

```
// 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
```

```
strategy(  
    "MFB - Value Reading Session - 15 min OR",  
    overlay = true,  
    max_bars_back = 5000,  
    max_lines_count = 500,  
    max_labels_count = 500,  
    pyramiding = 0,  
    default_qty_type = strategy.fixed,  
    default_qty_value = 1,  
    initial_capital = 1000000,  
    process_orders_on_close = true,  
    use_bar_magnifier = true  
)
```

```
// --- Constants ---
```

```
const string OPENING_RANGE_SESSION = "0930-0945:23456"
```

```
const string LOWER_TIMEFRAME = "1"
```

```
const string NY_TIMEZONE = "America/New_York"
```

```
const color COLOR_UP = #26a69a
```

```
const color COLOR_DOWN = #ef5350
const color COLOR_ENTRY = #5b9cf6
const color COLOR_VA = #6a1b9a
const color COLOR_MID = #fdd835
const color COLOR_BG = color.new(#9c27b0, 95)

// --- Inputs ---
string timezoneInput = input.string(
    NY_TIMEZONE,
    "Timezone",
    tooltip = "Timezone used to identify the 09:30-09:45 opening range."
)
float valueAreaPctInput = input.float(
    70.0,
    "Value Area Percentage",
    minval = 0.0,
    maxval = 100.0,
    tooltip = "Percentage of opening-range volume included in the value area."
) / 100.0
int rowCountInput = input.int(
    50,
    "Row Resolution",
    minval = 10,
```

```
        maxval = 100,

        tooltip = "Number of price rows used to build the 09:30-09:45 volume
profile."
    )
    int histWidthInput = input.int(
        40,
        "Histogram Width (Bars)",
        minval = 5,
        maxval = 200,
        tooltip = "Maximum horizontal width of the opening-range volume
        histogram."
    )
    bool showProfileInput = input.bool(
        true,
        "Show Opening-Range Profile",
        tooltip = "Displays the 15-minute fixed-range histogram and VAH, VAL, and
        VA Mid lines."
    )
    string labelSizeInput = input.string(
        size.small,
        "Profile Label Size",
        options = [size.tiny, size.small, size.normal, size.large],
        tooltip = "Size used for VAH, VAL, and VA Mid labels."
    )
```

```
int vaOpacityInput = input.int(  
    30,  
    "VA Opacity %",  
    minval = 0,  
    maxval = 100,  
    tooltip = "Transparency of histogram rows inside the value area."  
)  
  
int outsideOpacityInput = input.int(  
    70,  
    "Outside VA Opacity %",  
    minval = 0,  
    maxval = 100,  
    tooltip = "Transparency of histogram rows outside the value area."  
)  
  
int histThicknessInput = input.int(  
    3,  
    "Histogram Thickness",  
    minval = 1,  
    maxval = 5,  
    tooltip = "Line thickness used for the fixed-range histogram."  
)  
  
int stopOffsetTicksInput = input.int(  

```

```

2,
"Stop Offset Behind POC (Ticks)",
minval = 1,
tooltip = "Long stops are placed this many ticks below POC; short stops this
many ticks above POC."
)
float rewardRiskInput = input.float(
2.0,
"Reward/Risk Target",
minval = 0.1,
step = 0.1,
tooltip = "Default 2.0 produces a 1:2 risk-to-reward target."
)
bool oneTradePerDayInput = input.bool(
true,
"One Trade Per Opening Range",
tooltip = "When enabled, only the first valid 5-minute close through VAH or
VAL can create a trade each day."
)
bool showSignalsInput = input.bool(
true,
"Show Entry Signals",
tooltip = "Displays markers on confirmed 5-minute closes that trigger a
strategy entry."

```

```
)
```

```
// --- Profile Data ---
```

```
type BarInfo
```

```
    float price
```

```
    float volume
```

```
    bool isUp
```

```
[lowerTimeArray, lowerOpenArray, lowerHighArray, lowerLowArray,  
lowerCloseArray, lowerVolumeArray, lowerInOpeningArray] =  
request.security_lower_tf(  
    syminfo.tickerid,  
    LOWER_TIMEFRAME,  
    [time, open, high, low, close, volume, not na(time(LOWER_TIMEFRAME,  
OPENING_RANGE_SESSION, timezoneInput))],  
    ignore_invalid_timeframe = true  
)
```

```
var BarInfo[] openingRangeData = array.new<BarInfo>()
```

```
var int openingDateKey = na
```

```
var int openingStartBar = na
```

```
var bool openingWindowClosed = false
```

```
var bool profileReady = false
```

```
var bool tradeTaken = false
```

```

var float openingHigh = na
var float openingLow = na
var float pocPrice = na
var float vahPrice = na
var float valPrice = na
var float vaMidPrice = na
var float activeEntryPrice = na
var float activeStopPrice = na
var float activeTargetPrice = na
var line tradeEntryLine = na
var line tradeStopLine = na
var line tradeTargetLine = na
var line[] profileLines = array.new_line()
var label[] profileLabels = array.new_label()

// --- New Opening Range Detection and Data Collection ---

if array.size(lowerTimeArray) > 0
    for i = 0 to array.size(lowerTimeArray) - 1
        int lowerTime = array.get(lowerTimeArray, i)

        int lowerDateKey = year(lowerTime, timezoneInput) * 10000 +
month(lowerTime, timezoneInput) * 100 + dayofmonth(lowerTime,
timezoneInput)

        bool lowerInOpening = array.get(lowerInOpeningArray, i)

```

```
float lowerOpen = array.get(lowerOpenArray, i)
float lowerHigh = array.get(lowerHighArray, i)
float lowerLow = array.get(lowerLowArray, i)
float lowerClose = array.get(lowerCloseArray, i)
float lowerVolume = array.get(lowerVolumeArray, i)
```

```
if lowerInOpening
```

```
    if na(openingDateKey) or lowerDateKey != openingDateKey
```

```
        openingRangeData.clear()
```

```
        openingDateKey := lowerDateKey
```

```
        openingStartBar := bar_index
```

```
        openingWindowClosed := false
```

```
        profileReady := false
```

```
        tradeTaken := false
```

```
        openingHigh := lowerHigh
```

```
        openingLow := lowerLow
```

```
        pocPrice := na
```

```
        vahPrice := na
```

```
        valPrice := na
```

```
        vaMidPrice := na
```

```
        activeStopPrice := na
```

```
        activeTargetPrice := na
```

```
for profileLine in profileLines
```

```
    line.delete(profileLine)
```

```
profileLines.clear()
```

```
for profileLabel in profileLabels
```

```
    label.delete(profileLabel)
```

```
profileLabels.clear()
```

```
else
```

```
    openingHigh := math.max(openingHigh, lowerHigh)
```

```
    openingLow := math.min(openingLow, lowerLow)
```

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

```
    openingRangeData.push(BarInfo.new(lowerClose, lowerVolume,  
lowerClose >= nz(lowerOpen, lowerClose)))
```

```
    else if not na(openingDateKey) and lowerDateKey == openingDateKey and  
array.size(openingRangeData) > 0
```

```
        openingWindowClosed := true
```

```
// --- Opening-Range Volume Profile Calculation ---
```

```
if openingWindowClosed and not profileReady and  
array.size(openingRangeData) > 0
```

```
    float priceRange = openingHigh - openingLow
```

```
    if priceRange > 0
```

```

float priceStep = priceRange / rowCountInput

float[] rowUpVolume = array.new_float(rowCountInput, 0.0)
float[] rowDownVolume = array.new_float(rowCountInput, 0.0)
float[] rowTotalVolume = array.new_float(rowCountInput, 0.0)
float totalVolume = 0.0

for profileBar in openingRangeData
    int rowIndex = int(math.min(rowCountInput - 1,
math.floor((profileBar.price - openingLow) / priceStep)))
    if rowIndex >= 0 and rowIndex < rowCountInput
        if profileBar.isUp
            rowUpVolume.set(rowIndex, rowUpVolume.get(rowIndex) +
profileBar.volume)
        else
            rowDownVolume.set(rowIndex, rowDownVolume.get(rowIndex) +
profileBar.volume)
            rowTotalVolume.set(rowIndex, rowTotalVolume.get(rowIndex) +
profileBar.volume)
        totalVolume += profileBar.volume

float maxVolume = array.max(rowTotalVolume)
if maxVolume > 0 and totalVolume > 0
    int pocIndex = array.indexof(rowTotalVolume, maxVolume)
    pocPrice := openingLow + pocIndex * priceStep + priceStep / 2.0

```

```

float targetVolume = totalVolume * valueAreaPctInput

float valueAreaVolume = maxVolume

int upperIndex = pocIndex

int lowerIndex = pocIndex

int expansionSteps = 0


while valueAreaVolume < targetVolume and (upperIndex <
rowCountInput - 1 or lowerIndex > 0) and expansionSteps < rowCountInput
    expansionSteps += 1

    bool canExpandUp = upperIndex < rowCountInput - 1

    bool canExpandDown = lowerIndex > 0

    float upperVolume = canExpandUp ? rowTotalVolume.get(upperIndex
+ 1) : 0.0

    float lowerVolume = canExpandDown ?
rowTotalVolume.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 := openingLow + (upperIndex + 1) * priceStep
```

```
valPrice := openingLow + lowerIndex * priceStep
```

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

```
profileReady := true
```

```
if showProfileInput
```

```
  for i = 0 to rowCountInput - 1
```

```
    float rowY = openingLow + i * priceStep + priceStep / 2.0
```

```
    float rowVolume = rowTotalVolume.get(i)
```

```
    if rowVolume > 0
```

```
      int totalLength = int(math.round((rowVolume / maxVolume) *  
histWidthInput))
```

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

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

```
      profileLines.push(line.new(openingStartBar, rowY,  
openingStartBar + upLength, rowY, color = color.new(COLOR_UP,  
transparency), width = histThicknessInput))
```

```
      profileLines.push(line.new(openingStartBar + upLength, rowY,  
openingStartBar + totalLength, rowY, color = color.new(COLOR_DOWN,  
transparency), width = histThicknessInput))
```

```
profileLines.push(line.new(bar_index, vahPrice, bar_index + 1,
vahPrice, color = COLOR_VA, style = line.style_dashed, width = 2, extend =
extend.right))
```

```
profileLines.push(line.new(bar_index, valPrice, bar_index + 1, valPrice,
color = COLOR_VA, style = line.style_dashed, width = 2, extend =
extend.right))
```

```
profileLines.push(line.new(bar_index, vaMidPrice, bar_index + 1,
vaMidPrice, color = COLOR_MID, style = line.style_dotted, width = 2, extend =
extend.right))
```

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

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

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

```
// --- Five-Minute Close Entry Logic ---
```

```
// Apply this strategy on a 5-minute chart so each signal is evaluated exactly at
a confirmed 5-minute close.
```

```
bool isFiveMinuteChart = timeframe.in_seconds() == 300
```

```
bool longBreakout = isFiveMinuteChart and profileReady and
barstate.isconfirmed and close > vahPrice and close[1] <= vahPrice
```

```
bool shortBreakout = isFiveMinuteChart and profileReady and
barstate.isconfirmed and close < valPrice and close[1] >= valPrice
```

```
bool canTrade = not oneTradePerDayInput or not tradeTaken
```

```
float stopOffset = stopOffsetTicksInput * syminfo.mintick
```

```
bool validLongTrade = longBreakout and canTrade and strategy.position_size  
== 0 and pocPrice - stopOffset < close
```

```
bool validShortTrade = shortBreakout and canTrade and strategy.position_size  
== 0 and pocPrice + stopOffset > close
```

```
if validLongTrade
```

```
    activeEntryPrice := close
```

```
    activeStopPrice := pocPrice - stopOffset
```

```
    float riskDistance = close - activeStopPrice
```

```
    activeTargetPrice := close + riskDistance * rewardRiskInput
```

```
    tradeEntryLine := line.new(bar_index, activeEntryPrice, bar_index + 1,  
activeEntryPrice, color = COLOR_ENTRY, style = line.style_dotted, width = 2)
```

```
    tradeStopLine := line.new(bar_index, activeStopPrice, bar_index + 1,  
activeStopPrice, color = COLOR_DOWN, style = line.style_dashed, width = 2)
```

```
    tradeTargetLine := line.new(bar_index, activeTargetPrice, bar_index + 1,  
activeTargetPrice, color = COLOR_UP, style = line.style_dashed, width = 2)
```

```
    strategy.entry("Long", strategy.long, alert_message = "MFB 15 min OR  
LONG")
```

```
    strategy.exit("Long Exit", "Long", stop = activeStopPrice, limit =  
activeTargetPrice, alert_message = "MFB 15 min OR LONG EXIT")
```

```
    tradeTaken := true
```

```
    alert("MFB 15 min OR long entry", alert.freq_once_per_bar_close)
```

if validShortTrade

activeEntryPrice := close

activeStopPrice := pocPrice + stopOffset

float riskDistance = activeStopPrice - close

activeTargetPrice := close - riskDistance * rewardRiskInput

tradeEntryLine := line.new(bar_index, activeEntryPrice, bar_index + 1,
activeEntryPrice, color = COLOR_ENTRY, style = line.style_dotted, width = 2)

tradeStopLine := line.new(bar_index, activeStopPrice, bar_index + 1,
activeStopPrice, color = COLOR_DOWN, style = line.style_dashed, width = 2)

tradeTargetLine := line.new(bar_index, activeTargetPrice, bar_index + 1,
activeTargetPrice, color = COLOR_UP, style = line.style_dashed, width = 2)

strategy.entry("Short", strategy.short, alert_message = "MFB 15 min OR
SHORT")

strategy.exit("Short Exit", "Short", stop = activeStopPrice, limit =
activeTargetPrice, alert_message = "MFB 15 min OR SHORT EXIT")

tradeTaken := true

alert("MFB 15 min OR short entry", alert.freq_once_per_bar_close)

if strategy.position_size > 0 and not na(activeStopPrice) and not
na(activeTargetPrice)

strategy.exit("Long Exit", "Long", stop = activeStopPrice, limit =
activeTargetPrice, alert_message = "MFB 15 min OR LONG EXIT")

if strategy.position_size < 0 and not na(activeStopPrice) and not
na(activeTargetPrice)

```
strategy.exit("Short Exit", "Short", stop = activeStopPrice, limit =  
activeTargetPrice, alert_message = "MFB 15 min OR SHORT EXIT")
```

```
if not na(tradeEntryLine) and (strategy.position_size != 0 or not  
na(activeEntryPrice))
```

```
    line.set_x2(tradeEntryLine, bar_index)
```

```
    line.set_x2(tradeStopLine, bar_index)
```

```
    line.set_x2(tradeTargetLine, bar_index)
```

```
if strategy.position_size == 0 and strategy.position_size[1] != 0
```

```
    line.set_x2(tradeEntryLine, bar_index)
```

```
    line.set_x2(tradeStopLine, bar_index)
```

```
    line.set_x2(tradeTargetLine, bar_index)
```

```
    activeEntryPrice := na
```

```
    activeStopPrice := na
```

```
    activeTargetPrice := na
```

```
// --- Visual Elements ---
```

```
// Trade levels are drawn as finite horizontal segments from entry until the  
trade closes.
```

```
plotshape(showSignalsInput and validLongTrade, "Long Entry",  
shape.triangleup, location.belowbar, COLOR_UP, size = size.small, text =  
"LONG", textcolor = chart.fg_color)
```

```
plotshape(showSignalsInput and validShortTrade, "Short Entry",  
shape.triangledown, location.abovebar, COLOR_DOWN, size = size.small, text  
= "SHORT", textcolor = chart.fg_color)
```

```
bgcolor(not na(time(timeframe.period, OPENING_RANGE_SESSION,  
timezoneInput)) ? COLOR_BG : na, title = "Opening Range Background")
```