInput - 1,
math.floor((profileBar.price - windowLow) / priceStep)))

    if rowIndex >= 0 and rowIndex < rowCountInput

        if profileBar.isUp

            rowUpVolumeArray.set(rowIndex, rowUpVolumeArray.get(rowIndex)
+ profileBar.volume)

        else

            rowDownVolumeArray.set(rowIndex,
rowDownVolumeArray.get(rowIndex) + profileBar.volume)

            rowTotalVolumeArray.set(rowIndex,
rowTotalVolumeArray.get(rowIndex) + profileBar.volume)

            totalVolume += profileBar.volume

float maxVolume = array.max(rowTotalVolumeArray)

if maxVolume > 0 and totalVolume > 0

    int pocIndex = array.indexof(rowTotalVolumeArray, maxVolume)

    float pocPrice = windowLow + pocIndex * priceStep + priceStep / 2.0

float targetVolume = totalVolume * valueAreaPctInput / 100.0

float valueAreaVolume = maxVolume

```

```
int upperIndex = pocIndex
```

```
int lowerIndex = pocIndex
```

```
while valueAreaVolume < targetVolume and (upperIndex <
rowCountInput - 1 or lowerIndex > 0)
```

```
float upperVolume = upperIndex < rowCountInput - 1 ?
rowTotalVolumeArray.get(upperIndex + 1) : 0.0
```

```
float lowerVolume = lowerIndex > 0 ?
rowTotalVolumeArray.get(lowerIndex - 1) : 0.0
```

```
if upperVolume >= lowerVolume and upperIndex < rowCountInput - 1
```

```
    upperIndex += 1
```

```
    valueAreaVolume += upperVolume
```

```
else if lowerIndex > 0
```

```
    lowerIndex -= 1
```

```
    valueAreaVolume += lowerVolume
```

```
else
```

```
    break
```

```
float vahPrice = windowLow + (upperIndex + 1) * priceStep
```

```
float valPrice = windowLow + lowerIndex * priceStep
```

```
float vaMidPrice = valPrice + (vahPrice - valPrice) / 2.0
```

```
int levelStartTime = rangeComplete ? windowEndTime :
windowStartTime
```

```

// Draw the fixed-range histogram.

for i = 0 to rowCountInput - 1

    float rowY = windowLow + i * priceStep + priceStep / 2.0

    float rowVolume = rowTotalVolumeArray.get(i)

    if rowVolume > 0

        int totalLength = math.max(1, int(math.round(rowVolume /
maxVolume * histWidthInput)))

        int upLength = int(math.round(rowUpVolumeArray.get(i) /
rowVolume * totalLength))

        int transparency = i >= lowerIndex and i <= upperIndex ?
vaOpacityInput : outOpacityInput

        profileLinesArray.push(line.new(windowStartBar, rowY,
windowStartBar + upLength, rowY, color = color.new(COLOR_UP,
transparency), width = histThicknessInput))

        profileLinesArray.push(line.new(windowStartBar + upLength, rowY,
windowStartBar + totalLength, rowY, color = color.new(COLOR_DOWN,
transparency), width = histThicknessInput))

// Draw profile levels from the end of the completed range, or from
// the current window boundary while the profile is developing.

if extendLevelsInput and not na(levelStartTime)

    profileLinesArray.push(line.new(levelStartTime, pocPrice,
levelStartTime + ONE_MINUTE_MS, pocPrice, xloc = xloc.bar_time, color =
COLOR_POC, width = 2, extend = extend.right))

```

```
profileLinesArray.push(line.new(levelStartTime, vahPrice,
levelStartTime + ONE_MINUTE_MS, vahPrice, xloc = xloc.bar_time, color =
COLOR_VA, style = line.style_dashed, width = 2, extend = extend.right))
```

```
profileLinesArray.push(line.new(levelStartTime, valPrice,
levelStartTime + ONE_MINUTE_MS, valPrice, xloc = xloc.bar_time, color =
COLOR_VA, style = line.style_dashed, width = 2, extend = extend.right))
```

```
profileLinesArray.push(line.new(levelStartTime, vaMidPrice,
levelStartTime + ONE_MINUTE_MS, vaMidPrice, xloc = xloc.bar_time, color =
COLOR_MID, style = line.style_dotted, width = 2, extend = extend.right))
```

```
if showIBRangeInput and not na(windowStartTime)
```

```
profileLinesArray.push(line.new(windowStartTime, windowHigh,
windowStartTime + ONE_MINUTE_MS, windowHigh, xloc = xloc.bar_time,
color = COLOR_IB, style = line.style_dashed, width = 2, extend = extend.right))
```

```
profileLinesArray.push(line.new(windowStartTime, windowLow,
windowStartTime + ONE_MINUTE_MS, windowLow, xloc = xloc.bar_time,
color = COLOR_IB, style = line.style_dashed, width = 2, extend = extend.right))
```

```
if showLabelsInput
```

```
int labelBar = bar_index + 5
```

```
profileLabelsArray.push(label.new(labelBar, pocPrice, "POC", color =
#00000000, textcolor = COLOR_POC, style = label.style_label_left, size =
labelSizeInput))
```

```
profileLabelsArray.push(label.new(labelBar, vahPrice, "VAH", color =
#00000000, textcolor = COLOR_VA, style = label.style_label_left, size =
labelSizeInput))
```

```
profileLabelsArray.push(label.new(labelBar, valPrice, "VAL", color =  
#00000000, textcolor = COLOR_VA, style = label.style_label_left, size =  
labelSizeInput))
```

```
profileLabelsArray.push(label.new(labelBar, vaMidPrice, "VA MID",  
color = #00000000, textcolor = COLOR_MID, style = label.style_label_left, size  
= labelSizeInput))
```

```
if showIBRangeInput
```

```
profileLabelsArray.push(label.new(labelBar, windowHigh, "IB HIGH",  
color = #00000000, textcolor = COLOR_IB, style = label.style_label_left, size =  
labelSizeInput))
```

```
profileLabelsArray.push(label.new(labelBar, windowLow, "IB LOW",  
color = #00000000, textcolor = COLOR_IB, style = label.style_label_left, size =  
labelSizeInput))
```

```
// The reading is based on the latest price after the Initial Balance
```

```
// window, rather than on the opening price used by the prior-session  
version.
```

```
if showReadingInput and rangeComplete
```

```
bool bullishReading = close > vahPrice and pocPrice > vaMidPrice
```

```
bool bearishReading = close < valPrice and pocPrice < vaMidPrice
```

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

```
if bullishReading or bearishReading or mixedReading
```

```
string readingText = bullishReading ? "BULLISH IB VALUE\nPrice >  
VAH • POC > VA MID" : bearishReading ? "BEARISH IB VALUE\nPrice < VAL •  
POC < VA MID" : "MIXED IB VALUE\nPrice and POC disagree"
```

```
    string readingTooltip = bullishReading ? "Bullish Initial Balance value  
reading.\n\nAcceptance above VAH strengthens the bullish condition." :  
bearishReading ? "Bearish Initial Balance value reading.\n\nAcceptance  
below VAL strengthens the bearish condition." : "Mixed Initial Balance value  
reading.\n\nThe current price location and POC disagree."
```

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

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

```
    profileLabelsArray.push(label.new(bar_index, readingY, readingText,  
color = color.new(chart.bg_color, 15), textcolor = COLOR_MID, style =  
labelStyle, textalign = text.align_left, tooltip = readingTooltip, size =  
labelSizeInput))
```

```
// --- Background Highlighting ---
```

```
int currentSessionTime = time(timeframe.period, rangeSessionInput +  
":23456", timezoneInput)
```

```
bool chartBarInWindow = not na(currentSessionTime)
```

```
bgcolor(showBackgroundInput and chartBarInWindow ? COLOR_BG : na)
```