

```
// 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 - Order Flow 7", "MFB - Order Flow 7", overlay = true, format =  
format.volume, precision = 0, max_labels_count = 20)
```

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

```
string TIMEZONE = "America/New_York"
```

```
string ASIA_SESSION = "1800-0000:1234567"
```

```
string LONDON_SESSION = "0000-0600:1234567"
```

```
string INITIAL_BALANCE_SESSION = "0930-1030:1234567"
```

```
int SECONDS_IN_FIVE_MINUTES = 300
```

```
int DEFAULT_CANDLES_AFTER_TOUCH = 7
```

```
int DEFAULT_ROW_TICKS = 1
```

```
int DEFAULT_LABEL_OFFSET_BARS = 12
```

```
float DEFAULT_LABEL_OFFSET_ATR = 2.5
```

```
color BULL_COLOR = #089981
```

```
color BEAR_COLOR = #f23645
```

```
color NEUTRAL_COLOR = #5b9cf6
```

```
color LABEL_BACKGROUND = #4a4a4a
```

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

```
int candlesAfterTouchInput = input.int(DEFAULT_CANDLES_AFTER_TOUCH,  
"Completed 5-minute candles to compare", minval = 1, maxval = 20, group =
```

"Order Flow 7", tooltip = "Number of completed 5-minute candles after a key-level touch used for both cumulative footprint delta and net price progress. Price is measured from the first counted candle's open to the last counted candle's close.")

int rowTicksInput = input.int(DEFAULT_ROW_TICKS, "Ticks per footprint row", minval = 1, maxval = 1000, group = "Order Flow 7", tooltip = "Price size of each footprint row, expressed in instrument ticks.")

int proximityTicksInput = input.int(4, "Key-level touch proximity (ticks)", minval = 0, maxval = 1000, group = "Key Levels", tooltip = "Price must trade within this many ticks of a key level to start tracking its interaction.")

float activeLevelDistanceAtrInput = input.float(0.75, "Active reading distance (ATR)", minval = 0.1, maxval = 5.0, step = 0.05, group = "Key Levels", tooltip = "Keeps a reading or countdown current only while the 5-minute close remains within this ATR distance of its key level. Tracking is cleared if price moves farther away.")

bool usePdhInput = input.bool(true, "PDH / PDL", group = "Key Levels", tooltip = "Use the previous day high and previous day low as automatic key levels.")

bool usePwhInput = input.bool(true, "PWH / PWL", group = "Key Levels", tooltip = "Use the previous week high and previous week low as automatic key levels.")

bool useAsiaInput = input.bool(true, "Asia high / low", group = "Key Levels", tooltip = "Use the completed Asia session high and low. Asia session is 6:00pm to 12:00am New York time.")

bool useLondonInput = input.bool(true, "London high / low", group = "Key Levels", tooltip = "Use the completed London session high and low. London session is 12:00am to 6:00am New York time.")

bool useInitialBalanceInput = input.bool(true, "Initial Balance high / low", group = "Key Levels", tooltip = "Use the completed Initial Balance high and low. Initial Balance is 9:30am to 10:30am New York time.")

bool showReadingsInput = input.bool(true, "Show readings", group = "Display", tooltip = "Shows a reading only for the active key level after the selected 5-minute candle count, and only while price remains within the active ATR distance. It is context, not a buy or sell signal.")

bool showAwarenessInput = input.bool(false, "Show Awareness box", group = "Display", tooltip = "Shows a beginner-friendly explanation for the current active key level, including the price/order-flow relationship and structure to wait for. Hidden by default.")

string awarenessTextSizeInput = input.string("Normal", "Awareness box text size", options = ["Tiny", "Small", "Normal", "Large", "Huge"], group = "Display", tooltip = "Controls the text size inside the optional Awareness box independently of the on-chart reading size.")

bool showProgressInput = input.bool(true, "Show counting progress", group = "Display", tooltip = "Shows progress for the active key level while counting. Counting is cleared if price closes farther away than the Active reading distance setting.")

string textSizeInput = input.string("Large", "Reading text size", options = ["Tiny", "Small", "Peace", "Large", "Huge"], group = "Display", tooltip = "Controls the size of the completed MFB reading. The temporary counting-progress label stays compact and uses Small text. Defaults to Large; Huge is also available.")

int labelOffsetBarsInput = input.int(DEFAULT_LABEL_OFFSET_BARS, "Label horizontal offset (bars)", minval = 0, maxval = 20, group = "Display", tooltip = "Moves the single active label to the right of the bar that completes the reading or the current bar during counting. Defaults to 12 bars to keep text clear of nearby candles; adjust from 0 to 20 bars.")

float labelOffsetAtrInput = input.float(DEFAULT_LABEL_OFFSET_ATR, "Label vertical offset (ATR)", minval = 0.25, maxval = 20.0, step = 0.25, group = "Display", tooltip = "Places the first label this many ATRs below the touched key level.")

```
float labelStackGapAtrInput = input.float(5.0, "Label stacking gap (ATR)",  
minval = 1.0, maxval = 20.0, step = 0.25, group = "Display", tooltip = "Vertical  
distance between the completed reading and the smaller active counting  
label when both are visible. Defaults to 5 ATR to reduce overlap.")
```

```
color bullColorInput = input.color(BULL_COLOR, "Bullish color", group =  
"Style", tooltip = "Color used for bullish divergence and bullish agreement  
readings.")
```

```
color bearColorInput = input.color(BEAR_COLOR, "Bearish color", group =  
"Style", tooltip = "Color used for bearish divergence and bearish agreement  
readings.")
```

```
color neutralColorInput = input.color(NEUTRAL_COLOR, "Neutral color",  
group = "Style", tooltip = "Color used for balanced, unavailable, or unclassified  
readings.")
```

```
string textSizeValue = switch textSizeInput
```

```
  "Tiny" => size.tiny
```

```
  "Small" => size.small
```

```
  "Peace" => size.normal
```

```
  "Large" => size.large
```

```
  "Huge" => size.huge
```

```
  => size.large
```

```
string awarenessTextSizeValue = switch awarenessTextSizeInput
```

```
  "Tiny" => size.tiny
```

```
  "Small" => size.small
```

```
  "Normal" => size.normal
```

"Large" => size.large

"Huge" => size.huge

=> size.normal

// --- Helper functions ---

touchesLevel(float levelPrice, float proximityPrice) =>

not na(levelPrice) and levelPrice > 0 and high >= levelPrice - proximityPrice
and low <= levelPrice + proximityPrice

considerTouchedLevel(float bestLevel, string bestName, float candidateLevel,
string candidateName, float proximityPrice) =>

bool candidateTouched = touchesLevel(candidateLevel, proximityPrice)

float candidateDistance = math.abs(close - candidateLevel)

float bestDistance = na(bestLevel) ? na : math.abs(close - bestLevel)

float tieTolerance = syminfo.mintick * 0.5

bool candidatelsCloser = candidateTouched and (na(bestLevel) or
candidateDistance < bestDistance - tieTolerance)

bool candidatelsTied = candidateTouched and not na(bestLevel) and
math.abs(candidateDistance - bestDistance) <= tieTolerance

float selectedLevel = candidatelsCloser ? candidateLevel : bestLevel

string selectedName = candidatelsCloser ? candidateName :
candidatelsTied and not str.contains(bestName, candidateName) ?
bestName + " / " + candidateName : bestName

[selectedLevel, selectedName]

```

// --- Chart and automatic key levels ---

bool isFiveMinuteChart = timeframe.in_seconds() ==
SECONDS_IN_FIVE_MINUTES

float proximityPrice = proximityTicksInput * syminfo.mintick

float atrValue = ta.atr(14)

float safeAtrValue = math.max(nz(atrValue, syminfo.mintick * 20),
syminfo.mintick * 10)

float activeLevelDistancePrice = safeAtrValue * activeLevelDistanceAtrInput

float labelOffsetPrice = safeAtrValue * labelOffsetAtrInput


[previousDayHigh, previousDayLow] = request.security(syminfo.tickerid, "D",
[high[1], low[1]], lookahead = barmerge.lookahead_on)

[previousWeekHigh, previousWeekLow] = request.security(syminfo.tickerid,
"W", [high[1], low[1]], lookahead = barmerge.lookahead_on)


bool asiaInSession = not na(time(timeframe.period, ASIA_SESSION,
TIMEZONE))

bool londonInSession = not na(time(timeframe.period, LONDON_SESSION,
TIMEZONE))

bool initialBalanceInSession = not na(time(timeframe.period,
INITIAL_BALANCE_SESSION, TIMEZONE))

bool asiaStarts = asiaInSession and not asiaInSession[1]

bool londonStarts = londonInSession and not londonInSession[1]

bool initialBalanceStarts = initialBalanceInSession and not
initialBalanceInSession[1]

bool asiaEnds = not asiaInSession and asiaInSession[1]

```

bool londonEnds = not londonInSession and londonInSession[1]

bool initialBalanceEnds = not initialBalanceInSession and
initialBalanceInSession[1]

var float asiaBuildingHigh = na

var float asiaBuildingLow = na

var float asiaHigh = na

var float asiaLow = na

var float londonBuildingHigh = na

var float londonBuildingLow = na

var float londonHigh = na

var float londonLow = na

var float initialBalanceBuildingHigh = na

var float initialBalanceBuildingLow = na

var float initialBalanceHigh = na

var float initialBalanceLow = na

if asiaStarts

 asiaBuildingHigh := high

 asiaBuildingLow := low

else if asiaInSession

 asiaBuildingHigh := math.max(nz(asiaBuildingHigh, high), high)

 asiaBuildingLow := math.min(nz(asiaBuildingLow, low), low)

if asiaEnds

asiaHigh := asiaBuildingHigh

asiaLow := asiaBuildingLow

if londonStarts

londonBuildingHigh := high

londonBuildingLow := low

else if londonInSession

londonBuildingHigh := math.max(nz(londonBuildingHigh, high), high)

londonBuildingLow := math.min(nz(londonBuildingLow, low), low)

if londonEnds

londonHigh := londonBuildingHigh

londonLow := londonBuildingLow

if initialBalanceStarts

initialBalanceBuildingHigh := high

initialBalanceBuildingLow := low

else if initialBalanceInSession

initialBalanceBuildingHigh := math.max(nz(initialBalanceBuildingHigh, high), high)

initialBalanceBuildingLow := math.min(nz(initialBalanceBuildingLow, low), low)

if initialBalanceEnds

initialBalanceHigh := initialBalanceBuildingHigh

initialBalanceLow := initialBalanceBuildingLow

[closestLevel1, closestName1] = considerTouchedLevel(na, "", usePdhInput ? previousDayHigh : na, "PDH", proximityPrice)

[closestLevel2, closestName2] = considerTouchedLevel(closestLevel1, closestName1, usePdhInput ? previousDayLow : na, "PDL", proximityPrice)

[closestLevel3, closestName3] = considerTouchedLevel(closestLevel2, closestName2, usePwhInput ? previousWeekHigh : na, "PWH", proximityPrice)

[closestLevel4, closestName4] = considerTouchedLevel(closestLevel3, closestName3, usePwhInput ? previousWeekLow : na, "PWL", proximityPrice)

[closestLevel5, closestName5] = considerTouchedLevel(closestLevel4, closestName4, useAsiaInput ? asiaHigh : na, "Asia High", proximityPrice)

[closestLevel6, closestName6] = considerTouchedLevel(closestLevel5, closestName5, useAsiaInput ? asiaLow : na, "Asia Low", proximityPrice)

[closestLevel7, closestName7] = considerTouchedLevel(closestLevel6, closestName6, useLondonInput ? londonHigh : na, "London High", proximityPrice)

[closestLevel8, closestName8] = considerTouchedLevel(closestLevel7, closestName7, useLondonInput ? londonLow : na, "London Low", proximityPrice)

[closestLevel9, closestName9] = considerTouchedLevel(closestLevel8, closestName8, useInitialBalanceInput ? initialBalanceHigh : na, "Initial Balance High", proximityPrice)

[closestLevel10, closestName10] = considerTouchedLevel(closestLevel9, closestName9, useInitialBalanceInput ? initialBalanceLow : na, "Initial Balance Low", proximityPrice)

float touchedKeyLevel = closestLevel10

```
string touchedKeyName = closestName10
```

```
bool keyLevelTouched = isFiveMinuteChart and not na(touchedKeyLevel)
```

```
bool keyLevelTouchIsCurrent = keyLevelTouched and math.abs(close -  
touchedKeyLevel) <= activeLevelDistancePrice
```

```
// --- One TradingView footprint request ---
```

```
footprint footprintId = request.footprint(rowTicksInput)
```

```
bool hasFootprint = not na(footprintId)
```

```
float barDelta = hasFootprint ? footprint.delta(footprintId) : na
```

```
float barTotalVolume = hasFootprint ? footprint.total_volume(footprintId) : na
```

```
bool currentFootprintValid = hasFootprint and not na(barDelta) and not  
na(barTotalVolume)
```

```
// --- Key-level observation window ---
```

```
var bool isCounting = false
```

```
var int countedCandles = 0
```

```
var int validCandles = 0
```

```
var float runningDelta = 0.0
```

```
var float runningVolume = 0.0
```

```
var float countingStartPrice = na
```

```
var float countingKeyLevel = na
```

```
var string countingKeyName = ""
```

```
var label readingLabel = na
```

```
var float readingLabelY = na
```

```
var label progressLabel = na
```

```
var string latestAwarenessText = "No active key-level reading.\nWaiting for a  
key-level touch."
```

```
var table awarenessTable = table.new(position.middle_right, 1, 1, frame_color  
= color.new(LABEL_BACKGROUND, 0), border_color =  
color.new(LABEL_BACKGROUND, 0), border_width = 1)
```

```
if isFiveMinuteChart and barstate.isconfirmed
```

```
    bool differentLevelTouched = keyLevelTouchIsCurrent and not  
na(countingKeyLevel) and math.abs(touchedKeyLevel - countingKeyLevel) >  
syminfo.mintick * 0.5
```

```
    bool startNewWindow = keyLevelTouchIsCurrent and (not isCounting or  
differentLevelTouched)
```

```
    bool currentLevelTooFar = not na(countingKeyLevel) and math.abs(close -  
countingKeyLevel) > activeLevelDistancePrice
```

```
if startNewWindow
```

```
    isCounting := true
```

```
    countedCandles := 0
```

```
    validCandles := 0
```

```
    runningDelta := 0.0
```

```
    runningVolume := 0.0
```

```
    countingStartPrice := na
```

```
    countingKeyLevel := touchedKeyLevel
```

```

countingKeyName := touchedKeyName

readingLabelY := na

latestAwarenessText := "ACTIVE LEVEL: " + countingKeyName + "\nTracking
price and order flow.\nContext only—not a trade signal."

if not na(readingLabel)
    label.delete(readingLabel)
    readingLabel := na
if not na(progressLabel)
    label.delete(progressLabel)
    progressLabel := na
else if isCounting and currentLevelTooFar
    isCounting := false
    countedCandles := 0
    validCandles := 0
    runningDelta := 0.0
    runningVolume := 0.0
    countingStartPrice := na
    countingKeyLevel := na
    countingKeyName := ""
    readingLabelY := na

    latestAwarenessText := "No active key-level reading.\nPrice moved
outside the active range.\nWaiting for a new key-level touch."

    if not na(readingLabel)
        label.delete(readingLabel)

```

```

    readingLabel := na
    if not na(progressLabel)
        label.delete(progressLabel)
        progressLabel := na
    else if isCounting
        if countedCandles == 0
            countingStartPrice := open
        countedCandles += 1
        if currentFootprintValid
            validCandles += 1
            runningDelta += barDelta
            runningVolume += barTotalVolume

        if countedCandles >= candlesAfterTouchInput
            bool validReading = validCandles == candlesAfterTouchInput and
runningVolume > 0 and not na(countingStartPrice)
            bool priceUp = validReading and close > countingStartPrice
            bool priceDown = validReading and close < countingStartPrice
            bool deltaUp = validReading and runningDelta > 0
            bool deltaDown = validReading and runningDelta < 0
            bool bullishDivergence = priceUp and deltaDown
            bool bearishDivergence = priceDown and deltaUp
            bool bullishAgreement = priceUp and deltaUp

```

bool bearishAgreement = priceDown and deltaDown

string priceDeltaText = not validReading ? "PRICE / DELTA UNAVAILABLE"
: priceUp ? "PRICE ↑ / DELTA " + (deltaUp ? "↑" : deltaDown ? "↓" : "FLAT") :
priceDown ? "PRICE ↓ / DELTA " + (deltaUp ? "↑" : deltaDown ? "↓" : "FLAT") :
"PRICE FLAT / DELTA " + (deltaUp ? "↑" : deltaDown ? "↓" : "FLAT")

string displayPriceDeltaText = not validReading ? "Price / Delta
unavailable" : priceUp ? "Price ↑ / Delta " + (deltaUp ? "↑" : deltaDown ? "↓" :
"flat") : priceDown ? "Price ↓ / Delta " + (deltaUp ? "↑" : deltaDown ? "↓" :
"flat") : "Price flat / Delta " + (deltaUp ? "↑" : deltaDown ? "↓" : "flat")

string readingText = not validReading ? "FOOTPRINT DATA
UNAVAILABLE" : bullishDivergence ? "Bullish Price/Delta Divergence" :
bearishDivergence ? "Bearish Price/Delta Divergence" : bullishAgreement or
bearishAgreement ? "Price/Delta Agreement" : "No clear directional
classification"

string relationshipText = bullishDivergence ? "Buyer-side Price/Order
Flow divergence" : bearishDivergence ? "Seller-side Price/Order Flow
divergence" : bullishAgreement ? "Buyer-side Price and Order Flow
agreement" : bearishAgreement ? "Seller-side Price and Order Flow
agreement" : not validReading ? "Price and Order Flow relationship
unavailable" : "No clear Price and Order Flow direction"

string disclaimerText = bullishDivergence or bullishAgreement ?
"Context only—not a buy signal. Wait for structural confirmation." :
bearishDivergence or bearishAgreement ? "Context only—not a sell signal.
Wait for structural confirmation." : "Context only—not a trade signal. Wait for
clearer structure."

string compactText = "ACTIVE LEVEL: " + countingKeyName + "\n" +
relationshipText + "\n" + displayPriceDeltaText + "\n" + disclaimerText

string detailedExplanationText = bullishDivergence ? "Price rose while
net delta was negative.\nSelling pressure did not push price lower.\nWait for
bullish 5m/15m CISD." : bearishDivergence ? "Price fell while net delta was

positive.\nBuying pressure did not push price higher.\nWait for bearish 5m/15m CISD." : bullishAgreement ? "Price and net delta both rose.\nWait for a 5m close above the level,\nthen a retest that holds.\nContext only—not an automatic buy." : bearishAgreement ? "Price and net delta both fell.\nWait for a 5m close below the level,\nthen a sell-retrace/rejection.\nContext only—not an automatic sell." : not validReading ? "Footprint data was unavailable\nfor the full comparison window." : "Price or net delta was flat.\nNo clear directional relationship."

color readingColor = bullishDivergence or bullishAgreement ?
bullColorInput : bearishDivergence or bearishAgreement ? bearColorInput :
neutralColorInput

latestAwarenessText := "ACTIVE LEVEL: " + countingKeyName +
"\nAWARENESS DETAILS\nReading: " + readingText + "\nRelationship:\n" +
relationshipText + "\n" + displayPriceDeltaText + "\n\n" +
detailedExplanationText + "\n\nCISD = Change in State of Delivery.\nDelta is
context, not an entry signal.\nWait for structure."

if not na(progressLabel)

label.delete(progressLabel)

progressLabel := na

if showReadingsInput

if not na(readingLabel)

label.delete(readingLabel)

readingLabelY := countingKeyLevel - labelOffsetPrice

readingLabel := label.new(bar_index + labelOffsetBarsInput,
readingLabelY, compactText, xloc = xloc.bar_index, yloc = yloc.price, color =
color.new(LABEL_BACKGROUND, 15), style = label.style_label_left, textcolor

```
= readingColor, size = textSizeValue, textalign = text.align_left, force_overlay = true)
```

```
    isCounting := false
```

```
    countedCandles := 0
```

```
    validCandles := 0
```

```
    runningDelta := 0.0
```

```
    runningVolume := 0.0
```

```
    countingStartPrice := na
```

```
else if not na(countingKeyLevel) and currentLevelTooFar
```

```
    countingKeyLevel := na
```

```
    countingKeyName := ""
```

```
    readingLabelY := na
```

```
    latestAwarenessText := "No active key-level reading.\nPrice moved  
outside the active range.\nWaiting for a new key-level touch."
```

```
    if not na(readingLabel)
```

```
        label.delete(readingLabel)
```

```
        readingLabel := na
```

```
    if not na(progressLabel)
```

```
        label.delete(progressLabel)
```

```
        progressLabel := na
```

```
// --- Active key-level progress and awareness panel ---
```

```
if barstate.islast
```

```

    if showProgressInput and isCounting and countedCandles <
candlesAfterTouchInput

        string progressText = countingKeyName + " · Counting " +
str.toString(countedCandles) + "/" + str.toString(candlesAfterTouchInput) + "
(5m)"

        float progressY = countingKeyLevel - labelOffsetPrice

        if not na(readingLabelY)

            progressY := readingLabelY - safeAtrValue * labelStackGapAtrInput

            if na(progressLabel)

                progressLabel := label.new(bar_index + labelOffsetBarsInput, progressY,
progressText, xloc = xloc.bar_index, yloc = yloc.price, color =
color.new(LABEL_BACKGROUND, 15), style = label.style_label_left, textcolor
= neutralColorInput, size = size.small, textalign = text.align_left, force_overlay
= true)

            else

                label.set_xy(progressLabel, bar_index + labelOffsetBarsInput,
progressY)

                label.set_text(progressLabel, progressText)

                label.set_textcolor(progressLabel, neutralColorInput)

                label.set_color(progressLabel, color.new(LABEL_BACKGROUND, 15))

                label.set_size(progressLabel, size.small)

            else if not na(progressLabel)

                label.delete(progressLabel)

                progressLabel := na

        if showAwarenessInput

```

```
table.cell(awarenessTable, 0, 0, latestAwarenessText, text_color =  
color.new(color.rgb(219, 219, 219), 0), text_halign = text.align_left, text_valign  
= text.align_top, text_size = awarenessTextSizeValue, bgcolor =  
color.new(LABEL_BACKGROUND, 0))
```

```
else
```

```
table.clear(awarenessTable, 0, 0)
```

```
// --- Informational alerts ---
```

```
bool differentAlertLevel = keyLevelTouchIsCurrent and not  
na(countingKeyLevel[1]) and math.abs(touchedKeyLevel -  
countingKeyLevel[1]) > syminfo.mintick * 0.5
```

```
bool touchStarted = isFiveMinuteChart and barstate.isconfirmed and  
keyLevelTouchIsCurrent and (not isCounting[1] or differentAlertLevel)
```

```
alertcondition(touchStarted, "MFB - Order Flow 7 active level tracking  
started", "MFB - Order Flow 7: a key level was touched. The indicator is  
tracking only the active key-level interaction and will clear it if price moves  
outside the configured ATR distance.")
```