

```
#include "sierrachart.h"
```

```
#include "CandleStickPatternNames.h"
```

```
/*=====*/
SCSFExport scsf_WoodieCCITrend2(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_CCI = sc.Subgraph[0];
    SCSubgraphRef Subgraph_TrendDown = sc.Subgraph[1];
    SCSubgraphRef Subgraph_TrendNeutral = sc.Subgraph[2];
    SCSubgraphRef Subgraph_TrendUp = sc.Subgraph[3];
    SCSubgraphRef Subgraph_HiLevel = sc.Subgraph[4];
    SCSubgraphRef Subgraph_LowLevel = sc.Subgraph[5];
    SCSubgraphRef Subgraph_Consecutive = sc.Subgraph[6];
    SCSubgraphRef Subgraph_LastTrend = sc.Subgraph[7];
    SCSubgraphRef Subgraph_WorksheetOutput = sc.Subgraph[8];
    SCSubgraphRef Subgraph_ZLR = sc.Subgraph[9];

    SCInputRef Input_NumberOfBars = sc.Input[0];
    SCInputRef Input_NeutralBars = sc.Input[1];
    SCInputRef Input_CCILength = sc.Input[2];
    SCInputRef Input_Level = sc.Input[3];
    SCInputRef Input_CCIInputData = sc.Input[4];
    SCInputRef Input_Version = sc.Input[5];

    if(sc.SetDefaults)
    {
        sc.GraphName="Woodies CCI Trend";

        sc.AutoLoop = 1;

        Subgraph_TrendDown.Name = "TrendDown";
        Subgraph_TrendDown.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_TrendDown.PrimaryColor = RGB(255,0,0);
        Subgraph_TrendDown.DrawZeros = false;

        Subgraph_TrendNeutral.Name = "TrendNeutral";
        Subgraph_TrendNeutral.PrimaryColor = RGB(255,255,0);
        Subgraph_TrendNeutral.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_TrendNeutral.DrawZeros = false;

        Subgraph_TrendUp.Name = "TrendUp";
        Subgraph_TrendUp.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_TrendUp.PrimaryColor = RGB(0,0,255);
        Subgraph_TrendUp.DrawZeros = false;

        Subgraph_HiLevel.Name = "Hi Level";
        Subgraph_HiLevel.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_HiLevel.PrimaryColor = RGB(0,255,0);
        Subgraph_HiLevel.DrawZeros = false;

        Subgraph_LowLevel.Name = "Low Level";
        Subgraph_LowLevel.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_LowLevel.PrimaryColor = RGB(0,255,0);
        Subgraph_LowLevel.DrawZeros = false;

        Subgraph_CCI.Name = "CCI";
        Subgraph_CCI.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_CCI.PrimaryColor = RGB(98,98,98);
        Subgraph_CCI.DrawZeros = false;

        Subgraph_Consecutive.DrawStyle = DRAWSTYLE_IGNORE;
        Subgraph_Consecutive.DrawZeros = false;

        Subgraph_LastTrend.DrawStyle = DRAWSTYLE_IGNORE;
        Subgraph_LastTrend.DrawZeros = false;
    }
}
```

```

Subgraph_WorksheetOutput.Name = "Spreadsheet Ouput";
Subgraph_WorksheetOutput.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_WorksheetOutput.PrimaryColor = RGB(0,127,255);
Subgraph_WorksheetOutput.DrawZeros = false;

Subgraph_ZLR.Name = "ZLR Output";
Subgraph_ZLR.DrawStyle = DRAWSTYLE_POINT;
Subgraph_ZLR.PrimaryColor = RGB(255,0,0);
Subgraph_ZLR.SecondaryColor = RGB(0,255,0);
Subgraph_ZLR.SecondaryColorUsed = 1;
Subgraph_ZLR.LineWidth = 3;
Subgraph_ZLR.DrawZeros = false;

Input_CCILength.Name = "CCI Length";
Input_CCILength.SetInt(14);
Input_CCILength.SetIntLimits(1,MAX_STUDY_LENGTH);

Input_NumberOfBars.Name = "Number of Bars";
Input_NumberOfBars.SetInt(6);
Input_NumberOfBars.SetIntLimits(1,MAX_STUDY_LENGTH);

Input_NeutralBars.Name = "Number of Neutral Bars";
Input_NeutralBars.SetInt(5);
Input_NeutralBars.SetIntLimits(1,MAX_STUDY_LENGTH);

Input_Level.Name = "Level";
Input_Level.SetInt(100);
Input_Level.SetIntLimits(1,MAX_STUDY_LENGTH);

Input_CCInputData.Name = "Input Data";
Input_CCInputData.SetInputDataIndex(SC_HLC_AVG);

Input_Version.SetInt(1);

return;
}

//Current number of consecutive conditions CCI > 0 (or CCI < 0 if variable less zero)
float& consec = Subgraph_Consecutive[sc.Index];
const float& PrevConsec = Subgraph_Consecutive[sc.Index-1];

float& currTrend= Subgraph_LastTrend[sc.Index]; // -1 - TrendDown; 1 - TrendUp
float& lastTrend= Subgraph_LastTrend[sc.Index-1];

const float& currCCI = Subgraph_CCI[sc.Index];
int OccurGreatEdge=0;
int OccurLessEdge=0;

Subgraph_HiLevel[sc.Index] = static_cast<float>(Input_Level.GetInt());
Subgraph_LowLevel[sc.Index] = static_cast<float>(-Input_Level.GetInt());

sc.DataStartIndex= Input_CCILength.GetInt()-1;

sc.CCI(sc.BaseData[Input_CCInputData.GetInputDataIndex()], Subgraph_CCI, Input_CCILength.GetInt(), 0.015f);

if(sc.Index == 0)
{
    if(Subgraph_CCI[sc.Index] > 0)
        consec = 1;
    else if(Subgraph_CCI[sc.Index] < 0)
        consec = -1;
}

```

```

        else
            consec = 0;
    }
    else
    {
        if(currCCI > 0)
        {
            if(PrevConsec < 0)
                consec = 1;
            else
                consec = PrevConsec + 1;
        }
        else if(currCCI < 0)
        {
            if(PrevConsec > 0)
                consec = -1;
            else
                consec = PrevConsec - 1;
        }
    }

    for(int i = sc.Index; i>sc.Index-Input_NumberOfBars.GetInt(); i--)
    {
        if(Subgraph_CCI[i] > Input_Level.GetInt())
            OccurGreatEdge++;
        else if(Subgraph_CCI[i] < -Input_Level.GetInt())
            OccurLessEdge++;
    }

    bool trendUp = (currCCI > 0) && ((lastTrend == 1) || ((consec >= Input_NumberOfBars.GetInt()) && (OccurGreatEdge > 0)));
    bool trendDown = (currCCI < 0) && ((lastTrend == -1) || ((consec <= -Input_NumberOfBars.GetInt()) && (OccurLessEdge > 0)));

    //Zero out subgraphs that color the main subgraph. This needs to be done in case one of these conditions is no longer met when a bar is forming.
    Subgraph_TrendUp[sc.Index] = 0;
    Subgraph_TrendDown[sc.Index] = 0;
    Subgraph_TrendNeutral[sc.Index] = 0;
    Subgraph_WorksheetOutput[sc.Index] = 0;

    if(trendUp)
    {
        Subgraph_TrendUp[sc.Index] = currCCI;//trend up
        currTrend = 1;
        Subgraph_WorksheetOutput[sc.Index] = 3;
    }
    else if(trendDown)
    {
        Subgraph_TrendDown[sc.Index] = currCCI;//trend down
        currTrend = -1;
        Subgraph_WorksheetOutput[sc.Index] = 1;
    }
    else
    {
        currTrend = lastTrend;
        if ((consec >= Input_NeutralBars.GetInt() && !trendUp) || (-consec >= Input_NeutralBars.GetInt() && !trendDown) )
            Subgraph_TrendNeutral[sc.Index] = currCCI;//trend neutral
        Subgraph_WorksheetOutput[sc.Index] = 2;
    }

    //Calculate the ZLR
    SCFloatArrayRef ZLRHooks = Subgraph_ZLR.Arrays[0]; // First extra array of ZLR subgraph

    if ((Subgraph_CCI[sc.Index-1] > Subgraph_CCI[sc.Index-2] && Subgraph_CCI[sc.Index] < Subgraph_CCI[sc.Index-1])

```

```

    || (ZLRHooks[sc.Index-1] < 0 && Subgraph_CCI[sc.Index] <= Subgraph_CCI[sc.Index-1])
  )
  {
    ZLRHooks[sc.Index] = -1;
    if (ZLRHooks [sc.Index] <0 && Subgraph_CCI [sc.Index] <0)
    {
      Subgraph_ZLR[sc.Index] = -200;
      Subgraph_ZLR.DataColor[sc.Index] = Subgraph_ZLR.PrimaryColor;
    }
    else
      Subgraph_ZLR [sc.Index] = 0;
  }
  else if ((Subgraph_CCI[sc.Index-1] < Subgraph_CCI[sc.Index-2] && Subgraph_CCI[sc.Index] >
Subgraph_CCI[sc.Index-1])
    || (ZLRHooks[sc.Index-1] > 0 && Subgraph_CCI[sc.Index] >= Subgraph_CCI[sc.Index-1])
  )
  {
    ZLRHooks[sc.Index] = 1;
    if (ZLRHooks [sc.Index] >0 && Subgraph_CCI [sc.Index] >0)
    {
      Subgraph_ZLR[sc.Index] = 200;
      Subgraph_ZLR.DataColor[sc.Index] = Subgraph_ZLR.SecondaryColor;
    }
    else
      Subgraph_ZLR [sc.Index] = 0;
  }
  else
  {
    Subgraph_ZLR[sc.Index] = 0;
    ZLRHooks[sc.Index] = 0;
  }
}
}

/*=====*/

```

SCSFExport scsf_CCIPredictor(SCStudyInterfaceRef sc)

```

{
  SCSubgraphRef CCIProjHigh = sc.Subgraph[0];
  SCSubgraphRef CCIProjLow = sc.Subgraph[1];
  SCSubgraphRef CCIOutputHigh = sc.Subgraph[2];
  SCSubgraphRef CCIOutputLow = sc.Subgraph[3];

  SCInputRef CCILength = sc.Input[0];

  if (sc.SetDefaults)
  {
    // Set the configuration and defaults

    sc.GraphName = "CCI Predictor";

    sc.AutoLoop = 0;
    sc.GraphRegion = 1;

    CCIProjHigh.Name = "CCI Proj High";
    CCIProjHigh.DrawStyle = DRAWSTYLE_ARROW_DOWN;
    CCIProjHigh.PrimaryColor = RGB(255,255,255);
    CCIProjHigh.LineWidth = 3;
    CCIProjHigh.DrawZeros = false;

    CCIProjLow.Name = "CCI Proj Low";
  }
}

```

```

CCIProjLow.DrawStyle = DRAWSTYLE_ARROW_UP;
CCIProjLow.PrimaryColor = RGB(255,255,255);
CCIProjLow.LineWidth = 3;
CCIProjLow.DrawZeros = false;

CCILength.Name = "CCI Length";
CCILength.SetInt(14);

return;
}

n_ACSIL::s_BarPeriod BarPeriod;
sc.GetBarPeriodParameters(BarPeriod);

for(int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize; BarIndex++)
{
    //CCI High
    CCIProjHigh.Arrays[0][BarIndex] = sc.HLCAvg[BarIndex];

    if (BarIndex == sc.ArraySize - 1 && BarPeriod.AreRangeBars())
    {
        float ProjectedRangeHigh = sc.Low[BarIndex] + BarPeriod.IntradayChartBarPeriodParameter1;
        CCIProjHigh.Arrays[0][BarIndex] = (ProjectedRangeHigh + ProjectedRangeHigh + sc.Low[BarIndex]) / 3;
    }

    sc.CCI(CCIProjHigh.Arrays[0], CCIOutputHigh, BarIndex, CCILength.GetInt(), 0.015f);

    if (BarIndex == sc.ArraySize - 1)
        CCIProjHigh[BarIndex] = CCIOutputHigh[BarIndex];
    else
        CCIProjHigh[BarIndex] = 0;

    //CCI Low
    CCIProjLow.Arrays[0][BarIndex] = sc.HLCAvg[BarIndex];
    if (BarIndex == sc.ArraySize - 1 && BarPeriod.AreRangeBars())
    {
        float ProjectedRangeLow = sc.High[BarIndex] - BarPeriod.IntradayChartBarPeriodParameter1;
        CCIProjLow.Arrays[0][BarIndex] = (ProjectedRangeLow + ProjectedRangeLow + sc.High[BarIndex]) / 3;
    }

    sc.CCI(CCIProjLow.Arrays[0], CCIOutputLow, BarIndex, CCILength.GetInt(), 0.015f);

    if (BarIndex == sc.ArraySize - 1)
        CCIProjLow[BarIndex] = CCIOutputLow[BarIndex];
    else
        CCIProjLow[BarIndex] = 0;
}

}
/*=====*/
/*=====*/
// Candle Pattern Finder
/*=====*/

struct s_CandleStickPatternsFinderSettings
{
    int PriceRangeNumberOfBars;
    double PriceRangeMultiplier;
    int UseTrendDetection;

    s_CandleStickPatternsFinderSettings()

```

```

{
    // default values
    PriceRangeNumberOfBars = 100;
    PriceRangeMultiplier   = 0.01;
    UseTrendDetection = true;
}
};

// Candle dimensions

inline double CandleLength(SCBaseDataRef InData, int index);
inline double BodyLength(SCBaseDataRef InData, int index);
inline double UpperWickLength(SCBaseDataRef InData, int index);
inline double LowerWickLength(SCBaseDataRef InData, int index);

// Helper functions

int DetermineTrendForCandlestickPatterns(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings&
PatternsFinderSettings, int index, int num_of_candles);

inline bool IsBodyStrong(SCBaseDataRef InData, int index);
inline bool IsCandleStrength(SCBaseDataRef InData, int index);

inline bool IsDoji(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);

inline bool IsBodyStrong(SCBaseDataRef InData, int index);
inline bool IsCandleStrength(SCBaseDataRef InData, int index);

inline bool IsWhiteCandle(SCBaseDataRef InData, int index);
inline bool IsBlackCandle(SCBaseDataRef InData, int index);

inline double PercentOfCandleLength(SCBaseDataRef InData, int index, double percent);
inline double PercentOfBodyLength(SCBaseDataRef InData, int index, double percent);

inline bool IsUpperWickSmall(SCBaseDataRef InData, int index, double percent);
inline bool IsLowerWickSmall(SCBaseDataRef InData, int index, double percent);

inline bool IsNearEqual(double value1, double value2, SCBaseDataRef InData, int index, double percent);

// Formations

bool IsHammer(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsHangingMan(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsBullishEngulfing(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsBearishEngulfing(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsBullishEngulfingBodyOnly(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int
index);
bool IsBearishEngulfingBodyOnly(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int
index);
bool IsDarkCloudCover(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsPiercingLine(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsMorningStar(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsEveningStar(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsMorningDojiStar(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsEveningDojiStar(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsBullishAbandonedBaby(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsBearishAbandonedBaby(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsShootingStar(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsInvertedHammer(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsBearishHarami(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsBullishHarami(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsBearishHaramiCross(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsBullishHaramiCross(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsTweezerTop(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsTweezerBottom(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);

```

```

bool IsBearishBeltHoldLine(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsBullishBeltHoldLine(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsTwoCrows(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsThreeBlackCrows(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsBearishCounterattackLine(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsBullishCounterattackLine(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsThreeInsideUp(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsThreeOutsideUp(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsThreeInsideDown(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsThreeOutsideDown(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsKicker(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsKicking(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsThreeWhiteSoldiers(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsAdvanceBlock(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsDeliberation(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsBearishTriStar(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsBullishTriStar(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsUniqueThreeRiverBottom(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsBearishDojiStar(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsBullishDojiStar(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsBearishDragonflyDoji(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsBullishDragonflyDoji(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsBearishGravestoneDoji(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsBullishGravestoneDoji(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsBearishLongleggedDoji(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsBullishLongleggedDoji(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsBearishSideBySideWhiteLines(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsBullishSideBySideWhiteLines(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsFallingThreeMethods(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsRisingThreeMethods(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsBearishSeparatingLines(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsBullishSeparatingLines(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsDownsideTasukiGap(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsUpsideTasukiGap(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsBearishThreeLineStrike(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsBullishThreeLineStrike(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsDownsideGapThreeMethods(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsUpsideGapThreeMethods(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsOnNeck(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsInNeck(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsBearishThrusting(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);
bool IsMatHold(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index);

```

```

static const int CANDLESTICK_TREND_FLAT = 0;
static const int CANDLESTICK_TREND_UP = 1;
static const int CANDLESTICK_TREND_DOWN = -1;
static const int TREND_FOR_PATTERNS = 8;

```

```

/*=====*/

```

```

SCSFExport scsf_CandleStickPatternsFinder(SCStudyInterfaceRef sc)

```

```

{
    SCSubgraphRef Subgraph_Pattern1 = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Pattern2 = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Pattern3 = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Pattern4 = sc.Subgraph[3];
    SCSubgraphRef Subgraph_Pattern5 = sc.Subgraph[4];
    SCSubgraphRef Subgraph_Pattern6 = sc.Subgraph[5];

```



```
SCSubgraphRef Subgraph_TrendUp = sc.Subgraph[6];
SCSubgraphRef Subgraph_TrendDown = sc.Subgraph[7];
SCSubgraphRef Subgraph_TrendForPatterns = sc.Subgraph[TREND_FOR_PATTERNS];
```

```
SCInputRef Input_InputPattern1 = sc.Input[0];
SCInputRef Input_InputPattern2 = sc.Input[1];
SCInputRef Input_InputPattern3 = sc.Input[2];
SCInputRef Input_InputPattern4 = sc.Input[3];
SCInputRef Input_InputPattern5 = sc.Input[4];
SCInputRef Input_InputPattern6 = sc.Input[5];
SCInputRef Input_UseNumericValue = sc.Input[6];
SCInputRef Input_Distance = sc.Input[7];
SCInputRef Input_PriceRangeNumberOfBars = sc.Input[8];
SCInputRef Input_PriceRangeMultiplier = sc.Input[9];
SCInputRef Input_TrendDetection = sc.Input[10];
SCInputRef Input_TrendDetectionLength = sc.Input[11];
SCInputRef Input_DisplayText = sc.Input[12];
SCInputRef Input_DisplayAboveCandle = sc.Input[13];
```

```
if (sc.SetDefaults)
{
    sc.GraphName="CandleStick Patterns Finder";
    sc.StudyDescription="This study looks for candlestick patterns and identifies them on the chart with an abbreviation on the bar where they occur. It uses very advanced logic.";
```

```
    sc.AutoLoop = 1;
    sc.GraphRegion = 0;
```

```
    Subgraph_Pattern1.Name = "Pattern 1";
    Subgraph_Pattern1.DrawStyle = DRAWSTYLE_CUSTOM_TEXT;
    Subgraph_Pattern1.PrimaryColor = RGB(255, 255, 0);
    Subgraph_Pattern1.SecondaryColor = RGB(0, 0, 0);
    Subgraph_Pattern1.SecondaryColorUsed = TRUE;
    Subgraph_Pattern1.LineWidth = 8;
    Subgraph_Pattern1.DrawZeros = false;
```

```
    Subgraph_Pattern2.Name = "Pattern 2";
    Subgraph_Pattern2.DrawStyle = DRAWSTYLE_IGNORE;
    Subgraph_Pattern2.PrimaryColor = RGB(255, 255, 255);
    Subgraph_Pattern2.DrawZeros = false;
```

```
    Subgraph_Pattern3.Name = "Pattern 3";
    Subgraph_Pattern3.DrawStyle = DRAWSTYLE_IGNORE;
    Subgraph_Pattern3.PrimaryColor = RGB(255, 255, 255);
    Subgraph_Pattern3.DrawZeros = false;
```

```
    Subgraph_Pattern4.Name = "Pattern 4";
    Subgraph_Pattern4.DrawStyle = DRAWSTYLE_IGNORE;
    Subgraph_Pattern4.PrimaryColor = RGB(255, 255, 255);
    Subgraph_Pattern4.DrawZeros = false;
```

```
    Subgraph_Pattern5.Name = "Pattern 5";
    Subgraph_Pattern5.DrawStyle = DRAWSTYLE_IGNORE;
    Subgraph_Pattern5.PrimaryColor = RGB(255, 255, 255);
    Subgraph_Pattern5.DrawZeros = false;
```

```
    Subgraph_Pattern6.Name = "Pattern 6";
    Subgraph_Pattern6.DrawStyle = DRAWSTYLE_IGNORE;
    Subgraph_Pattern6.PrimaryColor = RGB(255, 255, 255);
    Subgraph_Pattern6.DrawZeros = false;
```

```
    Subgraph_TrendUp.Name = "Trend Up";
    Subgraph_TrendUp.DrawStyle = DRAWSTYLE_IGNORE;
    Subgraph_TrendUp.PrimaryColor = COLOR_GREEN;
    Subgraph_TrendUp.DrawZeros = false;
```



```

Subgraph_TrendDown.Name = "Trend Down";
Subgraph_TrendDown.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_TrendDown.PrimaryColor = COLOR_RED;
Subgraph_TrendDown.DrawZeros = false;

Input_InputPattern1.Name="Pattern 1";
Input_InputPattern1.SetCandleStickPatternIndex(0);

Input_InputPattern2.Name="Pattern 2";
Input_InputPattern2.SetCandleStickPatternIndex(0);

Input_InputPattern3.Name="Pattern 3";
Input_InputPattern3.SetCandleStickPatternIndex(0);

Input_InputPattern4.Name="Pattern 4";
Input_InputPattern4.SetCandleStickPatternIndex(0);

Input_InputPattern5.Name="Pattern 5";
Input_InputPattern5.SetCandleStickPatternIndex(0);

Input_InputPattern6.Name="Pattern 6";
Input_InputPattern6.SetCandleStickPatternIndex(0);

Input_UseNumericValue.Name="Use Numeric Values Instead of Letter Codes";
Input_UseNumericValue.SetYesNo(0);

Input_Distance.Name="Distance Between the Codes and the Candle as a Percentage";
Input_Distance.SetInt(5);

Input_PriceRangeNumberOfBars.Name="Number of Bars for Price Range Detection (Used for Trend Detection)";
Input_PriceRangeNumberOfBars.SetInt(100);

Input_PriceRangeMultiplier.Name="Price Range Multiplier for Determining Value Per Point";
Input_PriceRangeMultiplier.SetFloat(0.01f);

Input_TrendDetection.Name = "Use Trend Detection";
Input_TrendDetection.SetYesNo(false);

Input_TrendDetectionLength.Name = "Number of Bars Used For Trend Detection";
Input_TrendDetectionLength.SetInt(4);
Input_TrendDetectionLength.SetIntLimits(1,MAX_STUDY_LENGTH);

Input_DisplayText.Name= "Display Text";
Input_DisplayText.SetYesNo( true);

Input_DisplayAboveCandle.Name= "Display Above Candles";
Input_DisplayAboveCandle.SetYesNo(false);

return;
}

// This must always be custom text
Subgraph_Pattern1.DrawStyle = DRAWSTYLE_CUSTOM_TEXT;

// These must always be ignore
Subgraph_Pattern2.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_Pattern3.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_Pattern4.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_Pattern5.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_Pattern6.DrawStyle = DRAWSTYLE_IGNORE;

int & r_DrawingNumberForLastBar = sc.GetPersistentInt(1);
if (sc.IsFullRecalculation)
    r_DrawingNumberForLastBar = 0;

```

```

if (Subgraph_Pattern1.LineWidth <= 1)
{
    Subgraph_Pattern1.LineWidth = 8;
}

const int NumberOfPatternsToDetect = 6;

uint32_t PatternToLookFor[NumberOfPatternsToDetect];
PatternToLookFor[0] = Input_InputPattern1.GetCandleStickPatternIndex();
PatternToLookFor[1] = Input_InputPattern2.GetCandleStickPatternIndex();
PatternToLookFor[2] = Input_InputPattern3.GetCandleStickPatternIndex();
PatternToLookFor[3] = Input_InputPattern4.GetCandleStickPatternIndex();
PatternToLookFor[4] = Input_InputPattern5.GetCandleStickPatternIndex();
PatternToLookFor[5] = Input_InputPattern6.GetCandleStickPatternIndex();

s_CandleStickPatternsFinderSettings PatternsFinderSettings;

if (Input_PriceRangeNumberOfBars.GetInt()>0)
{
    PatternsFinderSettings.PriceRangeNumberOfBars = Input_PriceRangeNumberOfBars.GetInt();
}

if (Input_PriceRangeMultiplier.GetFloat()>0)
{
    PatternsFinderSettings.PriceRangeMultiplier = Input_PriceRangeMultiplier.GetFloat();
}

PatternsFinderSettings.UseTrendDetection = Input_TrendDetection.GetYesNo();

bool DisplayNumericValue = (Input_UseNumericValue.GetYesNo() != 0);

bool PatternDetected = false;
bool PatternCodesToDisplay = false;

int Direction = DetermineTrendForCandlestickPatterns(sc, PatternsFinderSettings,
sc.Index-1,Input_TrendDetectionLength.GetInt());

Subgraph_TrendForPatterns[sc.Index] = static_cast<float>(Direction);

if (Direction == CANDLESTICK_TREND_UP)
{
    Subgraph_TrendUp[sc.Index] = sc.Low[sc.Index];
    Subgraph_TrendDown[sc.Index] = 0;
}
else if (Direction == CANDLESTICK_TREND_DOWN)
{
    Subgraph_TrendUp[sc.Index] = 0;
    Subgraph_TrendDown[sc.Index] = sc.High[sc.Index];
}
else
{
    Subgraph_TrendUp[sc.Index] = 0;
    Subgraph_TrendDown[sc.Index] = 0;
}

SCString DisplayTextString;

for (int Index = 0; Index < NumberOfPatternsToDetect; Index++)
{
    sc.Subgraph[Index][sc.Index] = 0;
}

```

PatternDetected = false;

switch (PatternToLookFor[Index])

{

case 1:

PatternDetected = IsHammer(sc, PatternsFinderSettings, sc.CurrentIndex);

break;

case 2:

PatternDetected = IsHangingMan(sc, PatternsFinderSettings, sc.CurrentIndex);

break;

case 3:

PatternDetected = IsBullishEngulfing(sc, PatternsFinderSettings, sc.CurrentIndex);

break;

case 4:

PatternDetected = IsBearishEngulfing(sc, PatternsFinderSettings, sc.CurrentIndex);

break;

case 5:

PatternDetected = IsDarkCloudCover(sc, PatternsFinderSettings, sc.CurrentIndex);

break;

case 6:

PatternDetected = IsPiercingLine(sc, PatternsFinderSettings, sc.CurrentIndex);

break;

case 7:

PatternDetected = IsMorningStar(sc, PatternsFinderSettings, sc.CurrentIndex);

break;

case 8:

PatternDetected = IsEveningStar(sc, PatternsFinderSettings, sc.CurrentIndex);

break;

case 9:

PatternDetected = IsMorningDojiStar(sc, PatternsFinderSettings, sc.CurrentIndex);

break;

case 10:

PatternDetected = IsEveningDojiStar(sc, PatternsFinderSettings, sc.CurrentIndex);

break;

case 11:

PatternDetected = IsBullishAbandonedBaby(sc, PatternsFinderSettings, sc.CurrentIndex);

break;

case 12:

PatternDetected = IsBearishAbandonedBaby(sc, PatternsFinderSettings, sc.CurrentIndex);

break;

case 13:

PatternDetected = IsShootingStar(sc, PatternsFinderSettings, sc.CurrentIndex);

break;

case 14:

PatternDetected = IsInvertedHammer(sc, PatternsFinderSettings, sc.CurrentIndex);

break;

case 15:

PatternDetected = IsBearishHarami(sc, PatternsFinderSettings, sc.CurrentIndex);

break;

case 16:

PatternDetected = IsBullishHarami(sc, PatternsFinderSettings, sc.CurrentIndex);

break;

case 17:

PatternDetected = IsBearishHaramiCross(sc, PatternsFinderSettings, sc.CurrentIndex);

break;

case 18:

PatternDetected = IsBullishHaramiCross(sc, PatternsFinderSettings, sc.CurrentIndex);

break;

case 19:

PatternDetected = IsTweezerTop(sc, PatternsFinderSettings, sc.CurrentIndex);

break;

case 20:

PatternDetected = IsTweezerBottom(sc, PatternsFinderSettings, sc.CurrentIndex);

break;

case 21:

```

    PatternDetected = IsBearishBeltHoldLine(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 22:
    PatternDetected = IsBullishBeltHoldLine(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 23:
    PatternDetected = IsTwoCrows(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 24:
    PatternDetected = IsThreeBlackCrows(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 25:
    PatternDetected = IsBearishCounterattackLine(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 26:
    PatternDetected = IsBullishCounterattackLine(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 27:
    PatternDetected = IsThreeInsideUp(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 28:
    PatternDetected = IsThreeOutsideUp(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 29:
    PatternDetected = IsThreeInsideDown(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 30:
    PatternDetected = IsThreeOutsideDown(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 31:
    PatternDetected = IsKicker(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 32:
    PatternDetected = IsKicking(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 33:
    PatternDetected = IsThreeWhiteSoldiers(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 34:
    PatternDetected = IsAdvanceBlock(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 35:
    PatternDetected = IsDeliberation(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 36:
    PatternDetected = IsBearishTriStar(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 37:
    PatternDetected = IsBullishTriStar(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 38:
    PatternDetected = IsUniqueThreeRiverBottom(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 39:
    PatternDetected = IsBearishDojiStar(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 40:
    PatternDetected = IsBullishDojiStar(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 41:
    PatternDetected = IsBearishDragonflyDoji(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 42:
    PatternDetected = IsBullishDragonflyDoji(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;

```

```
case 43:
    PatternDetected = IsBearishGravestoneDoji(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 44:
    PatternDetected = IsBullishGravestoneDoji(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 45:
    PatternDetected = IsBearishLongleggedDoji(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 46:
    PatternDetected = IsBullishLongleggedDoji(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 47:
    PatternDetected = IsBearishSideBySideWhiteLines(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 48:
    PatternDetected = IsBullishSideBySideWhiteLines(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 49:
    PatternDetected = IsFallingThreeMethods(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 50:
    PatternDetected = IsRisingThreeMethods(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 51:
    PatternDetected = IsBearishSeparatingLines(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 52:
    PatternDetected = IsBullishSeparatingLines(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 53:
    PatternDetected = IsDownsideTasukiGap(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 54:
    PatternDetected = IsUpsideTasukiGap(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 55:
    PatternDetected = IsBearishThreeLineStrike(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 56:
    PatternDetected = IsBullishThreeLineStrike(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 57:
    PatternDetected = IsDownsideGapThreeMethods(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 58:
    PatternDetected = IsUpsideGapThreeMethods(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 59:
    PatternDetected = IsOnNeck(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 60:
    PatternDetected = IsInNeck(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 61:
    PatternDetected = IsBearishThrusting(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 62:
    PatternDetected = IsMatHold(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 63:
    PatternDetected = IsDoji(sc, PatternsFinderSettings, sc.CurrentIndex);
    break;
case 64:
    PatternDetected = IsBullishEngulfingBodyOnly(sc, PatternsFinderSettings, sc.CurrentIndex);
```

```

        break;
    case 65:
        PatternDetected = IsBearishEngulfingBodyOnly(sc, PatternsFinderSettings, sc.CurrentIndex);
        break;
    }

    if (PatternDetected)
    {
        sc.Subgraph[Index][sc.CurrentIndex] = static_cast<float>(PatternToLookFor[Index]);

        if (Input_DisplayText.GetYesNo() && !sc.HideStudy)
        {
            SCString DisplayTextStringCopy = DisplayTextString;

            PatternCodesToDisplay = true;

            if (DisplayNumericValue)
                DisplayTextString.Format("%s%.2d%s", DisplayTextStringCopy.GetChars(), PatternToLookFor[Index],
"\r\n");
            else
                DisplayTextString.Format("%s%s%s", DisplayTextStringCopy.GetChars(),
CandleStickPatternNames[PatternToLookFor[Index]][1], "\r\n");
        }
    }
}

if (PatternCodesToDisplay && sc.Index < sc.ArraySize - 1)
{
    s_UseTool Tool;

    Tool.ChartNumber = sc.ChartNumber;
    Tool.DrawingType = DRAWING_TEXT;
    // Tool.LineNumber;
    Tool.BeginIndex = sc.CurrentIndex;
    Tool.Region = sc.GraphRegion;

    if (Input_DisplayAboveCandle.GetYesNo())
    {
        Tool.BeginValue = sc.High[sc.Index] + ( (sc.High[sc.Index] - sc.Low[sc.Index]) * Input_Distance.GetInt() * 0.01f);
        Tool.TextAlignment = DT_CENTER | DT_BOTTOM;
    }
    else
    {
        Tool.BeginValue = sc.Low[sc.Index] - ( (sc.High[sc.Index] - sc.Low[sc.Index]) * Input_Distance.GetInt() * 0.01f);
        Tool.TextAlignment = DT_CENTER | DT_TOP;
    }

    Tool.Color = Subgraph_Pattern1.PrimaryColor;
    Tool.FontBackColor = Subgraph_Pattern1.SecondaryColor;
    Tool.FontSize = Subgraph_Pattern1.LineWidth;
    Tool.FontBold = TRUE;
    Tool.Text.Format("%s", DisplayTextString.GetChars());
    Tool.AddMethod = UTAM_ADD_ALWAYS;

    sc.UseTool(Tool);
}
else if (PatternCodesToDisplay && sc.Index == sc.ArraySize - 1) //last bar in chart
{
    s_UseTool Tool;

    Tool.ChartNumber = sc.ChartNumber;
    Tool.DrawingType = DRAWING_TEXT;
    if (r_DrawingNumberForLastBar != 0)
        Tool.LineNumber = r_DrawingNumberForLastBar;
}

```

```

Tool.BeginIndex = sc.CurrentIndex;
Tool.Region = sc.GraphRegion;

if (Input_DisplayAboveCandle.GetYesNo())
{
    Tool.BeginValue = sc.High[sc.Index] + ( (sc.High[sc.Index] - sc.Low[sc.Index]) * Input_Distance.GetInt() * 0.01f);
    Tool.TextAlignment = DT_CENTER | DT_BOTTOM;
}
else
{
    Tool.BeginValue = sc.Low[sc.Index] - ( (sc.High[sc.Index] - sc.Low[sc.Index]) * Input_Distance.GetInt() * 0.01f);
    Tool.TextAlignment = DT_CENTER | DT_TOP;
}

Tool.Color = Subgraph_Pattern1.PrimaryColor;
Tool.FontBackColor = Subgraph_Pattern1.SecondaryColor;
Tool.FontSize = Subgraph_Pattern1.LineWidth;
Tool.FontBold = TRUE;

if(DisplayTextString.GetLength() == 0)
    DisplayTextString = " ";

Tool.Text.Format("%s",DisplayTextString.GetChars());

Tool.AddMethod = UTAM_ADD_OR_ADJUST;

sc.UseTool(Tool);

r_DrawingNumberForLastBar = Tool.LineNumber;
}

return;
}

/*=====*/
inline double UpperWickLength(SCBaseDataRef InData, int index)
{
    double upperBoundary = max(InData[SC_LAST][index], InData[SC_OPEN][index]);

    double upperWickLength = InData[SC_HIGH][index] - upperBoundary;

    return upperWickLength;
}

/*=====*/
inline double LowerWickLength(SCBaseDataRef InData, int index)
{
    double lowerBoundary = min(InData[SC_LAST][index], InData[SC_OPEN][index]);

    double lowerWickLength = lowerBoundary - InData[SC_LOW][index];

    return lowerWickLength;
}

/*=====*/
inline double CandleLength(SCBaseDataRef InData, int index)
{
    return InData[SC_HIGH][index]-InData[SC_LOW][index];
}

/*=====*/
inline double BodyLength(SCBaseDataRef InData, int index)
{
    return fabs(InData[SC_OPEN][index] - InData[SC_LAST][index]);
}

```



```

}

/*=====*/
inline double PercentOfCandleLength(SCBaseDataRef InData, int index, double percent)
{
    return CandleLength(InData, index) * (percent / 100.0);
}

/*=====*/
inline double PercentOfBodyLength(SCBaseDataRef InData, int index, double percent)
{
    return BodyLength(InData, index) * percent / 100.0;
}

/*=====*/
const int k_Body_NUM_OF_CANDLES = 5;           // number of previous candles to calculate body strength

inline bool IsBodyStrong(SCBaseDataRef InData, int index)
{
    // detecting whether candle has a strong body or not
    bool ret_flag = false;

    // calculating average length of bodies of last NUM_OF_CANDLES
    float mov_aver = 0;
    for (int i = 1; i < k_Body_NUM_OF_CANDLES + 1; i++)
    {
        mov_aver += static_cast<float>(BodyLength(InData, index - i));
    }
    mov_aver /= k_Body_NUM_OF_CANDLES;

    // compare with length of current body with average length.
    if(BodyLength(InData, index) > mov_aver)
    {
        ret_flag = true;
    }
    return ret_flag;
}

/*=====*/
const int k_Candle_NUM_OF_CANDLES = 5;           // number of previous candles to calculate candle strength
const double k_CandleStrength_Multiplier = 1.0;

inline bool IsCandleStrength(SCBaseDataRef InData, int index)
{
    // detecting whether candle has a strong body or not
    bool ret_flag = false;

    // calculating average length of bodies of last NUM_OF_CANDLES
    float mov_aver = 0;
    for (int i = 1; i < k_Candle_NUM_OF_CANDLES + 1; i++)
        mov_aver += static_cast<float>(CandleLength(InData, index - i));

    mov_aver /= k_Candle_NUM_OF_CANDLES;

    // compare with length of body with average
    if(CandleLength(InData, index) > mov_aver * k_CandleStrength_Multiplier)
    {
        ret_flag = true;
    }
    return ret_flag;
}

/*=====*/

```

```

int DetermineTrendForCandlestickPatterns(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings&
PatternsFinderSettings, int index, int num_of_candles)
{
    // detecting trend
    // index: index of the last candle in trend
    // num_of_candles: minimum number of candles required

    const double UptrendPercent = 1.0;
    const double DowntrendPercent = -1.0;

    // Using linear regression to estimate the slope
    SCBaseDataRef InData = sc.BaseData;
    double sumx = 0.0;
    double sumy = 0.0;
    double sumxx = 0.0;
    double sumyy = 0.0;
    double sumxy = 0.0;

    for (int IndexOffset=1; IndexOffset<=num_of_candles;IndexOffset++)
    {
        double value = InData[SC_HL_AVG][index-(IndexOffset-1)];

        sumx += -IndexOffset;
        sumy += value;
        sumxx += IndexOffset*IndexOffset;
        sumyy += value*value;
        sumxy += -IndexOffset*value;
    }

    double n = double(num_of_candles);
    double Sxy = n*sumxy-sumx*sumy;
    double Sxx = n*sumxx-sumx*sumx;

    double Slope = Sxy/Sxx; // slope value

    // estimate the value per point:

    double high=sc.GetHighest(InData[SC_HIGH], index, PatternsFinderSettings.PriceRangeNumberOfBars);
    double low=sc.GetLowest(InData[SC_LOW], index, PatternsFinderSettings.PriceRangeNumberOfBars);

    double range = high - low;

    double valuePerPoint = range * PatternsFinderSettings.PriceRangeMultiplier;

    // detect trend

    double CorrectSlope = Slope / valuePerPoint;

    if (CorrectSlope>UptrendPercent)
    {
        return CANDLESTICK_TREND_UP;
    }
    else if (CorrectSlope<DowntrendPercent)
    {
        return CANDLESTICK_TREND_DOWN;
    }
    else
    {
        return CANDLESTICK_TREND_FLAT;
    }
}

```

```

}

/*=====*/

inline bool IsDoji(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    const double Doji_BodyPercent = 5.0;
    SCBaseDataRef InData = sc.BaseData;
    if (BodyLength(InData, index) <= PercentOfCandleLength(InData, index, Doji_BodyPercent))
    {
        return true;
    }

    return false;
}

/*=====*/
inline bool IsWhiteCandle(SCBaseDataRef InData, int index)
{
    return InData[SC_LAST][index]>InData[SC_OPEN][index];
}

/*=====*/
inline bool IsBlackCandle(SCBaseDataRef InData, int index)
{
    return InData[SC_LAST][index]<InData[SC_OPEN][index];
}

/*=====*/
inline bool IsNearEqual(double value1, double value2, SCBaseDataRef InData, int index, double percent)
{
    return abs(value1 - value2) < PercentOfCandleLength(InData, index, percent);
}

/*=====*/
inline bool IsUpperWickSmall(SCBaseDataRef InData, int index, double percent)
{
    return UpperWickLength(InData, index) < PercentOfCandleLength(InData, index, percent);
}

/*=====*/
inline bool IsLowerWickSmall(SCBaseDataRef InData, int index, double percent)
{
    return LowerWickLength(InData, index) < PercentOfCandleLength(InData, index, percent);
}

/*=====*/

bool IsHammer(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    const double LowerWickPercent = 200.0;
    const double UpperWickPercent = 7.0;

    // HAMMER
    // 1. Downtrend
    // 2. Body [index] of the candle in the upper part of the candle
    // 3. Lower wick [index] of candle is 2 longer then candle body
    // 4. Upper wick [index] of candle is either absent or small
    SCBaseDataRef InData = sc.BaseData;

```

```

bool ret_flag = false;

if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_DOWN) // downtrend
{
    // check that body is in the upper part of the candle
    if (((InData[SC_HIGH][index]-InData[SC_OPEN][index])<(InData[SC_OPEN][index]-InData[SC_LOW][index]))
        &&((InData[SC_HIGH][index]-InData[SC_LAST][index])<(InData[SC_LAST][index]-InData[SC_LOW][index]))
        && (InData[SC_OPEN][index] != InData[SC_LAST][index]))
    {

        if (// check of the length of the lower wick
            IsLowerWickSmall(InData, index, LowerWickPercent) &&
            // check of the length of the upper wick
            IsUpperWickSmall(InData, index, UpperWickPercent)
        )
        {
            ret_flag = true;
        }
    }
}
return ret_flag;
}

```

```

/*=====*/

```

```

const double k_HangingMan_LowerWickPercent = 200.0;
const double k_HangingMan_UpperWickPercent = 7.0;

```

```

bool IsHangingMan(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // HANGING MAN
    // 1. Uptrend
    // 2. Body [index] is in the upper part of the candle
    // 3. Lower wick [index] is 2 times longer then the body
    // 4. Upper wick [index] is either absent or small
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_UP) // check uptrend
    {
        // is body in the upper part of the candle
        if (((InData[SC_HIGH][index]-InData[SC_OPEN][index])<(InData[SC_OPEN][index]-InData[SC_LOW][index]))
            &&((InData[SC_HIGH][index]-InData[SC_LAST][index])<(InData[SC_LAST][index]-InData[SC_LOW][index]))
            && (InData[SC_OPEN][index] != InData[SC_LAST][index]))
        {
            // check the length of lower wick
            if(IsLowerWickSmall(InData, index, k_HangingMan_LowerWickPercent))
            {
                // check the length of the upper wick
                if(IsUpperWickSmall(InData, index, k_HangingMan_UpperWickPercent))
                {
                    ret_flag = true;
                }
            }
        }
    }
    return ret_flag;
}

```

```

/*=====*/

```

```

// Reference: http://www.investopedia.com/terms/b/bullishengulfingpattern.asp

```

```

bool IsBullishEngulfing(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{

```

```

SCBaseDataRef InData = sc.BaseData;
bool ret_flag = false;
if(settings.UseTrendDetection == false
    || sc.Subgraph[TREND_FOR_PATTERNS][index] == CANDLESTICK_TREND_DOWN)    // down
{
    if(InData[SC_LAST][index-1] < InData[SC_OPEN][index-1]
    && InData[SC_LAST][index] > InData[SC_OPEN][index])
    {
        if ((InData[SC_HIGH][index] > InData[SC_HIGH][index - 1]) &&
            (InData[SC_LOW][index] < InData[SC_LOW][index - 1]) &&
            (InData[SC_LAST][index] > InData[SC_OPEN][index - 1]) &&
            (InData[SC_OPEN][index] < InData[SC_LAST][index - 1]))
        {
            ret_flag = true;
        }
    }
}
return ret_flag;
}

/*=====*/
// Reference: http://www.investopedia.com/terms/b/bearishengulfingp.asp
bool IsBearishEngulfing(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false
        || sc.Subgraph[TREND_FOR_PATTERNS][index] == CANDLESTICK_TREND_UP) // up
    {
        if(InData[SC_LAST][index-1] > InData[SC_OPEN][index-1]
        && InData[SC_LAST][index] < InData[SC_OPEN][index])
        {
            if ((InData[SC_HIGH][index] > InData[SC_HIGH][index - 1]) &&
                (InData[SC_LOW][index] < InData[SC_LOW][index - 1]) &&
                (InData[SC_OPEN][index] > InData[SC_LAST][index - 1]) &&
                (InData[SC_LAST][index] < InData[SC_OPEN][index - 1]))
            {
                ret_flag = true;
            }
        }
    }
    return ret_flag;
}

/*=====*/
bool IsBullishEngulfingBodyOnly(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if (settings.UseTrendDetection == false
        || sc.Subgraph[TREND_FOR_PATTERNS][index] == CANDLESTICK_TREND_DOWN)    // down
    {
        if (InData[SC_LAST][index - 1] < InData[SC_OPEN][index - 1]
            && InData[SC_LAST][index] > InData[SC_OPEN][index])
        {
            if ((InData[SC_LAST][index] > InData[SC_OPEN][index - 1]) &&
                (InData[SC_OPEN][index] < InData[SC_LAST][index - 1]))
            {
                ret_flag = true;
            }
        }
    }
    return ret_flag;
}

/*=====*/

```

```

bool IsBearishEngulfingBodyOnly(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int
index)
{
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if (settings.UseTrendDetection == false
        || sc.Subgraph[TREND_FOR_PATTERNS][index] == CANDLESTICK_TREND_UP) // up
    {
        if (InData[SC_LAST][index - 1] > InData[SC_OPEN][index - 1]
            && InData[SC_LAST][index] < InData[SC_OPEN][index])
        {
            if ((InData[SC_OPEN][index] > InData[SC_LAST][index - 1]) &&
                (InData[SC_LAST][index] < InData[SC_OPEN][index - 1]))
            {
                ret_flag = true;
            }
        }
    }
    return ret_flag;
}
/*=====*/
const double k_DarkCloudCover_OverlapPercent = 50.0;

```

```

bool IsDarkCloudCover(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // DARK CLOUD COVER
    // 1. Uptrend
    // 2. Strong white body [index-1]
    // 3. Open [index] is higher then High of candle [index-1]
    // 4. Close [index] is lower then OVERLAP_PERCENT of white body of candle [index-1]

    /*1. Market is characterized by an uptrend.
    2. We see a long white candlestick in the first day.
    3. Then we see a black body characterized by an open above the high of the previous day on the second day.
    4. The second black candlestick closes within and below the midpoint of the previous white body.*/

    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(
        (settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_UP) // uptrend
        && IsWhiteCandle(InData, index-1) // check for white body
        && IsBodyStrong(InData, index-1)
        && IsBlackCandle(InData, index) // 0 candle is black
        && (InData[SC_OPEN][index] > InData[SC_HIGH][index-1]) // open 0 is higher then high of -1
        && (InData[SC_LAST][index] < (InData[SC_LAST][index-1] - PercentOfBodyLength(InData, index-1,
k_DarkCloudCover_OverlapPercent))) // check for the overlap between white body and black OVERLAP_PERCENT
    )
    {
        ret_flag = true;
    }

    return ret_flag;
}
/*=====*/
const double k_PiercingLine_OverlapPercent = 50.0;

```

```

bool IsPiercingLine(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // PIERCING LINE
    // 1. Downtrend
    // 2. Strong black body [index-1]

```

```

// 3. Open [index] less than low of candle [index-1]
// 4. Close [index] is higher than white body of candle [index-1] by OverlapPercent
SCBaseDataRef InData = sc.BaseData;
bool ret_flag = false;
if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_DOWN) // downtrend
{
    if (IsBlackCandle(InData, index-1)) // check for black body, -1th candle is black
    {
        if (IsWhiteCandle(InData, index) && // 0th candle is white
            (InData[SC_OPEN][index]<InData[SC_LOW][index-1])) // open of 0 the below low of -1th
        {
            // check for the overlap between black body and white by OverlapPercent
            if (InData[SC_LAST][index]>(InData[SC_OPEN][index-1]-PercentOfBodyLength(InData, index-1,
k_PiercingLine_OverlapPercent)))
            {
                if(IsBodyStrong(InData,index-1)) // strong body of -1th candle
                {
                    ret_flag = true;
                }
            }
        }
    }
}
return ret_flag;
}

/*=====*/
const double k_MorningStar_OverlapPercent = 50.0;

bool IsMorningStar(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // MORNING STAR
    // 1. Downtrend
    // 2. Strong black body of candle [index-2]
    // 3. Any weak body of candle [index-1], but not Doji
    // 4. Strong white body of candle [index] overlap more then OVERLAP_PERCENT of the body of candle [index-2]

    /*1. Market is characterized by downtrend.
    2. We see a long black candlestick in the first day.
    3. Then we see a small body on the second day gapping in the direction of the previous downtrend.
    4. Finally we see a white candlestick on the third day*/
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(
        (settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_DOWN)
        && IsBlackCandle(InData, index-2)
        && IsBodyStrong(InData,index-2)
        && (InData[SC_OPEN][index-1] < InData[SC_LOW][index-2])// body of -1 candle below low of -2 candle
        && (InData[SC_LAST][index-1] < InData[SC_LOW][index-2])
        //&& !IsDoji(sc, settings, index-1) // -1 candle is not Doji
        && !IsBodyStrong(InData,index-1) // body of -1 candle is weak
        && IsWhiteCandle(InData, index)
        && IsBodyStrong(InData,index)
        && (InData[SC_OPEN][index]<InData[SC_LAST][index-2]) // body of 0 candle starts below the body of -2 candle
        && (InData[SC_LAST][index]>(InData[SC_OPEN][index-2]-PercentOfBodyLength(InData, index,
k_MorningStar_OverlapPercent))) // body of the candle 0 overlaps the body of the candle -2
    )
    {
        ret_flag = true;
    }
    return ret_flag;
}

```



```

/*=====*/
const double k_EveningStar_OverlapPercent = 50.0;

bool IsEveningStar(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // EVENING STAR
    // 1. Uptrend
    // 2. Strong white body of candle [index-2]
    // 3. Any weak body of candle [index-1], but not Doji
    // 4. Strong black body of candle [index] overlaps more then OVERLAP_PERCENT of body of candle [index-2]. -I have
not found any info about overlapping
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_UP) // check for uptrend
    {
        if(IsWhiteCandle(InData, index-2)) // body of candle -2 is white
        {
            if(IsBodyStrong(InData, index-2)) // body of candle -2 is strong
            {
                if ((InData[SC_OPEN][index-1] > InData[SC_HIGH][index-2]) && // body of candle -1 higher then maximum. -
This line does not correspond to what I found online it should not matter, the gap should be between the bodies.
                (InData[SC_LAST][index-1] > InData[SC_LOW][index-2])
                //&& !IsDoji(sc, settings, index-1) // candle -1 is not Doji
                )
                {
                    if(!IsBodyStrong(InData, index-1)) // body of the candle -1 is weak
                    {
                        if (IsBlackCandle(InData, index) && // body of the candle 0 is black
                        (InData[SC_OPEN][index] > InData[SC_LAST][index-2]) && // body of the candle 0 starts higher then
body of candle -2
                        (InData[SC_LAST][index] < (InData[SC_LAST][index-2] - PercentOfBodyLength(InData, index-2,
k_EveningStar_OverlapPercent)))) // body of candle 0 overlaps body of candle -2. Did not find written reference to this,
but all images support this.
                        {
                            if (IsBodyStrong(InData, index)) // body of candle 0 is strong
                            {
                                ret_flag = true;
                            }
                        }
                    }
                }
            }
        }
    }
}

return ret_flag;
}

```

```

/*=====*/
const double k_MorningDojiStar_OverlapPercent = 50.0;

bool IsMorningDojiStar(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // MORNING DOJI STAR
    // 1. Downtrend
    // 2. Strong black body of candle [index-2]
    // 3. [index-1] candle is Doji. (2nd day is a doji which gaps below the 1st day's close.)
    // 4. Strong white body of candle [index] overlap more then OVERLAP_PERCENT of body of the candle [index-2]
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(
        (settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_DOWN) // check for downtrend
    )
    {
        if(IsBlackCandle(InData, index-2)) // body of candle -2 is black
        {
            if(IsDoji(InData, index-1)) // candle -1 is Doji
            {
                if(IsWhiteCandle(InData, index)) // body of candle 0 is white
                {
                    if (InData[SC_OPEN][index] < InData[SC_LAST][index-2] && // body of candle 0 starts lower then
body of candle -2
                    (InData[SC_LAST][index] > (InData[SC_LAST][index-2] - PercentOfBodyLength(InData, index-2,
k_MorningDojiStar_OverlapPercent)))) // body of candle 0 overlaps body of candle -2. Did not find written reference to this,
but all images support this.
                    {
                        ret_flag = true;
                    }
                }
            }
        }
    }
}

```

```

    && IsBlackCandle(InData,index-2) // body of the candle -2 is black
    && IsBodyStrong(InData,index-2) // body of the candle -2 is strong
    && (InData[SC_HIGH][index-1] < InData[SC_LAST][index-2])// the -1 candle is below the close of candle -2
    && IsDoji(sc, settings, index-1) // -1 candle is Doji
    && IsWhiteCandle(InData,index) // body of the candle 0 is white
    && (InData[SC_OPEN][index]<InData[SC_LAST][index-2]) // body of the candle 0 starts below the body of candle -2
    && (InData[SC_LAST][index]>(InData[SC_OPEN][index-2]-PercentOfBodyLength(InData, index-2,
k_MorningDojiStar_OverlapPercent))))// body of the candle 0 overlaps body of the candle -2
    && IsBodyStrong(InData,index) // body of the candle 0 is strong
    )
    {
        ret_flag = true;
    }
    return ret_flag;
}

```

```

/*=====*/
const double k_EveningDojiStar_OverlapPercent = 50.0;

```

```

bool IsEveningDojiStar(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // EVENING DOJI STAR
    // 1. Uptrend
    // 2. Strong white body of candle [index-2]
    // 3. candle [index-1] 2nd day is a doji which gaps above the [index-2] 1st day's close.
    // 4. Strong black body of candle [index]
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(
        (settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_UP) // check for uptrend
        && IsWhiteCandle(InData, index-2) // body of the -2 candle is white
        && IsBodyStrong(InData,index-2) // body of th -2 candle is strong
        && InData[SC_OPEN][index-1] > InData[SC_HIGH][index-2] // body of the -1 candle is higher then maximum of -2
candle
        && IsDoji(sc, settings, index-1) // -1 candle is Doji
        && IsBlackCandle(InData, index) // body of the 0 candle is black
        && InData[SC_OPEN][index]>InData[SC_LAST][index-2] // body of the 0 candle starts higher then body of -2 candle
        && (InData[SC_LAST][index]<(InData[SC_LAST][index-2]-PercentOfBodyLength(InData, index-2,
k_EveningDojiStar_OverlapPercent)))) // body of the 0 candle overlaps the body of -2 candle //This is not a requirement
        && IsBodyStrong(InData,index) // body of candle 0 is strong
        )
    {
        ret_flag = true;
    }
    return ret_flag;
}

```

```

/*=====*/
const double k_BullishAbandonedBaby_OverlapPercent = 50.0;

```

```

bool IsBullishAbandonedBaby(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // BULLISH ABANDONED BABY
    // 1. Downtrend
    // 2. Strong black body of candle [index-2]
    // 3. Candle [index-1] - Doji
    // 4. Strong white body of candle [index] overlaps more then OVERLAP_PERCENT of the body of candle [index-2]. Is
not included in web descriptions but is logical.
    // 5. Gap between -1 and 0 candles, -1 and -2 candles
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_DOWN) // check for downtrend
    {

```

```

    if(IsBlackCandle(InData, index-2)) // body of candle -2 is black
    {
        if(IsBodyStrong(InData,index-2)) // body of the candle -2 is strong
        {
            if ((InData[SC_OPEN][index-1] < InData[SC_LOW][index-2]) && // body of the candle -1 is below the minimum
of candle -2
                IsDoji(sc, settings, index-1)) // -1 candle is Doji
            {
                if (IsWhiteCandle(InData, index) && // body of candle 0 is white
(InData[SC_OPEN][index]<InData[SC_LAST][index-2]) &&// body of candle 0 starts below of the body of
the candle -2
                    (InData[SC_LAST][index]>(InData[SC_OPEN][index-2]-PercentOfBodyLength(InData, index-2,
k_BullishAbandonedBaby_OverlapPercent)))) // body of candle 0 overlaps with body of candle -2
                {
                    if (IsBodyStrong(InData,index)) // body of candle 0 is strong
                    {
                        if ((InData[SC_LOW][index-2]>InData[SC_HIGH][index-1]) && // gap between candle -2 and -1
(InData[SC_LOW][index]>InData[SC_HIGH][index-1])) // gap between candle -0 and -1
                        {
                            ret_flag = true;
                        }
                    }
                }
            }
        }
    }
}
return ret_flag;
}

/*=====*/
const double k_BearishAbandonedBaby_OverlapPercent = 50.0;

bool IsBearishAbandonedBaby(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // BEARISH ABANDONED BABY
    // 1. Uptrend
    // 2. Strong white body of candle [index-2]
    // 3. Weak body of candle [index-1]
    // 4. Candle [index-1] is Doji
    // 5. Strong black body of candle [index] overlaps more then OVERLAP_PERCENT of body of candle [index-2]
    // 6. Gap between candle -1 and 0, -1 and 2 candles
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_UP) // check for uptrend
    {
        if(IsWhiteCandle(InData, index-2)) // body of candle -2 is white
        {
            if(IsBodyStrong(InData,index-2)) // body of candle -2 is strong
            {
                if ((InData[SC_OPEN][index-1] > InData[SC_HIGH][index-2]) && // body of candle -1 is higher than maximum
of candle -2
                    IsDoji(sc, settings, index-1)) // candle -1 is Doji
                {
                    if (IsBlackCandle(InData, index) && // body candle 0 is black
(InData[SC_OPEN][index]>InData[SC_LAST][index-2]) &&// body of candle 0 starts higher then body of
candle -2
                        (InData[SC_LAST][index]<(InData[SC_LAST][index-2]-PercentOfBodyLength(InData, index-2,
k_BearishAbandonedBaby_OverlapPercent)))) // body of candle 0 overlaps body of candle -2
                    {
                        if (IsBodyStrong(InData,index)) // body of candle 0 is strong
                        {
                            if ((InData[SC_HIGH][index-2]<InData[SC_LOW][index-1]) && // gap between -2 and -1 candle
(InData[SC_HIGH][index]<InData[SC_LOW][index-1])) // gap between -0 and -1 candle

```

```

        {
            ret_flag = true;
        }
    }
}
}
}
}
return ret_flag;
}

/*=====*/
const double k_ShootingStar_LowerWickPercent = 7.0;
const double k_ShootingStar_UpperWickPercent = 300.0;

bool IsShootingStar(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // SHOOTING STAR
    // 1. Uptrend
    // 2. Small body of candle [index] is in the lower part of candle
    // 3. Upper wick of the candle [index] is at least three times as long as the body
    // 4. Lower wick [index] of the candle is either absent or small
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_UP) // uptrend
    {
        if (!IsBodyStrong(InData,index)) // weak body of the candle 0
        {
            // check that the body of the candle is in the lower part of the candle
            if (((InData[SC_HIGH][index]-InData[SC_OPEN][index])>(InData[SC_OPEN][index]-InData[SC_LOW][index]))
                &&((InData[SC_HIGH][index]-InData[SC_LAST][index])>(InData[SC_LAST][index]-InData[SC_LOW][index]))
            )
            {
                // Upper wick is 3 times longer then the body
                if(UpperWickLength(InData, index) >= PercentOfBodyLength(InData, index,
k_ShootingStar_UpperWickPercent))
                {
                    // Lower wick is insignificant
                    if(IsLowerWickSmall(InData, index, k_ShootingStar_LowerWickPercent))
                    {
                        ret_flag = true;
                    }
                }
            }
        }
    }
    return ret_flag;
}

/*=====*/
const double k_InvertedHammer_LowerWickPercent = 7.0;
const double k_InvertedHammer_UpperWickPercent = 200.0;

bool IsInvertedHammer(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // INVERTED HAMMER
    // 1. Downtrend
    // 2. Small body of candle [index] is in the lower part of the candle
    // 3. The upper shadow is no more than two times as long as the body
    // 4. Lower wick [index] of the candle is either absent or small
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==

```

```

CANDLESTICK_TREND_DOWN) // downtrend
{
    if (!IsBodyStrong(InData,index)) // candle 0 has weak body
    {
        // check that body is in the lower part of the candle
        if (((InData[SC_HIGH][index]-InData[SC_OPEN][index])>(InData[SC_OPEN][index]-InData[SC_LOW][index]))
            &&((InData[SC_HIGH][index]-InData[SC_LAST][index])>(InData[SC_LAST][index]-InData[SC_LOW][index]))
            && !IsDoji(sc, settings, index)) // candle is not Doji
        {
            // The upper shadow is no more than two times as long as the body.
            if(UpperWickLength(InData, index)<=PercentOfBodyLength(InData, index,
k_InvertedHammer_UpperWickPercent))
            {
                // lower wick is insignificant
                if(IsLowerWickSmall(InData, index, k_InvertedHammer_LowerWickPercent))
                {
                    ret_flag = true;
                }
            }
        }
    }
}
return ret_flag;
}
/*=====*/

bool IsBearishHarami(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // BEARISH HARAMI
    // 1. Uptrend
    // 2. Strong body of candle [index-1]
    // 3. Weak body of candle [index] does not extend beyond the limits of the body of candle [index-1]
    // 4. Candle [index] is not Doji
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_UP) // Uptrend
    {
        if (IsBodyStrong(InData,index-1)) // Strong body of candle -1
        {
            if(!IsDoji(sc, settings, index)) // candle 0 is not Doji
            {
                if(!IsBodyStrong(InData,index)) // weak body of candle 0
                {
                    if(InData[SC_OPEN][index-1] > InData[SC_LAST][index-1]) // part of a check that one body does not extend
beyond the limits of the other body
                    {
                        if(InData[SC_OPEN][index] > InData[SC_LAST][index]) // part of a check that one body does not extend
beyond the limits of the other body
                        {
                            // body of the candle 0 does not extend beyond the limits of the body of candle -1
                            if ((InData[SC_OPEN][index-1]>=InData[SC_OPEN][index]) &&
                                (InData[SC_LAST][index-1]<=InData[SC_LAST][index]))
                            {
                                ret_flag = true;
                            }
                        }
                    }
                }
            }
        }
        else // part of a check that one body does not extend beyond the limits of the other body
        {
            // body of the candle 0 does not extend beyond the limits of the body of candle -1
            if ((InData[SC_OPEN][index-1]>=InData[SC_LAST][index]) &&
                (InData[SC_LAST][index-1]<=InData[SC_OPEN][index]))
            {
                ret_flag = true;
            }
        }
    }
}

```

```

    }
}
else // part of a check that one body does not extend beyond the limits of the other body
{
    if(InData[SC_OPEN][index] > InData[SC_LAST][index]) // part of a check that one body does not
extend beyond the limits of the other body
    {
        // body of the candle 0 does not extend beyond the limits of the body of candle -1
        if ((InData[SC_LAST][index-1]>=InData[SC_OPEN][index]) &&
            (InData[SC_OPEN][index-1]<=InData[SC_LAST][index]))
        {
            ret_flag = true;
        }
    }
else // part of a check that one body does not extend beyond the limits of the other body
{
    // body of the candle 0 does not extend beyond the limits of the body of candle -1
    if ((InData[SC_LAST][index-1]>=InData[SC_LAST][index]) &&
        (InData[SC_OPEN][index-1]<=InData[SC_OPEN][index]))
    {
        ret_flag = true;
    }
}
}
}
}
}
}
return ret_flag;
}
/*=====*/

```

```

bool IsBullishHarami(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // BULLISH HARAMI
    // 1. Downtrend
    // 2. Strong body of candle [index-1]
    // 3. Weak body of candle [index] does not extend beyond the limits of the body of candle [index-1]
    // 4. Candle [index] is not Doji
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_DOWN) // downtrend
    {
        if (IsBodyStrong(InData,index-1)) // Strong body of candle [index-1]
        {
            if(!IsDoji(sc, settings, index)) // candle 0 is not Doji
            {
                if(!IsBodyStrong(InData,index)) // weak body of candle 0
                {
                    if(InData[SC_OPEN][index-1] > InData[SC_LAST][index-1]) // part of a check that one body does not extend
beyond the limits of the other body
                    {
                        if(InData[SC_OPEN][index] > InData[SC_LAST][index]) // part of a check that one body does not extend
beyond the limits of the other body
                        {
                            // body of the candle 0 does not extend beyond the limits of the body of candle -1
                            if ((InData[SC_OPEN][index-1]>=InData[SC_OPEN][index]) &&
                                (InData[SC_LAST][index-1]<=InData[SC_LAST][index]))
                            {
                                ret_flag = true;
                            }
                        }
                    }
else // part of a check that one body does not extend beyond the limits of the other body
{

```

```

        // body of the candle 0 does not extend beyond the limits of the body of candle -1
        if ((InData[SC_OPEN][index-1]>=InData[SC_LAST][index]) &&
            (InData[SC_LAST][index-1]<=InData[SC_OPEN][index]))
        {
            ret_flag = true;
        }
    }
}
else // part of a check that one body does not extend beyond the limits of the other body
{
    if(InData[SC_OPEN][index] >= InData[SC_LAST][index]) // part of a check that one body does not
extend beyond the limits of the other body
    {
        // body of the candle 0 does not extend beyond the limits of the body of candle -1
        if ((InData[SC_LAST][index-1]>=InData[SC_OPEN][index]) &&
            (InData[SC_OPEN][index-1]<=InData[SC_LAST][index]))
        {
            ret_flag = true;
        }
    }
    else // part of a check that one body does not extend beyond the limits of the other body
    {
        // body of the candle 0 does not extend beyond the limits of the body of candle -1
        if ((InData[SC_LAST][index-1]>=InData[SC_LAST][index]) &&
            (InData[SC_OPEN][index-1]<=InData[SC_OPEN][index]))
        {
            ret_flag = true;
        }
    }
}
}
}
}
}
}
return ret_flag;
}
/*=====*/

```

```

bool IsBearishHaramiCross(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // BEARISH HARAMI CROSS
    // 1. Uptrend
    // 2. Strong body of candle [index-1]
    // 3. Candle [index] is Doji
    // 4. Body of candle [index] does not extend beyond the limits of the body of the candle [index-1]
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_UP) // uptrend
    {
        if (IsBodyStrong(InData,index-1)) // strong body of the candle -1
        {
            if(IsDoji(sc, settings, index)) // candle 0 is Doji
            {
                if(InData[SC_OPEN][index-1] > InData[SC_LAST][index-1]) // part of a check that one body does not extend
beyond the limits of the other body
                {
                    // body of the candle 0 does not extend beyond the limits of the body of candle -1
                    if ((InData[SC_OPEN][index-1]>InData[SC_LAST][index]) &&
                        (InData[SC_LAST][index-1]<InData[SC_LAST][index]))
                    {
                        ret_flag = true;
                    }
                }
            }
        }
        else // part of a check that one body does not extend beyond the limits of the other body
    }
}

```



```

    {
        // body of the candle 0 does not extend beyond the limits of the body of candle -1
        if ((InData[SC_LAST][index-1]>=InData[SC_LAST][index]) &&
            (InData[SC_OPEN][index-1]<=InData[SC_LAST][index]))
        {
            ret_flag = true;
        }
    }
}
}
return ret_flag;
}
/*=====*/

```

```

bool IsBullishHaramiCross(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // BULLISH HARAMI CROSS
    // 1. Downtrend
    // 2. Strong body of the candle [index-1]
    // 3. candle [index] is Doji
    // 4. Candle [index] does not extend beyond the limits of the body of the candle [index-1]
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_DOWN) // Downtrend
    {
        if (IsBodyStrong(InData,index-1)) // strong body of the candle -1
        {
            if(IsDoji(sc, settings, index)) // candle 0 is Doji
            {
                if(InData[SC_OPEN][index-1] > InData[SC_LAST][index-1]) // part of a check that one body does not extend
beyond the limits of the other body
                {
                    // body of the candle 0 does not extend beyond the limits of the body of candle -1
                    if ((InData[SC_OPEN][index-1]>InData[SC_LAST][index]) &&
                        (InData[SC_LAST][index-1]<InData[SC_LAST][index]))
                    {
                        ret_flag = true;
                    }
                }
            }
            else // part of a check that one body does not extend beyond the limits of the other body
            {
                // body of the candle 0 does not extend beyond the limits of the body of candle -1
                if ((InData[SC_LAST][index-1]>InData[SC_LAST][index]) &&
                    (InData[SC_OPEN][index-1]<InData[SC_LAST][index]))
                {
                    ret_flag = true;
                }
            }
        }
    }
}
return ret_flag;
}
/*=====*/

```

```

const int k_TweezerTop_PAST_INDEX = 3;
const double k_TweezerTop_SimilarityPercent = 7.0;

```

```

bool IsTweezerTop(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // TWEezer TOP
    // 1. Maximum of the candle [index] is equal to the maximum of one of the [index-X] candles (where X is from 1 to
PAST_INDEX)

```

```
// 2. uptrend for the candle [index-X]
```

```
SCBaseDataRef InData = sc.BaseData;
bool ret_flag = false;
// Iterating previous candles
/*for (int i=1; i<k_TweezerTop_PAST_INDEX+1;i++)
{
// Searching for a candle with matching maximum
if(IsNearEqual(InData[SC_HIGH][index], InData[SC_HIGH][index-i], InData, index, k_TweezerTop_SimilarityPercent))
{
// checking the trend
if(settings.UseTrendDetection == false || DetectTendency(sc, settings,index-i,k_TweezerTop_TrendArea) == 1)
{
ret_flag = true;
break;
}
}
}*/

//-----
// 1. 1st day consists of a long body candle.
// 2. 2nd day consists of a short body candle that has a high equal to the prior day's high.
// 3. The color of each candle body is not considered in the matching of this pattern.

if(
(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_UP)
&& IsNearEqual(InData[SC_HIGH][index], InData[SC_HIGH][index-1], InData, index,
k_TweezerTop_SimilarityPercent)
&& IsBodyStrong(InData,index-1)
&& !IsBodyStrong(InData,index)
)
{
ret_flag = true;
}

return ret_flag;
}
/*=====*/
const int k_TweezerBottom_PAST_INDEX = 3;
const double k_TweezerBottom_SimilarityPercent = 7.0;

bool IsTweezerBottom(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
// TWEezer BOTTOM
// 1. Minimum of the candle [index] matches the minimum of one of the candles [index-X] (where X is from 1 to
PAST_INDEX)
// 2. Downtrend for the candle [index-X]
SCBaseDataRef InData = sc.BaseData;
bool ret_flag = false;
// Iterating previous candles
// for (int i=1; i<k_TweezerBottom_PAST_INDEX+1;i++)
// {
//Searching for a candle with matching minimum
// if(IsNearEqual(InData[SC_LOW][index], InData[SC_LOW][index-i], InData, index,
k_TweezerBottom_SimilarityPercent))
// {
//Checking the trend
// if(settings.UseTrendDetection == false || DetectTendency(sc, settings,index-i,k_TweezerBottom_TrendArea) == -1)
// {
// ret_flag = true;
// break;
// }
}
```

```

// }
// }

// 1. 1st day consists of a long body candle.
// 2. 2nd day consists of a short body candle that has a low equal to the prior day's low.
// 3. The color of each candle body is not considered in the matching of this pattern.

if(
    (settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_DOWN)
    && IsNearEqual(InData[SC_LOW][index], InData[SC_LOW][index-1], InData, index,
k_TweezerBottom_SimilarityPercent)
    && IsBodyStrong(InData,index-1)
    && !IsBodyStrong(InData,index)
    )
{
    ret_flag = true;
}
return ret_flag;
}
/*=====*/
const double k_BearishBeltHoldLine_LowerWickPercent = 7.0;
const double k_BearishBeltHoldLine_SimilarityPercent = 7.0;

bool IsBearishBeltHoldLine(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // BEARISH BELT HOLD LINE
    // 1. Uptrend
    // 2. Long black day where the open is equal to the high.
    // 3. No upper shadow.
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(
        (settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_UP) // uptrend
        && IsBlackCandle(InData, index) // Black body of the candle 0
        && IsBodyStrong(InData,index) //Strong body of candle0
        && IsNearEqual(InData[SC_HIGH][index], InData[SC_OPEN][index], InData, index,
k_BearishBeltHoldLine_SimilarityPercent) // OPEN = HIGH
        )
    {
        ret_flag = true;
    }
    return ret_flag;
}
/*=====*/
const double k_BullishBeltHoldLine_UpperWickPercent = 7.0;
const double k_BullishBeltHoldLine_SimilarityPercent = 7.0;

bool IsBullishBeltHoldLine(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // BULLISH BELT HOLD LINE
    // 1. Downtrend
    // 2. Long white day where the open is equal to the low.
    // 3. No lower shadow.
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(
        (settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_DOWN) // downtrend
        && IsWhiteCandle(InData,index) // white body of the candle 0
        && IsBodyStrong(InData,index) //Strong body of candle0
        && IsNearEqual(InData[SC_LOW][index], InData[SC_OPEN][index], InData, index,
k_BullishBeltHoldLine_SimilarityPercent) // OPEN = LOW

```

```

    )
    {
        ret_flag = true;
    }

    return ret_flag;
}
/*=====*/

```

bool IsTwoCrows(SCStudyInterfaceRef sc, **const** s_CandleStickPatternsFinderSettings& settings, **int** index)

```

{
    // UPSIDE-GAP TWO CROWS
    // 1. Uptrend
    // 2. White strong body of the candle [index-2]
    // 3. Candles [index-1] and [index] are black
    // 4. Gap between the bodies of candles [index-2] and [index-1]
    // 5. The body of the candle [index] engulfs the body of the candle [index-1]
    // 6. 3rd day closes above the close of the 1st day.
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(
        (settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_UP) // uptrend
        && IsWhiteCandle(InData, index-2) // the candle -2 is white
        && IsBodyStrong(InData, index-2) // the body of the candle -2 is strong
        && IsBlackCandle(InData, index-1) // the body of the candle -1 is black
        && IsBlackCandle(InData, index) // the body of the candle 0 is black
        && InData[SC_LAST][index-2] < InData[SC_LAST][index-1] // gap between the bodies of candles -2 and -1
        && InData[SC_OPEN][index-1] <= InData[SC_OPEN][index] // the body of the candle 0 engulfs the body of the candle -1
        && InData[SC_LAST][index-1] >= InData[SC_LAST][index]
        && InData[SC_LAST][index] > InData[SC_LAST][index-2] // 3rd day closes above the close of the 1st day.
    )
    {
        ret_flag = true;
    }

    return ret_flag;
}
/*=====*/

```

const double k_ThreeBlackCrows_LowerWickPercent= 7.0;

bool IsThreeBlackCrows(SCStudyInterfaceRef sc, **const** s_CandleStickPatternsFinderSettings& settings, **int** index)

```

{
    // THREE BLACK CROWS
    // 1. Uptrend
    // 2. Three consecutive black days with lower closes each day.
    // 3. Each day opens within the body of the previous day.
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(
        (settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_UP) // uptrend
        && (IsBlackCandle(InData, index-2)
        && IsBlackCandle(InData, index-1)
        && IsBlackCandle(InData, index))
        && InData[SC_LAST][index-2] > InData[SC_LAST][index-1]
        && InData[SC_LAST][index-1] > InData[SC_LAST][index]
        && InData[SC_OPEN][index-1] <= InData[SC_OPEN][index-2]
        && InData[SC_OPEN][index-1] >= InData[SC_LAST][index-2]
        && InData[SC_OPEN][index] <= InData[SC_OPEN][index-1]
        && InData[SC_OPEN][index] >= InData[SC_LAST][index-1])
    {
        ret_flag = true;
    }
}

```

```

    return ret_flag;
}
/*=====*/
const double k_BearishCounterattackLine_SimilarityPercent = 7.0;

bool IsBearishCounterattackLine(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // BEARISH COUNTERATTACK LINE
    // 1. Uptrend
    // 2. The candles [index] and [index-1] are of contrast colors and their close prices are equal
    //Bearish Counterattack Line
    //In an uptrending market, a large white candlestick is following by a large black candlestick that opens on a big gap higher and then slumps back during the period to close at the same price as the previous close. The bearish black candlestick needs followup action to the downside to confirm the turn to a downtrend.
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] == CANDLESTICK_TREND_UP) // uptrend
    {
        // close prices are equal
        if(IsNearEqual(InData[SC_LAST][index], InData[SC_LAST][index-1], InData, index, k_BearishCounterattackLine_SimilarityPercent))
        {
            // check for the colors contrast
            if (IsWhiteCandle(InData, index-1)
                && IsBodyStrong(InData, index-1))
            {
                if(IsBlackCandle(InData, index)
                    && IsBodyStrong(InData, index))
                {
                    ret_flag = true;
                }
            }
            /*else if (IsBlackCandle(InData, index-1))
            {
                if(IsWhiteCandle(InData, index))
                {
                    ret_flag = true;
                }
            }*/
        }
    }
    return ret_flag;
}
/*=====*/
const double k_BullishCounterattackLine_SimilarityPercent = 7.0;

bool IsBullishCounterattackLine(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // BULLISH COUNTERATTACK LINE
    // 1. Downtrend
    // 2. The candles [index] and [index-1] are of contrast colors and their close prices are equal
    //Bullish Counterattack Line
    //In a downtrending market, a large black candlestick is following by a large white candlestick that opens on a big gap lower and then rallies during the period to close at the same price as the previous close. The bullish white candlestick needs followup action to the upside to confirm the turn to an uptrend.
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] == CANDLESTICK_TREND_DOWN) // downtrend
    {
        // close prices are equal
        if(IsNearEqual(InData[SC_LAST][index], InData[SC_LAST][index-1], InData, index, k_BullishCounterattackLine_SimilarityPercent))

```

```

{
    // check for the colors contrast
    /*if (IsWhiteCandle(InData, index-1))
    {
        if(IsBlackCandle(InData, index))
        {
            ret_flag = true;
        }
    }
    else*/ if (IsBlackCandle(InData, index-1)
        && IsBodyStrong(InData,index-1))
    {
        if(IsWhiteCandle(InData, index)
            && IsBodyStrong(InData,index))
        {
            ret_flag = true;
        }
    }
}
}
return ret_flag;
}
/*=====*/

```

```

bool IsThreeInsideUp(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // THREE INSIDE UP
    // 1. Downtrend
    // 2. The candle [index] is white and the close price higher than the close price of the candle [index-1]
    // 3. The candles [index-2] and [index-3] form Bullish Harami Pattern
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_DOWN) // downtrend
    {
        if (IsWhiteCandle(InData,index) && // the candle 0 is white
            (InData[SC_LAST][index]>InData[SC_LAST][index-1])) // the close price of the candle 0 is higher than the close
price of the candle -1
        {
            if(IsBullishHarami(sc,settings,index-1))
            {
                ret_flag = true;
            }
        }
    }
    return ret_flag;
}
/*=====*/

```

```

bool IsThreeOutsideUp(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // THREE OUTSIDE UP
    // 1. Downtrend
    // 2. The candle [index] is white, and its close price is higher than the close price of the candle [index-1]
    // 3. Candles [index-2] and [index-3] form Bullish Engulfing Pattern
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_DOWN) // downtrend
    {
        if (IsWhiteCandle(InData, index) && // the candle 0 is white
            (InData[SC_LAST][index]>InData[SC_LAST][index-1])) // CLOSE of the candle 0 is higher than CLOSE of the
candle -1
        {
            if(IsBullishEngulfing(sc,settings,index-1))

```

```

        {
            ret_flag = true;
        }
    }
}
return ret_flag;
}
/*=====*/

```

```

bool IsThreeInsideDown(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // THREE INSIDE DOWN
    // 1. Uptrend
    // 2. The candle [index] is black, and its close price is lower than the close price of the the candle [index-1]
    // 3. The candles [index-2] and [index-3] form Bearish Harami Pattern
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_UP) // uptrend
    {
        if(IsBlackCandle(InData, index) && // the candle 0 is black
            (InData[SC_LAST][index]<InData[SC_LAST][index-1])) // CLOSE of the candle 0 is lower than CLOSE of the
candle -1
        {
            if(IsBearishHarami(sc,settings,index-1))
            {
                ret_flag = true;
            }
        }
    }
    return ret_flag;
}
/*=====*/

```

```

bool IsThreeOutsideDown(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // THREE OUTSIDE DOWN
    // 1. Uptrend
    // 2. The candle [index] is black, and its close price is lower than close price of the candle [index-1]
    // 3. The candles [index-2] and [index-3] form Bearish Engulfing Pattern
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_UP) // uptrend
    {
        if(IsBlackCandle(InData, index) && // the candle 0 is black
            (InData[SC_LAST][index]<InData[SC_LAST][index-1])) // CLOSE of the candle 0 is lower than CLOSE of the
candle -1
        {
            if(IsBearishEngulfing(sc,settings,index-1))
            {
                ret_flag = true;
            }
        }
    }
    return ret_flag;
}
/*=====*/

```

```

const double k_Kicker_UpperWickPercent = 7.0;
const double k_Kicker_LowerWickPercent = 7.0;

```

```

bool IsKicker(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // KICKER
    // 1. The candle [index-1] is black, without wicks

```



```

// 2. The candle [index] is white, without wicks
// 3. The bodies of the candles [index] and [index-1] are strong
// 4. A gap between the bodies of the candles [index] and [index-1]
SCBaseDataRef InData = sc.BaseData;
bool ret_flag = false;
// the body of the candle -1 is black and without wicks
if (IsBlackCandle(InData, index-1) &&
    IsUpperWickSmall(InData, index-1, k_Kicker_UpperWickPercent) &&
    IsLowerWickSmall(InData, index-1, k_Kicker_LowerWickPercent))
{
    // the body of the candle 0 is white and without wicks
    if (IsWhiteCandle(InData, index) &&
        IsUpperWickSmall(InData, index, k_Kicker_UpperWickPercent) &&
        IsLowerWickSmall(InData, index, k_Kicker_LowerWickPercent))
    {
        if(IsBodyStrong(InData,index-1)) // the body of the candle -1 is strong
        {
            if(IsBodyStrong(InData,index)) // the body of the candle 0 is strong
            {
                // a gap between bodies
                if(InData[SC_OPEN][index-1]<InData[SC_OPEN][index])
                {
                    ret_flag = true;
                }
            }
        }
    }
}
return ret_flag;
}

/*=====*/
const double k_Kicking_UpperWickPercent = 7.0;
const double k_Kicking_LowerWickPercent = 7.0;

bool IsKicking(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // KICKING
    // 1. The candle [index-1] is white and without wicks
    // 2. The candle [index] is black and without wicks
    // 3. Bodies of the candles [index] and [index-1] are strong
    // 4. A gap between the bodies of the candles [index] and [index-1]
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    // the body of the candle -1 is white and without wicks
    if (IsWhiteCandle(InData, index-1) &&
        IsUpperWickSmall(InData, index-1, k_Kicking_UpperWickPercent) &&
        IsLowerWickSmall(InData, index-1, k_Kicking_LowerWickPercent))
    {
        // the body of the candle 0 is black and without wicks
        if (IsBlackCandle(InData, index) &&
            IsUpperWickSmall(InData, index, k_Kicking_UpperWickPercent) &&
            IsLowerWickSmall(InData, index, k_Kicking_LowerWickPercent))
        {
            if(IsBodyStrong(InData,index-1)) // the body of the candle -1 is strong
            {
                if(IsBodyStrong(InData,index)) // the body of the candle 0 is strong
                {
                    // a gap between the bodies
                    if(InData[SC_OPEN][index]<InData[SC_OPEN][index-1])
                    {
                        ret_flag = true;
                    }
                }
            }
        }
    }
}

```

```

    }
}
return ret_flag;
}
/*=====*/

const double k_ThreeWhiteSoldiers_UpperWickPercent = 7.0;

bool IsThreeWhiteSoldiers(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // THREE WHITE SOLDIERS
    // 1. Downtrend till the candle 0
    // 2. The candles [index], [index-1] and [index-2] are white
    // 3. OPENs of candles [index], [index-1] are inside the bodies of the previous candles
    // 4. Upper wicks of the candles [index], [index-1] and [index-2] are insignificant
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if( settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_DOWN) // Downtrend
    {
        if (IsWhiteCandle(InData,index) && // the candle 0 is white
            IsWhiteCandle(InData, index-1) && // the candle -1 is white
            IsWhiteCandle(InData, index-2)) // the candle -2 is white
        {
            // OPEN of the candle 0 inside the body of the candle -1
            if ((InData[SC_OPEN][index]<=InData[SC_LAST][index-1])&&
                (InData[SC_OPEN][index]>=InData[SC_OPEN][index-1]))
            {
                // OPEN of the candle -1 inside the body of the candle -2
                if ((InData[SC_OPEN][index-1]<=InData[SC_LAST][index-2])&&
                    (InData[SC_OPEN][index-1]>=InData[SC_OPEN][index-2]))
                {
                    // check for upper wicks
                    if(IsUpperWickSmall(InData, index, k_ThreeWhiteSoldiers_UpperWickPercent) &&
                        IsUpperWickSmall(InData, index-1, k_ThreeWhiteSoldiers_UpperWickPercent)&&
                        IsUpperWickSmall(InData, index-2, k_ThreeWhiteSoldiers_UpperWickPercent)
                    )
                    {
                        ret_flag = true;
                    }
                }
            }
        }
    }
}
return ret_flag;
}
/*=====*/

```

```

bool IsAdvanceBlock(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // ADVANCE BLOCK
    // 1. Uptrend till the candle 0
    // 2. The candles [index], [index-1] and [index-2] are white
    // 3. Opens of candles [index], [index-1] are inside the bodies of the previous candles
    // 4. The body of the candle [index] is smaller than the body of the candle [index-1],
    //and the body of the candle [index-1] is smaller than the body of the candle [index-2]
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_UP) // Uptrend
    {
        if (IsWhiteCandle(InData, index) && // the candle 0 is white
            IsWhiteCandle(InData, index-1) && // the candle -1 is white

```

```

    IsWhiteCandle(InData, index-2)) // the candle -2 is white
{
    // OPEN of the candle 0 is inside the body of the candle -1
    if ((InData[SC_OPEN][index]<=InData[SC_LAST][index-1])&&
        (InData[SC_OPEN][index]>=InData[SC_OPEN][index-1]))
    {
        // OPEN of the candle -1 is inside the body of the candle -2
        if ((InData[SC_OPEN][index-1]<=InData[SC_LAST][index-2])&&
            (InData[SC_OPEN][index-1]>=InData[SC_OPEN][index-2]))
        {
            // check for the size of the body
            if(BodyLength(InData,index) < BodyLength(InData, index-1) &&
                BodyLength(InData, index-1) < BodyLength(InData, index-2))
            {
                ret_flag = true;
            }
        }
    }
}
}
return ret_flag;
}
/*=====*/

const int k_Deliberation_UpperWickPercent = 7;

bool IsDeliberation(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // DELIBERATION
    // 1. Uptrend till the candle 0
    // 2. The candles [index], [index-1] and [index-2] are white
    // 3. OPENs of the candle [index-1] is inside the body of the previous candle
    // 4. The candle [index-1] has a strong body
    // 5. Candle [index] has a weak body
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
    CANDLESTICK_TREND_UP) // uptrend
    {
        if (IsWhiteCandle(InData, index) && // Candle 0 is white
            IsWhiteCandle(InData, index-1) && // Candle -1 is white
            IsWhiteCandle(InData, index-2)) // Candle -2 is white
        {
            // OPEN of the candle 0 is inside of the body of the candle -1
            /*if ((InData[SC_OPEN][index]<InData[SC_LAST][index-1])&&
                (InData[SC_OPEN][index]>InData[SC_OPEN][index-1]))*/
            {
                // OPEN of the candle -1 is inside of the body of the candle -2
                if ((InData[SC_OPEN][index-1]<=InData[SC_LAST][index-2])&&
                    (InData[SC_OPEN][index-1]>=InData[SC_OPEN][index-2]))
                {
                    // body of the candle -1 is strong
                    if(IsBodyStrong(InData,index-1))
                    {
                        // candle -1 does not have an upper wick
                        //if (IsUpperWickSmall(InData, index-1, k_Deliberation_UpperWickPercent))
                        {
                            // body of the candle 0 is weak
                            if(!IsBodyStrong(InData,index))
                            {
                                ret_flag = true;
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

    }
}
}
}
return ret_flag;
}
/*=====*/

```

```
const int k_BearishTriStar_PAST_INDEX = 3;
```

```
bool IsBearishTriStar(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{

```

```

    // BEARISH TRI STAR
    //1. Market is characterized by uptrend.
    //2. We see three Dojis on three consecutive days.
    //3. The second day Doji has a gap above the first and third.

```

```

    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    // the candle 0 is Doji
    if((settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_UP)
        && IsDoji(sc, settings, index)
        && IsDoji(sc, settings, index-1)
        && IsDoji(sc, settings, index-2)
        && (InData[SC_OPEN][index-1] > InData[SC_LAST][index])
        && (InData[SC_OPEN][index-1] > InData[SC_LAST][index-2])
    )
    {
        ret_flag = true;
    }
    return ret_flag;
}
/*=====*/

```

```
const int k_BullishTriStar_PAST_INDEX = 3;
```

```
bool IsBullishTriStar(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{

```

```

    // BULLISH TRI STAR
    //1. Market is characterized by downtrend.
    //2. Then we see three consecutive Doji.
    //3. The second day Doji gaps below the first and third.

```

```

    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    // the candle 0 is Doji
    if((settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_DOWN)
        && IsDoji(sc, settings, index)
        && IsDoji(sc, settings, index-1)
        && IsDoji(sc, settings, index-2)
        && (InData[SC_OPEN][index-1] < InData[SC_LAST][index])
        && (InData[SC_OPEN][index-1] < InData[SC_LAST][index-2])
    )
    {
        ret_flag = true;
    }
    return ret_flag;
}
/*=====*/

```

```
bool IsUniqueThreeRiverBottom(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{

```

```

    // UNIQUE THREE RIVER BOTTOM

```

```

// 1. Downtrend till the candle [index-2]
// 2. The candle [index-2] is black and with a strong body
// 3. The candle [index-1] forms HAMMER PATTERN
// 4. The minimum of the candle [index-1] is smaller than the minimum of the candles [index-2] and [index]
// 5. The candle [index] has a weak body that lower than the body of the candle [index-1]
SCBaseDataRef InData = sc.BaseData;
bool ret_flag = false;
if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_DOWN) // Downtrend
{
    if(InData[SC_LAST][index-2]<InData[SC_OPEN][index-2]) //the candle -2 is black
    {
        if(IsBodyStrong(InData,index-2)) //the body of the candle -2 is strong
        {
            if(IsHammer(sc,settings,index-1)) // The candle -1 is HAMMER
            {
                // lower wick of the candle -1 forms new minimum
                if ((InData[SC_LOW][index-1]<InData[SC_LOW][index])&&
                    (InData[SC_LOW][index-1]<InData[SC_LOW][index-2]))
                {
                    if(IsWhiteCandle(InData, index)) // the candle 0 is white
                    {
                        if(!IsBodyStrong(InData,index)) // the body of the candle 0 is weak
                        {
                            if ((InData[SC_OPEN][index]<InData[SC_OPEN][index-1])&&
                                (InData[SC_OPEN][index]<InData[SC_LAST][index-1]))
                                // the body of the candle 0 is lower than the body of the candle -1
                                {
                                    ret_flag = true;
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
return ret_flag;
}
/*=====*/
const double k_BearishDojiStar_UpperWickPercent = 25.0;
const double k_BearishDojiStar_LowerWickPercent = 25.0;

bool IsBearishDojiStar(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // BEARISH DOJI STAR
    // 1. Uptrend
    // 2. A strong white body of the candle [index-1]
    // 3. The candle [index] is Doji
    // 4. The body of the candle [index] is higher than the maximum of the candle [index-1]
    // 5. The wicks of the candles are "not long"
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_UP) // check for the uptrend
    {
        if(IsWhiteCandle(InData, index-1)) // the body of the candle -1 is white
        {
            if(IsBodyStrong(InData,index-1)) // the body of the candle -1 is strong
            {
                if (IsDoji(sc, settings, index)) // the candle 0 is Doji
                {
                    if (InData[SC_LAST][index] > InData[SC_HIGH][index-1])
                        // the body of the candle 0 is higher than the maximum of the candle -1
                    {

```

```

        // check for the wicks length
        if (UpperWickLength(InData, index) <= PercentOfCandleLength(InData, index-1,
k_BearishDojiStar_UpperWickPercent) && // upper wick of Doji
            LowerWickLength(InData, index) <= PercentOfCandleLength(InData, index-1,
k_BearishDojiStar_LowerWickPercent))
        {
            ret_flag = true;
        }
    }
}
}
}
}
return ret_flag;
}
/*=====*/
const double k_BullishDojiStar_LowerWickPercent = 25.0;
const double k_BullishDojiStar_UpperWickPercent = 25.0;

bool IsBullishDojiStar(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // BULLISH DOJI STAR
    // 1. Downtrend
    // 2. Strong black body of the candle [index-1]
    // 3. The candle [index] is Doji
    // 4. The body of the candle [index] is lower than minimum of the candle [index-1]
    // 5. The wicks of the candle [index] are "not long"
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_DOWN) // check for the downtrend
    {
        if(IsBlackCandle(InData, index-1)) // the body of the candle -1 is black
        {
            if(IsBodyStrong(InData, index-1)) // the body of the candle -1 is strong
            {
                if (IsDoji(sc, settings, index)) // the candle 0 is Doji
                {
                    if (InData[SC_LAST][index] < InData[SC_LOW][index-1])
                        // the body of the candle 0 is lower than minimum of the candle -1
                    {
                        // check for the wicks length
                        if (UpperWickLength(InData, index) <= PercentOfCandleLength(InData, index-1,
k_BearishDojiStar_UpperWickPercent) && // upper wick of Doji
                            LowerWickLength(InData, index) <= PercentOfCandleLength(InData, index-1,
k_BearishDojiStar_LowerWickPercent)) // lower wick of Doji
                        {
                            ret_flag = true;
                        }
                    }
                }
            }
        }
    }
}
return ret_flag;
}
/*=====*/
const double k_BearishDragonflyDoji_UpperWickPercent = 7.0;

bool IsBearishDragonflyDoji(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // BEARISH DRAGONFLY DOJI
    // 1. Uptrend
    // 2. The candle [index] is Doji

```

```

// 3. The candle [index] does not have an upper wick
// 4. The lower wick of the candle [index] is very long
SCBaseDataRef InData = sc.BaseData;
bool ret_flag = false;
if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_UP) // check for the uptrend
{
    if (IsDoji(sc, settings, index)) // the candle 0 is Doji
    {
        if (IsUpperWickSmall(InData, index, k_BearishDragonflyDoji_UpperWickPercent)) //the candle 0 does not have an
upper wick
        {
            // The candle 0 has a long lower wick
            //
            // Note:
            // if body is small and upper wick is small while the candle is
            // long => the lower wick is long
            if (IsCandleStrength(InData,index))
            {
                ret_flag = true;
            }
        }
    }
}
return ret_flag;
}
/*=====*/

```

```

const double k_BullishDragonflyDoji_UpperWickPercent = 7.0;

```

```

bool IsBullishDragonflyDoji(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // BULLISH DRAGONFLY DOJI
    // 1. Downtrend
    // 2. The candle [index] is Doji
    // 3. The candle [index] does not have an upper wick
    // 4. The lower wick of the candle [index] is very long
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_DOWN) // check for the downtrend
    {
        if (IsDoji(sc, settings, index)) // the candle 0 is Doji
        {
            if (IsUpperWickSmall(InData, index, k_BullishDragonflyDoji_UpperWickPercent)) //the candle 0 does not have an
upper wick
            {
                // The candle 0 has a long lower wick
                //
                // Note:
                // if body is small and upper wick is small while the candle is
                // long => the lower wick is long
                if (IsCandleStrength(InData,index))
                {
                    ret_flag = true;
                }
            }
        }
    }
    return ret_flag;
}
/*=====*/

```

```

const double k_BearishGravestoneDoji_LowerWickPercent = 7.0;

```



```

bool IsBearishGravestoneDoji(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // BEARISH GRAVESTONE DOJI
    // 1. Uptrend
    // 2. The candle [index-1] has a white body
    // 3. The candle [index] is Doji
    // 4. The lower wick of the candle [index] is insignificant
    // 5. Upper wick of the candle [index] is very long
    // 6. OPEN of the candle [index] is higher than the maximum of the candle [index-1]
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_UP) // check for the uptrend
    {
        if(IsWhiteCandle(InData, index-1)) // the body of the candle -1 is white
        {
            if (IsDoji(sc, settings, index)) // the candle [index] is Doji
            {
                // lower wick of the candle 0 is insignificant
                if (IsLowerWickSmall(InData, index, k_BearishGravestoneDoji_LowerWickPercent))
                {
                    // upper wick of the candle 0 is long (please, note that it's Doji, so there is no body)
                    if (IsCandleStrength(InData,index))
                    {
                        // OPEN of the candle 0 is higher than the maximum of the candle -1
                        if(InData[SC_OPEN][index] > InData[SC_HIGH][index-1])
                        {
                            ret_flag = true;
                        }
                    }
                }
            }
        }
    }
}
return ret_flag;
}
/*=====*/

```

```

const double k_BullishGravestoneDoji_LowerWickPercent = 7;

```

```

bool IsBullishGravestoneDoji(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // BULLISH GRAVESTONE DOJI
    // 1. Downtrend
    // 2. The candle [index-1] has a black body
    // 3. The candle [index] is Doji
    // 4. The candle [index] does not have a lower wick
    // 5. The upper wick of the candle [index] is very long
    // 6. Absence of the gap between the candles [index] and [index-1] is necessary
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_DOWN) // check for the downtrend
    {
        if(IsBlackCandle(InData, index-1)) // the body of the candle -1 is black
        {
            if (IsDoji(sc, settings,index)) // the candle 0 is Doji
            {
                // candle 0 does not have a lower wick
                if (IsLowerWickSmall(InData, index, k_BullishGravestoneDoji_LowerWickPercent))
                {
                    // candle 0 has a long upper wick
                    //
                    // Note:
                    // if body is small and lower wick is small and the candle is

```

```

// long => the lower wick is long
// (please, note that it's Doji, so there is no body)
if (IsCandleStrength(InData, index))
{
    // check for the absence of the gap
    if (InData[SC_LOW][index-1] < InData[SC_HIGH][index])
    {
        ret_flag = true;
    }
}
}
}
}
}
return ret_flag;
}
/*=====*/

```

```

const double k_BearishLongleggedDoji_SimilarityPercent = 10.0;

```

```

bool IsBearishLongleggedDoji(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // BEARISH LONGLEGGED DOJI
    // 1. Uptrend
    // 2. The candle [index] is Doji
    // 3. The candle [index] is long
    // 4. The wicks of the candle [index] are nearly the same
    // 5. The body of the candle [index] is higher than the maximum of the candle [index-1]
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if (settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
    CANDLESTICK_TREND_UP) // check for the uptrend
    {
        if (IsDoji(sc, settings, index)) // the candle 0 is Doji
        {
            // the candle 0 is long
            if (IsCandleStrength(InData, index))
            {
                // the body of the candle 0 is higher than the maximum of the candle -1
                if (max(InData[SC_LAST][index], InData[SC_OPEN][index]) > InData[SC_HIGH][index-1])
                {
                    // the wicks of the candle 0 are nearly the same
                    if (abs(LowerWickLength(InData, index) - UpperWickLength(InData, index)) <
                        PercentOfCandleLength(InData, index, k_BearishLongleggedDoji_SimilarityPercent))
                    {
                        ret_flag = true;
                    }
                }
            }
        }
    }
}
return ret_flag;
}
/*=====*/

```

```

const double k_BullishLongleggedDoji_SimilarityPercent = 10.0;

```

```

bool IsBullishLongleggedDoji(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // BULLISH LONGLEGGED DOJI
    // 1. Downtrend
    // 2. The candle [index] is Doji
    // 3. The candle [index] is long
    // 4. The wicks of the candle [index] are nearly the same
    // 5. The body of the candle [index] is lower than the minimum of the candle [index-1]

```

```

SCBaseDataRef InData = sc.BaseData;
bool ret_flag = false;
if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_DOWN) // check for the downtrend
{
    if (IsDoji(sc, settings, index)) // the candle 0 is Doji
    {
        // The candle 0 is long
        if (IsCandleStrength(InData,index))
        {
            // the body of the candle 0 is lower than the minimum of the candle -1
            if (max(InData[SC_LAST][index], InData[SC_OPEN][index]) < InData[SC_LOW][index-1])
            {
                // the wicks of the candle 0 are nearly the same
                if (abs(LowerWickLength(InData, index) - UpperWickLength(InData, index))<
                    PercentOfCandleLength(InData, index, k_BullishLongleggedDoji_SimilarityPercent))
                {
                    ret_flag = true;
                }
            }
        }
    }
}
return ret_flag;
}
/*=====*/
//const int    k_BearishSideBySideWhiteLines_TrendArea = 3;
const int    k_BearishSideBySideWhiteLines_PAST_INDEX = 3;
const double k_BearishSideBySideWhiteLines_OpenSimilarityPercent = 14.0;
const double k_BearishSideBySideWhiteLines_CandleSimilarityPercent = 14.0;

bool IsBearishSideBySideWhiteLines(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int
index)
{
    // BEARISH SIDE-BY-SIDE WHITE LINES
    // 1. Downtrend
    // 2. The candles [index], [index-1] are white
    // 3. The candle [index-2] is black
    // 4. The maximum of the candle [index-1] is lower than the minimum of the candle [index-2]
    // 5. Open prices and the sizes of the candles [index-1] and [index] are ALMOST the same

    /* 1. 1st day is a black day.
    2. 2nd day is a white day which gaps below the 1st day's open.
    3. 3rd day is a white day about the same size as the 2nd day, opening at about the same price.*/

    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_DOWN) // check for the downtrend
    {
        if (IsWhiteCandle(InData, index) && // the candle 0 is white
            IsWhiteCandle(InData, index-1) && // the candle -1 is white
            IsBlackCandle(InData, index-2)) // the candle -2 is black
        {
            // the candle -2 is higher than the candle -1 with a gap
            if (InData[SC_LOW][index-2] > InData[SC_HIGH][index-1])
            {
                // compare of the OPENS of the candles 0 and -1
                if (IsNearEqual(InData[SC_OPEN][index-1], InData[SC_OPEN][index], InData, index,
k_BearishSideBySideWhiteLines_OpenSimilarityPercent))
                {
                    // compare of the length of the candles 0 and -1
                    if (IsNearEqual(CandleLength(InData,index), CandleLength(InData, index-1), InData, index,
k_BearishSideBySideWhiteLines_CandleSimilarityPercent))
                    {

```

```

        ret_flag = true;
    }
}
}
}
return ret_flag;
}
/*=====*/
//const int k_BullishSideBySideWhiteLines_TrendArea = 3;
const int k_BullishSideBySideWhiteLines_PAST_INDEX = 3;
const double k_BullishSideBySideWhiteLines_OpenSimilarityPercent = 7.0;
const double k_BullishSideBySideWhiteLines_CandleSimilarityPercent = 7.0;

bool IsBullishSideBySideWhiteLines(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // BULLISH SIDE-BY-SIDE WHITE LINES
    // 1. Uptrend
    // 2. The candles [index], [index-1] and [index-2] are white
    // 3. Maximum of the candle [index-2] is lower than minimum of the candle [index-1]
    // 4. Open prices and the sizes of the candles [index-1] and [index] are ALMOST the same

    /* 1. Market is characterized by uptrend.
    2. We see a white candlestick in the first day.
    3. Then we see another white candlestick on the second day with an upward gap.
    4. Finally, we see a white candlestick on the third day characterized by the same body length and whose closing price
    is equal to the close of the second day and a new high is established.*/

    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(
        (settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
        CANDLESTICK_TREND_UP) // check for the uptrend
        && IsWhiteCandle(InData, index)
        && IsWhiteCandle(InData, index-1)
        && IsWhiteCandle(InData, index-2)
        && (InData[SC_LOW][index-1] > InData[SC_HIGH][index-2])
        && (IsNearEqual(InData[SC_LAST][index-1], InData[SC_LAST][index], InData, index,
        k_BullishSideBySideWhiteLines_OpenSimilarityPercent))
        && (IsNearEqual(InData[SC_OPEN][index-1], InData[SC_OPEN][index], InData, index,
        k_BullishSideBySideWhiteLines_OpenSimilarityPercent))
        && (InData[SC_HIGH][index] > InData[SC_HIGH][index-1])
    )
    {
        ret_flag = true;
    }
    return ret_flag;
}
/*=====*/

const int k_FallingThreeMethods_PAST_INDEX = 3;
const double k_FallingThreeMethods_CandleCloseClosenessPercent = 30.0;

bool IsFallingThreeMethods(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // FALLING THREE METHODS

    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;

    // 1. Market is characterized by downtrend.
    // 2. We see a long black candlestick in the first day.
    // 3. We then see three small real bodies defining a brief uptrend on the second, third, and fourth days. However these

```

bodies stay within the range of the first day.

// 4. Finally we see a long black candlestick on the fifth day opening near the previous day's close and also closing below the close of the initial day to define a new low.

```
if(
    (settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_DOWN)

    && IsBodyStrong(InData,index-4)
    && IsBlackCandle(InData, index-4)

    //&& IsWhiteCandle(InData, index-1)
    //&& IsWhiteCandle(InData, index-2)
    //&& IsWhiteCandle(InData, index-3)
    && !IsBodyStrong(InData,index-1)
    && !IsBodyStrong(InData,index-2)
    && !IsBodyStrong(InData,index-3)
    && (InData[SC_OHLC_AVG][index-3] < InData[SC_OHLC_AVG][index-2])
    && (InData[SC_OHLC_AVG][index-2] < InData[SC_OHLC_AVG][index-1])

    && (InData[SC_HIGH][index-3] < InData[SC_HIGH][index-4])
    && (InData[SC_HIGH][index-2] < InData[SC_HIGH][index-4])
    && (InData[SC_HIGH][index-1] < InData[SC_HIGH][index-4])
    && (InData[SC_LOW][index-3] > InData[SC_LOW][index-4])
    && (InData[SC_LOW][index-2] > InData[SC_LOW][index-4])
    && (InData[SC_LOW][index-1] > InData[SC_LOW][index-4])

    && IsBodyStrong(InData,index)
    && IsBlackCandle(InData, index)
    && (IsNearEqual(InData[SC_OPEN][index], InData[SC_LAST][index-1], InData, index,
k_FallingThreeMethods_CandleCloseClosenessPercent))
    && (InData[SC_LAST][index] < InData[SC_LAST][index-4])
    )
{
    ret_flag = true;
}

return ret_flag;
}
/*=====*/

const int k_RisingThreeMethods_PAST_INDEX = 3;

bool IsRisingThreeMethods(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // RISING THREE METHODS

    // . Market is characterized by downtrend.
    // 1. 1st day is a long white day.
    // 2. Three small body candlesticks follow the 1st day. Each trends downward and closes within the range of the 1st
day.
    // 3. The last day is a long white day and closes above the 1st day's close.
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;

    if(
        (settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_UP)

        && IsBodyStrong(InData,index-4)
        && IsWhiteCandle(InData, index-4)

        && !IsBodyStrong(InData,index-1)
```

```

    && !IsBodyStrong(InData,index-2)
    && !IsBodyStrong(InData,index-3)
    && (InData[SC_OHLC_AVG][index-3] > InData[SC_OHLC_AVG][index-2])
    && (InData[SC_OHLC_AVG][index-2] > InData[SC_OHLC_AVG][index-1])

    && (InData[SC_HIGH][index-3] <= InData[SC_HIGH][index-4])
    && (InData[SC_HIGH][index-2] <= InData[SC_HIGH][index-4])
    && (InData[SC_HIGH][index-1] <= InData[SC_HIGH][index-4])
    && (InData[SC_LOW][index-3] >= InData[SC_LOW][index-4])
    && (InData[SC_LOW][index-2] >= InData[SC_LOW][index-4])
    && (InData[SC_LOW][index-1] >= InData[SC_LOW][index-4])

    && IsBodyStrong(InData,index)
    && IsWhiteCandle(InData, index)
    && (InData[SC_LAST][index] > InData[SC_LAST][index-4])
)
{
    ret_flag = true;
}

return ret_flag;
}
/*=====*/

const double k_BearishSeparatingLines_SimilarityPercent = 5.0;

bool IsBearishSeparatingLines(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // BEARISH SEPARATING LINE
    // 1. Downtrend
    // 2. The candles [index] and [index-1] are of contrast colors and their open prices are equal
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if( settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_DOWN) // downtrend
    {
        // OPENS are equal
        if(IsNearEqual(InData[SC_OPEN][index], InData[SC_OPEN][index-1], InData, index,
k_BearishSeparatingLines_SimilarityPercent))
        {
            // check for the colors contrast
            bool colorIndex0 = IsWhiteCandle(InData, index);
            bool colorIndex1 = IsWhiteCandle(InData, index-1);

            ret_flag = (colorIndex1!=colorIndex0);
        }
    }
    return ret_flag;
}
/*=====*/

const double k_BullishSeparatingLines_SimilarityPercent = 5.0;

bool IsBullishSeparatingLines(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // BULLISH SEPARATING LINE
    // 1. Uptrend
    // 2. The candles [index] and [index-1] are of contrast colors and their open prices are equal
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_UP) // uptrend
    {
        // OPENS are equal
        if(IsNearEqual(InData[SC_OPEN][index], InData[SC_OPEN][index-1], InData, index,

```

```

k_BullishSeparatingLines_SimilarityPercent))
{
    // check for the colors contrast
    // check for the colors contrast
    bool colorIndex0 = IsWhiteCandle(InData, index);
    bool colorIndex1 = IsWhiteCandle(InData, index-1);

    ret_flag = (colorIndex1!=colorIndex0);
}
}
return ret_flag;
}
/*=====*/

bool IsDownsideTasukiGap(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // DOWNSIDE TASUKI GAP
    // 1. Downtrend
    // 2. The candle [index] is white and its open price is inside the body of the candle [index-1]
    // 3. The candles [index-1] and [index-2] are black with strong bodies
    // 4. There is a gap between the candles [index-1] and [index-2]
    // 5. CLOSE of the candle [index] is in the gap between the candles [index-1] and [index-2]
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_DOWN) // downtrend
    {
        if(IsWhiteCandle(InData, index) && // the candle 0 is white
            IsBlackCandle(InData, index-1) && // the candle -1 is black
            IsBlackCandle(InData, index-2)) // the candle -2 is black
        {
            if(IsBodyStrong(InData, index-1)) // the body of the candle -1 is strong
            {
                if(IsBodyStrong(InData, index-2)) // the body of the candle -2 is strong
                {
                    if (InData[SC_LOW][index-2] > InData[SC_HIGH][index-1]) // a gap between the candles -1 and -2
                    {
                        // OPEN of the candle 0 is inside the body of the candle -1
                        if ((InData[SC_OPEN][index] > InData[SC_LAST][index-1]) &&
                            (InData[SC_OPEN][index] < InData[SC_OPEN][index-1]))
                        {
                            // CLOSE of the candle 0 is higher than maximum of the candle -1 and lower than the minimum of the
candle -2
                            if ((InData[SC_LAST][index]>InData[SC_HIGH][index-1]) &&
                                (InData[SC_LAST][index]<InData[SC_LOW][index-2]))
                            {
                                ret_flag = true;
                            }
                        }
                    }
                }
            }
        }
    }
}
return ret_flag;
}
/*=====*/

```

```

bool IsUpsideTasukiGap(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // UPSIDE TASUKI GAP
    // 1. Uptrend
    // 2. The candle [index] is black and its open price is inside the body of the candle [index-1]
    // 3. The candles [index-1] and [index-2] are white with strong bodies

```

```

// 4. There is a gap between the candles [index-1] and [index-2]
// 5. CLOSE of the candle [index] is in the gap between the candles [index-1] and [index-2]
SCBaseDataRef InData = sc.BaseData;
bool ret_flag = false;
if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_UP) // uptrend
{
    if (IsBlackCandle(InData, index) && // the candle 0 is black
        IsWhiteCandle(InData, index-1) && // the candle -1 is white
        IsWhiteCandle(InData, index-2)) // the candle -2 is white
    {
        if(IsBodyStrong(InData, index-1)) // the body of the candle -1 is strong
        {
            if(IsBodyStrong(InData, index-2)) // the body of the candle -2 is strong
            {
                if (InData[SC_HIGH][index-2] < InData[SC_LOW][index-1]) // a gap between the candles -1 and -2
                {
                    // OPEN of the candle 0 is inside the body of the candle -1
                    if ((InData[SC_OPEN][index] > InData[SC_OPEN][index-1]) &&
                        (InData[SC_OPEN][index] < InData[SC_LAST][index-1]))
                    {
                        // CLOSE of the candle 0 is lower than minimum of the candle -1 and higher than the maximum of the
candle -2
                        if ((InData[SC_LAST][index]>InData[SC_HIGH][index-2]) &&
                            (InData[SC_LAST][index]<InData[SC_LOW][index-1]))
                        {
                            ret_flag = true;
                        }
                    }
                }
            }
        }
    }
}
return ret_flag;
}
/*=====*/

```

```

bool IsBearishThreeLineStrike(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // BEARISH THREE LINE STRIKE
    // 1. Downtrend
    // 2. The candle [index] is white with a strong body
    // 3. The candles [index-1], [index-2] and [index-3] are black with strong bodies
    // 4. OPEN of the candle [index] is lower than CLOSE of the candle [index-1]
    // 5. CLOSE of the candle [index] is higher than OPEN of the candle [index-3]
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_DOWN) // downtrend
    {
        if (IsWhiteCandle(InData, index) && // the candle 0 is white
            IsBlackCandle(InData, index-1) && // the candle -1 is black
            IsBlackCandle(InData, index-2) && // the candle -2 is black
            IsBlackCandle(InData, index-3)) // the candle -3 is black
        {
            if(IsBodyStrong(InData, index-1)) // the body of the candle -1 is strong
            {
                if(IsBodyStrong(InData, index-2)) // the body of the candle -2 is strong
                {
                    if(IsBodyStrong(InData, index-3)) // the body of the candle -1 is strong
                    {
                        // OPEN of the candle 0 is lower than CLOSE of the candle -1
                        if(InData[SC_OPEN][index] < InData[SC_LAST][index-1])
                        {

```



```

        // CLOSE of the candle 0 is higher than OPEN of the candle -3
        if(InData[SC_LAST][index]>InData[SC_OPEN][index-3])
        {
            ret_flag = true;
        }
    }
}
}
}
}
}
return ret_flag;
}
/*=====*/

```

```

bool IsBullishThreeLineStrike(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // BULLISH THREE LINE STRIKE
    // 1. Uptrend
    // 2. The candle [index] is black with a strong body
    // 3. The candles [index-1], [index-2] and [index-3] are white with strong bodies
    // 4. OPEN of the candle [index] is higher than CLOSE of the candle [index-1]
    // 5. CLOSE of the candle [index] is lower than OPEN of the candle [index-3]
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_UP) // uptrend
    {
        if (IsBlackCandle(InData, index) && // the candle 0 is black
            IsWhiteCandle(InData, index-1) && // the candle -1 is white
            IsWhiteCandle(InData, index-2) && // the candle -2 is white
            IsWhiteCandle(InData, index-3)) // the candle -3 is white
        {
            if(IsBodyStrong(InData, index-1)) // the body of the candle -1 is strong
            {
                if(IsBodyStrong(InData, index-2)) // the body of the candle -2 is strong
                {
                    if(IsBodyStrong(InData, index-3)) // the body of the candle -3 is strong
                    {
                        // OPEN of the candle 0 is higher than CLOSE of the candle -1
                        if(InData[SC_OPEN][index] > InData[SC_LAST][index-1])
                        {
                            // CLOSE of the candle 0 is lower than OPEN of the candle -3
                            if(InData[SC_LAST][index]<InData[SC_OPEN][index-3])
                            {
                                ret_flag = true;
                            }
                        }
                    }
                }
            }
        }
    }
}
return ret_flag;
}
/*=====*/

```

```

bool IsDownsideGapThreeMethods(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int
index)
{
    // DOWNSIDE GAP THREE METHODS
    // 1. Downtrend
    // 2. The candle [index] is white and its OPEN is inside the body of the candle [index-1]
    // 3. The candles [index-1] and [index-2] are black with a strong bodies

```

```

// 4. There is a gap between the candles [index-1] and [index-2]
// 5. The body of the candle [index] closes the gap between the candles [index-1] and [index-2]
SCBaseDataRef InData = sc.BaseData;
bool ret_flag = false;
if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_DOWN) // downtrend
{
    if (IsWhiteCandle(InData, index)    && // the candle 0 is white
        IsBlackCandle(InData, index-1)  && // the candle -1 is black
        IsBlackCandle(InData, index-2)) // the candle -2 is black
    {
        if(IsBodyStrong(InData, index-1)) // the body of the candle -1 is strong
        {
            if(IsBodyStrong(InData, index-2)) // the body of the candle -2 is strong
            {
                if (InData[SC_LOW][index-2] > InData[SC_HIGH][index-1]) // a gap between the candles -1 and -2
                {
                    // OPEN of the candle 0 is inside the body of the candle -1
                    if ((InData[SC_OPEN][index] > InData[SC_LAST][index-1]) &&
                        (InData[SC_OPEN][index] < InData[SC_OPEN][index-1]))
                    {
                        // CLOSE of the candle 0 is higher than minimum of the candle -2
                        if(InData[SC_LAST][index]>InData[SC_LOW][index-2])
                        {
                            ret_flag = true;
                        }
                    }
                }
            }
        }
    }
}
return ret_flag;
}
/*=====*/

```

```

bool IsUpsideGapThreeMethods(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int
index)
{
    // UPSIDE GAP THREE METHODS
    // 1. Uptrend
    // 2. The candle [index] is black and its OPEN is inside the body of the candle [index-1]
    // 3. The candles [index-1] and [index-2] are white with a strong bodies
    // 4. There is a gap between the candles [index-1] and [index-2]
    // 5. The body of the candle [index] closes the gap between the candles [index-1] and [index-2]
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if( settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_UP) // uptrend
    {
        if (IsBlackCandle(InData, index)    && // the candle 0 is black
            IsWhiteCandle(InData, index-1)  && // the candle -1 is white
            IsWhiteCandle(InData, index-2)) // the candle -2 is white
        {
            if(IsBodyStrong(InData, index-1)) // the body of the candle -1 is strong
            {
                if(IsBodyStrong(InData, index-2)) // the body of the candle -1 is strong
                {
                    if (InData[SC_HIGH][index-2] < InData[SC_LOW][index-1]) // a gap between the candles -1 and -2
                    {
                        // OPEN of the candle 0 is inside the body of the candle -1
                        if ((InData[SC_OPEN][index] > InData[SC_OPEN][index-1]) &&
                            (InData[SC_OPEN][index] < InData[SC_LAST][index-1]))
                        {
                            // CLOSE of the candle 0 is lower than maximum of the candle -2

```

```

        if(InData[SC_LAST][index]<InData[SC_HIGH][index-2])
        {
            ret_flag = true;
        }
    }
}
}
}
}
}
return ret_flag;
}
/*=====*/

```

```

const double k_OnNeck_SimilarityPercent = 15.0;

```

```

bool IsOnNeck(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // ON NECK
    // 1. Downtrend
    // 2. The candle [index-1] is black and with a strong body
    // 3. The candle [index] is white
    // 4. CLOSE of the candle [index] is ALMOST equal the minimum of the candle [index-1]
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_DOWN) // downtrend
    {
        if(IsBlackCandle(InData, index-1)) // the candle -1 is black
        {
            if(IsBodyStrong(InData, index-1)) // the body of the candle -1 is strong
            {
                if(IsWhiteCandle(InData, index)) // the candle 0 is white
                {
                    // CLOSE of the candle 0 is ALMOST equal the minimum of the candle -1
                    if (IsNearEqual(InData[SC_LAST][index], InData[SC_LOW][index-1], InData, index,
k_OnNeck_SimilarityPercent))
                    {
                        ret_flag = true;
                    }
                }
            }
        }
    }
}
return ret_flag;
}
/*=====*/

```

```

const double k_InNeck_SimilarityPercent = 15.0;

```

```

bool IsInNeck(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // IN NECK
    // 1. Downtrend
    // 2. The candle [index-1] is black and with a strong body
    // 3. The candle [index] is white
    // 4. OPEN of the candle [index] is lower than the minimum of the candle [index-1]
    // 5. CLOSE of the candle [index] is a little bit higher or equal the CLOSE of the candle -1
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_DOWN) // downtrend
    {
        if(IsBlackCandle(InData, index-1)) // the candle -1 is black
        {

```

```

    if(IsBodyStrong(InData, index-1))    // the body of the candle -1 is strong
    {
        if(IsWhiteCandle(InData, index))    // the candle 0 is white
        {
            // OPEN of the candle 0 is lower than the minimum of the candle -1,
            // and CLOSE of the candle 0 is a higher or equal the CLOSE of the candle -1
            if ((InData[SC_LOW][index-1] > InData[SC_OPEN][index]) &&
                (InData[SC_LAST][index] >= InData[SC_LAST][index-1]))
            {
                // CLOSE of the candle 0 is equal the CLOSE of the candle -1
                if(IsNearEqual(InData[SC_LAST][index], InData[SC_LAST][index-1], InData, index,
k_InNeck_SimilarityPercent))
                {
                    ret_flag = true;
                }
                // CLOSE of the candle 0 is a little bit higher or equal the CLOSE of the candle -1
                else if((InData[SC_LAST][index]-InData[SC_LAST][index-1])<
                    PercentOfCandleLength(InData, index, k_InNeck_SimilarityPercent))
                {
                    ret_flag = true;
                }
            }
        }
    }
}
return ret_flag;
}
/*=====*/

```

```

const double BearishThrusting_SignificantlyLowerPercent = 10.0;

```

```

bool IsBearishThrusting(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // Bearish Thrusting
    // 1. Downtrend
    // 2. Candle [index-1] is black
    // 3. Candle [index] is white
    // 4. OPEN of the candle [index] SIGNIFICANTLY lower the minimum of the candle [index-1]
    // 5. CLOSE of the candle [index] is higher than the CLOSE of candle [index-1], but lower than the middle of body of
candle [index-1]
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if(settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_DOWN) // downtrend
    {
        if(IsBlackCandle(InData, index-1))    // candle -1 is black
        {
            if(IsWhiteCandle(InData, index))    // candle 0 is white
            {
                // OPEN of candle 0 is SIGNIFICANTLY lower than the minimum of candle -1
                if ((InData[SC_LOW][index-1] > InData[SC_OPEN][index]) &&
                    (InData[SC_LOW][index-1]-InData[SC_OPEN][index])>
                    PercentOfCandleLength(InData, index, BearishThrusting_SignificantlyLowerPercent))
                {
                    // CLOSE of candle 0 is higher than the CLOSE of -1, but lower than the middle of body of the candle -1
                    if ((InData[SC_LAST][index] > InData[SC_LAST][index-1]) &&
                        (InData[SC_LAST][index] < (InData[SC_LAST][index-1]+(InData[SC_OPEN][index-1]-InData[SC_LAST]
[index-1])/2)))
                    {
                        ret_flag = true;
                    }
                }
            }
        }
    }
}

```

```

    }
}
return ret_flag;
}
/*=====*/

const int k_MatHold_PAST_INDEX = 3;

bool IsMatHold(SCStudyInterfaceRef sc, const s_CandleStickPatternsFinderSettings& settings, int index)
{
    // MAT HOLD
    // 1. Candle [index] is white with strong body
    // 2. OPEN of th candle [index] is higher than CLOSE of candle [index-1]
    // 3. Candles [index-1] and [index-2] has weak bodies
    // 4. CLOSE of candle [index-2] is higher than the CLOSE of candle [index-1]
    // 5. Candle [index-X] has strong white body (where X is from 3 to PAST_INDEX+1)
    // 6. CLOSE of candle [index] is higher than CLOSE of candle [index-X]
    // 7. Uptrend till candle [index-X]
    // 8. Candles [index-Y] has weak bodies (where Y is from 1 to X-1)
    // 9. CLOSE of candle [index-Y] is higher than CLOSE of [index-Y-1] (where Y is from 1 to X-1)
    // 10. Minimum of candle [index-Y+1] is higher than maximum of candle [index-Y]
    SCBaseDataRef InData = sc.BaseData;
    bool ret_flag = false;
    if (IsWhiteCandle(InData, index) && // candle 0 is white
        (InData[SC_OPEN][index] > InData[SC_LAST][index-1])) // OPEN of candle 0 is higher than CLOSE of -1
    {
        if (IsBodyStrong(InData, index)) // body of the candle 0 is strong
        {
            if (InData[SC_LAST][index-2] > InData[SC_LAST][index-1]) // CLOSE of candle -2 is higher than CLOSE of
candle -1
            {
                // bodies of candles -1 and -2 are weak
                if ((!IsBodyStrong(InData, index-1)) && (!IsBodyStrong(InData, index-2)))
                {
                    int X = 0;
                    // searching for a strong white body
                    for (int i = 3; i < (k_MatHold_PAST_INDEX + 2); i++)
                    {
                        if (IsBodyStrong(InData, index-i) && IsWhiteCandle(InData, index-i))
                        {
                            // CLOSE of candle 0 is higher than CLOSE of candle -X
                            if (InData[SC_LAST][index] > InData[SC_LAST][index-i])
                            {
                                // found candle -X
                                X = i;
                            }
                            break;
                        }
                    }
                    if (X != 0)
                    {
                        // uptrend till candle -X
                        if (settings.UseTrendDetection == false || sc.Subgraph[TREND_FOR_PATTERNS][index] ==
CANDLESTICK_TREND_UP)
                        {
                            // Minimum of candle [index-Y+1] is higher than maximum of candle [index-Y]; candle [index-Y+1] is
black
                            if ((InData[SC_LOW][index-X+1] > InData[SC_HIGH][index-X]) &&
                                IsBlackCandle(InData, index-X+1))
                            {
                                ret_flag = true;
                                // checking the candles between two white candles
                                if ((IsBodyStrong(InData, index-1)) || // check if body is strong
                                    (InData[SC_HIGH][index-1] > InData[SC_HIGH][index-X]) || // check if maximum is higher than
maximum of candle -X

```

```

        (InData[SC_LOW][index-1] < InData[SC_LOW][index-X])) // check if minimum is lower than
minimum of candle -Ö
    {
        ret_flag = false;
    }
    else
    {
        for (int i = 2;i<X;i++)
        {
            // searching for a presence of a candle that does not match conditions
            if ((IsBodyStrong(InData,index-i)) || // check whether body is strong
                (InData[SC_LAST][index-i] < InData[SC_LAST][index-i+1])) // check whether CLOSE is
lower than the CLOSE of a next candle
            {
                ret_flag = false;
                break;
            }
        }
    }
}
}
}
}
}
}
}
}
}
return ret_flag;
}

```

```

/*=====*/
SCSFExport scsf_WoodiesZLR(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_LongTrade = sc.Subgraph[0];
    SCSubgraphRef Subgraph_ShortTrade = sc.Subgraph[1];
    SCSubgraphRef Subgraph_ChopzoneAverage = sc.Subgraph[2];

    SCInputRef Input_CCITrendRef = sc.Input[0];
    SCInputRef Input_SideWinderRef = sc.Input[1];
    SCInputRef Input_ChopzoneRef = sc.Input[2];
    SCInputRef Input_LowestCCILevelforShort = sc.Input[3];
    SCInputRef Input_HighestCCIDifferenceforShort = sc.Input[4];
    SCInputRef Input_LowestCCIDifferenceforLong = sc.Input[5];
    SCInputRef Input_HighestCCILevelforLong = sc.Input[6];
    SCInputRef Input_ChopzoneLength = sc.Input[7];
    SCInputRef Input_ChopzoneThreshold = sc.Input[8];
    SCInputRef Input_AlertNumber = sc.Input[9];

    if (sc.SetDefaults)
    {
        // Name and description
        sc.GraphName = "Woodies ZLR System";
        sc.StudyDescription = "Woodies ZLR for using in \"Woodies for Range Charts-New\" Study Collection.";

        // Settings
        sc.AutoLoop = 0;
        sc.GraphRegion = 1;
        sc.CalculationPrecedence = LOW_PREC_LEVEL;

        // Subgraphs
        Subgraph_LongTrade.Name = "Long Trade";
        Subgraph_LongTrade.DrawStyle = DRAWSTYLE_ARROW_UP;
        Subgraph_LongTrade.PrimaryColor = RGB(0,255,0);
        Subgraph_LongTrade.LineWidth = 2;
    }
}

```

```

Subgraph_LongTrade.DrawZeros = false;

Subgraph_ShortTrade.Name = "Short Trade";
Subgraph_ShortTrade.DrawStyle = DRAWSTYLE_ARROW_DOWN;
Subgraph_ShortTrade.PrimaryColor = RGB(255,0,0);
Subgraph_ShortTrade.LineWidth = 2;
Subgraph_ShortTrade.DrawZeros = false;

Subgraph_ChopzoneAverage.Name = "Chopzone Average";
Subgraph_ChopzoneAverage.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_ChopzoneAverage.PrimaryColor = RGB(255,255,0);
Subgraph_ChopzoneAverage.DrawZeros = false;

// Inputs
Input_CCITrendRef.Name = "Woodies CCI Trend Study Reference";
Input_CCITrendRef.SetStudyID(1);

Input_SideWinderRef.Name = "Sidewinder Study Reference";
Input_SideWinderRef.SetStudyID(1);

Input_ChopzoneRef.Name = "Chopzone Study Reference";
Input_ChopzoneRef.SetStudyID(1);

Input_LowestCCILevelforShort.Name = "Lowest CCI Level for Short";
Input_LowestCCILevelforShort.SetFloat(-120);

Input_HighestCCIDifferenceforShort.Name = "Highest CCI Difference for Short";
Input_HighestCCIDifferenceforShort.SetFloat(-15);

Input_LowestCCIDifferenceforLong.Name = "Lowest CCI Difference for Long";
Input_LowestCCIDifferenceforLong.SetFloat(15);

Input_HighestCCILevelforLong.Name = "Highest CCI Level for Long";
Input_HighestCCILevelforLong.SetFloat(120);

Input_ChopzoneThreshhold.Name = "Chopzone Threshold";
Input_ChopzoneThreshhold.SetFloat(4.5);

Input_ChopzoneLength.Name = "Chopzone Length";
Input_ChopzoneLength.SetInt(7);

Input_AlertNumber.Name = "Alert Number";
Input_AlertNumber.SetAlertSoundNumber(0);

return;
}

// see CCI Trend New study to find correct color values
enum CCITrendColors
{
    CCI_TREND_UP_COLOR = 3,
    CCI_TREND_DOWN_COLOR = 1
};

// see Sidewinder study to find correct color values
enum SidewinderColors
{
    SW_TREND_FLAT = 0
};

const int UniqueID = 1546833579;
const int LastBarUniqueID = UniqueID+1;

```

```

SCFloatArray CCITrendColor;
sc.GetStudyArrayUsingID(Input_CCITrendRef.GetStudyID(), 8, CCITrendColor);

SCFloatArray CCI;
sc.GetStudyArrayUsingID(Input_CCITrendRef.GetStudyID(), 0, CCI);

SCFloatArray SidewinderColors;
sc.GetStudyArrayUsingID(Input_SideWinderRef.GetStudyID(), 3, SidewinderColors);

SCFloatArray ChopzoneOutput;
sc.GetStudyArrayUsingID(Input_ChopzoneRef.GetStudyID(), 2, ChopzoneOutput);

SCFloatArray HooksConditions;
sc.GetStudyArrayUsingID(Input_CCITrendRef.GetStudyID(), 9, HooksConditions);

bool ZLRLongTrade = false;
bool ZLRShortTrade = false;

for (int index = sc.UpdateStartIndex; index < sc.ArraySize; index++)
{
    if(index == 0)
        continue;

    float CCIDiff = CCI[index] - CCI[index-1];

    ZLRLongTrade = false;
    ZLRShortTrade = false;
    Subgraph_LongTrade[index] = 0;
    Subgraph_ShortTrade[index] = 0;

    if(sc.GetBarHasClosedStatus(index)==BHCS_BAR_HAS_NOT_CLOSED)
    {
        sc.DeleteACSCChartDrawing(sc.ChartNumber, TOOL_DELETE_CHARTDRAWING, LastBarUniqueID);
    }

    // Clear the alert state
    sc.SetAlert(0);

    // check ZLR LONG TRADE CONDITION
    if(CCITrendColor[index] == CCI_TREND_UP_COLOR) // trend is UP
    {
        if(CCIDiff < Input_LowestCCIDifferenceforLong.GetFloat())
            continue;

        if(CCI[index] > Input_HighestCCILevelforLong.GetFloat())
            continue;

        if(CCI[index] < 0)
            continue;

        if(SidewinderColors[index] == SW_TREND_FLAT)
            continue;

        if(HooksConditions[index] <= 0)
            continue;

        // check Chopzone
        SCFloatArrayRef ChopzoneAVG = Subgraph_ChopzoneAverage;
        sc.WeightedMovingAverage(ChopzoneOutput,ChopzoneAVG,index,Input_ChopzoneLength.GetInt());

        float ChopzoneAVGval = ChopzoneAVG[index];

        if (ChopzoneAVG[index] >= Input_ChopzoneThreshold.GetFloat())

```



```

{
    ZLRLongTrade = true;
}

if(!ZLRLongTrade)
    continue;
}
else if(CCI_TrendColor[index] == CCI_TREND_DOWN_COLOR) // trend is DOWN
{
    if(CCIDiff > Input_HighestCCIDifferenceforShort.GetFloat())
        continue;

    if(CCI[index] < Input_LowestCCILevelforShort.GetFloat())
        continue;

    if(CCI[index] >= 0)
        continue;

    if(SidewinderColors[index] == SW_TREND_FLAT)
        continue;

    if(HooksConditions[index] >= 0)
        continue;

    // check Chopzone
    SFloatArrayRef ChopzoneAVG = Subgraph_ChopzoneAverage;
    sc.WeightedMovingAverage(ChopzoneOutput,ChopzoneAVG,index,Input_ChopzoneLength.GetInt());

    if (ChopzoneAVG[index] <= -1 * Input_ChopzoneThreshold.GetFloat())
    {
        ZLRShortTrade = true;
    }

    if(!ZLRShortTrade)
        continue;
}
else
    continue;

// put values to arrays
s_UseTool Tool;
Tool.Clear(); // reset tool structure for our next use
Tool.ChartNumber = sc.ChartNumber;
Tool.DrawingType = DRAWING_TEXT;
Tool.Region = sc.GraphRegion;
Tool.DrawingType = DRAWING_TEXT;
Tool.ReverseTextColor = 0;
Tool.Text = "ZLR";
Tool.BeginDateTime = sc.BaseDateTimeIn[index];
Tool.LineNumber = UniqueID;
Tool.FontSize = 8;
Tool.AddMethod = UTAM_ADD_ALWAYS;

if(sc.GetBarHasClosedStatus(index)==BHCS_BAR_HAS_NOT_CLOSED)
{
    Tool.LineNumber = LastBarUniqueID;
}

if(ZLRLongTrade)
{
    Subgraph_LongTrade[index] = -200;
}

```

```

Tool.BeginValue    = -200;
Tool.Color         = Subgraph_LongTrade.PrimaryColor;
Tool.TextAlignment = DT_CENTER | DT_BOTTOM;

sc.UseTool(Tool);

if (Input_AlertNumber.GetAlertSoundNumber() > 0)
    sc.SetAlert(Input_AlertNumber.GetAlertSoundNumber() - 1, "Long Trade" );
}

if(ZLRShortTrade)
{
    Subgraph_ShortTrade[index] = 200;

    Tool.BeginValue    = 200;
    Tool.Color         = Subgraph_ShortTrade.PrimaryColor;
    Tool.TextAlignment = DT_CENTER | DT_TOP;

    sc.UseTool(Tool);

    if (Input_AlertNumber.GetAlertSoundNumber() > 0)
        sc.SetAlert(Input_AlertNumber.GetAlertSoundNumber() - 1, "Short Trade");
}
}

}

/*=====*/
SCSFExport scsf_ChopZone(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Czi1 = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Czi2 = sc.Subgraph[1];
    SCSubgraphRef Subgraph_OutWorksheets = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Ema = sc.Subgraph[3];
    SCSubgraphRef Subgraph_ColorControl3 = sc.Subgraph[4];
    SCSubgraphRef Subgraph_ColorControl4 = sc.Subgraph[5];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_PosCzi1 = sc.Input[2];
    SCInputRef Input_PosCzi2 = sc.Input[3];
    SCInputRef Input_EmaLength = sc.Input[5];
    SCInputRef Input_TickValue = sc.Input[6];

    if(sc.SetDefaults)
    {
        sc.GraphName = "Chop Zone";
        sc.StudyDescription = "Chop Zone";

        sc.AutoLoop = 1;

        Subgraph_Czi1.Name = "CZI Top";
        Subgraph_Czi1.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Czi1.LineWidth = 2;
        Subgraph_Czi1.PrimaryColor = RGB(255,255,0);
        Subgraph_Czi1.SecondaryColor = RGB(170,255,85);
        Subgraph_Czi1.SecondaryColorUsed = true;
        Subgraph_Czi1.DrawZeros = false;

        Subgraph_Czi2.Name = "CZI Bottom";
        Subgraph_Czi2.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Czi2.LineWidth = 2;
        Subgraph_Czi2.PrimaryColor = RGB(85,255,170);
        Subgraph_Czi2.SecondaryColor = RGB(0,255,255);
        Subgraph_Czi2.SecondaryColorUsed = true;
    }
}

```

```

Subgraph_Czi2.DrawZeros = false;

Subgraph_OutWorksheets.Name = "Output for Spreadsheets";
Subgraph_OutWorksheets.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_OutWorksheets.PrimaryColor = RGB(255,255,255);
Subgraph_OutWorksheets.DrawZeros = false;

Subgraph_ColorControl3.Name = "Down Colors 1 & 2";
Subgraph_ColorControl3.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_ColorControl3.PrimaryColor = RGB(213,170,0);
Subgraph_ColorControl3.SecondaryColor = RGB(170,85,0);
Subgraph_ColorControl3.SecondaryColorUsed = true;
Subgraph_ColorControl3.DrawZeros = false;

Subgraph_ColorControl4.Name = "Down Color 3";
Subgraph_ColorControl4.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_ColorControl4.PrimaryColor = RGB(128,0,0);
Subgraph_ColorControl4.DrawZeros = false;

Input_InputData.Name = "Input Data";
Input_InputData.SetInputDataIndex(SC_LAST);

Input_PosCzi1.Name = "Y Pos CZI 1";
Input_PosCzi1.SetFloat(100.0f);

Input_PosCzi2.Name = "Y Pos CZI 2";
Input_PosCzi2.SetFloat(-100.0f);

Input_EmaLength.Name = "EMA Length";
Input_EmaLength.SetInt(34);
Input_EmaLength.SetIntLimits(1,MAX_STUDY_LENGTH);

Input_TickValue.Name = "Tick Value (Enter number to override default)";
Input_TickValue.SetFloat(0.0f);

return;
}

if (Subgraph_Czi1.PrimaryColor == RGB(255,255,0)
    && Subgraph_Czi1.SecondaryColor == RGB(0,255,255))
{
    Subgraph_Czi1.PrimaryColor = RGB(255,255,0);
    Subgraph_Czi1.SecondaryColor = RGB(170,255,85);
    Subgraph_Czi2.PrimaryColor = RGB(85,255,170);
    Subgraph_Czi2.SecondaryColor = RGB(0,255,255);
    Subgraph_ColorControl3.PrimaryColor = RGB(213,170,0);
    Subgraph_ColorControl3.SecondaryColor = RGB(170,85,0);
    Subgraph_ColorControl4.PrimaryColor = RGB(128,0,0);
}

float inTickSize = sc.TickSize;

if(Input_TickValue.GetFloat() != 0)
    inTickSize = Input_TickValue.GetFloat();

float &PriorTickSize = sc.GetPersistentFloat(1);

if(PriorTickSize != inTickSize)
{
    sc.UpdateStartIndex = 0;
    PriorTickSize = inTickSize;
}

// Colors

```

```

uint32_t ColorLevel = Subgraph_Czi1.PrimaryColor;
uint32_t ColorUp1 = Subgraph_Czi1.SecondaryColor;
uint32_t ColorUp2 = Subgraph_Czi2.PrimaryColor;
uint32_t ColorUp3 = Subgraph_Czi2.SecondaryColor;
uint32_t ColorDown1 = Subgraph_ColorControl3.PrimaryColor;
uint32_t ColorDown2 = Subgraph_ColorControl3.SecondaryColor;
uint32_t ColorDown3 = Subgraph_ColorControl4.PrimaryColor;

sc.DataStartIndex = Input_EmaLength.GetInt();

int pos = sc.Index;

sc.ExponentialMovAvg(sc.BaseData[Input_InputData.GetInputDataIndex()], Subgraph_Ema, pos,
Input_EmaLength.GetInt());

int TickDifference = sc.Round((sc.BaseData[Input_InputData.GetInputDataIndex()][pos] -
Subgraph_Ema[pos])/inTickSize);

Subgraph_Czi1[pos] = Input_PosCzi1.GetFloat();
Subgraph_Czi2[pos] = Input_PosCzi2.GetFloat();

/*
cyan   Woodies CZI >= 5
dark green Woodies CZI >= 4 And Woodies CZI < 5
light green   Woodies CZI >= 3 And Woodies CZI < 4
lime         Woodies CZI > 1 And Woodies CZI < 3

yellow      -1 <= (Woodies CZI) <= 1

light orange Woodies CZI > -3 And Woodies CZI < -1
dark orange  Woodies CZI > -4 And Woodies CZI <= -3
light red    Woodies CZI > -5 And Woodies CZI <= -4
brown        <=-5
*/

if (TickDifference <= 1 && TickDifference >= -1)
    Subgraph_Czi1.DataColor[pos] = Subgraph_Czi2.DataColor[pos] = ColorLevel;
else if (TickDifference > 1 && TickDifference < 3)
    Subgraph_Czi1.DataColor[pos] = Subgraph_Czi2.DataColor[pos] = ColorUp1;
else if (TickDifference >= 3 && TickDifference < 4)
    Subgraph_Czi1.DataColor[pos] = Subgraph_Czi2.DataColor[pos] = ColorUp2;
else if (TickDifference >= 4)
    Subgraph_Czi1.DataColor[pos] = Subgraph_Czi2.DataColor[pos] = ColorUp3;

else if (TickDifference < -1 && TickDifference > -3)
    Subgraph_Czi1.DataColor[pos] = Subgraph_Czi2.DataColor[pos] = ColorDown1;
else if (TickDifference <= -3 && TickDifference > -4)
    Subgraph_Czi1.DataColor[pos] = Subgraph_Czi2.DataColor[pos] = ColorDown2;
else if (TickDifference <= -4)
    Subgraph_Czi1.DataColor[pos] = Subgraph_Czi2.DataColor[pos] = ColorDown3;

// for worksheets
Subgraph_OutWorksheets[pos] = static_cast<float>(TickDifference);
}

/*=====*/
SCSFExport scsf_StochBands(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_SubgraphK = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SubgraphD = sc.Subgraph[1];

```

```

SCSubgraphRef Subgraph_SubgraphK2 = sc.Subgraph[2];
SCSubgraphRef Subgraph_SubgraphD2 = sc.Subgraph[3];

SCSubgraphRef Subgraph_SubgraphBackgroundCalcForStoch1 = sc.Subgraph[8];
SCSubgraphRef Subgraph_SubgraphBackgroundCalcForStoch2 = sc.Subgraph[9];

SCInputRef Input_InputDataType = sc.Input[0];
SCInputRef Input_InputMAType = sc.Input[1];
SCInputRef Input_InputLength = sc.Input[2];
SCInputRef Input_InputK = sc.Input[3];
SCInputRef Input_InputD = sc.Input[4];
SCInputRef Input_InputLength2 = sc.Input[5];
SCInputRef Input_InputK2 = sc.Input[6];
SCInputRef Input_InputD2 = sc.Input[7];

if (sc.SetDefaults)
{
    sc.GraphName = "Stochastic Bands";
    sc.StudyDescription = "Stochastic Bands";

    sc.AutoLoop = 1;

    Subgraph_SubgraphK.Name = "K";
    Subgraph_SubgraphK.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_SubgraphK.LineWidth = 2;
    Subgraph_SubgraphK.PrimaryColor = RGB(0, 200, 0);
    Subgraph_SubgraphK.SecondaryColor = RGB(0, 200, 0);
    Subgraph_SubgraphK.SecondaryColorUsed = 1;
    Subgraph_SubgraphK.DrawZeros = true;

    Subgraph_SubgraphD.Name = "D";
    Subgraph_SubgraphD.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_SubgraphD.LineWidth = 2;
    Subgraph_SubgraphD.PrimaryColor = RGB(0, 200, 0);
    Subgraph_SubgraphD.SecondaryColor = RGB(0, 200, 0);
    Subgraph_SubgraphD.SecondaryColorUsed = 1;
    Subgraph_SubgraphD.DrawZeros = true;

    Subgraph_SubgraphK2.Name = "K2";
    Subgraph_SubgraphK2.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_SubgraphK2.LineWidth = 2;
    Subgraph_SubgraphK2.PrimaryColor = RGB(0, 200, 0);
    Subgraph_SubgraphK2.SecondaryColor = RGB(0, 200, 0);
    Subgraph_SubgraphK2.SecondaryColorUsed = 1;
    Subgraph_SubgraphK2.DrawZeros = true;

    Subgraph_SubgraphD2.Name = "D2";
    Subgraph_SubgraphD2.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_SubgraphD2.LineWidth = 2;
    Subgraph_SubgraphD2.PrimaryColor = RGB(0, 200, 0);
    Subgraph_SubgraphD2.SecondaryColor = RGB(0, 200, 0);
    Subgraph_SubgraphD2.SecondaryColorUsed = 1;
    Subgraph_SubgraphD2.DrawZeros = true;

    Input_InputDataType.Name = "Input Data";
    Input_InputDataType.SetInputDataIndex(SC_HL_AVG);

    Input_InputMAType.Name = "Moving Average Type";
    Input_InputMAType.SetMovAvgType(MOVAVGTYPE_SIMPLE);

    Input_InputLength.Name = "Length";
    Input_InputLength.SetInt(7);

    Input_InputK.Name = "K";

```

```

Input_InputK.SetInt(3);

Input_InputD.Name = "D";
Input_InputD.SetInt(3);

Input_InputLength2.Name = "Length2";
Input_InputLength2.SetInt(21);

Input_InputK2.Name = "K2";
Input_InputK2.SetInt(10);

Input_InputD2.Name = "D2";
Input_InputD2.SetInt(4);

sc.DataStartIndex = 50;

return;
}

if (sc.Index < 1)
    return;

int Length = Input_InputLength.GetInt();
int K = Input_InputK.GetInt();
int D = Input_InputD.GetInt();

int Length2 = Input_InputLength2.GetInt();
int K2 = Input_InputK2.GetInt();
int D2 = Input_InputD2.GetInt();

int DataIndex = Input_InputDataType.GetInputDataIndex();
int MAType = Input_InputMAType.GetMovAvgType();

// First Stochastic
sc.Stochastic(sc.BaseDataIn[DataIndex], sc.BaseDataIn[DataIndex], sc.BaseDataIn[DataIndex],
    Subgraph_SubgraphBackgroundCalcForStoch1, sc.Index,
    Length, K, D, MAType);

float sk = Subgraph_SubgraphBackgroundCalcForStoch1.Arrays[0][sc.Index];
float sd = Subgraph_SubgraphBackgroundCalcForStoch1.Arrays[1][sc.Index];

// Second Stochastic
sc.Stochastic(sc.BaseDataIn[DataIndex], sc.BaseDataIn[DataIndex], sc.BaseDataIn[DataIndex],
    Subgraph_SubgraphBackgroundCalcForStoch2, sc.Index,
    Length2, K2, D2, MAType);

float sk2 = Subgraph_SubgraphBackgroundCalcForStoch2.Arrays[0][sc.Index];
float sd2 = Subgraph_SubgraphBackgroundCalcForStoch2.Arrays[1][sc.Index];

Subgraph_SubgraphK[sc.Index] = sk;
Subgraph_SubgraphD[sc.Index] = sd;

if (sk > sd)
{
    Subgraph_SubgraphK.DataColor[sc.Index] = Subgraph_SubgraphK.PrimaryColor;
    Subgraph_SubgraphD.DataColor[sc.Index] = Subgraph_SubgraphD.PrimaryColor;
}
else
{
    Subgraph_SubgraphK.DataColor[sc.Index] = Subgraph_SubgraphK.SecondaryColor;
    Subgraph_SubgraphD.DataColor[sc.Index] = Subgraph_SubgraphD.SecondaryColor;
}

```

```

Subgraph_SubgraphK2[sc.Index] = sk2;
Subgraph_SubgraphD2[sc.Index] = sd2;

if (sk2 > sd2)
{
    Subgraph_SubgraphK2.DataColor[sc.Index] = Subgraph_SubgraphK2.PrimaryColor;
    Subgraph_SubgraphD2.DataColor[sc.Index] = Subgraph_SubgraphD2.PrimaryColor;
}
else
{
    Subgraph_SubgraphK2.DataColor[sc.Index] = Subgraph_SubgraphK2.SecondaryColor;
    Subgraph_SubgraphD2.DataColor[sc.Index] = Subgraph_SubgraphD2.SecondaryColor;
}
}

/*=====*/
SCSFExport scsf_BLine(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Baseline = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Oversold = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Overbought = sc.Subgraph[2];

    SCSubgraphRef Subgraph_BackgroundCalcForStoch = sc.Subgraph[5];

    SCInputRef Input_MATypeSimple = sc.Input[0];
    SCInputRef Input_Length = sc.Input[1];
    SCInputRef Input_K = sc.Input[2];
    SCInputRef Input_D = sc.Input[3];
    SCInputRef Input_UseHL = sc.Input[4];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Buffys B-Line";
        sc.StudyDescription = "Buffys B-Line";

        sc.AutoLoop = 1;

        Subgraph_Baseline.Name = "Baseline";
        Subgraph_Baseline.DrawStyle=DRAWSTYLE_LINE;
        Subgraph_Baseline.PrimaryColor=RGB(255,255,255);

        Subgraph_Oversold.Name = "Oversold";
        Subgraph_Oversold.DrawStyle=DRAWSTYLE_POINT;
        Subgraph_Oversold.PrimaryColor=RGB(0,255,0);

        Subgraph_Overbought.Name = "Overbought";
        Subgraph_Overbought.DrawStyle=DRAWSTYLE_POINT;
        Subgraph_Overbought.PrimaryColor=RGB(255,0,0);

        Input_MATypeSimple.Name = "MA Type: Use Simple instead of Ensign Exponential";
        Input_MATypeSimple.SetYesNo(false);

        Input_Length.Name = "Length";
        Input_Length.SetInt(21);

        Input_K.Name = "K";
        Input_K.SetInt(10);

        Input_D.Name = "D";
        Input_D.SetInt(4);

        Input_UseHL.Name = "Use HL Average Instead of Close";
        Input_UseHL.SetYesNo(false);
    }
}

```

```

    sc.DataStartIndex = 50;

    return;
}

int Length = Input_Length.GetInt();
int K = Input_K.GetInt();
int D = Input_D.GetInt();

int DataIndex = SC_LAST;
if (Input_UseHL.GetBoolean())
    DataIndex = SC_HL_AVG;

sc.Stochastic(sc.BaseDataIn[DataIndex], sc.BaseDataIn[DataIndex],
sc.BaseDataIn[DataIndex], Subgraph_BackgroundCalcForStoch, sc.Index,
    Length, K, D, MOVAVGTYPE_SIMPLE);

float FastK = Subgraph_BackgroundCalcForStoch[sc.Index];

int skArrayIndex, sdArrayIndex;
if (Input_MATypeSimple.GetBoolean())
{
    skArrayIndex = 0;
    sdArrayIndex = 1;
}
else
{
    skArrayIndex = 2;
    sdArrayIndex = 3;
}

float PriorStochasticK, PriorStochasticD;
if (sc.Index)
{
    PriorStochasticK = Subgraph_BackgroundCalcForStoch.Arrays[skArrayIndex][sc.Index-1];
    PriorStochasticD = Subgraph_BackgroundCalcForStoch.Arrays[sdArrayIndex][sc.Index-1];
}
else
    PriorStochasticK = PriorStochasticD = sc.BaseDataIn[DataIndex][0];

if (Input_MATypeSimple.GetBoolean() == false)
{
    // Calculate Ensign Exponential Moving Average
    float temp = Subgraph_BackgroundCalcForStoch.Arrays[2][sc.Index] = (FastK-PriorStochasticK)/K +
PriorStochasticK;
    Subgraph_BackgroundCalcForStoch.Arrays[3][sc.Index] = (temp-PriorStochasticD)/D + PriorStochasticD;
}

float sk = Subgraph_BackgroundCalcForStoch.Arrays[skArrayIndex][sc.Index];
float sd = Subgraph_BackgroundCalcForStoch.Arrays[sdArrayIndex][sc.Index];

Subgraph_Baseline[sc.Index] = sk;
if(sk>=80)
    Subgraph_Oversold[sc.Index] = sk;
else if (sk <= 20)
    Subgraph_Overbought[sc.Index] = sk;
}

/*=====*/
SCSFExport scsf_BuffyMALines(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_SMA1 = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SMA2 = sc.Subgraph[1];

```



```

SCSubgraphRef Subgraph_WMA1 = sc.Subgraph[2];
SCSubgraphRef Subgraph_WMA2 = sc.Subgraph[3];

SCSubgraphRef Subgraph_BackgroundCalcForWMA1 = sc.Subgraph[6];
SCSubgraphRef Subgraph_BackgroundCalcForWMA2 = sc.Subgraph[7];

SCSubgraphRef Subgraph_BackgroundCalcForSMA1 = sc.Subgraph[8];
SCSubgraphRef Subgraph_BackgroundCalcForSMA2 = sc.Subgraph[9];

SCInputRef Input_SMA1DataInput = sc.Input[0];
SCInputRef Input_SMA1Length = sc.Input[1];
SCInputRef Input_SMA2DataInput = sc.Input[2];
SCInputRef Input_SMA2Length = sc.Input[3];

SCInputRef Input_WMA1DataInput = sc.Input[4];
SCInputRef Input_WMA1Length = sc.Input[5];
SCInputRef Input_WMA2DataInput = sc.Input[6];
SCInputRef Input_WMA2Length = sc.Input[7];

if (sc.SetDefaults)
{
    sc.GraphName = "Buffy's MA Lines";
    sc.StudyDescription = "Buffy's MA Lines";

    sc.AutoLoop = 1;

    Subgraph_SMA1.Name = "SMA1Dn";
    Subgraph_SMA1.DrawStyle = DRAWSTYLE_STAIR_STEP;
    Subgraph_SMA1.LineWidth = 5;
    Subgraph_SMA1.PrimaryColor = RGB(0, 0, 255);
    Subgraph_SMA1.DrawZeros = false;

    Subgraph_SMA2.Name = "SMA2Up";
    Subgraph_SMA2.DrawStyle = DRAWSTYLE_STAIR_STEP;
    Subgraph_SMA2.LineWidth = 5;
    Subgraph_SMA2.PrimaryColor = RGB(255, 255, 0);
    Subgraph_SMA2.DrawZeros = false;

    Subgraph_WMA1.Name = "WMA1Up";
    Subgraph_WMA1.DrawStyle = DRAWSTYLE_STAIR_STEP;
    Subgraph_WMA1.LineWidth = 2;
    Subgraph_WMA1.PrimaryColor = RGB(0, 0, 0);
    Subgraph_WMA1.DrawZeros = false;

    Subgraph_WMA2.Name = "WMA2Dn";
    Subgraph_WMA2.DrawStyle = DRAWSTYLE_STAIR_STEP;
    Subgraph_WMA2.LineWidth = 2;
    Subgraph_WMA2.PrimaryColor = RGB(0, 0, 0);
    Subgraph_WMA2.DrawZeros = false;

    Input_SMA1DataInput.Name = "SMA1 Input Data";
    Input_SMA1DataInput.SetInputDataIndex(SC_HIGH);

    Input_SMA1Length.Name = "SMA1 Length";
    Input_SMA1Length.SetInt(12);
    Input_SMA1Length.SetIntLimits(1, MAX_STUDY_LENGTH);

    Input_SMA2DataInput.Name = "SMA2 Input Data";
    Input_SMA2DataInput.SetInputDataIndex(SC_LOW);

    Input_SMA2Length.Name = "SMA2 Length";
    Input_SMA2Length.SetInt(12);
    Input_SMA2Length.SetIntLimits(1, MAX_STUDY_LENGTH);

```

```

Input_WMA1DataInput.Name = "WMA1 Input Data";
Input_WMA1DataInput.SetInputDataIndex(SC_HIGH);

Input_WMA1Length.Name = "WMA1 (Rounded To Tick): Length";
Input_WMA1Length.SetInt(6);
Input_WMA1Length.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_WMA2DataInput.Name = "WMA2 Input Data";
Input_WMA2DataInput.SetInputDataIndex(SC_LOW);

Input_WMA2Length.Name = "WMA2 (Rounded To Tick): Length";
Input_WMA2Length.SetInt(6);
Input_WMA2Length.SetIntLimits(1, MAX_STUDY_LENGTH);

sc.DataStartIndex = 10;

return;
}

sc.SimpleMovAvg(sc.BaseDataIn[Input_SMA1DataInput.GetInputDataIndex()], Subgraph_BackgroundCalcForSMA1,
sc.Index, Input_SMA1Length.GetInt());
sc.SimpleMovAvg(sc.BaseDataIn[Input_SMA2DataInput.GetInputDataIndex()], Subgraph_BackgroundCalcForSMA2,
sc.Index, Input_SMA2Length.GetInt());

sc.WeightedMovingAverage(sc.BaseDataIn[Input_WMA1DataInput.GetInputDataIndex()],
Subgraph_BackgroundCalcForWMA1, sc.Index, Input_WMA1Length.GetInt());
sc.WeightedMovingAverage(sc.BaseDataIn[Input_WMA2DataInput.GetInputDataIndex()],
Subgraph_BackgroundCalcForWMA2, sc.Index, Input_WMA2Length.GetInt());

if (sc.Index < 10)
return; // nothing else to do

// If Falling, Show High SMA
if (Subgraph_BackgroundCalcForSMA1[sc.Index-1] >= Subgraph_BackgroundCalcForSMA1[sc.Index])
Subgraph_SMA1[sc.Index] = Subgraph_BackgroundCalcForSMA1[sc.Index];

// If Rising, Show Low SMA
if (Subgraph_BackgroundCalcForSMA2[sc.Index-1] <= Subgraph_BackgroundCalcForSMA2[sc.Index])
Subgraph_SMA2[sc.Index] = Subgraph_BackgroundCalcForSMA2[sc.Index];

// If Falling, Show High WMA
if (Subgraph_BackgroundCalcForWMA1[sc.Index-1] >= Subgraph_BackgroundCalcForWMA1[sc.Index])
Subgraph_WMA1[sc.Index] = static_cast<float>
(sc.RoundToTickSize(Subgraph_BackgroundCalcForWMA1[sc.Index], sc.TickSize));

// If Rising, Show Low WMA
if (Subgraph_BackgroundCalcForWMA2[sc.Index-1] <= Subgraph_BackgroundCalcForWMA2[sc.Index])
Subgraph_WMA2[sc.Index] = static_cast<float>
(sc.RoundToTickSize(Subgraph_BackgroundCalcForWMA2[sc.Index], sc.TickSize));
}

/*****/
SCSFExport scsf_AutomaticTrendlines(SCStudyInterfaceRef sc)
{
SCSubgraphRef Subgraph_UpTrendline = sc.Subgraph[0];
SCSubgraphRef Subgraph_DownTrendline = sc.Subgraph[1];

SCInputRef Input_LengthInput = sc.Input[0];
SCInputRef Input_SkipLastBars = sc.Input[1];
SCInputRef Input_ShowTrendlineUp = sc.Input[3];
SCInputRef Input_ShowTrendlineDown = sc.Input[4];
SCInputRef Input_NewBarsBeforeAdjust = sc.Input[5];
SCInputRef Input_ForwardProjectAutoTrendLines = sc.Input[6];

```

```

SCInputRef Input_VersionUpdate = sc.Input[7];
SCInputRef Input_InputDataHigh = sc.Input[8];
SCInputRef Input_InputDataLow = sc.Input[9];

if (sc.SetDefaults)
{
    sc.GraphName = "Automatic Trendlines";

    sc.GraphRegion = 0;
    sc.AutoLoop = 0;
    sc.ValueFormat = VALUEFORMAT_INHERITED;

    // initialize subgraphs
    Subgraph_UpTrendline.Name = "Up Trendline";
    Subgraph_UpTrendline.PrimaryColor = RGB(0, 128, 255);
    Subgraph_UpTrendline.LineWidth = 2;
    Subgraph_UpTrendline.DrawZeros = false;

    Subgraph_DownTrendline.Name = "Down Trendline";
    Subgraph_DownTrendline.PrimaryColor = RGB(0, 128, 255);
    Subgraph_DownTrendline.LineWidth = 2;
    Subgraph_DownTrendline.DrawZeros = false;

    // Number of bars to use when determining the trendline
    Input_LengthInput.Name = "Length";
    Input_LengthInput.SetInt(50);
    Input_LengthInput.SetIntLimits(2, MAX_STUDY_LENGTH);

    // Number of bars back from the end to begin consideration for the trendline. This allows the user to have the most
    recent X bars ignored when determining the trendline
    Input_SkipLastBars.Name = "Skip Last N Bars";
    Input_SkipLastBars.SetInt(0);
    Input_SkipLastBars.SetIntLimits(0, MAX_STUDY_LENGTH);

    Input_ShowTrendlineUp.Name = "Show Up Trendline";
    Input_ShowTrendlineUp.SetYesNo(1);

    Input_ShowTrendlineDown.Name = "Show Down Trendline";
    Input_ShowTrendlineDown.SetYesNo(1);

    Input_NewBarsBeforeAdjust.Name = "Number Of New Bars Before Auto-Adjustment";
    Input_NewBarsBeforeAdjust.SetInt(5);
    Input_NewBarsBeforeAdjust.SetIntLimits(0, MAX_STUDY_LENGTH);

    Input_ForwardProjectAutoTrendLines.Name = "Forward Project Automatic Trendlines";
    Input_ForwardProjectAutoTrendLines.SetYesNo(1);

    Input_InputDataHigh.Name = "Input Data High";
    Input_InputDataHigh.SetInputDataIndex(SC_HIGH);

    Input_InputDataLow.Name = "Input Data Low";
    Input_InputDataLow.SetInputDataIndex(SC_LOW);

    // version update
    Input_VersionUpdate.SetInt(2);

    return;
}

if (Input_VersionUpdate.GetInt() < 1)
{
    Input_ForwardProjectAutoTrendLines.SetYesNo(1);
    Input_VersionUpdate.SetInt(1);
}

```

```

if (Input_VersionUpdate.GetInt() < 2)
{
    Input_InputDataHigh.SetInputDataIndex(SC_HIGH);
    Input_InputDataLow.SetInputDataIndex(SC_LOW);
}

int& r_UpLineIndex1 = sc.GetPersistentInt(1);
int& r_UpLineIndex2 = sc.GetPersistentInt(2);
int& r_DownLineIndex1 = sc.GetPersistentInt(3);
int& r_DownLineIndex2 = sc.GetPersistentInt(4);
int& r_PrevIndex = sc.GetPersistentInt(5);

float& r_UpLine_A = sc.GetPersistentFloat(1);
float& r_UpLine_B = sc.GetPersistentFloat(2);
float& r_DownLine_A = sc.GetPersistentFloat(3);
float& r_DownLine_B = sc.GetPersistentFloat(4);

int NumberOfExtendedElements = 0;

if(Input_ForwardProjectAutoTrendLines.GetYesNo() != 0)
{
    NumberOfExtendedElements = 10 ;
    Subgraph_UpTrendline.ExtendedArrayElementsToGraph = 10;
    Subgraph_DownTrendline.ExtendedArrayElementsToGraph = 10;
}
else
{
    NumberOfExtendedElements = 0;
    Subgraph_UpTrendline.ExtendedArrayElementsToGraph = 0;
    Subgraph_DownTrendline.ExtendedArrayElementsToGraph = 0;
}

// set index from which study will look back
int LastIndex = sc.ArraySize - 1;
int LastExtendedIndex = LastIndex + NumberOfExtendedElements;
int StartIndex = LastIndex - Input_SkipLastBars.GetInt();

SCFloatArrayRef Highs = sc.BaseData[Input_InputDataHigh.GetInputDataIndex()];
SCFloatArrayRef Lows = sc.BaseData[Input_InputDataLow.GetInputDataIndex()];

if(sc.UpdateStartIndex == 0)
{
    // On initial calculation set r_PrevIndex to last index in array
    // StartIndex already set to correct value
    r_PrevIndex = sc.ArraySize-1;

    r_UpLineIndex1 = 0;
    r_UpLineIndex2 = 0;
    r_DownLineIndex1 = 0;
    r_DownLineIndex2 = 0;
    r_PrevIndex = -1;
}
else
{
    // set initial value of variable or adjust on replay
    if(r_PrevIndex == -1 || r_PrevIndex > LastIndex)
        r_PrevIndex = LastIndex;

    // perform adjust only after specified number of new bars were added
    int BarIndexDifference = LastIndex - r_PrevIndex;

    // if we should readjust trendline on each bar's changing then
    // do not any additional check
    if(Input_NewBarsBeforeAdjust.GetInt() > 0)
    {

```

```

// if NewBarsBeforeAdjust greater then 0 then
// we should readjust trendline only when Nth bar will be closed
// so if we should readjust at 3rd bar we should wait when 3rd bar will be closed and then readjust trendlines
if ((BarIndexDifference < Input_NewBarsBeforeAdjust.GetInt()))
{
    // we shouldn't readjust trendline
    // Only extend it to the last bar
    if(r_UpLineIndex2 < LastExtendedIndex || r_DownLineIndex2 < LastExtendedIndex)
    {
        if(r_UpLineIndex2 > 0)
        {
            for(int Index=r_UpLineIndex1; Index<=LastExtendedIndex; Index++ )
            {
                Subgraph_UpTrendline[Index] = static_cast<float>(r_UpLine_A * Index + r_UpLine_B);
            }

            r_UpLineIndex2 = LastExtendedIndex;
        }

        if(r_DownLineIndex2 > 0)
        {
            for(int Index=r_DownLineIndex1; Index<=LastExtendedIndex; Index++ )
            {
                Subgraph_DownTrendline[Index] = static_cast<float>(r_DownLine_A * Index + r_DownLine_B);
            }

            r_DownLineIndex2 = LastExtendedIndex;
        }
    }

    // Exit. Do not readjust trendlines
    return;
}

// update previous index and StartIndex to re-adjust
r_PrevIndex = LastIndex;
StartIndex = LastIndex - Input_SkipLastBars.GetInt();
}

if(Input_ShowTrendlineUp.GetYesNo() == 0 && Input_ShowTrendlineDown.GetYesNo() == 0)
    return;

if(StartIndex < Input_LengthInput.GetInt())
    return;

int StartIndex1 = StartIndex;
int LastIndex1 = StartIndex - Input_LengthInput.GetInt();

bool TLDrawn = false;

// the variables which describe trendline equality
double K = 0;
double A = 0;
double B = 0;

// calculate UP trendlines
if(Input_ShowTrendlineUp.GetYesNo() != 0)
{
    // find LowestLow Index
    int LowestLowIndex1 = StartIndex1;
    int LowestLowIndex2 = -1;

```

```

for (int Index = StartIndex1; Index >= LastIndex1; Index--)
{
    if(Lows[Index] < Lows[LowestLowIndex1])
        LowestLowIndex1 = Index;
}

// set second lowest low index as first lowest low + min trend line length
LowestLowIndex2 = LowestLowIndex1;

// search second LowestLow
while(!TLDrawn)
{
    // create new region where search lowest low
    int StartIndex2 = LowestLowIndex2 + 1;
    int LastIndex2 = StartIndex;

    if(StartIndex2 > LastIndex2)
    {
        TLDrawn = false;
        break;
    }

    LowestLowIndex2 = StartIndex2;
    for(int Index = StartIndex2; Index <= LastIndex2; Index++)
    {
        if(Lows[Index] < Lows[LowestLowIndex2])
            LowestLowIndex2 = Index;
    }

    // check if the line does not cross any other bars

    // calculate trendline between Low on Index1 and Low on Index2
    // A and B - these are coefficients of line equation
    K = static_cast<double>(LowestLowIndex2 - LowestLowIndex1);
    A = 0;
    B = 0;

    if(K != 0)
    {
        A = (Lows[LowestLowIndex2] - Lows[LowestLowIndex1])/K;
        B = Lows[LowestLowIndex1] - A * LowestLowIndex1;
    }

    // check if any of the bar between Index2 and Index1 cross the trendline
    TLDrawn = true;
    for(int Index=StartIndex1; Index >= LastIndex1; Index--)
    {
        // don't check former bars
        if(Index == LowestLowIndex2 || Index == LowestLowIndex1)
            continue;

        double TLValue = A*Index + B;
        if(TLValue > Lows[Index])
        {
            TLDrawn = false;
            break;
        }
    }
}

// clean previous trendline
for(int Index=r_UpLineIndex1; Index <= LastExtendedIndex; Index++)
    Subgraph_UpTrendline[Index] = 0;

r_UpLineIndex1 = 0;

```

```

r_UpLineIndex2 = 0;

// draw trendline if it was found
if(TLDrawn)
{
    // We found correct trend line. Draw it
    Subgraph_UpTrendline[LowestLowIndex1] = Lows[LowestLowIndex1];

    // fill all elements
    for (int Index = LowestLowIndex1 + 1; Index <= LastExtendedIndex; Index++)
    {
        Subgraph_UpTrendline[Index] = static_cast<float>(A * Index + B);
    }

    // save indexes in order to fast clean last drawn trendline
    r_UpLineIndex1 = LowestLowIndex1;
    r_UpLineIndex2 = LastExtendedIndex;

    r_UpLine_A = static_cast<float>(A);
    r_UpLine_B = static_cast<float>(B);
}

}

// calculate down trendlines

TLDrawn = false;

if(Input_ShowTrendlineDown.GetYesNo() != 0)
{
    // find LowestLow Index
    int HighestHighIndex1 = StartIndex1;
    int HighestHighIndex2 = -1;

    for (int Index = StartIndex1; Index >= LastIndex1; Index--)
    {
        if(Highs[Index] > Highs[HighestHighIndex1])
            HighestHighIndex1 = Index;
    }

    // set second lowest low index as first lowest low + min trend line length
    HighestHighIndex2 = HighestHighIndex1;

    // search second LowestLow
    while(!TLDrawn)
    {
        // create new region where search lowest low
        int StartIndex2 = HighestHighIndex2 + 1;
        int LastIndex2 = StartIndex;

        if(StartIndex2 > LastIndex2)
        {
            TLDrawn = false;
            break;
        }

        HighestHighIndex2 = StartIndex2;

        for(int Index=StartIndex2; Index <= LastIndex2; Index++)
        {
            if(Highs[Index] > Highs[HighestHighIndex2])
                HighestHighIndex2 = Index;
        }

        // check if the line does not cross any other bars

```

```

// calculate trendline between Low on Index1 and Low on Index2
// A and B - these are coefficients of line equation
K = static_cast<double>(HighestHighIndex2 - HighestHighIndex1);
A = 0;
B = 0;

if(K != 0)
{
    A = (Highs[HighestHighIndex2] - Highs[HighestHighIndex1])/K;
    B = Highs[HighestHighIndex1] - A * HighestHighIndex1;
}

// check if any of the bar between Index2 and Index1 cross the trendline
TLDrawn = true;
for(int Index=StartIndex1; Index >= LastIndex1; Index--)
{
    // do not check former bars
    if(Index == HighestHighIndex2 || Index == HighestHighIndex1)
        continue;

    double TLValue = A*Index + B;
    if(TLValue < Highs[Index])
    {
        TLDrawn = false;
        break;
    }
}

// clean previous trendline
for(int Index=r_DownLineIndex1; Index<=LastExtendedIndex; Index++)
    Subgraph_DownTrendline[Index] = 0;

r_DownLineIndex1 = 0;
r_DownLineIndex2 = 0;

// draw trendline if it was found
if(TLDrawn)
{
    // We found correct trend line. Draw it
    Subgraph_DownTrendline[HighestHighIndex1] = Highs[HighestHighIndex1];

    // fill all elements
    for (int Index = HighestHighIndex1 + 1; Index <= LastExtendedIndex; Index++)
    {
        Subgraph_DownTrendline[Index] = static_cast<float>(A * Index + B);
    }

    // save indexes in order to fast clean last drawn trendline
    r_DownLineIndex1 = HighestHighIndex1;
    r_DownLineIndex2 = LastExtendedIndex;

    r_DownLine_A = static_cast<float>(A);
    r_DownLine_B = static_cast<float>(B);
}
}

}

/*=====*/
SCSFExport scsf_ArnaudLegouxMovingAverage(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_ALMA = sc.Subgraph[0];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_Length = sc.Input[1];

```



```

SCInputRef Input_Sigma = sc.Input[2];
SCInputRef Input_Offset = sc.Input[3];

if (sc.SetDefaults)
{
    sc.GraphName = "Moving Average - Arnaud Legoux";

    sc.GraphRegion = 0;
    sc.ValueFormat = 3;
    sc.AutoLoop = 1;

    Subgraph_ALMA.Name = "ALMA";
    Subgraph_ALMA.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_ALMA.PrimaryColor = RGB(0, 255, 0);

    Input_InputData.Name = "Input Data";
    Input_InputData.SetInputDataIndex(SC_LAST);

    Input_Length.Name = "Length";
    Input_Length.SetInt(9);
    Input_Length.SetIntLimits(1, MAX_STUDY_LENGTH);

    Input_Sigma.Name = "Sigma";
    Input_Sigma.SetFloat(6.0f);
    Input_Sigma.SetFloatLimits(0.0001f, static_cast<float>(MAX_STUDY_LENGTH));

    Input_Offset.Name = "Offset";
    Input_Offset.SetFloat(0.5f);
    Input_Offset.SetFloatLimits(0.0001f, static_cast<float>(MAX_STUDY_LENGTH));

    return;
}

sc.DataStartIndex = Input_Length.GetInt() - 1;

sc.ArnaudLegouxMovingAverage(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_ALMA, sc.Index,
Input_Length.GetInt(), Input_Sigma.GetFloat(), Input_Offset.GetFloat());
}

/*=====*/
SCSFExport scsf_GannTrendOscillator(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_GTO = sc.Subgraph[0];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Gann Trend Oscillator";

        sc.GraphRegion = 1;
        sc.AutoLoop = 1;

        Subgraph_GTO.Name = "Gann Trend Oscillator";
        Subgraph_GTO.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_GTO.PrimaryColor = RGB(0, 0, 255);
    }

    int IsUptrend = 0;
    int IsDowntrend = 0;

    if (sc.Index == 0)
    {
        int IsUptrend = 0;
        int IsDowntrend = 0;

        return;
    }

```

```

}

if (sc.High[sc.Index - 1] < sc.High[sc.Index] && sc.Low[sc.Index - 1] < sc.Low[sc.Index])
    IsUptrend = 1;
else
    IsUptrend = 0;

if (sc.High[sc.Index - 1] > sc.High[sc.Index] && sc.Low[sc.Index - 1] > sc.Low[sc.Index])
    IsDowntrend = 1;
else
    IsDowntrend = 0;

if (IsUptrend == 1)
    Subgraph_GTO[sc.Index] = 1.0f;
else if (IsDowntrend == 1)
    Subgraph_GTO[sc.Index] = -1.0f;
else
    Subgraph_GTO[sc.Index] = 0.0f;
}

/*=====*/
SCSFExport scsf_GannSwingOscillator(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Upswing = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Downswing = sc.Subgraph[1];
    SCSubgraphRef Subgraph_GSO = sc.Subgraph[2];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Gann Swing Oscillator";

        sc.GraphRegion = 1;
        sc.AutoLoop = 1;

        Subgraph_Upswing.Name = "Upswing";
        Subgraph_Upswing.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_Downswing.Name = "Downswing";
        Subgraph_Downswing.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_GSO.Name = "Gann Swing Oscillator";
        Subgraph_GSO.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_GSO.PrimaryColor = RGB(128, 0, 128);

        return;
    }

    int& r_PriorIsUptrend = sc.GetPersistentInt(0);
    int& r_IsUptrend = sc.GetPersistentInt(1);
    int& r_PriorIsDowntrend = sc.GetPersistentInt(2);
    int& r_IsDowntrend = sc.GetPersistentInt(3);

    if (sc.Index == 0)
    {
        r_PriorIsUptrend = 0;
        r_IsUptrend = 0;
        r_PriorIsDowntrend = 0;
        r_IsDowntrend = 0;
    }

    // Detect Uptrend
    r_PriorIsUptrend = r_IsUptrend;

    if (sc.High[sc.Index - 1] < sc.High[sc.Index] && sc.Low[sc.Index - 1] < sc.Low[sc.Index])
        r_IsUptrend = 1;

```

```

else
    r_IsUptrend = 0;

// Detect Downtrend
r_PriorIsDowntrend = r_IsDowntrend;

if (sc.High[sc.Index - 1] > sc.High[sc.Index] && sc.Low[sc.Index - 1] > sc.Low[sc.Index])
    r_IsDowntrend = 1;
else
    r_IsDowntrend = 0;

// Detect Upswing
if (r_PriorIsDowntrend == 1 && r_IsUptrend == 1)
    Subgraph_Upswing[sc.Index - 1] = sc.Low[sc.Index - 1];
else
    Subgraph_Upswing[sc.Index - 1] = 0.0f;

//Detect Downswing
if (r_PriorIsUptrend == 1 && r_IsDowntrend == 1)
    Subgraph_Downswing[sc.Index - 1] = sc.High[sc.Index - 1];
else
    Subgraph_Downswing[sc.Index - 1] = 0.0f;

// Calculate Gann Swing Oscillator
if (sc.Index < 2)
    Subgraph_GSO[sc.Index] = 0.0f;
else
{
    if (Subgraph_Upswing[sc.Index - 2] == sc.Low[sc.Index - 2] && (sc.High[sc.Index - 2] < sc.High[sc.Index - 1] &&
sc.High[sc.Index - 1] < sc.High[sc.Index]))
        Subgraph_GSO[sc.Index] = 1.0f;
    else if (Subgraph_Downswing[sc.Index - 2] == sc.High[sc.Index - 2] && (sc.Low[sc.Index - 2] > sc.Low[sc.Index - 1]
&& sc.Low[sc.Index - 1] > sc.Low[sc.Index]))
        Subgraph_GSO[sc.Index] = -1.0f;
    else
        Subgraph_GSO[sc.Index] = Subgraph_GSO[sc.Index - 1];
}
}

/*=====*/
SCSFExport scsf_VariableIndexDynamicMovingAverage(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_VIDYA = sc.Subgraph[0];
    SCSubgraphRef Subgraph_TopBand = sc.Subgraph[1];
    SCSubgraphRef Subgraph_BottomBand = sc.Subgraph[2];

    SCFloatArrayRef Array_StdDev = sc.Subgraph[0].Arrays[0];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_VIDYALength = sc.Input[1];
    SCInputRef Input_StdDevLength = sc.Input[2];
    SCInputRef Input_ReferenceStandardDeviation = sc.Input[3];
    SCInputRef Input_PercentageOffset = sc.Input[4];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Moving Average - Variable Index Dynamic";

        sc.GraphRegion = 0;
        sc.AutoLoop = 1;

        Subgraph_VIDYA.Name = "VIDYA";
        Subgraph_VIDYA.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_VIDYA.PrimaryColor = RGB(0, 0, 255);
        Subgraph_VIDYA.DrawZeros = false;
    }
}

```

```

Subgraph_TopBand.Name = "Top Band";
Subgraph_TopBand.DrawStyle = DRAWSTYLE_LINE;
Subgraph_TopBand.PrimaryColor = RGB(0, 255, 0);
Subgraph_TopBand.DrawZeros = false;

Subgraph_BottomBand.Name = "Bottom Band";
Subgraph_BottomBand.DrawStyle = DRAWSTYLE_LINE;
Subgraph_BottomBand.PrimaryColor = RGB(255, 0, 0);
Subgraph_BottomBand.DrawZeros = false;

Input_InputData.Name = "Input Data";
Input_InputData.SetInputDataIndex(SC_LAST);

Input_VIDYALength.Name = "VIDYA Length";
Input_VIDYALength.SetInt(12);
Input_VIDYALength.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_StdDevLength.Name = "Std Dev Length";
Input_StdDevLength.SetInt(10);
Input_StdDevLength.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_ReferenceStandardDeviation.Name = "Reference Standard Deviation";
Input_ReferenceStandardDeviation.SetFloat(4.0f);
Input_ReferenceStandardDeviation.SetFloatLimits(0.0f, FLT_MAX);

Input_PercentageOffset.Name = "Percentage Offset";
Input_PercentageOffset.SetFloat(1.0f);
Input_PercentageOffset.SetFloatLimits(0.0f, 100.0f);

return;
}

if (sc.Index < Input_StdDevLength.GetInt() - 2)
{
return;
}
else if (sc.Index == Input_StdDevLength.GetInt() - 2)
{
Subgraph_VIDYA[sc.Index] = sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index];
}
else
{
sc.StdDeviation(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Array_StdDev, Input_StdDevLength.GetInt());

float InputData = sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index];
float VIDYALength = static_cast<float>(Input_VIDYALength.GetInt());
float RefStdDev = Input_ReferenceStandardDeviation.GetFloat();
float StdDev = Array_StdDev[sc.Index];
float Multiplier1 = 2.0f / (VIDYALength + 1.0f) * (StdDev / RefStdDev);
float Multiplier2 = 1.0f - Multiplier1;

Subgraph_VIDYA[sc.Index] = Multiplier1 * InputData + Multiplier2 * Subgraph_VIDYA[sc.Index - 1];
}

Subgraph_TopBand[sc.Index] = Subgraph_VIDYA[sc.Index] + Input_PercentageOffset.GetFloat() / 100.0f *
Subgraph_VIDYA[sc.Index];
Subgraph_BottomBand[sc.Index] = Subgraph_VIDYA[sc.Index] - Input_PercentageOffset.GetFloat() / 100.0f *
Subgraph_VIDYA[sc.Index];
}

/*=====*/
SCSFExport scsf_RapidAdaptiveVarianceIndicator(SCStudyInterfaceRef sc)
{
SCSubgraphRef Subgraph_RAVI = sc.Subgraph[0];

```

```

SCFloatArrayRef Array_LongVIDYA = sc.Subgraph[0].Arrays[0];
SCFloatArrayRef Array_LongStdDev = sc.Subgraph[0].Arrays[1];
SCFloatArrayRef Array_ShortVIDYA = sc.Subgraph[0].Arrays[2];
SCFloatArrayRef Array_ShortStdDev = sc.Subgraph[0].Arrays[3];

SCInputRef Input_InputData = sc.Input[0];
SCInputRef Input_LongVIDYALength = sc.Input[1];
SCInputRef Input_LongStdDevLength = sc.Input[2];
SCInputRef Input_LongReferenceStandardDeviation = sc.Input[3];
SCInputRef Input_ShortVIDYALength = sc.Input[4];
SCInputRef Input_ShortStdDevLength = sc.Input[5];
SCInputRef Input_ShortReferenceStandardDeviation = sc.Input[6];

if (sc.SetDefaults)
{
    sc.GraphName = "RAVI";

    sc.GraphRegion = 1;
    sc.AutoLoop = 1;

    Subgraph_RAVI.Name = "RAVI";
    Subgraph_RAVI.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_RAVI.PrimaryColor = RGB(0, 0, 255);
    Subgraph_RAVI.DrawZeros = false;

    Input_InputData.Name = "Input Data";
    Input_InputData.SetInputDataIndex(SC_LAST);

    Input_LongVIDYALength.Name = "Long VIDYA Length";
    Input_LongVIDYALength.SetInt(25);
    Input_LongVIDYALength.SetIntLimits(1, MAX_STUDY_LENGTH);

    Input_LongStdDevLength.Name = "Long Std Dev Length";
    Input_LongStdDevLength.SetInt(13);
    Input_LongStdDevLength.SetIntLimits(1, MAX_STUDY_LENGTH);

    Input_LongReferenceStandardDeviation.Name = "Long Reference Standard Deviation";
    Input_LongReferenceStandardDeviation.SetFloat(6.0f);
    Input_LongReferenceStandardDeviation.SetFloatLimits(0.0f, FLT_MAX);

    Input_ShortVIDYALength.Name = "Short VIDYA Length";
    Input_ShortVIDYALength.SetInt(12);
    Input_ShortVIDYALength.SetIntLimits(1, MAX_STUDY_LENGTH);

    Input_ShortStdDevLength.Name = "Short Std Dev Length";
    Input_ShortStdDevLength.SetInt(10);
    Input_ShortStdDevLength.SetIntLimits(1, MAX_STUDY_LENGTH);

    Input_ShortReferenceStandardDeviation.Name = "Short Reference Standard Deviation";
    Input_ShortReferenceStandardDeviation.SetFloat(4.0f);
    Input_ShortReferenceStandardDeviation.SetFloatLimits(0.0f, FLT_MAX);

    return;
}

if (sc.Index < max(Input_LongStdDevLength.GetInt(), Input_ShortStdDevLength.GetInt()) - 2)
{
    return;
}
else if (sc.Index == max(Input_LongStdDevLength.GetInt(), Input_ShortStdDevLength.GetInt()) - 2)
{
    Array_LongVIDYA[sc.Index] = sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index];
    Array_ShortVIDYA[sc.Index] = sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index];
}

```

```

        Subgraph_RAVI[sc.Index] = Array_ShortVIDYA[sc.Index] - Array_LongVIDYA[sc.Index];
    }
    else
    {
        sc.StdDeviation(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Array_LongStdDev,
Input_LongStdDevLength.GetInt());
        sc.StdDeviation(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Array_ShortStdDev,
Input_ShortStdDevLength.GetInt());

        float InputData = sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index];
        float LongVIDYALength = static_cast<float>(Input_LongVIDYALength.GetInt());
        float LongRefStdDev = Input_LongReferenceStandardDeviation.GetFloat();
        float LongStdDev = Array_LongStdDev[sc.Index];
        float LongMultiplier1 = 2.0f / (LongVIDYALength + 1.0f) * (LongStdDev / LongRefStdDev);
        float LongMultiplier2 = 1.0f - LongMultiplier1;
        float ShortVIDYALength = static_cast<float>(Input_ShortVIDYALength.GetInt());
        float ShortRefStdDev = Input_ShortReferenceStandardDeviation.GetFloat();
        float ShortStdDev = Array_ShortStdDev[sc.Index];
        float ShortMultiplier1 = 2.0f / (ShortVIDYALength + 1.0f) * (ShortStdDev / ShortRefStdDev);
        float ShortMultiplier2 = 1.0f - ShortMultiplier1;

        Array_LongVIDYA[sc.Index] = LongMultiplier1 * InputData + LongMultiplier2 * Array_LongVIDYA[sc.Index - 1];
        Array_ShortVIDYA[sc.Index] = ShortMultiplier1 * InputData + ShortMultiplier2 * Array_ShortVIDYA[sc.Index - 1];

        Subgraph_RAVI[sc.Index] = Array_ShortVIDYA[sc.Index] - Array_LongVIDYA[sc.Index];
    }
}

/*=====*/
SCSFExport scsf_TurboMACD(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_RAVI = sc.Subgraph[0];
    SCSubgraphRef Subgraph_TMACD = sc.Subgraph[1];
    SCSubgraphRef Subgraph_CenterLine = sc.Subgraph[2];

    SCFloatArrayRef Array_LongVIDYA = sc.Subgraph[0].Arrays[0];
    SCFloatArrayRef Array_LongStdDev = sc.Subgraph[0].Arrays[1];
    SCFloatArrayRef Array_ShortVIDYA = sc.Subgraph[0].Arrays[2];
    SCFloatArrayRef Array_ShortStdDev = sc.Subgraph[0].Arrays[3];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_LongVIDYALength = sc.Input[1];
    SCInputRef Input_LongStdDevLength = sc.Input[2];
    SCInputRef Input_LongReferenceStandardDeviation = sc.Input[3];
    SCInputRef Input_ShortVIDYALength = sc.Input[4];
    SCInputRef Input_ShortStdDevLength = sc.Input[5];
    SCInputRef Input_ShortReferenceStandardDeviation = sc.Input[6];
    SCInputRef Input_TurboMACDLength = sc.Input[7];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Turbo MACD";

        sc.GraphRegion = 1;
        sc.AutoLoop = 1;

        Subgraph_RAVI.Name = "RAVI";
        Subgraph_RAVI.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_TMACD.Name = "Turbo MACD";
        Subgraph_TMACD.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_TMACD.PrimaryColor = RGB(0, 0, 255);
        Subgraph_TMACD.DrawZeros = true;

        Subgraph_CenterLine.Name = "Center Line";
    }
}

```

```

Subgraph_CenterLine.DrawStyle = DRAWSTYLE_LINE;
Subgraph_CenterLine.PrimaryColor = RGB(128, 0, 128);
Subgraph_CenterLine.DrawZeros = true;

Input_InputData.Name = "Input Data";
Input_InputData.SetInputDataIndex(SC_LAST);

Input_LongVIDYALength.Name = "Long VIDYA Length";
Input_LongVIDYALength.SetInt(25);
Input_LongVIDYALength.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_LongStdDevLength.Name = "Long Std Dev Length";
Input_LongStdDevLength.SetInt(13);
Input_LongStdDevLength.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_LongReferenceStandardDeviation.Name = "Long Reference Standard Deviation";
Input_LongReferenceStandardDeviation.SetFloat(6.0f);
Input_LongReferenceStandardDeviation.SetFloatLimits(0.0f, FLT_MAX);

Input_ShortVIDYALength.Name = "Short VIDYA Length";
Input_ShortVIDYALength.SetInt(12);
Input_ShortVIDYALength.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_ShortStdDevLength.Name = "Short Std Dev Length";
Input_ShortStdDevLength.SetInt(10);
Input_ShortStdDevLength.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_ShortReferenceStandardDeviation.Name = "Short Reference Standard Deviation";
Input_ShortReferenceStandardDeviation.SetFloat(4.0f);
Input_ShortReferenceStandardDeviation.SetFloatLimits(0.0f, FLT_MAX);

Input_TurboMACDLength.Name = "Turbo MACD Length";
Input_TurboMACDLength.SetInt(9);
Input_TurboMACDLength.SetIntLimits(1, MAX_STUDY_LENGTH);

return;
}

if (sc.Index < max(Input_LongStdDevLength.GetInt(), Input_ShortStdDevLength.GetInt()) - 2)
{
return;
}
else if (sc.Index == max(Input_LongStdDevLength.GetInt(), Input_ShortStdDevLength.GetInt()) - 2)
{
Array_LongVIDYA[sc.Index] = sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index];
Array_ShortVIDYA[sc.Index] = sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index];

Subgraph_RAVI[sc.Index] = Array_ShortVIDYA[sc.Index] - Array_LongVIDYA[sc.Index];
Subgraph_TMACD[sc.Index] = Subgraph_RAVI[sc.Index];
Subgraph_CenterLine[sc.Index] = 0.0f;
}
else
{
sc.StdDeviation(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Array_LongStdDev,
Input_LongStdDevLength.GetInt());
sc.StdDeviation(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Array_ShortStdDev,
Input_ShortStdDevLength.GetInt());

float InputData = sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index];
float LongVIDYALength = static_cast<float>(Input_LongVIDYALength.GetInt());
float LongRefStdDev = Input_LongReferenceStandardDeviation.GetFloat();
float LongStdDev = Array_LongStdDev[sc.Index];
float LongMultiplier1 = 2.0f / (LongVIDYALength + 1.0f) * (LongStdDev / LongRefStdDev);
float LongMultiplier2 = 1.0f - LongMultiplier1;
float ShortVIDYALength = static_cast<float>(Input_ShortVIDYALength.GetInt());

```

```

float ShortRefStdDev = Input_ShortReferenceStandardDeviation.GetFloat();
float ShortStdDev = Array_ShortStdDev[sc.Index];
float ShortMultiplier1 = 2.0f / (ShortVIDYALength + 1.0f) * (ShortStdDev / ShortRefStdDev);
float ShortMultiplier2 = 1.0f - ShortMultiplier1;
float TMACDLength = static_cast<float>(Input_TurboMACDLength.GetInt());

Array_LongVIDYA[sc.Index] = LongMultiplier1 * InputData + LongMultiplier2 * Array_LongVIDYA[sc.Index - 1];
Array_ShortVIDYA[sc.Index] = ShortMultiplier1 * InputData + ShortMultiplier2 * Array_ShortVIDYA[sc.Index - 1];

Subgraph_RAVI[sc.Index] = Array_ShortVIDYA[sc.Index] - Array_LongVIDYA[sc.Index];
Subgraph_TMADC[sc.Index] = Subgraph_RAVI[sc.Index] - (2.0f / (TMACDLength + 1.0f) *
Subgraph_RAVI[sc.Index] + (1.0f - 2.0f / (TMACDLength + 1.0f)) * Subgraph_RAVI[sc.Index - 1]);
Subgraph_CenterLine[sc.Index] = 0.0f;
}
}

/*=====*/
SCSFExport scsf_OneTimeFraming(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_OTFU = sc.Subgraph[0];
    SCSubgraphRef Subgraph_OTFD = sc.Subgraph[1];

    SCInputRef Input_PercentageOffset = sc.Input[0];

    if (sc.SetDefaults)
    {
        sc.GraphName = "One Time Framing";

        sc.GraphRegion = 0;
        sc.AutoLoop = 1;

        Subgraph_OTFU.Name = "OTF - Up";
        Subgraph_OTFU.DrawStyle = DRAWSTYLE_ARROWUP;
        Subgraph_OTFU.PrimaryColor = RGB(0, 255, 0);
        Subgraph_OTFU.SecondaryColor = RGB(255, 0, 0);
        Subgraph_OTFU.SecondaryColorUsed = true;
        Subgraph_OTFU.LineWidth = 2;
        Subgraph_OTFU.DrawZeros = false;

        Subgraph_OTFD.Name = "OTF - Down";
        Subgraph_OTFD.DrawStyle = DRAWSTYLE_ARROWDOWN;
        Subgraph_OTFD.PrimaryColor = RGB(0, 255, 0);
        Subgraph_OTFD.SecondaryColor = RGB(255, 0, 0);
        Subgraph_OTFD.SecondaryColorUsed = true;
        Subgraph_OTFD.LineWidth = 2;
        Subgraph_OTFD.DrawZeros = false;

        Input_PercentageOffset.Name = "Arrow Offset Percentage";
        Input_PercentageOffset.SetFloat(1.0f);
        Input_PercentageOffset.SetFloatLimits(0.0f, FLT_MAX);
    }

    int& r_PriorEnterOTFU = sc.GetPersistentInt(0);
    int& r_EnterOTFU = sc.GetPersistentInt(1);
    int& r_PriorIsOTFU = sc.GetPersistentInt(2);
    int& r_IsOTFU = sc.GetPersistentInt(3);
    int& r_PriorEnterOTFD = sc.GetPersistentInt(4);
    int& r_EnterOTFD = sc.GetPersistentInt(5);
    int& r_PriorIsOTFD = sc.GetPersistentInt(6);
    int& r_IsOTFD = sc.GetPersistentInt(7);

    if (sc.Index == 0)
    {
        r_PriorEnterOTFU = 0;
        r_EnterOTFU = 0;
    }

```



```

    r_PriorIsOTFU = 0;
    r_IsOTFU = 0;
    r_PriorEnterOTFD = 0;
    r_EnterOTFD = 0;
    r_PriorIsOTFD = 0;
    r_IsOTFD = 0;

    return;
}

if (sc.GetBarHasClosedStatus() == BHCS_BAR_HAS_NOT_CLOSED)
    return;

// OTF - Up
r_PriorEnterOTFU = r_EnterOTFU;
r_PriorIsOTFU = r_IsOTFU;

if (sc.Index > 0)
{
    if (sc.High[sc.Index] > sc.High[sc.Index - 1] && sc.Low[sc.Index] > sc.Low[sc.Index - 1])
        r_EnterOTFU = 1;
    else
        r_EnterOTFU = 0;

    if ((r_PriorEnterOTFU == 1 || r_PriorIsOTFU == 1) && (sc.High[sc.Index] >= sc.High[sc.Index - 1] &&
sc.Low[sc.Index] > sc.Low[sc.Index - 1]))
        r_IsOTFU = 1;
    else
        r_IsOTFU = 0;
}

if (r_PriorEnterOTFU == 1 && (r_PriorIsOTFU == 0 && r_IsOTFU == 1))
{
    Subgraph_OTFU.DataColor[sc.Index] = Subgraph_OTFU.PrimaryColor;
    Subgraph_OTFU[sc.Index] = sc.Low[sc.Index] - (Input_PercentageOffset.GetFloat() / 100.0f) * sc.Low[sc.Index];
}
else if (r_PriorIsOTFU == 1 && r_IsOTFU == 0)
{
    Subgraph_OTFU.DataColor[sc.Index] = Subgraph_OTFU.SecondaryColor;
    Subgraph_OTFU[sc.Index] = sc.Low[sc.Index] - (Input_PercentageOffset.GetFloat() / 100.0f) * sc.Low[sc.Index];
}
else
    Subgraph_OTFU[sc.Index] = 0.0f;

// OTF - Down
r_PriorEnterOTFD = r_EnterOTFD;
r_PriorIsOTFD = r_IsOTFD;

if (sc.Index > 0)
{
    if (sc.Low[sc.Index] < sc.Low[sc.Index - 1] && sc.High[sc.Index] < sc.High[sc.Index - 1])
        r_EnterOTFD = 1;
    else
        r_EnterOTFD = 0;

    if ((r_PriorEnterOTFD == 1 || r_PriorIsOTFD == 1) && (sc.High[sc.Index] < sc.High[sc.Index - 1] && sc.Low[sc.Index]
<= sc.Low[sc.Index - 1]))
        r_IsOTFD = 1;
    else
        r_IsOTFD = 0;
}

if (r_PriorEnterOTFD == 1 && (r_PriorIsOTFD == 0 && r_IsOTFD == 1))
{
    Subgraph_OTFD.DataColor[sc.Index] = Subgraph_OTFD.PrimaryColor;

```

```

    Subgraph_OTFD[sc.Index] = sc.High[sc.Index] + (Input_PercentageOffset.GetFloat() / 100.0f) * sc.High[sc.Index];
}
else if (r_PriorIsOTFD == 1 && r_IsOTFD == 0)
{
    Subgraph_OTFD.DataColor[sc.Index] = Subgraph_OTFD.SecondaryColor;
    Subgraph_OTFD[sc.Index] = sc.High[sc.Index] + (Input_PercentageOffset.GetFloat() / 100.0f) * sc.High[sc.Index];
}
else
    Subgraph_OTFD[sc.Index] = 0.0f;
}

/*=====*/
SCSFExport scsf_Covariance(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Covariance = sc.Subgraph[0];

    SCFloatArrayRef Array_Average1 = sc.Subgraph[0].Arrays[0];
    SCFloatArrayRef Array_Average2 = sc.Subgraph[0].Arrays[1];
    SCFloatArrayRef Array_JointArray = sc.Subgraph[0].Arrays[2];
    SCFloatArrayRef Array_JointAverage = sc.Subgraph[0].Arrays[3];

    SCInputRef Input_InputArray1 = sc.Input[0];
    SCInputRef Input_InputArray2 = sc.Input[1];
    SCInputRef Input_Length = sc.Input[2];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Covariance";

        sc.GraphRegion = 1;

        Subgraph_Covariance.Name = "Covariance";
        Subgraph_Covariance.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Covariance.PrimaryColor = RGB(0, 255, 0);
        Subgraph_Covariance.DrawZeros = true;

        Input_InputArray1.Name = "Input Array 1";
        Input_InputArray1.SetStudySubgraphValues(0, 0);

        Input_InputArray2.Name = "Input Array 2";
        Input_InputArray2.SetStudySubgraphValues(0, 0);

        Input_Length.Name = "Input_Length";
        Input_Length.SetInt(200);

        sc.AutoLoop = 1;
        sc.CalculationPrecedence = LOW_PREC_LEVEL;

        return;
    }

    sc.DataStartIndex = Input_Length.GetInt();

    SCFloatArray Array1;
    SCFloatArray Array2;

    sc.GetStudyArrayUsingID(Input_InputArray1.GetStudyID(), Input_InputArray1.GetSubgraphIndex(), Array1);

    if (Array1.GetArraySize() < sc.ArraySize)
        return;

    sc.GetStudyArrayUsingID(Input_InputArray2.GetStudyID(), Input_InputArray2.GetSubgraphIndex(), Array2);

    if (Array2.GetArraySize() < sc.ArraySize)
        return;

```

```

Array_JointArray[sc.Index] = sc.BaseDataIn[Input_InputArray1.GetSubgraphIndex()][sc.Index] *
sc.BaseDataIn[Input_InputArray2.GetSubgraphIndex()][sc.Index];

```

```

sc.SimpleMovAvg(Array1, Array_Average1, Input_Length.GetInt());
sc.SimpleMovAvg(Array2, Array_Average2, Input_Length.GetInt());
sc.SimpleMovAvg(Array_JointArray, Array_JointAverage, Input_Length.GetInt());

```

```

Subgraph_Covariance[sc.Index] = Array_JointAverage[sc.Index] - Array_Average1[sc.Index] *
Array_Average2[sc.Index];
}

```

```

/*=====*/

```

```

SCSFExport scsf_SuperTrend(SCStudyInterfaceRef sc)

```

```

{
    SCSubgraphRef Subgraph_SuperTrend = sc.Subgraph[0];
    SCSubgraphRef Subgraph_HullATR = sc.Subgraph[1];
    SCSubgraphRef Subgraph_TrueRange = sc.Subgraph[2];
    SCSubgraphRef Subgraph_AvgTrueRange = sc.Subgraph[3];
    SCSubgraphRef Subgraph_UpperBandBasic = sc.Subgraph[4];
    SCSubgraphRef Subgraph_LowerBandBasic = sc.Subgraph[5];
    SCSubgraphRef Subgraph_UpperBand = sc.Subgraph[6];
    SCSubgraphRef Subgraph_LowerBand = sc.Subgraph[7];

    SCFloatArrayRef Array_TrueRange = Subgraph_SuperTrend.Arrays[0];
    SCFloatArrayRef Array_AvgTrueRange = Subgraph_SuperTrend.Arrays[1];
    SCFloatArrayRef Array_UpperBandBasic = Subgraph_SuperTrend.Arrays[2];
    SCFloatArrayRef Array_LowerBandBasic = Subgraph_SuperTrend.Arrays[3];
    SCFloatArrayRef Array_UpperBand = Subgraph_SuperTrend.Arrays[4];
    SCFloatArrayRef Array_LowerBand = Subgraph_SuperTrend.Arrays[5];

```

```

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_ATRMultiplier = sc.Input[1];
    SCInputRef Input_ATRPeriod = sc.Input[2];
    SCInputRef Input_ATRMovAvgType = sc.Input[3];

```

```

    if (sc.SetDefaults)
    {

```

```

        sc.GraphName = "SuperTrend";

        sc.StudyDescription = "";
        sc.DrawZeros = true;
        sc.GraphRegion = 0;
        sc.ValueFormat = sc.BaseGraphValueFormat;

```

```

        sc.AutoLoop = 1;

```

```

        Subgraph_SuperTrend.Name = "SuperTrend";
        Subgraph_SuperTrend.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_SuperTrend.LineWidth = 2;
        Subgraph_SuperTrend.PrimaryColor = COLOR_RED;
        Subgraph_SuperTrend.SecondaryColor = COLOR_GREEN;
        Subgraph_SuperTrend.SecondaryColorUsed = 1;

```

```

        Subgraph_TrueRange.Name = "True Range";
        Subgraph_TrueRange.DrawStyle = DRAWSTYLE_IGNORE;

```

```

        Subgraph_AvgTrueRange.Name = "Average True Range";
        Subgraph_AvgTrueRange.DrawStyle = DRAWSTYLE_IGNORE;

```

```

        Subgraph_UpperBandBasic.Name = "Basic Upper Band";
        Subgraph_UpperBandBasic.DrawStyle = DRAWSTYLE_IGNORE;

```

```

        Subgraph_LowerBandBasic.Name = "Basic Lower Band";
        Subgraph_LowerBandBasic.DrawStyle = DRAWSTYLE_IGNORE;

```

```

Subgraph_UpperBand.Name = "Upper Band";
Subgraph_UpperBand.DrawStyle = DRAWSTYLE_IGNORE;

Subgraph_LowerBand.Name = "Lower Band";
Subgraph_LowerBand.DrawStyle = DRAWSTYLE_IGNORE;

Input_InputData.Name = "Input Data";
Input_InputData.SetInputDataIndex(SC_HL);

Input_ATRMultiplier.Name = "ATR Multiplier";
Input_ATRMultiplier.SetFloat(2);
Input_ATRMultiplier.SetFloatLimits(0.000001f, (float)MAX_STUDY_LENGTH);

Input_ATRPeriod.Name = "ATR Period";
Input_ATRPeriod.SetInt(3);
Input_ATRPeriod.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_ATRMovAvgType.Name = "ATR Moving Average Type";
Input_ATRMovAvgType.SetCustomInputStrings("Exponential Moving Avg;Linear Regression Moving Avg;Simple
Moving Avg;Weighted Moving Avg;Wilders Moving Avg;Simple Moving Avg SkipZeros;Smoothed Moving Avg;Hull Moving
Avg");
Input_ATRMovAvgType.SetCustomInputIndex(7);

return;
}

// Calculate TR and ATR
sc.TrueRange(sc.BaseDataIn, Array_TrueRange);

if (Input_ATRMovAvgType.GetIndex() == 0)
    sc.ATR(sc.BaseDataIn, Array_TrueRange, Array_AvgTrueRange, sc.Index, Input_ATRPeriod.GetInt(),
MOVAVGTYPE_EXPONENTIAL);
else if (Input_ATRMovAvgType.GetIndex() == 1)
    sc.ATR(sc.BaseDataIn, Array_TrueRange, Array_AvgTrueRange, sc.Index, Input_ATRPeriod.GetInt(),
MOVAVGTYPE_LINEARREGRESSION);
else if (Input_ATRMovAvgType.GetIndex() == 2)
    sc.ATR(sc.BaseDataIn, Array_TrueRange, Array_AvgTrueRange, sc.Index, Input_ATRPeriod.GetInt(),
MOVAVGTYPE_SIMPLE);
else if (Input_ATRMovAvgType.GetIndex() == 3)
    sc.ATR(sc.BaseDataIn, Array_TrueRange, Array_AvgTrueRange, sc.Index, Input_ATRPeriod.GetInt(),
MOVAVGTYPE_WEIGHTED);
else if (Input_ATRMovAvgType.GetIndex() == 4)
    sc.ATR(sc.BaseDataIn, Array_TrueRange, Array_AvgTrueRange, sc.Index, Input_ATRPeriod.GetInt(),
MOVAVGTYPE_WILDERS);
else if (Input_ATRMovAvgType.GetIndex() == 5)
    sc.ATR(sc.BaseDataIn, Array_TrueRange, Array_AvgTrueRange, sc.Index, Input_ATRPeriod.GetInt(),
MOVAVGTYPE_SIMPLE_SKIP_ZEROS);
else if (Input_ATRMovAvgType.GetIndex() == 6)
    sc.ATR(sc.BaseDataIn, Array_TrueRange, Array_AvgTrueRange, sc.Index, Input_ATRPeriod.GetInt(),
MOVAVGTYPE_SMOOTHED);
else
{
    sc.HullMovingAverage(Array_TrueRange, Subgraph_HullATR, Input_ATRPeriod.GetInt());
    Array_AvgTrueRange[sc.Index] = Subgraph_HullATR[sc.Index];
}

// Calculate Basic Upper and Lower Bands
float Price = sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index];

Array_UpperBandBasic[sc.Index] = Price + Input_ATRMultiplier.GetFloat() * Array_AvgTrueRange[sc.Index];
Array_LowerBandBasic[sc.Index] = Price - Input_ATRMultiplier.GetFloat() * Array_AvgTrueRange[sc.Index];

// Calculate Upper and Lower Bands
if (Array_UpperBandBasic[sc.Index] < Array_UpperBand[sc.Index - 1] || sc.Close[sc.Index - 1] >

```

```

Array_UpperBand[sc.Index - 1])
    Array_UpperBand[sc.Index] = Array_UpperBandBasic[sc.Index];
else
    Array_UpperBand[sc.Index] = Array_UpperBand[sc.Index - 1];

if (Array_LowerBandBasic[sc.Index] > Array_LowerBand[sc.Index - 1] || sc.Close[sc.Index - 1] <
Array_LowerBand[sc.Index - 1])
    Array_LowerBand[sc.Index] = Array_LowerBandBasic[sc.Index - 1];
else
    Array_LowerBand[sc.Index] = Array_LowerBand[sc.Index - 1];

// Extra Subgraphs
Subgraph_TrueRange[sc.Index] = Array_TrueRange[sc.Index];
Subgraph_AvgTrueRange[sc.Index] = Array_AvgTrueRange[sc.Index];
Subgraph_UpperBandBasic[sc.Index] = Array_UpperBandBasic[sc.Index];
Subgraph_LowerBandBasic[sc.Index] = Array_LowerBandBasic[sc.Index];
Subgraph_UpperBand[sc.Index] = Array_UpperBand[sc.Index];
Subgraph_LowerBand[sc.Index] = Array_LowerBand[sc.Index];

// Calculate SuperTrend
if (sc.Index == 0)
    Subgraph_SuperTrend[sc.Index] = Array_UpperBand[sc.Index];

if (Subgraph_SuperTrend[sc.Index - 1] == Array_UpperBand[sc.Index - 1] && sc.Close[sc.Index] <
Array_UpperBand[sc.Index])
    Subgraph_SuperTrend[sc.Index] = Array_UpperBand[sc.Index];
else if (Subgraph_SuperTrend[sc.Index - 1] == Array_UpperBand[sc.Index - 1] && sc.Close[sc.Index] >
Array_UpperBand[sc.Index])
    Subgraph_SuperTrend[sc.Index] = Array_LowerBand[sc.Index];
else if (Subgraph_SuperTrend[sc.Index - 1] == Array_LowerBand[sc.Index - 1] && sc.Close[sc.Index] >
Array_LowerBand[sc.Index])
    Subgraph_SuperTrend[sc.Index] = Array_LowerBand[sc.Index];
else if (Subgraph_SuperTrend[sc.Index - 1] == Array_LowerBand[sc.Index - 1] && sc.Close[sc.Index] <
Array_LowerBand[sc.Index])
    Subgraph_SuperTrend[sc.Index] = Array_UpperBand[sc.Index];
else
    Subgraph_SuperTrend[sc.Index] = Subgraph_SuperTrend[sc.Index - 1];

if (Subgraph_SuperTrend[sc.Index] == Array_UpperBand[sc.Index])
    Subgraph_SuperTrend.DataColor[sc.Index] = Subgraph_SuperTrend.PrimaryColor;
else
    Subgraph_SuperTrend.DataColor[sc.Index] = Subgraph_SuperTrend.SecondaryColor;
}

```