

// 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.")
```

```
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
```

```
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
```

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

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

```
float proximityPrice = proximityTicksInput * 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)
```

```
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

bool pdhTouched = usePdhInput and (touchesLevel(previousDayHigh, proximityPrice) or touchesLevel(previousDayLow, proximityPrice))

bool pwhTouched = usePwhInput and (touchesLevel(previousWeekHigh, proximityPrice) or touchesLevel(previousWeekLow, proximityPrice))

bool asiaTouched = useAsiaInput and (touchesLevel(asiaHigh, proximityPrice) or touchesLevel(asiaLow, proximityPrice))

bool londonTouched = useLondonInput and (touchesLevel(londonHigh, proximityPrice) or touchesLevel(londonLow, proximityPrice))

bool initialBalanceTouched = useInitialBalanceInput and (touchesLevel(initialBalanceHigh, proximityPrice) or touchesLevel(initialBalanceLow, proximityPrice))

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)
```

```
// --- 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
```

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

```
    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 < 0
bool isPositiveDelta = validReading and runningDelta > 0
bool directionalReading = isNegativeDelta or isPositiveDelta
bool canPrintReading = showReadingsInput and directionalReading
color readingColor = isNegativeDelta ? bearColorInput : isPositiveDelta
? bullColorInput : neutralColorInput

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

string contextText = isNegativeDelta ? "Check for Bullish 15m CISD
structure." : isPositiveDelta ? "Check for Bearish 15m CISD structure." :
"Balanced or unavailable reading."

if canPrintReading
    if not na(latestReadingLabel)
        label.delete(latestReadingLabel)

        string compactText = countingKeyName + " — " + readingText + "\n" +
contextText

        latestReadingLabel := label.new(bar_index, countingKeyLevel -
atrValue * 2.5, 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

// --- Optional status marker for the active count ---
var label progressLabel = na
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 * 3.5
        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
```

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

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

```
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.")
```