```
// This Pine Script™ code is subject to the terms of the Mozilla Public License 2.0 at
https://mozilla.org/MPL/2.0/
// © fluxchart

//@version=5
const bool DEBUG = false
const bool DEBUGOBFVG = false
const int maxBoxesCount = 500
const int maxDistanceToLastBar = 100000 // Affects Running Time
const int showLastXFVGs = 2
const int maxOrderBlocks = 5
const int maxBarsBack = 50
const int extendLastXFVGsCount = 20
const int minimumFVGSize = 2
const int minimumIFVGSize = 2
const float overlapThresholdPercentage = 0
const int atrLen = 10

indicator("Candle Range Theory | Flux Charts", shorttitle = "CRT | Flux Charts", overlay = true,
max_boxes_count = maxBoxesCount, max_labels_count = maxBoxesCount, max_lines_count =
maxBoxesCount, max_bars_back = 100)
var int curTFMMS = timeframe.in_seconds()
//#region Settings
higherTF = input.timeframe("240", "Higher Timeframe", group = "General Configuration", tooltip = "M15 -> H4
is recommended.\n\nAll Recommended Timeframes :\nWeekly -> Monthly\nDaily -> Weekly\nH4 ->
Daily\nM15 -> H4\nM3 -> M30")
bulkyCandleATRStr = input.string("Big", "HTF Candle Size", options = ["Big", "Normal", "Small"], group =
"General Configuration", tooltip = "The size of the higher timeframe candle. A candle will be considered
eligible for CRT if it's range obliges to this setting.")
bulkyCandleATR = bulkyCandleATRStr == "Big" ? 2.1 : bulkyCandleATRStr == "Normal" ? 1.6 : 1.3
entryMode = input.string("FVGs", "Entry Mode", ["FVGs", "Order Blocks"], group = "General Configuration")
requireRetracement = input.bool(false, "Require Retracement", tooltip = "A retracement to the FVG or Order
Block will be required for entry confirmation if enabled.", group = "General Configuration")
showHTFLines = input.bool(true, "Show HTF Candle Lines", group = "General Configuration")

fvgSensitivityText = input.string("High", "FVG Detection Sensitivity", options = ["All", "Extreme", "High",
"Normal", "Low"], group = "Fair Value Gaps")
showFVG = input.bool(true, "Show FVGs", inline = "bb", group = "Fair Value Gaps")

OBsEnabled = true
orderBlockVolumetricInfo = false
obEndMethod = "Close"
```

```
swingLength = input.int(10, 'Swing Length', minval = 3, maxval = 45, tooltip="Swing length is used when finding order block formations. Smaller values will result in finding smaller order blocks.",group = "Order Blocks", display = display.none)
bullOrderBlockColor = #08998180
bearOrderBlockColor = #f2364680
showOB = input.bool(true, "Show Order Blocks", group = "Order Blocks")

BBsEnabled = false
breakBlockVolumetricInfo = false
bbEndMethod = "Wick"
breakersFull = true

dbgLabelSize = DEBUG ? input.string("Normal", "[DBG] Label Size", ["Normal", "Small", "Tiny"], group = "General Configuration") : "Normal"
lblSize = (dbgLabelSize == "Small" ? size.small : dbgLabelSize == "Normal" ? size.normal : size.tiny)
dbgShowBBFVG = DEBUG ? input.bool(false, "[DBG] Show All BB and FVGs", group = "General Configuration") : false

showTPSL = input.bool(true, "Enabled", group = "TP / SL")
tpslMethod = input.string("Dynamic", "TP / SL Method", options = ["Dynamic", "Fixed"], group = "TP / SL")
riskAmount = input.string("High", "Dynamic Risk", options = ["Highest", "High", "Normal", "Low", "Lowest"], group = "TP / SL", tooltip = "The risk amount when Dynamic TP / SL method is selected.\n\nDifferent assets may have different volatility so changing this setting may result in change of performance of the indicator.")
customSLATRMult = DEBUG ? input.float(6.5, "[DBG] Dynamic Custom Risk Mult", group = "TP / SL") : 6.5

slATRMult = riskAmount == "Highest" ? 10 : riskAmount == "High" ? 8 : riskAmount == "Normal" ? 6.5 : riskAmount == "Low" ? 5 : riskAmount == "Lowest" ? 3 : customSLATRMult
tpPercent = input.float(0.3, "Fixed Take Profit %", step = 0.1, group = "TP / SL")
slPercent = input.float(0.4, "Fixed Stop Loss %", step = 0.1, group = "TP / SL")

backtestDisplayEnabled = input.bool(true, "Enabled", group = "Backtesting Dashboard", display = display.none)
backtestingLocation = input.string("Top Center", "Position", options = ["Top Right", "Right Center", "Top Center"], group = "Backtesting Dashboard", display = display.none)
fillBackgrounds = input.bool(true, "Fill Backgrounds", group = "Backtesting Dashboard", display = display.none)
screenerColor = input.color(#1B1F2B, 'Background', inline = "1", group = 'Backtesting Dashboard', display = display.none)

DynamicRR = DEBUG ? input.float(0.39, "[DBG] Dynamic Risk:Reward Ratio", group = "Debug") : 0.39

buyAlertEnabled = input.bool(true, "Buy Signal", inline = "BS", group = "Alerts")
sellAlertEnabled = input.bool(true, "Sell Signal", inline = "BS", group = "Alerts")
tpAlertEnabled = input.bool(true, "Take-Profit Signal", inline = "TS", group = "Alerts")
```

```
slAlertEnabled = input.bool(true, "Stop-Loss Signal ", inline = "TS", group = "Alerts")

dbgTPSLVersion = input.string("Default", "TP / SL Layout", options = ["Default", "Alternative"], group =
"Visuals")

fvgBullColor = input(#08998180, 'Bullish FVG', inline = 'fvgColor', group = 'Visuals', display = display.none)
fvgBearColor = input(#f2364680, 'Bearish FVG', inline = 'fvgColor', group = 'Visuals', display = display.none)

highColor = input.color(#089981, "Buy", inline = "colors", group = "Visuals")
lowColor = input.color(#f23646, "Sell", inline = "colors", group = "Visuals")
textColor = input.color(#ffffff, "Text", inline = "colors", group = "Visuals")

showInvalidated = DEBUGOBFVG ? input.bool(true, "Show Historic Zones", group = "General Configuration",
display = display.none) : true

//#region FVG Settings
fvgEnabled = DEBUGOBFVG ? input.bool(true, "Enabled", group = "Fair Value Gaps", inline="EV", display =
display.none) : true
fvgVolumetricInfo = DEBUGOBFVG ? input.bool(false, "Volumetric Info", group = "Fair Value Gaps",
inline="EV", display = display.none) : false
fvgEndMethod = DEBUGOBFVG ? input.string("Close", "Zone Invalidation", options = ["Wick", "Close"],  group
= "Fair Value Gaps") : "Close"
fvgFilterMethod = DEBUGOBFVG ? input.string("Average Range", "Zone Filtering", options = ["Average
Range", "Volume Threshold"],  group = "Fair Value Gaps") : "Average Range"
volumeThresholdPercent = DEBUGOBFVG ? input.int(50, "Volume Threshold %", group = "Fair Value Gaps",
tooltip = "Only taken into calculation when filter method is Volume Threshold.", minval = 1, maxval = 200) : 50
fvgBars = DEBUGOBFVG ? input.string("Same Type", "FVG Detection", options = ["Same Type", "All"], tooltip
= "Same Type -> All 3 bars that formed the FVG should be the same type. (Bullish / Bearish) \n\nAll -> Bar
types may vary between bullish / bearish.", group = "Fair Value Gaps") : "Same Type"
fvgSensEnabled = DEBUGOBFVG ? input.bool(true, "", "", "sens", "Fair Value Gaps") : true
//fvgSensitivityText = DEBUGOBFVG ? input.string("Extreme", "Detection Sensitivity", inline = "sens", options
= ["Extreme", "High", "Normal", "Low"], group = "Fair Value Gaps") : "Extreme"
combineFVGs = DEBUGOBFVG ? input.bool(false, "Combine Zones", group = "Fair Value Gaps") : false
allowGaps = DEBUGOBFVG ? input.bool(false, "Allow Gaps Between Bars", group = "Fair Value Gaps",
tooltip = "On tickers that can have a different opening price than the previous bar's closing price, the indicator
will not analyze those bars for FVGs if disabled.\n\nFor Example, if the today's opening price is different from
the yesterday's closing price in a stock ticker.") : false
deleteUntouched = DEBUGOBFVG ? input.bool(true, "", group = "Fair Value Gaps", inline =
"deleteUntouched") : true
deleteUntouchedAfterXBars = DEBUGOBFVG ? input.int(200, "Delete Untouched Zones After", group = "Fair
Value Gaps", minval = 5, maxval = 200, inline = "deleteUntouched") : 200

ifvgEnabled = DEBUGOBFVG ? input.bool(false, "Enabled", inline="EV", group = "Inversion Fair Value Gaps")
: false
```

```
ifvgVolumetricInfo = DEBUGOBFVG ? input.bool(false, "Volumetric Info", group = "Inversion Fair Value Gaps",
inline="EV", display = display.none) : false
ifvgEndMethod = DEBUGOBFVG ? input.string("Wick", "IFVG Zone Invalidation", options = ["Wick", "Close"],
group = "Inversion Fair Value Gaps") : "Wick"
ifvgFull = DEBUGOBFVG ? input.bool(true, "IFVG Full", group = "Inversion Fair Value Gaps", display =
display.none) : true
bullishInverseColor = DEBUGOBFVG ? input(#08998180, "Bullish", inline = 'breakerColor', group = 'Inversion
Fair Value Gaps', display = display.none) : #08998180
bearishInverseColor = DEBUGOBFVG ? input(#f2364580, "Bearish", inline = 'breakerColor', group =
'Inversion Fair Value Gaps', display = display.none) : #f2364580
//#endregion

//#region DEBUG Settings
maxATRMult = (DEBUG and DEBUGOBFVG) ? input.float(3.5,"Max ATR Multiplier", group = "General
Configuration", display = display.none) : 3.5
extendZonesBy = (DEBUG and DEBUGOBFVG) ? input.int(15, "Extend Zones", group = "Style", minval = 1,
maxval = 30, inline = "ExtendZones") : 15
extendZonesDynamic = (DEBUG and DEBUGOBFVG) ? input.bool(true, "Dynamic", group = "Style", inline =
"ExtendZones") : true
extendLastFVGs = (DEBUG and DEBUGOBFVG) ? input.bool(true, "Extend Last Zones", group = "Style") :
true
changeCombinedFVGsColor = (DEBUG and DEBUGOBFVG) ? input.bool(false, "Change Combined Zones
Color", group = "Style", inline = "CombinedColor") : false
combinedText = (DEBUG and DEBUGOBFVG) ? input.bool(false, "Combined Text", group = "Style", inline =
"CombinedColor") : false
combinedColor = (DEBUG and DEBUGOBFVG) ? input.color(#fff70080, DEBUG ? "" : "Combined Zone
Color", group = "Style", inline = "CombinedColor") : #fff70080
startZoneFrom = (DEBUG and DEBUGOBFVG) ? input.string("Last Bar", "Start FVG Zones From", options =
["First Bar", "Last Bar"], group = "Style"): "Last Bar"
volumeBarsPlace = (DEBUG and DEBUGOBFVG) ? input.string("Left", "Show Volume Bars At", options =
["Left", "Right"], group = "Style", inline = "volumebars") : "Left"
mirrorVolumeBars = (DEBUG and DEBUGOBFVG) ? input.bool(true, "Mirror Volume Bars", group = "Style",
inline = "volumebars") : true
//#endregion DEBUG Settings
//#endregion Settings

var tfCheck = false
if not tfCheck
    if timeframe.in_seconds() > timeframe.in_seconds(higherTF)
        runtime.error("Higher timeframe must be higher than current timeframe.")
    tfCheck := true

//#region UDTs
```

```
type orderBlockInfo
    float top
    float bottom
    float obVolume
    string obType
    int startTime
    float bbVolume
    float obLowVolume
    float obHighVolume
    bool breaker = false
    int breakTime
    int breakerEndTime
    string timeframeStr
    bool disabled = false
    string combinedTimeframesStr = na
    bool combined = false

type orderBlock
    orderBlockInfo info
    bool isRendered = false

    box orderBox = na
    box breakerBox = na

    line orderBoxLineTop = na
    line orderBoxLineBottom = na
    line breakerBoxLineTop = na
    line breakerBoxLineBottom = na
    //
    box orderBoxText = na
    box orderBoxPositive = na
    box orderBoxNegative = na

    line orderSeperator = na
    line orderTextSeperator = na

type FVGInfo
    float max = na
    float min = na
    bool isBull = na
    int t = na
    float totalVolume = na
    int startBarIndex = na
    int endBarIndex = na
```

```
    int startTime = na
    int endTime = na
    bool extendInfinite = false
    bool combined = false
    string combinedTimeframesStr = na
    bool disabled = false
    string timeframeStr = na

    float lowVolume = na
    float highVolume = na
    bool isInverse = false
    int lastTouched = na
    int lastTouchedIFVG = na
    int inverseEndIndex = na
    int inverseEndTime = na
    float inverseVolume

createFVGInfo (h,l,bull,t,tv) =>
    FVGInfo newFVGInfo = FVGInfo.new(h, l, bull, t, tv)
    newFVGInfo

type FVG
    FVGInfo info = na

    bool isRendered = false

    box fvgBox = na
    box ifvgBox = na
    box fvgBoxText = na
    box fvgBoxPositive = na
    box fvgBoxNegative = na

    line fvgSeperator = na
    line fvgTextSeperator = na

createFVG (FVGInfo FVGInfoF) =>
    FVG newFVG = FVG.new(FVGInfoF)
    newFVG

safeDeleteFVG (FVG fvg) =>
    fvg.isRendered := false

    box.delete(fvg.fvgBox)
    box.delete(fvg.ifvgBox)
```

```
    box.delete(fvg.fvgBoxText)
    box.delete(fvg.fvgBoxPositive)
    box.delete(fvg.fvgBoxNegative)

    line.delete(fvg.fvgSeperator)
    line.delete(fvg.fvgTextSeperator)
//#region Definitions

volumeBarsLeftSide = (volumeBarsPlace == "Left")
fvgSensitivity = fvgSensitivityText == "All" ? 100 : fvgSensitivityText == "Extreme" ? 6 : fvgSensitivityText ==
"High" ? 2 : fvgSensitivityText == "Normal" ? 1.5 : 1
extendZonesByTime = extendZonesBy * timeframe.in_seconds(timeframe.period) * 1000

atr = ta.atr(atrLen)
volCheck = (ta.cum(volume) > 0)

const int maxArrCount = 200
var array<float> lo = array.new<float>()
var array<float> hi = array.new<float>()
var array<int> ti = array.new<int>()
lo.unshift(low)
hi.unshift(high)
ti.unshift(time)
if lo.size() > maxArrCount
    lo.pop()
    hi.pop()
    ti.pop()

findValRtnTime (valToFind, toSearch, searchMode) =>
    int rtnTime = na
    float minDiff = na
    if toSearch == "Low" and not na(lo)
        if (lo).size() > 0
            for i = (lo).size() - 1 to 0
                curLow = (lo).get(i)
                if searchMode == "Nearest"
                    curDiff = math.abs(valToFind - curLow)
                    if na(minDiff)
                        rtnTime := (ti).get(i)
                        minDiff := curDiff
                    else
                        if curDiff <= minDiff
                            minDiff := curDiff
                            rtnTime := (ti).get(i)
```

```
            if searchMode == "Higher"
                if curLow >= valToFind
                    rtnTime := (ti).get(i)
            if searchMode == "Lower"
                if curLow <= valToFind
                    rtnTime := (ti).get(i)
    else if not na(hi)
        if (hi).size() > 0
            for i = (hi).size() - 1 to 0
                curHigh = (hi).get(i)
                if searchMode == "Nearest"
                    curDiff = math.abs(valToFind - curHigh)
                    if na(minDiff)
                        rtnTime := (ti).get(i)
                        minDiff := curDiff
                    else
                        if curDiff <= minDiff
                            minDiff := curDiff
                            rtnTime := (ti).get(i)
                if searchMode == "Higher"
                    if curHigh >= valToFind
                        rtnTime := (ti).get(i)
                if searchMode == "Lower"
                    if curHigh <= valToFind
                        rtnTime := (ti).get(i)
    rtnTime

moveLine(_line, _x, _y, _x2) =>
    line.set_xy1(_line, _x,  _y)
    line.set_xy2(_line, _x2, _y)

moveBox (_box, _topLeftX, _topLeftY, _bottomRightX, _bottomRightY) =>
    box.set_lefttop(_box, _topLeftX, _topLeftY)
    box.set_rightbottom(_box, _bottomRightX, _bottomRightY)

colorWithTransparency (colorF, transparencyX) =>
    color.new(colorF, color.t(colorF) * transparencyX)

createFVGBox (boxColor, transparencyX = 1.0, xlocType = xloc.bar_time) =>
    box.new(na, na, na, na, text_size = size.normal, xloc = xlocType, extend = extend.none, bgcolor =
colorWithTransparency(boxColor, transparencyX), text_color = textColor, text_halign = text.align_center,
border_color = #00000000)

var FVGInfo[] FVGInfoList = array.new<FVGInfo>(0)
```

```
var FVG[] allFVGList = array.new<FVG>(0)
//#endregion

//#region FVGs
arrHasFVG (FVG[] arr, FVG fvgF) =>
    hasFVG = false
    if arr.size() > 0
        for i = 0 to arr.size() - 1
            FVG fvg1 = arr.get(i)
            if fvg1.info.startTime == fvgF.info.startTime
                hasFVG := true
                break
    hasFVG

arrHasIFVG (FVG[] arr, FVG fvgF) =>
    hasIFVG = false
    if arr.size() > 0
        for i = 0 to arr.size() - 1
            FVG fvg1 = arr.get(i)
            if fvg1.info.isInverse
                if fvg1.info.startTime == fvgF.info.startTime
                    hasIFVG := true
                    break
    hasIFVG

renderFVG (FVG fvg, int customEndTime) =>
    fvg.isRendered := true
    if fvgEnabled and ((not ifvgEnabled) or (not fvg.info.isInverse)) and (showInvalidated or
na(fvg.info.endTime))
        fvg.fvgBox := createFVGBox((fvg.info.combined and changeCombinedFVGsColor) ? combinedColor :
fvg.info.isBull ? fvgBullColor : fvgBearColor, 1.5)
        fvg.fvgBoxText := createFVGBox(color.new(color.white, 100))
        if fvgVolumetricInfo
            fvg.fvgBoxPositive := createFVGBox(fvgBullColor)
            fvg.fvgBoxNegative := createFVGBox(fvgBearColor)
            fvg.fvgSeperator := line.new(na,na,na,na,xloc.bar_time,extend.none,textColor,line.style_dashed,1)
            fvg.fvgTextSeperator := line.new(na,na,na,na,xloc.bar_time,extend.none,textColor,line.style_solid,1)

        zoneSize = extendZonesDynamic ? na(fvg.info.endTime) ? extendZonesByTime : (fvg.info.endTime -
fvg.info.startTime) : extendZonesByTime
        if na(fvg.info.endTime) and fvg.info.extendInfinite
            zoneSize := (time + 1) - fvg.info.startTime

        if not na(customEndTime)
```

```
            zoneSize := customEndTime - fvg.info.startTime

        startX = volumeBarsLeftSide ? fvg.info.startTime : fvg.info.startTime + zoneSize - zoneSize / 3
        maxEndX = volumeBarsLeftSide ? fvg.info.startTime + zoneSize / 3 : fvg.info.startTime + zoneSize

        moveBox(fvg.fvgBox, fvg.info.startTime, fvg.info.max, fvg.info.startTime + zoneSize, fvg.info.min)
        moveBox(fvg.fvgBoxText, volumeBarsLeftSide ? maxEndX : fvg.info.startTime, fvg.info.max,
volumeBarsLeftSide ? fvg.info.startTime + zoneSize : startX, fvg.info.min)

        percentage = int((math.min(fvg.info.highVolume, fvg.info.lowVolume) / math.max(fvg.info.highVolume,
fvg.info.lowVolume)) * 100.0)
        //FVGText = (na(fvg.info.combinedTimeframesStr) ? formatTimeframeString(fvg.info.timeframeStr) :
fvg.info.combinedTimeframesStr) + " FVG"
        FVGText = ""
        box.set_text(fvg.fvgBoxText, (fvgVolumetricInfo ? str.tostring(fvg.info.totalVolume, format.volume) + " (" +
str.tostring(percentage) + "%)\n" : "") + (combinedText and fvg.info.combined ? "[Combined]\n" : "") +
FVGText)

        if fvg.info.combined and not changeCombinedFVGsColor
            fvg.fvgBox.set_bgcolor(colorWithTransparency(fvg.info.isBull ? fvgBullColor : fvgBearColor, 1.1))

        if fvgVolumetricInfo
            showHighLowBoxText = false

            curEndXHigh = int(math.ceil((fvg.info.highVolume / fvg.info.totalVolume) * (maxEndX - startX) +
startX))
            curEndXLow = int(math.ceil((fvg.info.lowVolume / fvg.info.totalVolume) * (maxEndX - startX) + startX))

            moveBox(fvg.fvgBoxPositive, mirrorVolumeBars ? startX : curEndXLow, fvg.info.max,
mirrorVolumeBars ? curEndXHigh : maxEndX, (fvg.info.min + fvg.info.max) / 2)
            box.set_text(fvg.fvgBoxPositive, showHighLowBoxText ? str.tostring(fvg.info.highVolume,
format.volume) : "")

            moveBox(fvg.fvgBoxNegative, mirrorVolumeBars ? startX : curEndXHigh, fvg.info.min,
mirrorVolumeBars ? curEndXLow : maxEndX, (fvg.info.min + fvg.info.max) / 2)
            box.set_text(fvg.fvgBoxNegative, showHighLowBoxText ? str.tostring(fvg.info.lowVolume,
format.volume) : "")

            moveLine(fvg.fvgSeperator, volumeBarsLeftSide ? startX : maxEndX, (fvg.info.min + fvg.info.max) / 2,
volumeBarsLeftSide ? maxEndX : startX)

            line.set_xy1(fvg.fvgTextSeperator, volumeBarsLeftSide ? maxEndX : startX, fvg.info.max)
            line.set_xy2(fvg.fvgTextSeperator, volumeBarsLeftSide ? maxEndX : startX, fvg.info.min)
```

```
    // IFVG
    if fvg.info.isInverse and ifvgEnabled
        inverseColor = fvg.info.isBull ? bearishInverseColor : bullishInverseColor
        fvg.ifvgBox := createFVGBox(inverseColor)
        startTimeIFVG = ifvgFull ? fvg.info.startTime : fvg.info.endTime
        inverseZoneSize = na(fvg.info.inverseEndTime) ? ((time + 1) - startTimeIFVG) : (fvg.info.inverseEndTime
- startTimeIFVG)
        moveBox(fvg.ifvgBox, startTimeIFVG, fvg.info.max, startTimeIFVG + inverseZoneSize, fvg.info.min)
        IFVGText = ""
        box.set_text(fvg.ifvgBox, (ifvgVolumetricInfo ? str.tostring(fvg.info.inverseVolume, format.volume) + "\n" :
"") + (combinedText and fvg.info.combined ? "[Combined]\n" : "") + IFVGText)


areaOfFVG (FVGInfo FVGInfoF) =>
    float XA1 = FVGInfoF.startTime
    float XA2 = na(FVGInfoF.endTime) ? time + 1 : FVGInfoF.endTime
    float YA1 = FVGInfoF.max
    float YA2 = FVGInfoF.min
    float edge1 = math.sqrt((XA2 - XA1) * (XA2 - XA1) + (YA2 - YA2) * (YA2 - YA2))
    float edge2 = math.sqrt((XA2 - XA2) * (XA2 - XA2) + (YA2 - YA1) * (YA2 - YA1))
    float totalArea = edge1 * edge2
    totalArea

doFVGsTouch (FVGInfo FVGInfo1, FVGInfo FVGInfo2) =>
    float XA1 = FVGInfo1.startTime
    float XA2 = na(FVGInfo1.endTime) ? time + 1 : FVGInfo1.endTime
    float YA1 = FVGInfo1.max
    float YA2 = FVGInfo1.min

    float XB1 = FVGInfo2.startTime
    float XB2 = na(FVGInfo2.endTime) ? time + 1 : FVGInfo2.endTime
    float YB1 = FVGInfo2.max
    float YB2 = FVGInfo2.min
    float intersectionArea = math.max(0, math.min(XA2, XB2) - math.max(XA1, XB1)) * math.max(0,
math.min(YA1, YB1) - math.max(YA2, YB2))
    float unionArea = areaOfFVG(FVGInfo1) + areaOfFVG(FVGInfo2) - intersectionArea

    float overlapPercentage = (intersectionArea / unionArea) * 100.0

    //if FVGInfo1.isBull and str.contains(str.tostring(FVGInfo1.totalVolume, format.volume),"137M") and
str.contains(str.tostring(FVGInfo2.totalVolume, format.volume),"221M")
    //    log.info(str.tostring(XA2) + " | " + str.tostring(XB2))

    if overlapPercentage > overlapThresholdPercentage
```

```
        true
    else
        false

isFVGValid (FVGInfo FVGInfoF) =>
    valid = true
    if (not showInvalidated) and (not na(FVGInfoF.endTime))
        valid := false
    else if FVGInfoF.disabled
        valid := false
    valid

isIFVGValid (FVGInfo FVGInfoF) =>
    valid = true
    if not ifvgEnabled
        valid := false
    else if (not showInvalidated) and (not na(FVGInfoF.inverseEndTime))
        valid := false
    else if FVGInfoF.disabled
        valid := false
    valid

isFVGValidInTimeframe (FVGInfo FVGInfoF) =>
    valid = true
    if (not showInvalidated) and (not na(FVGInfoF.endTime))
        valid := false
    else if FVGInfoF.disabled
        valid := false
    else if not na(FVGInfoF.endBarIndex) and (FVGInfoF.endBarIndex - FVGInfoF.startBarIndex) <
minimumFVGSize
        valid := false
    else if na(FVGInfoF.endBarIndex) and deleteUntouched and (bar_index - FVGInfoF.lastTouched) >
deleteUntouchedAfterXBars
        valid := false
    valid

isIFVGValidInTimeframe (FVGInfo FVGInfoF) =>
    valid = true
    if not ifvgEnabled
        valid := false
    else if (not showInvalidated) and (not na(FVGInfoF.inverseEndIndex))
        valid := false
    else if not na(FVGInfoF.inverseEndIndex) and (FVGInfoF.inverseEndIndex - FVGInfoF.endBarIndex) <
minimumIFVGSize
```

```
        valid := false
    else if na(FVGInfoF.inverseEndIndex) and deleteUntouched and (bar_index - FVGInfoF.lastTouchedIFVG) >
deleteUntouchedAfterXBars
        valid := false
    valid

combineFVGsFunc () =>
    if allFVGList.size() > 0
        lastCombinations = 999
        while lastCombinations > 0
            lastCombinations := 0
            for i = 0 to allFVGList.size() - 1
                curFVG1 = allFVGList.get(i)
                for j = 0 to allFVGList.size() - 1
                    curFVG2 = allFVGList.get(j)
                    if i == j
                        continue
                    if not isFVGValid(curFVG1.info) or not isFVGValid(curFVG2.info)
                        continue
                    if curFVG1.info.isBull != curFVG2.info.isBull
                        continue
                    if doFVGsTouch(curFVG1.info, curFVG2.info)
                        curFVG1.info.disabled := true
                        curFVG2.info.disabled := true
                        FVG newFVG = createFVG(createFVGInfo(math.max(curFVG1.info.max, curFVG2.info.max),
math.min(curFVG1.info.min, curFVG2.info.min), curFVG1.info.isBull, math.min(curFVG1.info.t,
curFVG2.info.t), 0))
                        newFVG.info.startTime := math.min(curFVG1.info.startTime, curFVG2.info.startTime)
                        newFVG.info.startBarIndex := math.min(curFVG1.info.startBarIndex,
curFVG2.info.startBarIndex)
                        newFVG.info.endTime := math.max(nz(curFVG1.info.endTime), nz(curFVG2.info.endTime))
                        newFVG.info.endTime := newFVG.info.endTime == 0 ? na : newFVG.info.endTime
                        newFVG.info.endBarIndex := math.max(nz(curFVG1.info.endBarIndex),
nz(curFVG2.info.endBarIndex))
                        newFVG.info.endBarIndex := newFVG.info.endBarIndex == 0 ? na : newFVG.info.endBarIndex
                        newFVG.info.timeframeStr := curFVG1.info.timeframeStr
                        newFVG.info.extendInfinite := curFVG1.info.extendInfinite or curFVG2.info.extendInfinite

                        newFVG.info.totalVolume := curFVG1.info.totalVolume + curFVG2.info.totalVolume
                        newFVG.info.lowVolume := curFVG1.info.lowVolume + curFVG2.info.lowVolume
                        newFVG.info.highVolume := curFVG1.info.highVolume + curFVG2.info.highVolume
                        newFVG.info.lastTouched := math.max(curFVG1.info.lastTouched, curFVG2.info.lastTouched)
                        newFVG.info.lastTouchedIFVG := math.max(curFVG1.info.lastTouchedIFVG,
curFVG2.info.lastTouchedIFVG)
```

```
                    // Combine IFVG
                    newFVG.info.inverseEndIndex := math.max(nz(curFVG1.info.inverseEndIndex),
nz(curFVG2.info.inverseEndIndex))
                    newFVG.info.inverseEndIndex := newFVG.info.inverseEndIndex == 0 ? na :
newFVG.info.inverseEndIndex

                    newFVG.info.inverseEndTime := math.max(nz(curFVG1.info.inverseEndTime),
nz(curFVG2.info.inverseEndTime))
                    newFVG.info.inverseEndTime := newFVG.info.inverseEndTime == 0 ? na :
newFVG.info.inverseEndTime
                    if curFVG1.info.isInverse or curFVG2.info.isInverse
                        newFVG.info.inverseVolume := nz(curFVG1.info.inverseVolume) +
nz(curFVG2.info.inverseVolume)
                        newFVG.info.isInverse := true

                    newFVG.info.combined := true
                    if timeframe.in_seconds(curFVG1.info.timeframeStr) !=
timeframe.in_seconds(curFVG2.info.timeframeStr)
                        newFVG.info.combinedTimeframesStr := ""
                    allFVGList.unshift(newFVG)
                    lastCombinations += 1

handleFVGsFinal () =>
    FVG[] newFVGsToAdd = array.new<FVG>(0)

    alertTimeFVG = ""
    newBullishFVGAlert = false
    newBearishFVGAlert = false

    alertTimeIFVG = ""
    newBullishIFVGAlert = false
    newBearishIFVGAlert = false

    // Add Timeframe FVGs
    if not na(FVGInfoList)
        if FVGInfoList.size() > 0
            for j = 0 to FVGInfoList.size() - 1
                newFVG = createFVG(FVGInfo.copy(FVGInfoList.get(j)))
                newFVG.info.timeframeStr := ""
                newFVGsToAdd.unshift(newFVG)

    // Check New FVGs
    if newFVGsToAdd.size () > 0
        for i = 0 to newFVGsToAdd.size() - 1
```

```
            curFVG = newFVGsToAdd.get(i)
            if not arrHasFVG(allFVGList, curFVG)
                alertTimeFVG := curFVG.info.timeframeStr
                if curFVG.info.isBull
                    newBullishFVGAlert := true
                else
                    newBearishFVGAlert := true
            if curFVG.info.isInverse
                if not arrHasIFVG(allFVGList, curFVG)
                    alertTimeIFVG := curFVG.info.timeframeStr
                    if curFVG.info.isBull
                        newBullishIFVGAlert := true
                    else
                        newBearishIFVGAlert := true

    // Delete Old FVGs
    if allFVGList.size () > 0
        for i = 0 to allFVGList.size() - 1
            safeDeleteFVG(allFVGList.get(i))
    allFVGList.clear()

    // Add New FVGs
    if newFVGsToAdd.size () > 0
        for i = 0 to newFVGsToAdd.size() - 1
            allFVGList.unshift(newFVGsToAdd.get(i))

    totalFoundFVGs = allFVGList.size()

    // Combine FVGs
    if combineFVGs
        combineFVGsFunc()

    // Render FVGs
    renderedFVGCount = 0
    extendedLastXFVGsCount = 0
    if allFVGList.size() > 0
        for i = 0 to allFVGList.size() - 1
            curFVG = allFVGList.get(i)
            if not isFVGValid(curFVG.info) and (not isIFVGValid(curFVG.info))
                continue
            if extendLastFVGs and na(curFVG.info.endTime) and extendedLastXFVGsCount <
extendLastXFVGsCount
                extendedLastXFVGsCount += 1
                curFVG.info.extendInfinite := true
```

```
            if dbgShowBBFVG
                if curFVG.isRendered
                    safeDeleteFVG(curFVG)
                renderedFVGCount += 1
                renderFVG(curFVG, na)

    if DEBUG and false
        log.info("Rendered " + str.tostring(renderedFVGCount) + " / " + str.tostring(totalFoundFVGs) + " FVGs.")

    [alertTimeFVG, newBullishFVGAlert, newBearishFVGAlert, alertTimeIFVG, newBullishIFVGAlert,
newBearishIFVGAlert]

shortVol = ta.sma(volume, 5)
longVol = ta.sma(volume, 15)
if (bar_index > last_bar_index - maxDistanceToLastBar) and barstate.isconfirmed
    // Add Found FVG
    bearCondition = false
    bullCondition = false

    shortTerm = volCheck ? shortVol : 1
    longTerm = volCheck ? longVol : 0

    firstBarSize = math.max(open, close) - math.min(open, close)
    secondBarSize = math.max(open[1], close[1]) - math.min(open[1], close[1])
    thirdBarSize = math.max(open[2], close[2]) - math.min(open[2], close[2])
    barSizeSum = firstBarSize + secondBarSize + thirdBarSize

    barSizeCheck = true
    //if (secondBarSize < math.max(firstBarSize, thirdBarSize))
        //barSizeCheck := false

    fvgBarsCheck = false
    if fvgBars == "Same Type"
        if (open > close and open[1] > close[1] and open[2] > close[2]) or (open <= close and open[1] <= close[1]
and open[2] <= close[2])
            fvgBarsCheck := true
    else
        fvgBarsCheck := true

    if fvgBarsCheck and barSizeCheck
        maxCODiff = math.max(math.abs(close[2] - open[1]), math.abs(close[1] - open))
        if fvgFilterMethod == "Average Range"
```

```
        bearCondition := ((not fvgSensEnabled) or (barSizeSum * fvgSensitivity > atr / 1.5)) and (allowGaps or
(maxCODiff <= atr))
        bullCondition := ((not fvgSensEnabled) or (barSizeSum * fvgSensitivity > atr / 1.5)) and (allowGaps or
(maxCODiff <= atr))
      else if fvgFilterMethod == "Volume Threshold"
        thresholdMultiplier = (volumeThresholdPercent / 100.0)
        bearCondition := shortTerm > longTerm * thresholdMultiplier and (allowGaps or (maxCODiff <= atr))
        bullCondition := shortTerm > longTerm * thresholdMultiplier and (allowGaps or (maxCODiff <= atr))

  bearFVG = high < low[2] and close[1] < low[2] and bearCondition
  bullFVG = low > high[2] and close[1] > high[2] and bullCondition

  volSum3 = math.sum(volume, 3)
  float totalVolume = volCheck ? volSum3 : 0
  FVGInfo newFVGInfo = bearFVG ? createFVGInfo(low[2], high, false, time, totalVolume) : bullFVG ?
createFVGInfo(low, high[2], true, time, totalVolume) : na
  FVGSize = bearFVG ? math.abs(low[2] - high) : bullFVG ? math.abs(low - high[2]) : 0

  FVGSizeEnough = (FVGSize * fvgSensitivity > atr)
  if FVGSizeEnough
    if not na(newFVGInfo)
      newFVGInfo.startTime := (startZoneFrom == "First Bar" ? time[2] : time)
      newFVGInfo.startBarIndex := bar_index - (startZoneFrom == "First Bar" ? 2 : 0)
      newFVGInfo.lastTouched := bar_index
      if bearFVG
        newFVGInfo.lowVolume := volCheck ? (volume + volume[1]) : 0
        newFVGInfo.highVolume := volCheck ? volume[2] : 0
      else
        newFVGInfo.lowVolume := volCheck ? volume[2] : 0
        newFVGInfo.highVolume := volCheck ? (volume + volume[1]) : 0

    if not na(newFVGInfo)
      FVGInfoList.unshift(newFVGInfo)
      while FVGInfoList.size() > showLastXFVGs
        FVGInfoList.pop()

  // Find Closed FVGs
  if FVGInfoList.size () > 0
    for i = 0 to FVGInfoList.size() - 1
      curFVG = FVGInfoList.get(i)
      // Is Touched FVG
      if ((curFVG.isBull) and low <= curFVG.max) or ((not curFVG.isBull) and high >= curFVG.min)
        curFVG.lastTouched := bar_index
```

```
        if ((not curFVG.isBull) and low <= curFVG.max) or ((curFVG.isBull) and high >= curFVG.min)
            curFVG.lastTouchedIFVG := bar_index

        // IFVG Close
        if curFVG.isInverse and na(curFVG.inverseEndIndex)
            if (not curFVG.isBull) and (ifvgEndMethod == "Wick" ? low < curFVG.min : close < curFVG.min)
                curFVG.inverseEndIndex := bar_index
                curFVG.inverseEndTime := time
            if curFVG.isBull and (ifvgEndMethod == "Wick" ? high > curFVG.max : close > curFVG.max)
                curFVG.inverseEndIndex := bar_index
                curFVG.inverseEndTime := time

        if na(curFVG.endBarIndex)
            // FVG End
            if curFVG.isBull and (fvgEndMethod == "Wick" ? (low < curFVG.min) : (close < curFVG.min))
                curFVG.endBarIndex := bar_index
                curFVG.endTime := time
                curFVG.isInverse := true
                curFVG.inverseVolume := nz(volume)
                curFVG.lastTouchedIFVG := bar_index
                //newIFVGTick := true
            if (not curFVG.isBull) and (fvgEndMethod == "Wick" ? (high > curFVG.max) : (close > curFVG.max))
                curFVG.endBarIndex := bar_index
                curFVG.endTime := time
                curFVG.isInverse := true
                curFVG.inverseVolume := nz(volume)
                curFVG.lastTouchedIFVG := bar_index
                //newIFVGTick := true

// Remove Old FVGs
FVGInfostoRemove = array.new<int>(0)

if FVGInfoList.size() > 0
    for i = 0 to FVGInfoList.size() - 1
        curIndex = FVGInfoList.size() - 1 - i
        curFVGInfo = FVGInfoList.get(curIndex)
        if (not curFVGInfo.isInverse) and not isFVGValidInTimeframe(curFVGInfo)
            FVGInfostoRemove.push(curIndex)
        if curFVGInfo.isInverse and not isIFVGValidInTimeframe(curFVGInfo)
            FVGInfostoRemove.push(curIndex)

if FVGInfostoRemove.size () > 0
    for i = 0 to FVGInfostoRemove.size() - 1
        deleteIndex = FVGInfostoRemove.get(i)
```

```
        FVGInfoList.remove(deleteIndex)
//#endregion

//#region Order Blocks

type obSwing
    int x = na
    float y = na
    bool crossed = false

var orderBlockInfo[] orderBlockInfoList = array.new<orderBlockInfo>(0)
var allOrderBlocksList = array.new<orderBlock>(0)

createOrderBlock (orderBlockInfo orderBlockInfoF) =>
    orderBlock newOrderBlock = orderBlock.new(orderBlockInfoF)
    newOrderBlock

safeDeleteOrderBlock (orderBlock orderBlockF) =>
    orderBlockF.isRendered := false

    box.delete(orderBlockF.orderBox)
    box.delete(orderBlockF.breakerBox)
    box.delete(orderBlockF.orderBoxText)
    box.delete(orderBlockF.orderBoxPositive)
    box.delete(orderBlockF.orderBoxNegative)

    line.delete(orderBlockF.orderBoxLineTop)
    line.delete(orderBlockF.orderBoxLineBottom)
    line.delete(orderBlockF.breakerBoxLineTop)
    line.delete(orderBlockF.breakerBoxLineBottom)
    line.delete(orderBlockF.orderSeperator)
    line.delete(orderBlockF.orderTextSeperator)

arrHasOB (orderBlock[] arr, orderBlock obF) =>
    hasOB = false
    if arr.size() > 0
        for i = 0 to arr.size() - 1
            orderBlock ob1 = arr.get(i)
            if (ob1.info.startTime == obF.info.startTime) and (ob1.info.breaker == obF.info.breaker)
                hasOB := true
                break
    hasOB

renderOrderBlock (orderBlock ob) =>
```

```
    orderBlockInfo info = ob.info
    ob.isRendered := true
    orderColor = ob.info.obType == "Bull" ? bullOrderBlockColor : bearOrderBlockColor
    if OBsEnabled and (not breakersFull or not (BBsEnabled and info.breaker)) and not (not showInvalidated
and info.breaker)
        ob.orderBox := createFVGBox(orderColor, 1.5)
        if ob.info.combined and not changeCombinedFVGsColor
            ob.orderBox.set_bgcolor(colorWithTransparency(orderColor, 1.1))
        ob.orderBoxText := createFVGBox(color.new(color.white, 100))
        if orderBlockVolumetricInfo
            ob.orderBoxPositive := createFVGBox(bullOrderBlockColor)
            ob.orderBoxNegative := createFVGBox(bearOrderBlockColor)
            ob.orderSeperator := line.new(na,na,na,na,xloc.bar_time,extend.none,textColor,line.style_dashed,1)
            ob.orderTextSeperator := line.new(na,na,na,na,xloc.bar_time,extend.none,textColor,line.style_solid,1)

        zoneSize = extendZonesDynamic ? na(info.breakTime) ? extendZonesByTime : ((info.breakTime -
curTFMMS * 1000) - info.startTime) : extendZonesByTime
        if na(info.breakTime)
            zoneSize := (time + 1) - info.startTime

        startX = volumeBarsLeftSide ? info.startTime : info.startTime + zoneSize - zoneSize / 3
        maxEndX = volumeBarsLeftSide ? info.startTime + zoneSize / 3 : info.startTime + zoneSize

        moveBox(ob.orderBox, info.startTime, info.top, info.startTime + zoneSize, info.bottom)
        moveBox(ob.orderBoxText, volumeBarsLeftSide ? maxEndX : info.startTime, info.top,
volumeBarsLeftSide ? info.startTime + zoneSize : startX, info.bottom)

        percentage = int((math.min(info.obHighVolume, info.obLowVolume) / math.max(info.obHighVolume,
info.obLowVolume)) * 100.0)
        OBText = (na(ob.info.combinedTimeframesStr) ? "" : ob.info.combinedTimeframesStr) + ""
        box.set_text(ob.orderBoxText, (orderBlockVolumetricInfo ? str.tostring(ob.info.obVolume, format.volume)
+ " (" + str.tostring(percentage) + "%)\n" : "") + (combinedText and ob.info.combined ? "[Combined]\n" : "") +
OBText)

        if orderBlockVolumetricInfo
            showHighLowBoxText = false

            curEndXHigh = int(math.ceil((info.obHighVolume / info.obVolume) * (maxEndX - startX) + startX))
            curEndXLow = int(math.ceil((info.obLowVolume / info.obVolume) * (maxEndX - startX) + startX))

            curEndXHigh := nz(curEndXHigh, maxEndX)
            curEndXLow := nz(curEndXLow, maxEndX)
```

```
        moveBox(ob.orderBoxPositive, mirrorVolumeBars ? startX : curEndXLow, info.top, mirrorVolumeBars
? curEndXHigh : maxEndX, (info.bottom + info.top) / 2)
        box.set_text(ob.orderBoxPositive, showHighLowBoxText ? str.tostring(info.obHighVolume,
format.volume) : "")

        moveBox(ob.orderBoxNegative, mirrorVolumeBars ? startX : curEndXHigh, info.bottom,
mirrorVolumeBars ? curEndXLow : maxEndX, (info.bottom + info.top) / 2)
        box.set_text(ob.orderBoxNegative, showHighLowBoxText ? str.tostring(info.obLowVolume,
format.volume) : "")

        moveLine(ob.orderSeperator, volumeBarsLeftSide ? startX : maxEndX, (info.bottom + info.top) / 2,
volumeBarsLeftSide ? maxEndX : startX)

        line.set_xy1(ob.orderTextSeperator, volumeBarsLeftSide ? maxEndX : startX, info.top)
        line.set_xy2(ob.orderTextSeperator, volumeBarsLeftSide ? maxEndX : startX, info.bottom)

handleOrderBlocksFinal () =>
    if DEBUG and false
        log.info("Order Block Count " + str.tostring(orderBlockInfoList.size()))

    newBullishOBAlert = false
    newBearishOBAlert = false

    newBullishBBAlert = false
    newBearishBBAlert = false

    alertTimeOB = ""
    alertTimeBB = ""

    orderBlock[] orderBlocksToAdd = array.new<orderBlock>(0)

    if not na(orderBlockInfoList)
        if orderBlockInfoList.size() > 0
            for j = 0 to orderBlockInfoList.size() - 1
                orderBlockInfoF = orderBlockInfoList.get(j)
                orderBlockInfoF.timeframeStr := ""
                orderBlocksToAdd.unshift(createOrderBlock(orderBlockInfo.copy(orderBlockInfoF)))

    // Check New Order & Breaker Blocks
    if orderBlocksToAdd.size () > 0
        for i = 0 to orderBlocksToAdd.size() - 1
            obToTest = orderBlocksToAdd.get(i)
            if obToTest.info.breaker == false
                if not arrHasOB(allOrderBlocksList, obToTest)
```

```
                alertTimeOB := obToTest.info.timeframeStr
                if obToTest.info.obType == "Bull"
                    newBullishOBAlert := true
                else
                    newBearishOBAlert := true
            else
                if not arrHasOB(allOrderBlocksList, obToTest)
                    alertTimeBB := obToTest.info.timeframeStr
                    if obToTest.info.obType == "Bull"
                        newBearishBBAlert := true
                    else
                        newBullishBBAlert := true

    // Delete Old Order Blocks
    if allOrderBlocksList.size () > 0
        for i = 0 to allOrderBlocksList.size() - 1
            safeDeleteOrderBlock(allOrderBlocksList.get(i))
    allOrderBlocksList.clear()

    // Add New Order Blocks
    if orderBlocksToAdd.size () > 0
        for i = 0 to orderBlocksToAdd.size() - 1
            allOrderBlocksList.unshift(orderBlocksToAdd.get(i))

    if allOrderBlocksList.size() > 0
        for i = 0 to allOrderBlocksList.size() - 1
            curOB = allOrderBlocksList.get(i)
            if dbgShowBBFVG
                renderOrderBlock(curOB)

    [alertTimeOB, alertTimeBB, newBullishOBAlert, newBearishOBAlert, newBullishBBAlert,
newBearishBBAlert]

findOrderBlocks() =>
    if (bar_index > last_bar_index - maxDistanceToLastBar and barstate.isconfirmed) and (OBsEnabled or
BBsEnabled)
        upper = ta.highest(swingLength)
        lower = ta.lowest(swingLength)

        var swingType = 0
        var obSwing top = obSwing.new(na, na)
        var obSwing btm = obSwing.new(na, na)

        li = low[swingLength]
```

```
hi = high[swingLength]
bi = bar_index[swingLength]

swingType := hi > upper ? 0 : li < lower ? 1 : swingType

if swingType == 0 and swingType[1] != 0
    top := obSwing.new(bi, hi)

if swingType == 1 and swingType[1] != 1
    btm := obSwing.new(bi, li)

boxBtmBull = high[1]
boxTopBull = low[1]
boxLocBull = time[1]

for i = 1 to (bar_index - top.x) - 1
    if i > maxBarsBack - 2
        break
    boxBtmBull := math.min(low[i], boxBtmBull)
    boxTopBull := boxBtmBull == low[i] ? high[i] : boxTopBull
    boxLocBull := boxBtmBull == low[i] ? time[i] : boxLocBull

boxBtmBear = low[1]
boxTopBear = high[1]
boxLocBear = time[1]

for i = 1 to (bar_index - btm.x) - 1
    if i > maxBarsBack - 2
        break
    boxTopBear := math.max(high[i], boxTopBear)
    boxBtmBear := boxTopBear == high[i] ? low[i] : boxBtmBear
    boxLocBear := boxTopBear == high[i] ? time[i] : boxLocBear

// Invalidations
obListSize = orderBlockInfoList.size()
if obListSize > 0
    for i = 0 to obListSize - 1
        currentOB = orderBlockInfoList.get(i)
        if currentOB.obType == "Bull"
            if not currentOB.breaker
                if (obEndMethod == "Wick" ? low : math.min(open, close)) < currentOB.bottom
                    currentOB.breaker := true
                    currentOB.breakTime := time_close
                    currentOB.bbVolume := volume
```

```
            else
                if (bbEndMethod == "Wick" ? high : close) > currentOB.top and na(currentOB.breakerEndTime)
                    currentOB.breakerEndTime := time_close
        else
            if not currentOB.breaker
                if (obEndMethod == "Wick" ? high : math.max(open, close)) > currentOB.top
                    currentOB.breaker := true
                    currentOB.breakTime := time_close
                    currentOB.bbVolume := volume
            else
                if (bbEndMethod == "Wick" ? low : close) < currentOB.bottom and
na(currentOB.breakerEndTime)
                    currentOB.breakerEndTime := time_close

    // Bullish Order Block
    if close > top.y and not top.crossed
        top.crossed := true
        obSize = math.abs(boxTopBull - boxBtmBull)
        if obSize <= atr * maxATRMult
            newOrderBlockInfo = orderBlockInfo.new(boxTopBull, boxBtmBull, volume + volume[1] + volume[2],
"Bull", boxLocBull)
            newOrderBlockInfo.obLowVolume := volume[2]
            newOrderBlockInfo.obHighVolume := volume + volume[1]
            orderBlockInfoList.unshift(newOrderBlockInfo)
            if orderBlockInfoList.size() > maxOrderBlocks
                orderBlockInfoList.pop()

    // Bearish Order Block
    if close < btm.y and not btm.crossed
        btm.crossed := true
        obSize = math.abs(boxTopBear - boxBtmBear)
        if obSize <= atr * maxATRMult
            newOrderBlockInfo = orderBlockInfo.new(boxTopBear, boxBtmBear, volume + volume[1] +
volume[2], "Bear", boxLocBear)
            newOrderBlockInfo.obLowVolume := volume + volume[1]
            newOrderBlockInfo.obHighVolume := volume[2]
            orderBlockInfoList.unshift(newOrderBlockInfo)
            if orderBlockInfoList.size() > maxOrderBlocks
                orderBlockInfoList.pop()
  orderBlockInfoList

findOrderBlocks()
//#endregion
```

```
//#region Setup
if barstate.isconfirmed and (bar_index > last_bar_index - maxDistanceToLastBar)
    [alertTimeFVG, newBullishFVGAlert, newBearishFVGAlert, alertTimeIFVG, newBullishIFVGAlert,
newBearishIFVGAlert] = handleFVGsFinal()
    [alertTimeOB, alertTimeBB, newBullishOBAlert, newBearishOBAlert, newBullishBBAlert,
newBearishBBAlert] = handleOrderBlocksFinal()
//#endregion

//#region ICT CRT
const int maxCRT = 75
const int atrLenCRT = 50
var initRun = true

buyAlertTick = false
sellAlertTick = false
tpAlertTick = false
slAlertTick = false

getPosition (positionText) =>
    if positionText == "Top Right"
        position.top_right
    else if positionText == "Top Center"
        position.top_center
    else if positionText == "Right Center"
        position.middle_right
    else if positionText == "Left Center"
        position.middle_left
    else if positionText == "Bottom Center"
        position.bottom_center
    else if positionText == "Middle Center"
        position.middle_center

//#region Bulky Candle
type barInfo
    float o
    float h
    float l
    float c
    float tr
    float atr

curBar = barInfo.new(open, high, low, close, ta.tr, atr)
higherTFBar = request.security(syminfo.tickerid, higherTF, curBar)
oldHigherTFBar = higherTFBar[1]
```

```
var float lastHigh = na
var float lastLow = na
bool newBulkyCandle = false
if not na(oldHigherTFBar)
    if oldHigherTFBar.h != higherTFBar.h and (higherTFBar.tr > higherTFBar.atr * bulkyCandleATR)
        newBulkyCandle := true
        lastHigh := higherTFBar.h
        lastLow := higherTFBar.l

//plotchar(newBulkyCandle ? high : na, "", "✅", size = size.tiny)
//#endregion

type CRT
    string state
    int startTime

    string overlapDirection
    int bulkyTimeLow
    int bulkyTimeHigh
    float bulkyHigh
    float bulkyLow
    int breakTime

    FVG fvg
    int fvgEndTime
    orderBlock ob

    float slTarget
    float tpTarget
    string entryType
    int entryTime
    int exitTime
    float entryPrice
    float exitPrice
    int dayEndedBeforeExit

var lineX = array.new<line>()
var boxX = array.new<box>()
var labelX = array.new<label>()

var CRT[] crtList = array.new<CRT>(0)
var CRT lastCRT = na

atrCRT = ta.atr(atrLenCRT)
```

```
diffPercent (float val1, float val2) =>
    (math.abs(val1 - val2) / val2) * 100.0

var FVG latestFVG = na
if allFVGList.size() > 0
    latestFVG := allFVGList.get(0)

var orderBlock latestOB = na
if allOrderBlocksList.size() > 0
    latestOB := allOrderBlocksList.get(0)

//#region CRT
if bar_index > last_bar_index - maxDistanceToLastBar and barstate.isconfirmed
    if true
        createNewCRT = true
        if not na(lastCRT)
            if na(lastCRT.exitPrice) and lastCRT.state != "Aborted"
                createNewCRT := false // Don't enter if a trade is already entered

        if createNewCRT
            newCRT = CRT.new("Waiting For Bulky Candle", time)
            crtList.unshift(newCRT)
            lastCRT := newCRT
            log.info("Waiting For Bulky Candle")

        if not na(lastCRT)
            // Waiting For Bulky Candle
            if lastCRT.state == "Waiting For Bulky Candle" and newBulkyCandle
                lastCRT.bulkyHigh := lastHigh
                lastCRT.bulkyLow := lastLow
                lastCRT.bulkyTimeLow := findValRtnTime(lastLow, "Low", "Nearest")
                lastCRT.bulkyTimeHigh := findValRtnTime(lastHigh, "High", "Nearest")
                lastCRT.state := "Waiting For Side Retest"
                log.info("Waiting For Side Retest")

            // Waiting For Side Retest
            else if lastCRT.state == "Waiting For Side Retest"
                if close > lastCRT.bulkyHigh
                    lastCRT.state := "Aborted"
                    log.info("Aborted")
                else if close < lastCRT.bulkyLow
                    lastCRT.state := "Aborted"
                    log.info("Aborted")
```

```
    if lastCRT.state != "Aborted"
        bearOverlap = false
        bullOverlap = false
        if high > lastCRT.bulkyHigh and close <= lastCRT.bulkyHigh
            bearOverlap := true
        if low < lastCRT.bulkyLow and close >= lastCRT.bulkyLow
            bullOverlap := true

        if bearOverlap and not bullOverlap
            lastCRT.overlapDirection := "Bear"
            lastCRT.breakTime := time
            if entryMode == "FVGs"
                lastCRT.state := "Waiting For FVG"
                log.info("Waiting For Bearish FVG @" + str.tostring(bar_index))
            else
                lastCRT.state := "Waiting For OB"
                log.info("Waiting For Bearish OB @" + str.tostring(bar_index))

        if bullOverlap and not bearOverlap
            lastCRT.overlapDirection := "Bull"
            lastCRT.breakTime := time
            if entryMode == "FVGs"
                lastCRT.state := "Waiting For FVG"
                log.info("Waiting For Bullish FVG @" + str.tostring(bar_index))
            else
                lastCRT.state := "Waiting For OB"
                log.info("Waiting For Bullish OB @" + str.tostring(bar_index))

if lastCRT.state == "Waiting For OB"
    if lastCRT.overlapDirection == "Bear" // Bearish OB
        if not na(latestOB)
            if (latestOB.info.startTime > lastCRT.breakTime) and latestOB.info.obType == "Bear"
                lastCRT.ob := latestOB
                if not requireRetracement
                    lastCRT.state := "Enter Position"
                else
                    lastCRT.state := "Waiting For OB Retracement"
                    log.info("Waiting For OB Retracement")
    else // Bullish OB
        if not na(latestOB)
            if (latestOB.info.startTime > lastCRT.breakTime) and latestOB.info.obType == "Bull"
                lastCRT.ob := latestOB
                if not requireRetracement
```

```
                lastCRT.state := "Enter Position"
            else
                lastCRT.state := "Waiting For OB Retracement"
                log.info("Waiting For OB Retracement")
// Waiting For FVG
if lastCRT.state == "Waiting For FVG"
    if lastCRT.overlapDirection == "Bear" // Bearish FVG
        if not na(latestFVG)
            if latestFVG.info.startTime == time and not latestFVG.info.isBull
                lastCRT.fvg := latestFVG
                if not requireRetracement
                    lastCRT.state := "Enter Position"
                else
                    lastCRT.state := "Waiting For FVG Retracement"
                    log.info("Waiting For FVG Retracement")
    else // Bullish FVG
        if not na(latestFVG)
            if latestFVG.info.startTime == time and latestFVG.info.isBull
                lastCRT.fvg := latestFVG
                if not requireRetracement
                    lastCRT.state := "Enter Position"
                else
                    lastCRT.state := "Waiting For FVG Retracement"
                    log.info("Waiting For FVG Retracement")

// Update FVG & FVG Retests
if not na(lastCRT.fvg)
    if na(lastCRT.fvgEndTime)
        if lastCRT.state == "Waiting For FVG Retracement" and time > lastCRT.fvg.info.startTime
            if lastCRT.fvg.info.isBull and low <= lastCRT.fvg.info.max
                lastCRT.state := "Enter Position"
            if (not lastCRT.fvg.info.isBull) and high >= lastCRT.fvg.info.min
                lastCRT.state := "Enter Position"
        // Invalidation
        if lastCRT.fvg.info.isBull and low < lastCRT.fvg.info.min
            lastCRT.fvgEndTime := time
        if (not lastCRT.fvg.info.isBull) and high > lastCRT.fvg.info.max
            lastCRT.fvgEndTime := time
// Update OB & OB Retests
if not na(lastCRT.ob)
    if na(lastCRT.ob.info.breakTime)
        if lastCRT.state == "Waiting For OB Retracement" and time > lastCRT.ob.info.startTime
            if lastCRT.ob.info.obType == "Bull" and low <= lastCRT.ob.info.top
                lastCRT.state := "Enter Position"
```

```
                if lastCRT.ob.info.obType == "Bear" and high >= lastCRT.ob.info.bottom
                    lastCRT.state := "Enter Position"
            // Invalidation
            if lastCRT.ob.info.obType == "Bull" and low < lastCRT.ob.info.bottom
                lastCRT.ob.info.breakTime := time
            if lastCRT.ob.info.obType == "Bear" and high > lastCRT.ob.info.top
                lastCRT.ob.info.breakTime := time
    // Enter Position
    if not na(lastCRT)
        if lastCRT.state == "Enter Position"
            log.info("Entry Taken")
            lastCRT.state := "Entry Taken"
            lastCRT.entryTime := time
            lastCRT.entryPrice := close
            if lastCRT.overlapDirection == "Bull"
                lastCRT.entryType := "Long"
                buyAlertTick := true
                if tpslMethod == "Fixed"
                    lastCRT.slTarget := lastCRT.entryPrice * (1 - slPercent / 100.0)
                    lastCRT.tpTarget := lastCRT.entryPrice * (1 + tpPercent / 100.0)
                else if tpslMethod == "Dynamic"
                    lastCRT.slTarget := lastCRT.entryPrice - atrCRT * slATRMult
                    lastCRT.tpTarget := lastCRT.entryPrice + (math.abs(lastCRT.entryPrice - lastCRT.slTarget) *
DynamicRR)
            else
                lastCRT.entryType := "Short"
                sellAlertTick := true
                if tpslMethod == "Fixed"
                    lastCRT.slTarget := lastCRT.entryPrice * (1 + slPercent / 100.0)
                    lastCRT.tpTarget := lastCRT.entryPrice * (1 - tpPercent / 100.0)
                else if tpslMethod == "Dynamic"
                    lastCRT.slTarget := lastCRT.entryPrice + atrCRT * slATRMult
                    lastCRT.tpTarget := lastCRT.entryPrice - (math.abs(lastCRT.entryPrice - lastCRT.slTarget) *
DynamicRR)
    // Entry Taken
    if not na(lastCRT)
        if lastCRT.state == "Entry Taken" and time > lastCRT.entryTime
            if tpslMethod == "Fixed"
                // Take Profit
                if lastCRT.entryType == "Long" and ((high / lastCRT.entryPrice) - 1) * 100 >= tpPercent
                    tpAlertTick := true
                    lastCRT.exitPrice := lastCRT.entryPrice * (1 + tpPercent / 100.0)
                    lastCRT.exitTime := time
                    lastCRT.state := "Take Profit"
```

```
        if na(lastCRT.fvgEndTime)
            lastCRT.fvgEndTime := time
        if not na(lastCRT.ob)
            if na(lastCRT.ob.info.breakTime)
                lastCRT.ob.info.breakTime := time
    if lastCRT.entryType == "Short" and ((low / lastCRT.entryPrice) - 1) * 100 <= -tpPercent
        tpAlertTick := true
        lastCRT.exitPrice := lastCRT.entryPrice * (1 - tpPercent / 100.0)
        lastCRT.exitTime := time
        lastCRT.state := "Take Profit"
        if na(lastCRT.fvgEndTime)
            lastCRT.fvgEndTime := time
        if not na(lastCRT.ob)
            if na(lastCRT.ob.info.breakTime)
                lastCRT.ob.info.breakTime := time

    // Stop Loss
    if lastCRT.entryType == "Long" and ((low / lastCRT.entryPrice) - 1) * 100 <= -slPercent
        slAlertTick := true
        lastCRT.exitPrice := lastCRT.entryPrice * (1 - slPercent / 100.0)
        lastCRT.exitTime := time
        lastCRT.state := "Stop Loss"
        if na(lastCRT.fvgEndTime)
            lastCRT.fvgEndTime := time
        if not na(lastCRT.ob)
            if na(lastCRT.ob.info.breakTime)
                lastCRT.ob.info.breakTime := time
    if lastCRT.entryType == "Short" and ((high / lastCRT.entryPrice) - 1) * 100 >= slPercent
        slAlertTick := true
        lastCRT.exitPrice := lastCRT.entryPrice * (1 + slPercent / 100.0)
        lastCRT.exitTime := time
        lastCRT.state := "Stop Loss"
        if na(lastCRT.fvgEndTime)
            lastCRT.fvgEndTime := time
        if not na(lastCRT.ob)
            if na(lastCRT.ob.info.breakTime)
                lastCRT.ob.info.breakTime := time
else
    // Take Profit
    if lastCRT.entryType == "Long" and high >= lastCRT.tpTarget
        tpAlertTick := true
        lastCRT.exitPrice := lastCRT.tpTarget
        lastCRT.exitTime := time
        lastCRT.state := "Take Profit"
```

```
            if na(lastCRT.fvgEndTime)
                lastCRT.fvgEndTime := time
            if not na(lastCRT.ob)
                if na(lastCRT.ob.info.breakTime)
                    lastCRT.ob.info.breakTime := time
        if lastCRT.entryType == "Short" and low <= lastCRT.tpTarget
            tpAlertTick := true
            lastCRT.exitPrice := lastCRT.tpTarget
            lastCRT.exitTime := time
            lastCRT.state := "Take Profit"
            if na(lastCRT.fvgEndTime)
                lastCRT.fvgEndTime := time
            if not na(lastCRT.ob)
                if na(lastCRT.ob.info.breakTime)
                    lastCRT.ob.info.breakTime := time

        // Stop Loss
        if lastCRT.entryType == "Long" and low <= lastCRT.slTarget
            slAlertTick := true
            lastCRT.exitPrice := lastCRT.slTarget
            lastCRT.exitTime := time
            lastCRT.state := "Stop Loss"
            if na(lastCRT.fvgEndTime)
                lastCRT.fvgEndTime := time
            if not na(lastCRT.ob)
                if na(lastCRT.ob.info.breakTime)
                    lastCRT.ob.info.breakTime := time
        if lastCRT.entryType == "Short" and high >= lastCRT.slTarget
            slAlertTick := true
            lastCRT.exitPrice := lastCRT.slTarget
            lastCRT.exitTime := time
            lastCRT.state := "Stop Loss"
            if na(lastCRT.fvgEndTime)
                lastCRT.fvgEndTime := time
            if not na(lastCRT.ob)
                if na(lastCRT.ob.info.breakTime)
                    lastCRT.ob.info.breakTime := time
//#endregion

//#region Render CRT
renderTopSL = false
renderBottomSL = false
renderTopTP = false
renderBottomTP = false
```

```
if not na(lastCRT)
    if lastCRT.state == "Stop Loss" and time >= lastCRT.exitTime
        if lastCRT.entryType == "Long"
            renderBottomSL := true
        else
            renderTopSL := true
        lastCRT.state := "Done"
    if lastCRT.state == "Take Profit"
        if lastCRT.entryType == "Long"
            renderTopTP := true
        else
            renderBottomTP := true
        lastCRT.state := "Done"

plotshape(renderTopSL, "", shape.circle, location.abovebar, color.red, textcolor = textColor, text = "SL", size =
size.tiny)
plotshape(renderBottomSL, "", shape.circle, location.belowbar, color.red, textcolor = textColor, text = "SL",
size = size.tiny)
plotshape(renderTopTP, "", shape.xcross, location.abovebar, color.blue, textcolor = textColor, text = "TP", size
= size.tiny)
plotshape(renderBottomTP, "", shape.xcross, location.belowbar, color.blue, textcolor = textColor, text = "TP",
size = size.tiny)
//#endregion

//#region Alerts
if barstate.islastconfirmedhistory
    initRun := false

alertcondition(buyAlertTick and not initRun, "Buy Signal", "")
alertcondition(sellAlertTick and not initRun, "Sell Signal", "")
alertcondition(tpAlertTick and not initRun, "Take-Profit Signal", "")
alertcondition(slAlertTick and not initRun, "Stop-Loss Signal", "")

if not initRun
    if buyAlertTick and buyAlertEnabled
        alert("Buy Signal")
    if sellAlertTick and sellAlertEnabled
        alert("Sell Signal")

    if tpAlertTick and tpAlertEnabled
        alert("Take-Profit Signal")
    if slAlertTick and slAlertEnabled
        alert("Stop-Loss Signal")
```

```
//#endregion

//#region Backtesting Dashboard
if barstate.islast and backtestDisplayEnabled
    var table backtestDisplay = table.new(getPosition(backtestingLocation), 2, 10, bgcolor = screenerColor,
frame_width = 2, frame_color = color.black, border_width = 1, border_color = color.black)

    float totalTSProfitPercent = 0
    int successfulTrades = 0
    int unsuccessfulTrades = 0

    if crtList.size() > 0
        for i = 0 to crtList.size() - 1
            curCRT = crtList.get(i)
            if not na(curCRT.entryPrice)
                isSuccess = false
                if not na(curCRT.exitPrice)
                    if (curCRT.entryType == "Long" and curCRT.exitPrice > curCRT.entryPrice) or (curCRT.entryType
== "Short" and curCRT.exitPrice < curCRT.entryPrice)
                        totalTSProfitPercent += math.abs(diffPercent(curCRT.entryPrice, curCRT.exitPrice))
                        isSuccess := true
                    else
                        totalTSProfitPercent -= math.abs(diffPercent(curCRT.entryPrice, curCRT.exitPrice))
                        isSuccess := false

                if isSuccess
                    successfulTrades += 1
                else
                    unsuccessfulTrades += 1

    // Header
    table.merge_cells(backtestDisplay, 0, 0, 1, 0)
    table.cell(backtestDisplay, 0, 0, "CRT Backtesting", text_color = color.white, bgcolor = screenerColor)

    // Total ORBs
    table.cell(backtestDisplay, 0, 1, "Total Entries", text_color = color.white, bgcolor = screenerColor)
    table.cell(backtestDisplay, 1, 1, str.tostring(successfulTrades + unsuccessfulTrades), text_color =
color.white, bgcolor = screenerColor)

    // Wins
    table.cell(backtestDisplay, 0, 2, "Wins", text_color = color.white, bgcolor = screenerColor)
    table.cell(backtestDisplay, 1, 2, str.tostring(successfulTrades), text_color = color.white, bgcolor =
screenerColor)
```

```
    // Losses
    table.cell(backtestDisplay, 0, 3, "Losses", text_color = color.white, bgcolor = screenerColor)
    table.cell(backtestDisplay, 1, 3, str.tostring(unsuccessfulTrades), text_color = color.white, bgcolor =
screenerColor)

    // Winrate
    table.cell(backtestDisplay, 0, 4, "Winrate", text_color = color.white, bgcolor = screenerColor)
    table.cell(backtestDisplay, 1, 4, str.tostring(100.0 * (successfulTrades / (successfulTrades +
unsuccessfulTrades)), "#.##") + "%", text_color = color.white, bgcolor = screenerColor)

    // Average Profit %
    table.cell(backtestDisplay, 0, 5, "Average Profit", text_color = color.white, bgcolor = screenerColor)
    table.cell(backtestDisplay, 1, 5, str.tostring(totalTSProfitPercent / (successfulTrades + unsuccessfulTrades),
"#.##") + "%", text_color = color.white, bgcolor = screenerColor)

    // Total Profit %
    table.cell(backtestDisplay, 0, 6, "Total Profit", text_color = color.white, bgcolor = screenerColor)
    table.cell(backtestDisplay, 1, 6, str.tostring(totalTSProfitPercent, "#.##") + "%", text_color = color.white,
bgcolor = screenerColor)
//#endregion

if barstate.isconfirmed
    if lineX.size() > 0
        for i = 0 to lineX.size() - 1
            line.delete(lineX.get(i))

    if boxX.size() > 0
        for i = 0 to boxX.size() - 1
            box.delete(boxX.get(i))

    if labelX.size() > 0
        for i = 0 to labelX.size() - 1
            label.delete(labelX.get(i))

    lineX.clear()
    boxX.clear()
    labelX.clear()

    if crtList.size() > 0
        for i = 0 to math.min(maxCRT, crtList.size() - 1)
            curTS = crtList.get(i)
            // TP / SL
            if not na(curTS.entryTime)
```

```
            // Liq Grab
            if showHTFLines
                lineX.push(line.new(curTS.bulkyTimeLow, curTS.bulkyLow, curTS.breakTime, curTS.bulkyLow,
color = lowColor, xloc = xloc.bar_time, width = 2, style = curTS.entryType == "Long" ? line.style_dotted :
line.style_solid))
                lineX.push(line.new(curTS.bulkyTimeHigh, curTS.bulkyHigh, curTS.breakTime, curTS.bulkyHigh,
color = highColor, xloc = xloc.bar_time, width = 2, style = curTS.entryType == "Long" ? line.style_solid:
line.style_dotted))

            // FVG
            if showFVG and not na(curTS.fvg)
                safeDeleteFVG(curTS.fvg)
                renderFVG(curTS.fvg, math.max(curTS.fvg.info.startTime + timeframe.in_seconds() * 1000 * 5,
curTS.fvgEndTime))

            // OB
            if showOB and not na(curTS.ob)
                safeDeleteOrderBlock(curTS.ob)
                renderOrderBlock(curTS.ob)

            // Entry Label
            if curTS.entryType == "Long"
                labelX.push(label.new(curTS.entryTime, close, "Buy", xloc = xloc.bar_time, yloc = yloc.belowbar,
textcolor = color.new(textColor, 0) , color = highColor, style = label.style_label_up, size = lblSize))
            else
                labelX.push(label.new(curTS.entryTime, close, "Sell", xloc = xloc.bar_time, yloc = yloc.abovebar,
textcolor = color.new(textColor, 0), color = lowColor, style = label.style_label_down, size = lblSize))

            // TP / SL
            if showTPSL
                if dbgTPSLVersion == "Alternative"
                    offset = atrCRT / 3.0
                    endTime = nz(curTS.exitTime, time("", -15))
                    boxX.push(box.new(curTS.entryTime, curTS.tpTarget + offset, endTime, curTS.tpTarget -
offset, text = "TAKE PROFIT (" + str.tostring(curTS.tpTarget, format.mintick) + ")", text_color = textColor, xloc =
xloc.bar_time, border_width = 0, bgcolor = color.new(highColor, 50), text_size = size.small))
                    boxX.push(box.new(curTS.entryTime, curTS.slTarget + offset, endTime, curTS.slTarget - offset,
text = "STOP LOSS (" + str.tostring(curTS.slTarget, format.mintick) + ")", text_color = textColor, xloc =
xloc.bar_time, border_width = 0, bgcolor = color.new(lowColor, 50) , text_size = size.small))
                else if dbgTPSLVersion == "Default"
                    endTime = nz(curTS.exitTime, time("", -15))
                    lineX.push(line.new(curTS.entryTime, curTS.entryPrice, curTS.entryTime, curTS.tpTarget, xloc
= xloc.bar_time, color = highColor, style = line.style_dashed))
```

```
                lineX.push(line.new(curTS.entryTime, curTS.tpTarget, endTime, curTS.tpTarget, xloc =
xloc.bar_time, color = highColor, style = line.style_dashed))
                labelX.push(label.new(endTime, curTS.tpTarget, "TP", xloc = xloc.bar_time, yloc = yloc.price,
textcolor = color.new(textColor, 0), color = color.new(highColor, 50), style = label.style_label_left, size =
lblSize))
                //
                lineX.push(line.new(curTS.entryTime, curTS.entryPrice, curTS.entryTime, curTS.slTarget, xloc
= xloc.bar_time, color = lowColor, style = line.style_dashed))
                lineX.push(line.new(curTS.entryTime, curTS.slTarget, endTime, curTS.slTarget, xloc =
xloc.bar_time, color = lowColor, style = line.style_dashed))
                labelX.push(label.new(endTime, curTS.slTarget, "SL", xloc = xloc.bar_time, yloc = yloc.price,
textcolor = color.new(textColor, 0), color = color.new(lowColor, 50), style = label.style_label_left, size =
lblSize))

        if not na(curTS.dayEndedBeforeExit)
            labelX.push(label.new(curTS.dayEndedBeforeExit, close, "Exit", xloc = xloc.bar_time, yloc =
yloc.belowbar, textcolor = textColor, color = color.yellow, style = label.style_circle, size = size.tiny))
//#endregion
```