

// 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 - One chart - 1 min", shorttitle = "MFB - One chart - 1 min", overlay = true,
max_lines_count = 200, max_labels_count = 200)

// --- Constants ---

string TOOLTIP_PIVOT = "Bars used on each side to confirm swing highs and lows. Lower
values detect nearer-term levels."

string TOOLTIP_BUFFER = "Minimum distance price must trade beyond a level before the
sweep is recognized."

string TOOLTIP_WINDOW = "Maximum number of bars allowed between a liquidity sweep
and its market structure shift."

string TOOLTIP_WICK = "Minimum rejection wick as a percentage of the full candle range."

string TOOLTIP_CLOSE_POSITION = "Bearish candles must close at or below this fraction
of the candle range. Bullish candles use the mirrored minimum."

string TOOLTIP_ATR_FILTER = "When enabled, the key-level sweep depth must be between
the selected minimum and maximum ATR multiples."

string SESSION_TIMEZONE = "America/New_York"

color BULL_COLOR = #089981

color BEAR_COLOR = #f23645

color LEVEL_COLOR = #5b9cf6

// --- Inputs ---

int pivotLeftInput = input.int(3, "Pivot left bars", minval = 1, group = "Detection", tooltip =
TOOLTIP_PIVOT)

int pivotRightInput = input.int(3, "Pivot right bars", minval = 1, group = "Detection", tooltip =
TOOLTIP_PIVOT)

```
float sweepBufferInput = input.float(0.0, "Sweep buffer (ATR)", minval = 0.0, step = 0.05,
group = "Detection", tooltip = TOOLTIP_BUFFER)
```

```
int confirmationWindowInput = input.int(20, "MSS confirmation window", minval = 1,
maxval = 100, group = "Confirmation", tooltip = TOOLTIP_WINDOW)
```

```
bool requireMssInput = input.bool(true, "Require market structure shift", group =
"Confirmation", tooltip = "When enabled, standard entry markers require price to break the
internal swing in the sweep direction after the rejection.")
```

```
int atrLengthInput = input.int(14, "ATR length", minval = 1, group = "Confirmation", tooltip =
"ATR length used to calculate the optional sweep buffer and HTF PWR ATR filter.")
```

```
bool useDailyLevelsInput = input.bool(true, "Use PDH / PDL", group = "Key Levels", tooltip =
"Uses the prior completed TradingView daily high and low as key liquidity levels.")
```

```
bool useWeeklyLevelsInput = input.bool(true, "Use PWH / PWL", group = "Key Levels",
tooltip = "Uses the prior completed TradingView weekly high and low as key liquidity
levels.")
```

```
bool useAsiaLevelsInput = input.bool(true, "Use Asia high / low", group = "Key Levels",
tooltip = "Uses the prior completed Asia session, defined as 6:00 PM to midnight New York
time.")
```

```
bool useLondonLevelsInput = input.bool(true, "Use London high / low", group = "Key
Levels", tooltip = "Uses the prior completed London session, defined as midnight to 6:00
AM New York time.")
```

```
bool enableKeyLevelAlertsInput = input.bool(true, "Enable key-level entry alerts", group =
"Key Levels", tooltip = "Enables alerts for a key-level sweep followed by a close beyond the
most recent confirmed pre-sweep swing.")
```

```
float rejectionWickRatioInput = input.float(0.35, "Minimum rejection wick ratio", minval =
0.0, maxval = 1.0, step = 0.05, group = "HTF PWR", tooltip = TOOLTIP_WICK)
```

```
float bearishClosePositionInput = input.float(0.40, "Bearish close position maximum",
minval = 0.0, maxval = 1.0, step = 0.05, group = "HTF PWR", tooltip =
TOOLTIP_CLOSE_POSITION)
```

```
float bullishClosePositionInput = input.float(0.60, "Bullish close position minimum", minval
= 0.0, maxval = 1.0, step = 0.05, group = "HTF PWR", tooltip = TOOLTIP_CLOSE_POSITION)
```

```
bool useAtrPowerFilterInput = input.bool(false, "Use ATR - HTF Key Level Power Candle",
group = "HTF PWR", tooltip = TOOLTIP_ATR_FILTER)
```

float minimumNormalizedSweepInput = input.float(0.05, "Minimum sweep depth (ATR)", minval = 0.0, step = 0.05, group = "HTF PWR", tooltip = "Minimum distance the candle must pierce beyond the key level, measured in ATR.")

float maximumNormalizedSweepInput = input.float(0.40, "Maximum sweep depth (ATR)", minval = 0.0, step = 0.05, group = "HTF PWR", tooltip = "Maximum distance the candle may pierce beyond the key level, measured in ATR.")

bool enablePowerReversalAlertsInput = input.bool(true, "Enable HTF PWR alerts", group = "HTF PWR", tooltip = "Enables alerts when a key level is swept and the same confirmed candle closes back inside with the required rejection characteristics.")

bool enableFvg123Input = input.bool(true, "Enable FVG-123 signals", group = "FVG-123", tooltip = "Detects a three-candle fair value gap, its later retest, and a confirmed close breaking out in the gap direction.")

bool requireFvg123EmaCrossInput = input.bool(true, "Require 9/20 EMA cross", group = "FVG-123", tooltip = "Requires the matching-direction EMA cross after the qualifying key-level touch. One FVG-123 signal is allowed per EMA cross.")

bool requireFvg123KeyTouchInput = input.bool(true, "Require prior key-level touch", group = "FVG-123", tooltip = "Requires price to touch an enabled PDH/PDL, PWH/PWL, or completed Asia/London extreme before the FVG forms.")

int fvg123KeyTouchLookbackInput = input.int(30, "Key-level touch lookback (bars)", minval = 1, group = "FVG-123", tooltip = "Maximum bars between the prior key-level touch and the FVG-123 breakout signal.")

int fvg123MaxAgeInput = input.int(100, "FVG-123 maximum age (bars)", minval = 1, group = "FVG-123", tooltip = "Maximum age of a fair value gap when its breakout signal occurs.")

float fvg123MinWidthTicksInput = input.float(0.0, "FVG-123 minimum width (ticks)", minval = 0.0, group = "FVG-123", tooltip = "Optional minimum gap width in ticks. Zero allows any positive-width FVG.")

int fvgFastEmaLengthInput = input.int(9, "Fast EMA length", minval = 1, group = "FVG-123", tooltip = "Fast EMA used for FVG-123 trend and cross confirmation.")

int fvgSlowEmaLengthInput = input.int(20, "Slow EMA length", minval = 1, group = "FVG-123", tooltip = "Slow EMA used for FVG-123 trend and cross confirmation.")

float fvgRewardRiskInput = input.float(2.0, "Target risk/reward multiple", minval = 0.1, step = 0.1, group = "FVG-123 Risk/Reward", tooltip = "Target distance as a multiple of entry-to-stop risk. Default 2.0 means a 1:2 risk/reward target.")

int fvgProjectionBarsInput = input.int(10, "Risk/reward line length (bars)", minval = 1, maxval = 100, group = "FVG-123 Risk/Reward", tooltip = "Number of bars to project the compact dotted entry, stop, and target guides to the right of each signal.")

int fvgRiskRewardLineWidthInput = input.int(3, "Trade line width", minval = 1, maxval = 5, group = "FVG-123 Risk/Reward", tooltip = "Line thickness for FVG-123 entry, stop, and target guides.")

int maxDisplayedFvgSetupsInput = input.int(5, "Maximum displayed FVG setups", minval = 1, maxval = 20, group = "FVG-123 Risk/Reward", tooltip = "Limits how many recent FVG-123 entry/stop/target guides remain visible.")

bool showFvg123SignalsInput = input.bool(true, "Show FVG-123 signal markers", group = "Style", tooltip = "Displays bullish and bearish FVG-123 breakout markers on the signal bars.")

bool showFvgRiskRewardInput = input.bool(true, "Show FVG-123 entry/stop/target lines", group = "Style", tooltip = "Draws projected entry, stop, and configured risk/reward target guides for recent FVG-123 signals.")

bool showFvgRiskRewardLabelsInput = input.bool(false, "Show entry/SL/TP text tags", group = "Style", tooltip = "Adds Entry, SL, and target-multiple text tags to the FVG-123 risk/reward lines. Off by default for a cleaner chart.")

bool showSweepMarkersInput = input.bool(false, "Show sweep labels", group = "Style", tooltip = "Displays the bullish and bearish liquidity-sweep markers on price bars.")

bool showPivotPointsInput = input.bool(false, "Show confirmed pivots", group = "Style", tooltip = "Displays the confirmed swing points used as liquidity levels.")

bool showSweepLevelsInput = input.bool(false, "Show blue swing sweep-reference lines", group = "Style", tooltip = "Displays the blue horizontal lines at confirmed swing highs and lows used by the standard liquidity-sweep detector.")

bool showKeyLevelSweepLinesInput = input.bool(true, "Show swept key-level lines", group = "Style", tooltip = "Displays the latest swept prior-day, prior-week, Asia, and London key levels.")

bool showPowerReversalInput = input.bool(false, "Show HTF PWR labels", group = "Style", tooltip = "Displays PWR to the left of the usual Sweep marker when a key-level power reversal candle is confirmed.")

```
bool hidePowerReversalOnOneMinuteInput = input.bool(false, "Hide HTF PWR labels on 1m", group = "Style", tooltip = "Automatically hides PWR labels on the 1-minute chart while keeping the PWR alert conditions active.")
```

```
color bullishColorInput = input.color(BULL_COLOR, "Bullish color", group = "Style", tooltip = "Color used for bullish sweep and entry signals.")
```

```
color bearishColorInput = input.color(BEAR_COLOR, "Bearish color", group = "Style", tooltip = "Color used for bearish sweep and entry signals.")
```

```
color levelColorInput = input.color(LEVEL_COLOR, "Level color", group = "Style", tooltip = "Color used for liquidity level lines.")
```

```
// --- Higher-timeframe key levels ---
```

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

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

```
// --- Completed Asia and London session levels ---
```

```
int currentNewYorkHour = hour(time, SESSION_TIMEZONE)
```

```
bool inAsiaSession = currentNewYorkHour >= 18
```

```
bool inLondonSession = currentNewYorkHour < 6
```

```
var bool asiaSessionActive = false
```

```
var float asiaSessionHigh = na
```

```
var float asiaSessionLow = na
```

```
var float previousAsiaHigh = na
```

```
var float previousAsiaLow = na
```

```
var bool londonSessionActive = false
```

```
var float londonSessionHigh = na
```

```
var float londonSessionLow = na
var float previousLondonHigh = na
var float previousLondonLow = na
```

```
if inAsiaSession
```

```
  if not asiaSessionActive
```

```
    asiaSessionHigh := high
```

```
    asiaSessionLow := low
```

```
    asiaSessionActive := true
```

```
  else
```

```
    asiaSessionHigh := math.max(asiaSessionHigh, high)
```

```
    asiaSessionLow := math.min(asiaSessionLow, low)
```

```
else if asiaSessionActive
```

```
  previousAsiaHigh := asiaSessionHigh
```

```
  previousAsiaLow := asiaSessionLow
```

```
  asiaSessionActive := false
```

```
if inLondonSession
```

```
  if not londonSessionActive
```

```
    londonSessionHigh := high
```

```
    londonSessionLow := low
```

```
    londonSessionActive := true
```

```
  else
```

```
    londonSessionHigh := math.max(londonSessionHigh, high)
```

```
    londonSessionLow := math.min(londonSessionLow, low)
```

```
else if londonSessionActive
```

```

previousLondonHigh := londonSessionHigh
previousLondonLow := londonSessionLow
londonSessionActive := false

// --- Confirmed pivots and volatility buffer ---
float pivotHigh = ta.pivohigh(high, pivotLeftInput, pivotRightInput)
float pivotLow = ta.pivotlow(low, pivotLeftInput, pivotRightInput)
float atrValue = ta.atr(atrLengthInput)
float sweepBuffer = atrValue * sweepBufferInput

var float lastSwingHighPrice = na
var int lastSwingHighIndex = na
var float lastSwingLowPrice = na
var int lastSwingLowIndex = na

// --- Standard sweep state ---
var int lastSweptHighIndex = na
var int lastSweptLowIndex = na
var bool pendingBullish = false
var bool pendingBearish = false
var int bullishSweepIndex = na
var int bearishSweepIndex = na
var float bullishTriggerPrice = na
var float bearishTriggerPrice = na
var line bullishLevelLine = na
var line bearishLevelLine = na

```

```
// --- Standard liquidity sweep detection ---
```

```
bool freshHighLevel = na(lastSweptHighIndex) ? true : lastSweptHighIndex !=  
lastSwingHighIndex
```

```
bool freshLowLevel = na(lastSweptLowIndex) ? true : lastSweptLowIndex !=  
lastSwingLowIndex
```

```
bool bullishSweep = not na(lastSwingLowPrice) and freshLowLevel and low <  
lastSwingLowPrice - sweepBuffer and close > lastSwingLowPrice
```

```
bool bearishSweep = not na(lastSwingHighPrice) and freshHighLevel and high >  
lastSwingHighPrice + sweepBuffer and close < lastSwingHighPrice
```

```
// A candle that sweeps both sides is ignored to keep the directional signal unambiguous.
```

```
bool bullishSweepEvent = bullishSweep and not bearishSweep
```

```
bool bearishSweepEvent = bearishSweep and not bullishSweep
```

```
if bullishSweepEvent
```

```
    lastSweptLowIndex := lastSwingLowIndex
```

```
    pendingBullish := true
```

```
    pendingBearish := false
```

```
    bullishSweepIndex := bar_index
```

```
    bullishTriggerPrice := lastSwingHighPrice
```

```
    if not na(bullishLevelLine)
```

```
        line.delete(bullishLevelLine)
```

```
    if showSweepLevelsInput
```

```
        bullishLevelLine := line.new(lastSwingLowIndex, lastSwingLowPrice, bar_index,  
lastSwingLowPrice, xloc = xloc.bar_index, extend = extend.right, color =  
color.new(levelColorInput, 15), style = line.style_dashed, width = 1)
```

```

if bearishSweepEvent

    lastSweptHighIndex := lastSwingHighIndex

    pendingBearish := true

    pendingBullish := false

    bearishSweepIndex := bar_index

    bearishTriggerPrice := lastSwingLowPrice

    if not na(bearishLevelLine)

        line.delete(bearishLevelLine)

    if showSweepLevelsInput

        bearishLevelLine := line.new(lastSwingHighIndex, lastSwingHighPrice, bar_index,
lastSwingHighPrice, xloc = xloc.bar_index, extend = extend.right, color =
color.new(levelColorInput, 15), style = line.style_dashed, width = 1)

// --- Key-level sweep candidates ---

bool pdhSweep = useDailyLevelsInput and not na(previousDayHigh) and high >
previousDayHigh + sweepBuffer and close < previousDayHigh

bool pdlSweep = useDailyLevelsInput and not na(previousDayLow) and low <
previousDayLow - sweepBuffer and close > previousDayLow

bool pwhSweep = useWeeklyLevelsInput and not na(previousWeekHigh) and high >
previousWeekHigh + sweepBuffer and close < previousWeekHigh

bool pwlSweep = useWeeklyLevelsInput and not na(previousWeekLow) and low <
previousWeekLow - sweepBuffer and close > previousWeekLow

bool asiaHighSweep = useAsiaLevelsInput and not na(previousAsiaHigh) and high >
previousAsiaHigh + sweepBuffer and close < previousAsiaHigh

bool asiaLowSweep = useAsiaLevelsInput and not na(previousAsiaLow) and low <
previousAsiaLow - sweepBuffer and close > previousAsiaLow

bool londonHighSweep = useLondonLevelsInput and not na(previousLondonHigh) and high
> previousLondonHigh + sweepBuffer and close < previousLondonHigh

```

```
bool londonLowSweep = useLondonLevelsInput and not na(previousLondonLow) and low <
previousLondonLow - sweepBuffer and close > previousLondonLow
```

```
var string lastBullishKeyName = ""
```

```
var float lastBullishKeyPrice = na
```

```
var string lastBearishKeyName = ""
```

```
var float lastBearishKeyPrice = na
```

```
bool pdllsNew = lastBullishKeyName != "PDL" ? true : lastBullishKeyPrice !=
previousDayLow
```

```
bool pwllsNew = lastBullishKeyName != "PWL" ? true : lastBullishKeyPrice !=
previousWeekLow
```

```
bool asiaLowlsNew = lastBullishKeyName != "Asia Low" ? true : lastBullishKeyPrice !=
previousAsiaLow
```

```
bool londonLowlsNew = lastBullishKeyName != "London Low" ? true : lastBullishKeyPrice
!= previousLondonLow
```

```
bool pdhlsNew = lastBearishKeyName != "PDH" ? true : lastBearishKeyPrice !=
previousDayHigh
```

```
bool pwhlsNew = lastBearishKeyName != "PWH" ? true : lastBearishKeyPrice !=
previousWeekHigh
```

```
bool asiaHighlsNew = lastBearishKeyName != "Asia High" ? true : lastBearishKeyPrice !=
previousAsiaHigh
```

```
bool londonHighlsNew = lastBearishKeyName != "London High" ? true :
lastBearishKeyPrice != previousLondonHigh
```

```
float bullishKeyCandidatePrice = na
```

```
string bullishKeyCandidateName = ""
```

```
if pdlSweep and pdllsNew
```

```
    bullishKeyCandidatePrice := previousDayLow
```

```
    bullishKeyCandidateName := "PDL"
else if pwlSweep and pwllsNew
    bullishKeyCandidatePrice := previousWeekLow
    bullishKeyCandidateName := "PWL"
else if asiaLowSweep and asiaLowlsNew
    bullishKeyCandidatePrice := previousAsiaLow
    bullishKeyCandidateName := "Asia Low"
else if londonLowSweep and londonLowlsNew
    bullishKeyCandidatePrice := previousLondonLow
    bullishKeyCandidateName := "London Low"

float bearishKeyCandidatePrice = na
string bearishKeyCandidateName = ""
if pdhSweep and pdhlsNew
    bearishKeyCandidatePrice := previousDayHigh
    bearishKeyCandidateName := "PDH"
else if pwhSweep and pwhlsNew
    bearishKeyCandidatePrice := previousWeekHigh
    bearishKeyCandidateName := "PWH"
else if asiaHighSweep and asiaHighlsNew
    bearishKeyCandidatePrice := previousAsiaHigh
    bearishKeyCandidateName := "Asia High"
else if londonHighSweep and londonHighlsNew
    bearishKeyCandidatePrice := previousLondonHigh
    bearishKeyCandidateName := "London High"
```

// A candle that sweeps key levels on both sides is ignored for the key-level state machine.

bool bullishKeySweepEvent = bullishKeyCandidateName != "" and
bearishKeyCandidateName == ""

bool bearishKeySweepEvent = bearishKeyCandidateName != "" and
bullishKeyCandidateName == ""

// --- HTF Key Level Power Reversal Candle ---

float candleRange = high - low

float upperWick = high - math.max(open, close)

float lowerWick = math.min(open, close) - low

float closePosition = candleRange > 0.0 ? (close - low) / candleRange : na

float bearishNormalizedSweep = not na(bearishKeyCandidatePrice) and not na(atrValue)
and atrValue > 0.0 ? (high - bearishKeyCandidatePrice) / atrValue : na

float bullishNormalizedSweep = not na(bullishKeyCandidatePrice) and not na(atrValue)
and atrValue > 0.0 ? (bullishKeyCandidatePrice - low) / atrValue : na

bool bearishAtrQuality = not useAtrPowerFilterInput or not na(bearishNormalizedSweep)
and bearishNormalizedSweep >= minimumNormalizedSweepInput and
bearishNormalizedSweep <= maximumNormalizedSweepInput

bool bullishAtrQuality = not useAtrPowerFilterInput or not na(bullishNormalizedSweep)
and bullishNormalizedSweep >= minimumNormalizedSweepInput and
bullishNormalizedSweep <= maximumNormalizedSweepInput

// The candle must sweep the key level, reclaim it on close, show a dominant rejection
wick,

// and finish in the opposite portion of its range. PWR is evaluated only on a confirmed
close.

bool bearishPowerReversal = barstate.isconfirmed and bearishKeySweepEvent and
candleRange > 0.0 and upperWick / candleRange >= rejectionWickRatioInput and
closePosition <= bearishClosePositionInput and bearishAtrQuality

```
bool bullishPowerReversal = barstate.isconfirmed and bullishKeySweepEvent and  
candleRange > 0.0 and lowerWick / candleRange >= rejectionWickRatioInput and  
closePosition >= bullishClosePositionInput and bullishAtrQuality
```

```
bool bullishPowerReversalAlert = enablePowerReversalAlertsInput and  
bullishPowerReversal
```

```
bool bearishPowerReversalAlert = enablePowerReversalAlertsInput and  
bearishPowerReversal
```

```
bool anyPowerReversalAlert = bullishPowerReversalAlert or bearishPowerReversalAlert
```

```
// --- Key-level sweep state ---
```

```
var bool pendingBullishKey = false
```

```
var bool pendingBearishKey = false
```

```
var int bullishKeySweepIndex = na
```

```
var int bearishKeySweepIndex = na
```

```
var float bullishKeyTriggerPrice = na
```

```
var float bearishKeyTriggerPrice = na
```

```
var string bullishKeyName = ""
```

```
var string bearishKeyName = ""
```

```
var line bullishKeyLevelLine = na
```

```
var line bearishKeyLevelLine = na
```

```
if bullishKeySweepEvent
```

```
    lastBullishKeyName := bullishKeyCandidateName
```

```
    lastBullishKeyPrice := bullishKeyCandidatePrice
```

```
    pendingBullishKey := true
```

```
    pendingBearishKey := false
```

```
    bullishKeySweepIndex := bar_index
```

```

bullishKeyTriggerPrice := lastSwingHighPrice

bullishKeyName := bullishKeyCandidateName

if not na(bullishKeyLevelLine)

    line.delete(bullishKeyLevelLine)

if showKeyLevelSweepLinesInput

    bullishKeyLevelLine := line.new(bar_index, bullishKeyCandidatePrice, bar_index,
bullishKeyCandidatePrice, xloc = xloc.bar_index, extend = extend.right, color =
color.new(bullishColorInput, 25), style = line.style_dotted, width = 2)

if bearishKeySweepEvent

    lastBearishKeyName := bearishKeyCandidateName

    lastBearishKeyPrice := bearishKeyCandidatePrice

    pendingBearishKey := true

    pendingBullishKey := false

    bearishKeySweepIndex := bar_index

    bearishKeyTriggerPrice := lastSwingLowPrice

    bearishKeyName := bearishKeyCandidateName

    if not na(bearishKeyLevelLine)

        line.delete(bearishKeyLevelLine)

    if showKeyLevelSweepLinesInput

        bearishKeyLevelLine := line.new(bar_index, bearishKeyCandidatePrice, bar_index,
bearishKeyCandidatePrice, xloc = xloc.bar_index, extend = extend.right, color =
color.new(bearishColorInput, 25), style = line.style_dotted, width = 2)

// --- FVG-123 key-touch and EMA context ---

bool fvgPdhTouch = useDailyLevelsInput and not na(previousDayHigh) and high >=
previousDayHigh and low <= previousDayHigh

```

```

bool fvgPdlTouch = useDailyLevelsInput and not na(previousDayLow) and high >=
previousDayLow and low <= previousDayLow

bool fvgPwhTouch = useWeeklyLevelsInput and not na(previousWeekHigh) and high >=
previousWeekHigh and low <= previousWeekHigh

bool fvgPwlTouch = useWeeklyLevelsInput and not na(previousWeekLow) and high >=
previousWeekLow and low <= previousWeekLow

bool fvgAsiaHighTouch = useAsiaLevelsInput and not na(previousAsiaHigh) and high >=
previousAsiaHigh and low <= previousAsiaHigh

bool fvgAsiaLowTouch = useAsiaLevelsInput and not na(previousAsiaLow) and high >=
previousAsiaLow and low <= previousAsiaLow

bool fvgLondonHighTouch = useLondonLevelsInput and not na(previousLondonHigh) and
high >= previousLondonHigh and low <= previousLondonHigh

bool fvgLondonLowTouch = useLondonLevelsInput and not na(previousLondonLow) and
high >= previousLondonLow and low <= previousLondonLow

bool fvgKeyLevelTouch = fvgPdhTouch or fvgPdlTouch or fvgPwhTouch or fvgPwlTouch or
fvgAsiaHighTouch or fvgAsiaLowTouch or fvgLondonHighTouch or fvgLondonLowTouch

var int lastFvgKeyLevelTouchBar = na

if fvgKeyLevelTouch
    lastFvgKeyLevelTouchBar := bar_index

float fvgFastEma = ta.ema(close, fvgFastEmaLengthInput)
float fvgSlowEma = ta.ema(close, fvgSlowEmaLengthInput)
bool fvgBullEmaCross = ta.crossover(fvgFastEma, fvgSlowEma)
bool fvgBearEmaCross = ta.crossunder(fvgFastEma, fvgSlowEma)
var int lastFvgBullEmaCrossBar = na
var int lastFvgBearEmaCrossBar = na

if fvgBullEmaCross
    lastFvgBullEmaCrossBar := bar_index

```

```

if fvgBearEmaCross

    lastFvgBearEmaCrossBar := bar_index

bool fvgBullEmaContext = not na(lastFvgBullEmaCrossBar) and bar_index >=
lastFvgBullEmaCrossBar and fvgFastEma > fvgSlowEma

bool fvgBearEmaContext = not na(lastFvgBearEmaCrossBar) and bar_index >=
lastFvgBearEmaCrossBar and fvgFastEma < fvgSlowEma


// --- FVG-123 setup tracking ---

type Fvg123Setup

    float top

    float bottom

    bool isBull

    int createdBar

    int keyTouchBar

    bool touched

    int touchBar

    bool done


var array<Fvg123Setup> fvg123SetupsArray = array.new<Fvg123Setup>()

var int fvg123EmaCrossUsedBar = na

var array<line> fvgRiskRewardLinesArray = array.new<line>()

var array<label> fvgRiskRewardLabelsArray = array.new<label>()

bool fvgLongSignal = false

bool fvgShortSignal = false

float fvgSignalEntry = na

float fvgSignalStop = na

float fvgSignalTarget = na

```

```

if enableFvg123Input and bar_index >= 2

  if low > high[2]

    array.push(fvg123SetupsArray, Fvg123Setup.new(low, high[2], true, bar_index,
lastFvgKeyLevelTouchBar, false, na, false))

  if high < low[2]

    array.push(fvg123SetupsArray, Fvg123Setup.new(low[2], high, false, bar_index,
lastFvgKeyLevelTouchBar, false, na, false))

  if array.size(fvg123SetupsArray) > 100

    array.shift(fvg123SetupsArray)

  if array.size(fvg123SetupsArray) > 0

    for i = array.size(fvg123SetupsArray) - 1 to 0

      Fvg123Setup fvgObj = array.get(fvg123SetupsArray, i)

      if not fvgObj.done

        bool fullyFilled = fvgObj.isBull ? low <= fvgObj.bottom : high >= fvgObj.top

        if fullyFilled

          fvgObj.done := true

        else

          bool afterCreation = bar_index > fvgObj.createdBar

          if afterCreation and high >= fvgObj.bottom and low <= fvgObj.top and
(fvgObj.isBull ? close >= fvgObj.bottom : close <= fvgObj.top)

            fvgObj.touched := true

            fvgObj.touchBar := bar_index

            if afterCreation and (fvgObj.isBull ? close < fvgObj.bottom : close > fvgObj.top)

              fvgObj.done := true

            bool hasLaterTouch = fvgObj.touched and not na(fvgObj.touchBar) and bar_index
> fvgObj.touchBar

```

```
bool fvgBreakout = (fvgObj.isBull and hasLaterTouch and close > fvgObj.top) or  
(not fvgObj.isBull and hasLaterTouch and close < fvgObj.bottom)
```

```
bool fvgAgeValid = bar_index - fvgObj.createdBar <= fvg123MaxAgeInput
```

```
bool fvgWidthValid = math.abs(fvgObj.top - fvgObj.bottom) / syminfo.mintick >=  
fvg123MinWidthTicksInput
```

```
bool keyTouchBeforeFvg = not na(fvgObj.keyTouchBar) and fvgObj.keyTouchBar <  
fvgObj.createdBar and bar_index - fvgObj.keyTouchBar <= fvg123KeyTouchLookbackInput
```

```
bool fvgKeyTouchAllowed = not requireFvg123KeyTouchInput or  
keyTouchBeforeFvg
```

```
if fvgBreakout
```

```
bool fvgTrendFlowAllowed = fvgObj.isBull ? fvgFastEma > fvgSlowEma and  
close > fvgFastEma and close > fvgSlowEma : fvgFastEma < fvgSlowEma and close <  
fvgFastEma and close < fvgSlowEma
```

```
bool fvgCrossAfterKeyTouch = not requireFvg123KeyTouchInput or not  
na(fvgObj.keyTouchBar) and not na(fvgObj.isBull ? lastFvgBullEmaCrossBar :  
lastFvgBearEmaCrossBar) and fvgObj.keyTouchBar < (fvgObj.isBull ?  
lastFvgBullEmaCrossBar : lastFvgBearEmaCrossBar)
```

```
bool fvgEmaCrossAvailable = fvgObj.isBull ? fvgBullEmaContext and  
fvgCrossAfterKeyTouch and (na(fvg123EmaCrossUsedBar) or fvg123EmaCrossUsedBar !=  
lastFvgBullEmaCrossBar) : fvgBearEmaContext and fvgCrossAfterKeyTouch and  
(na(fvg123EmaCrossUsedBar) or fvg123EmaCrossUsedBar != lastFvgBearEmaCrossBar)
```

```
bool fvgEmaCrossAllowed = not requireFvg123EmaCrossInput or  
fvgEmaCrossAvailable
```

```
if fvgAgeValid and fvgWidthValid and fvgKeyTouchAllowed and  
fvgTrendFlowAllowed and fvgEmaCrossAllowed and barstate.isconfirmed and not  
fvgLongSignal and not fvgShortSignal
```

```
float entryPrice = close
```

```
float stopPrice = fvgObj.isBull ? fvgObj.bottom - syminfo.mintick : fvgObj.top +  
syminfo.mintick
```

```
float risk = fvgObj.isBull ? entryPrice - stopPrice : stopPrice - entryPrice
```

```
if risk > 0.0
```

```

        fvgSignalEntry := entryPrice

        fvgSignalStop := stopPrice

        fvgSignalTarget := fvgObj.isBull ? entryPrice + risk * fvgRewardRiskInput :
entryPrice - risk * fvgRewardRiskInput

        if requireFvg123EmaCrossInput

            fvg123EmaCrossUsedBar := fvgObj.isBull ? lastFvgBullEmaCrossBar :
lastFvgBearEmaCrossBar

            if fvgObj.isBull

                fvgLongSignal := true

            else

                fvgShortSignal := true

            fvgObj.done := true

            array.set(fvg123SetupsArray, i, fvgObj)

```

// --- Update liquidity levels after processing the current bar ---

```

if not na(pivotHigh)

    lastSwingHighPrice := pivotHigh

    lastSwingHighIndex := bar_index - pivotRightInput

```

```

if not na(pivotLow)

    lastSwingLowPrice := pivotLow

    lastSwingLowIndex := bar_index - pivotRightInput

```

// --- Market structure shift confirmation ---

```

bool bullishMssCross = ta.crossover(close, bullishTriggerPrice)

bool bearishMssCross = ta.crossunder(close, bearishTriggerPrice)

bool bullishKeyMssCross = ta.crossover(close, bullishKeyTriggerPrice)

```

bool bearishKeyMssCross = ta.crossunder(close, bearishKeyTriggerPrice)

bool bullishWithinWindow = not na(bullishSweepIndex) and bar_index - bullishSweepIndex
<= confirmationWindowInput

bool bearishWithinWindow = not na(bearishSweepIndex) and bar_index -
bearishSweepIndex <= confirmationWindowInput

bool bullishKeyWithinWindow = not na(bullishKeySweepIndex) and bar_index -
bullishKeySweepIndex <= confirmationWindowInput

bool bearishKeyWithinWindow = not na(bearishKeySweepIndex) and bar_index -
bearishKeySweepIndex <= confirmationWindowInput

bool bullishEntrySignal = requireMssInput ? pendingBullish and bullishWithinWindow and
bar_index > bullishSweepIndex and bullishMssCross : bullishSweepEvent

bool bearishEntrySignal = requireMssInput ? pendingBearish and bearishWithinWindow
and bar_index > bearishSweepIndex and bearishMssCross : bearishSweepEvent

bool bullishKeyEntrySignal = pendingBullishKey and bullishKeyWithinWindow and
bar_index > bullishKeySweepIndex and bullishKeyMssCross

bool bearishKeyEntrySignal = pendingBearishKey and bearishKeyWithinWindow and
bar_index > bearishKeySweepIndex and bearishKeyMssCross

bool bullishKeyEntryAlert = enableKeyLevelAlertsInput and bullishKeyEntrySignal

bool bearishKeyEntryAlert = enableKeyLevelAlertsInput and bearishKeyEntrySignal

bool anyKeyLevelEntryAlert = bullishKeyEntryAlert or bearishKeyEntryAlert

if bullishEntrySignal

 pendingBullish := false

if bearishEntrySignal

 pendingBearish := false

```
if bullishKeyEntrySignal
```

```
    pendingBullishKey := false
```

```
if bearishKeyEntrySignal
```

```
    pendingBearishKey := false
```

```
if pendingBullish and not bullishWithinWindow
```

```
    pendingBullish := false
```

```
if pendingBearish and not bearishWithinWindow
```

```
    pendingBearish := false
```

```
if pendingBullishKey and not bullishKeyWithinWindow
```

```
    pendingBullishKey := false
```

```
if pendingBearishKey and not bearishKeyWithinWindow
```

```
    pendingBearishKey := false
```

```
// --- FVG-123 risk/reward guides ---
```

```
if showFvgRiskRewardInput and (fvgLongSignal or fvgShortSignal)
```

```
    int objectLimit = maxDisplayedFvgSetupsInput * 3
```

```
    while array.size(fvgRiskRewardLinesArray) + 3 > objectLimit
```

```
        for i = 0 to 2
```

```
            line.delete(array.shift(fvgRiskRewardLinesArray))
```

```
        if array.size(fvgRiskRewardLabelsArray) >= 3
```

```
            for i = 0 to 2
```

```

        label.delete(array.shift(fvgRiskRewardLabelsArray))

    int guideEndBar = bar_index + fvgProjectionBarsInput

    line entryLine = line.new(bar_index, fvgSignalEntry, guideEndBar, fvgSignalEntry, xloc =
xloc.bar_index, color = color.new(color.gray, 10), style = line.style_dotted, width =
fvgRiskRewardLineWidthInput)

    line stopLine = line.new(bar_index, fvgSignalStop, guideEndBar, fvgSignalStop, xloc =
xloc.bar_index, color = color.new(bearishColorInput, 5), style = line.style_dotted, width =
fvgRiskRewardLineWidthInput)

    line targetLine = line.new(bar_index, fvgSignalTarget, guideEndBar, fvgSignalTarget, xloc =
xloc.bar_index, color = color.new(bullishColorInput, 5), style = line.style_dotted, width =
fvgRiskRewardLineWidthInput)

    array.push(fvgRiskRewardLinesArray, entryLine)

    array.push(fvgRiskRewardLinesArray, stopLine)

    array.push(fvgRiskRewardLinesArray, targetLine)

    if showFvgRiskRewardLabelsInput

        label entryLabel = label.new(guideEndBar, fvgSignalEntry, "Entry", xloc =
xloc.bar_index, yloc = yloc.price, color = color.new(color.gray, 15), style =
label.style_label_left, textcolor = chart.fg_color, size = size.tiny)

        label stopLabel = label.new(guideEndBar, fvgSignalStop, "SL", xloc = xloc.bar_index,
yloc = yloc.price, color = color.new(bearishColorInput, 15), style = label.style_label_left,
textcolor = chart.fg_color, size = size.tiny)

        label targetLabel = label.new(guideEndBar, fvgSignalTarget, "TP " +
str.toString(fvgRewardRiskInput, "#.##") + "R", xloc = xloc.bar_index, yloc = yloc.price, color
= color.new(bullishColorInput, 15), style = label.style_label_left, textcolor = chart.fg_color,
size = size.tiny)

        array.push(fvgRiskRewardLabelsArray, entryLabel)

        array.push(fvgRiskRewardLabelsArray, stopLabel)

        array.push(fvgRiskRewardLabelsArray, targetLabel)

// --- Visual elements ---

```

```
bool showPowerReversalLabel = showPowerReversalInput and (not  
hidePowerReversalOnOneMinuteInput or timeframe.period != "1")
```

```
plotshape(showPivotPointsInput and not na(pivotHigh), title = "Confirmed swing high", style  
= shape.circle, location = location.abovebar, offset = -pivotRightInput, color =  
bearishColorInput, size = size.tiny)
```

```
plotshape(showPivotPointsInput and not na(pivotLow), title = "Confirmed swing low", style  
= shape.circle, location = location.belowbar, offset = -pivotRightInput, color =  
bullishColorInput, size = size.tiny)
```

```
plotshape(showSweepMarkersInput and bullishSweepEvent, title = "Bullish liquidity  
sweep", style = shape.circle, location = location.belowbar, color = bullishColorInput, text =  
"Sweep", textcolor = chart.fg_color, size = size.tiny)
```

```
plotshape(showSweepMarkersInput and bearishSweepEvent, title = "Bearish liquidity  
sweep", style = shape.circle, location = location.abovebar, color = bearishColorInput, text =  
"Sweep", textcolor = chart.fg_color, size = size.tiny)
```

```
plotshape(bullishEntrySignal, title = "Bullish sweep entry confirmation", style =  
shape.triangleup, location = location.belowbar, color = bullishColorInput, text = "LONG",  
textcolor = chart.fg_color, size = size.small)
```

```
plotshape(bearishEntrySignal, title = "Bearish sweep entry confirmation", style =  
shape.triangledown, location = location.abovebar, color = bearishColorInput, text =  
"SHORT", textcolor = chart.fg_color, size = size.small)
```

```
plotshape(showFvg123SignalsInput and fvgLongSignal, title = "Bullish FVG-123 2R signal",  
style = shape.triangleup, location = location.belowbar, color = bullishColorInput, text =  
"FVG-123", textcolor = chart.fg_color, size = size.small)
```

```
plotshape(showFvg123SignalsInput and fvgShortSignal, title = "Bearish FVG-123 2R signal",  
style = shape.triangledown, location = location.abovebar, color = bearishColorInput, text =  
"FVG-123", textcolor = chart.fg_color, size = size.small)
```

```
// label.style_label_right anchors the label at the candle and places its text to the left.
```

```
// Both labels are positioned above the candle with a directional background and key-level  
name.
```

```
float powerLabelOffset = not na(atrValue) ? math.max(candleRange * 0.25, atrValue * 0.15)  
: math.max(candleRange * 0.25, syminfo.mintick * 10.0)
```

```
float powerLabelPrice = high + powerLabelOffset
```

```
if showPowerReversalLabel and bullishPowerReversal
```

```
    label.new(bar_index, powerLabelPrice, "PWR " + bullishKeyCandidateName, xloc =  
xloc.bar_index, yloc = yloc.price, color = color.new(bullishColorInput, 15), style =  
label.style_label_right, textcolor = chart.fg_color, size = size.small)
```

```
if showPowerReversalLabel and bearishPowerReversal
```

```
    label.new(bar_index, powerLabelPrice, "PWR " + bearishKeyCandidateName, xloc =  
xloc.bar_index, yloc = yloc.price, color = color.new(bearishColorInput, 15), style =  
label.style_label_right, textcolor = chart.fg_color, size = size.small)
```

```
// --- Dynamic key-level alerts ---
```

```
if bullishKeyEntryAlert
```

```
    alert("Bullish key-level sweep + MSS confirmed at " + bullishKeyName + " on " +  
syminfo.ticker + " " + timeframe.period + ".", alert.freq_once_per_bar_close)
```

```
if bearishKeyEntryAlert
```

```
    alert("Bearish key-level sweep + MSS confirmed at " + bearishKeyName + " on " +  
syminfo.ticker + " " + timeframe.period + ".", alert.freq_once_per_bar_close)
```

```
if bullishPowerReversalAlert
```

```
    alert("Bullish HTF PWR at " + bullishKeyCandidateName + " on " + syminfo.ticker + " " +  
timeframe.period + ".", alert.freq_once_per_bar_close)
```

```
if bearishPowerReversalAlert
```

```
    alert("Bearish HTF PWR at " + bearishKeyCandidateName + " on " + syminfo.ticker + " " +  
timeframe.period + ".", alert.freq_once_per_bar_close)
```

// --- Alerts ---

alertcondition(bullishSweepEvent, title = "Bullish liquidity sweep", message = "Bullish liquidity sweep detected on {{ticker}} {{interval}}.")

alertcondition(bearishSweepEvent, title = "Bearish liquidity sweep", message = "Bearish liquidity sweep detected on {{ticker}} {{interval}}.")

alertcondition(bullishEntrySignal, title = "Bullish sweep entry confirmation", message = "Bullish liquidity sweep confirmed by market structure shift on {{ticker}} {{interval}}.")

alertcondition(bearishEntrySignal, title = "Bearish sweep entry confirmation", message = "Bearish liquidity sweep confirmed by market structure shift on {{ticker}} {{interval}}.")

alertcondition(bullishKeyEntryAlert, title = "Bullish key-level sweep entry confirmation", message = "Bullish key-level sweep and close above the most recent confirmed pre-sweep swing high on {{ticker}} {{interval}}.")

alertcondition(bearishKeyEntryAlert, title = "Bearish key-level sweep entry confirmation", message = "Bearish key-level sweep and close below the most recent confirmed pre-sweep swing low on {{ticker}} {{interval}}.")

alertcondition(anyKeyLevelEntryAlert, title = "Any key-level sweep entry confirmation", message = "Bullish or bearish key-level sweep and pre-sweep structure confirmation detected on {{ticker}} {{interval}}.")

alertcondition(bullishPowerReversalAlert, title = "Bullish Sweep PWR Candle", message = "Bullish Sweep PWR Candle detected on {{ticker}} {{interval}}.")

alertcondition(bearishPowerReversalAlert, title = "Bearish Sweep PWR Candle", message = "Bearish Sweep PWR Candle detected on {{ticker}} {{interval}}.")

alertcondition(anyPowerReversalAlert, title = "Any Sweep PWR Candle", message = "Bullish or bearish Sweep PWR Candle detected on {{ticker}} {{interval}}.")

alertcondition(fvgLongSignal, title = "Bullish FVG-123", message = "Bullish FVG-123 signal confirmed with configured risk/reward guides on {{ticker}} {{interval}}.")

alertcondition(fvgShortSignal, title = "Bearish FVG-123", message = "Bearish FVG-123 signal confirmed with configured risk/reward guides on {{ticker}} {{interval}}.")