

```
// 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 = 50)
```

```
// --- 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
```

```
color BULL_COLOR = #089981
```

```
color BEAR_COLOR = #f23645
```

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

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

```
int candlesAfterTouchInput = input.int(DEFAULT_CANDLES_AFTER_TOUCH,  
"Candles after key-level touch", minval = 1, maxval = 20, group = "Order Flow  
7", tooltip = "Number of completed 5-minute footprint candles counted after  
the key-level touch. The default is 7.")
```

```
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 proximity (ticks)", minval = 0,  
maxval = 1000, group = "Key Levels", tooltip = "Distance from a key level within  
which price is considered to have touched that level.")
```

```
float balancedDeltaThresholdInput = input.float(25.0, "Balanced Delta  
threshold (contracts)", minval = 0.0, maxval = 100000.0, step = 1.0, group =  
"Alerts", tooltip = "Absolute cumulative Delta within this range is classified as  
Balanced. Adjust for the instrument and session.")
```

```
int suppressionWindowInput = input.int(14, "Suppression window (5m  
candles)", minval = 0, maxval = 200, group = "Alerts", tooltip = "Number of  
completed 5-minute candles before the same key level can alert again.")
```

```
int displacementTicksInput = input.int(4, "15m displacement threshold  
(ticks)", minval = 0, maxval = 1000, group = "Alerts", tooltip = "Distance beyond  
a key level required for a confirmed 15-minute close to be treated as a  
decisive continuation break.")
```

```
bool enableAlertsInput = input.bool(true, "Enable context alerts", group =  
"Alerts", tooltip = "Alerts are context notifications only. They never issue a buy,  
sell, position-size, or risk instruction.")
```

```
bool showAlertLabelInput = input.bool(true, "Show alert reading label", group  
= "Alerts", tooltip = "Prints the latest alert reading and routing instruction on  
the chart.")
```

```
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 = "Prints one compact reading beneath the touched key level after the  
selected number of completed 5-minute candles.")
```

```
bool showProgressInput = input.bool(true, "Show counting progress", group =  
"Display", tooltip = "Shows a small temporary progress message beneath the  
touched key level while the footprint candles are being counted.")
```

```
string textSizeInput = input.string("Peace", "Reading text size", options =  
["Tiny", "Small", "Peace", "Large"], group = "Display", tooltip = "Controls the size  
of both the completed MFB reading and the temporary counting-progress  
text.")
```

```
string textSizeValue = switch textSizeInput
```

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

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

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

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

```
    => size.normal
```

```
float completedLabelOffsetAtrInput = input.float(5.0, "Completed reading  
offset (ATR)", minval = 1.0, maxval = 20.0, step = 0.5, group = "Display", tooltip
```

= "Vertical distance below the key level for the completed alert reading.
Increase this value to keep text farther from candles and companion labels.")

color bullColorInput = input.color(BULL_COLOR, "Bullish color", group =
"Style", tooltip = "Color used for positive delta readings.")

color bearColorInput = input.color(BEAR_COLOR, "Bearish color", group =
"Style", tooltip = "Color used for negative delta readings.")

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

// --- Helper functions ---

touchesLevel(float levelPrice, float proximityPrice) =>

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

isLowLevel(string keyName) =>

keyName == "PDL" or keyName == "PWL" or keyName == "Asia Low" or
keyName == "London Low" or keyName == "Initial Balance Low"

isHighLevel(string keyName) =>

keyName == "PDH" or keyName == "PWH" or keyName == "Asia High" or
keyName == "London High" or keyName == "Initial Balance High"

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

bool isFiveMinuteChart = timeframe.in_seconds() ==
SECONDS_IN_FIVE_MINUTES

float proximityPrice = proximityTicksInput * syminfo.mintick

```
float displacementPrice = displacementTicksInput * syminfo.mintick
```

```
float atrValue = ta.atr(14)
```

```
[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)
```

```
[previous15Open, previous15Close] = request.security(syminfo.tickerid, "15",  
[open[1], close[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

float touchedKeyLevel = na

string touchedKeyName = ""

if usePdhInput and touchesLevel(previousDayHigh, proximityPrice)

```
touchedKeyLevel := previousDayHigh
touchedKeyName := "PDH"
else if usePdhInput and touchesLevel(previousDayLow, proximityPrice)
    touchedKeyLevel := previousDayLow
    touchedKeyName := "PDL"
else if usePwhInput and touchesLevel(previousWeekHigh, proximityPrice)
    touchedKeyLevel := previousWeekHigh
    touchedKeyName := "PWH"
else if usePwhInput and touchesLevel(previousWeekLow, proximityPrice)
    touchedKeyLevel := previousWeekLow
    touchedKeyName := "PWL"
else if useAsiaInput and touchesLevel(asiaHigh, proximityPrice)
    touchedKeyLevel := asiaHigh
    touchedKeyName := "Asia High"
else if useAsiaInput and touchesLevel(asiaLow, proximityPrice)
    touchedKeyLevel := asiaLow
    touchedKeyName := "Asia Low"
else if useLondonInput and touchesLevel(londonHigh, proximityPrice)
    touchedKeyLevel := londonHigh
    touchedKeyName := "London High"
else if useLondonInput and touchesLevel(londonLow, proximityPrice)
    touchedKeyLevel := londonLow
    touchedKeyName := "London Low"
```

```
else if useInitialBalanceInput and touchesLevel(initialBalanceHigh,  
proximityPrice)
```

```
    touchedKeyLevel := initialBalanceHigh
```

```
    touchedKeyName := "Initial Balance High"
```

```
else if useInitialBalanceInput and touchesLevel(initialBalanceLow,  
proximityPrice)
```

```
    touchedKeyLevel := initialBalanceLow
```

```
    touchedKeyName := "Initial Balance Low"
```

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

```
// --- 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)
```

```
// --- Alert suppression state ---
```

```
var array<string> alertKeyNames = array.from("PDH", "PDL", "PWH", "PWL",  
"Asia High", "Asia Low", "London High", "London Low", "Initial Balance High",  
"Initial Balance Low")
```

```
var array<int> suppressionCounters = array.new_int(10, -1)
```

```

// --- Touch-to-seven-candle state machine ---

var bool isCounting = false
var int countedCandles = 0
var int validCandles = 0
var float runningDelta = 0.0
var float runningVolume = 0.0
var float countingKeyLevel = na
var string countingKeyName = ""
var label latestReadingLabel = na
var label progressLabel = na
bool contextAlertThisBar = false

if isFiveMinuteChart and barstate.isconfirmed
    int suppressionCountSize = suppressionCounters.size()
    if suppressionCountSize > 0
        for counterIndex = 0 to suppressionCountSize - 1
            int counterValue = suppressionCounters.get(counterIndex)
            if counterValue >= 0
                suppressionCounters.set(counterIndex, counterValue + 1)

if isCounting
    countedCandles += 1
    if currentFootprintValid

```

validCandles += 1

runningDelta += barDelta

runningVolume += barTotalVolume

if countedCandles >= candlesAfterTouchInput

bool validReading = validCandles == candlesAfterTouchInput and
runningVolume > 0

bool isNegativeDelta = validReading and runningDelta < -
balancedDeltaThresholdInput

bool isPositiveDelta = validReading and runningDelta >
balancedDeltaThresholdInput

bool isBalancedDelta = validReading and not isNegativeDelta and not
isPositiveDelta

string readingText = not validReading ? "FOOTPRINT DATA
UNAVAILABLE" : isNegativeDelta ? "NEGATIVE DELTA" : isPositiveDelta ?
"POSITIVE DELTA" : "BALANCED DELTA"

string routingText = "No Delta preference — Rely on structure"

bool bullish15mBreak = validReading and previous15Close >
countingKeyLevel + displacementPrice and previous15Open <=
countingKeyLevel

bool bearish15mBreak = validReading and previous15Close <
countingKeyLevel - displacementPrice and previous15Open >=
countingKeyLevel

if bullish15mBreak

```
        routingText := isPositiveDelta ? "Check 5m Bullish FVG-123" :  
isNegativeDelta ? "Conflict: Breakout vs Delta — Wait" : "No Delta preference  
— Rely on structure"
```

```
    else if bearish15mBreak
```

```
        routingText := isNegativeDelta ? "Check 5m Bearish FVG-123" :  
isPositiveDelta ? "Conflict: Breakout vs Delta — Wait" : "No Delta preference  
— Rely on structure"
```

```
    else if isLowLevel(countingKeyName)
```

```
        routingText := isNegativeDelta ? "Check 15m Bullish CISD" :  
isPositiveDelta ? "Conflict: Delta opposes level type — Wait" : "No Delta  
preference — Rely on structure"
```

```
    else if isHighLevel(countingKeyName)
```

```
        routingText := isPositiveDelta ? "Check 15m Bearish CISD" :  
isNegativeDelta ? "Conflict: Delta opposes level type — Wait" : "No Delta  
preference — Rely on structure"
```

```
int currentKeyIndex = alertKeyNames.indexOf(countingKeyName)
```

```
bool suppressionExpired = true
```

```
if currentKeyIndex >= 0
```

```
    int currentSuppression = suppressionCounters.get(currentKeyIndex)
```

```
    suppressionExpired := currentSuppression < 0 or currentSuppression  
>= suppressionWindowInput
```

```
bool alertReady = validReading and suppressionExpired
```

```
bool alertFired = false
```

```
if enableAlertsInput and alertReady
```

```
    string alertMessage = countingKeyName + " — " + readingText + " — " +  
routingText
```

```
    alert(alertMessage, alert.freq_once_per_bar_close)
```

```
    contextAlertThisBar := true
```

```
    alertFired := true
```

```
    if currentKeyIndex >= 0
```

```
        suppressionCounters.set(currentKeyIndex, 0)
```

```
    bool directionalReading = isNegativeDelta or isPositiveDelta
```

```
    bool shouldDisplayReading = (showReadingsInput and  
directionalReading) or (showAlertLabelInput and alertFired)
```

```
    color readingColor = isNegativeDelta ? bearColorInput : isPositiveDelta  
? bullColorInput : neutralColorInput
```

```
    if shouldDisplayReading
```

```
        if not na(latestReadingLabel)
```

```
            label.delete(latestReadingLabel)
```

```
            string compactText = countingKeyName + " — " + readingText + "\n" +  
routingText
```

```
            latestReadingLabel := label.new(bar_index, countingKeyLevel -  
atrValue * completedLabelOffsetAtrInput, compactText, xloc =  
xloc.bar_index, yloc = yloc.price, color = color(na), style = label.style_none,  
textcolor = readingColor, size = textSizeValue, textalign = text.align_center,  
force_overlay = true)
```

```
    isCounting := false
```

```
    countedCandles := 0
```

```

    validCandles := 0
    runningDelta := 0.0
    runningVolume := 0.0
    countingKeyLevel := na
    countingKeyName := ""
else if keyLevelTouched
    isCounting := true
    countedCandles := 0
    validCandles := 0
    runningDelta := 0.0
    runningVolume := 0.0
    countingKeyLevel := touchedKeyLevel
    countingKeyName := touchedKeyName

// --- Temporary counting-progress label ---
if barstate.islast
    if showProgressInput and isCounting and countedCandles <
candlesAfterTouchInput
        string progressText = "MFB - Order Flow 7 · " + countingKeyName +
"\nCounting: " + str.toString(countedCandles) + " / " +
str.toString(candlesAfterTouchInput) + " completed 5m candles"
        float progressY = countingKeyLevel - atrValue *
(completedLabelOffsetAtrInput + 1.5)
        if na(progressLabel)

```

```
    progressLabel := label.new(bar_index, progressY, progressText, xloc =  
xloc.bar_index, yloc = yloc.price, color = color(na), style = label.style_none,  
textcolor = neutralColorInput, size = textSizeValue, textalign =  
text.align_center, force_overlay = true)
```

```
    else
```

```
        label.set_xy(progressLabel, bar_index, progressY)
```

```
        label.set_text(progressLabel, progressText)
```

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

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

```
    progressLabel := na
```

```
// --- Alert conditions ---
```

```
bool touchStarted = isFiveMinuteChart and barstate.isconfirmed and  
keyLevelTouched and not isCounting[1]
```

```
alertcondition(contextAlertThisBar, "MFB - Order Flow 7 context alert", "MFB -  
Order Flow 7 context alert fired. Use Any alert() function call for the full  
dynamic key-level routing message.")
```

```
alertcondition(touchStarted, "MFB - Order Flow 7 counting started", "MFB -  
Order Flow 7: price touched an automatic key level. The indicator is now  
counting completed 5-minute footprint candles.")
```