

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

```
/**/
```

```
SCSFExport scsf_ChaikinOsc(SCStudyInterfaceRef sc)
```

```
{
    SCSubgraphRef Subgraph_Oscillator = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Overbought = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Oversold = sc.Subgraph[3];
```

```
    SCInputRef Input_LongMALength = sc.Input[2];
    SCInputRef Input_ShortMALength = sc.Input[3];
    SCInputRef Input_Divisor = sc.Input[4];
    SCInputRef Input_OverboughtValue = sc.Input[5];
    SCInputRef Input_OversoldValue = sc.Input[6];
```

```
    if(sc.SetDefaults)
```

```
    {
        sc.GraphName = "Chaikin Oscillator";
```

```
        Subgraph_Oscillator.Name = "Oscillator";
        Subgraph_Oscillator.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Oscillator.PrimaryColor = RGB(0,255,0);
```

```
        Subgraph_Overbought.Name = "Overbought";
        Subgraph_Overbought.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Overbought.PrimaryColor = RGB(255,255,0);
        Subgraph_Overbought.DrawZeros = false;
```

```
        Subgraph_Oversold.Name = "Oversold";
        Subgraph_Oversold.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Oversold.PrimaryColor = RGB(255,127,0);
        Subgraph_Oversold.DrawZeros = false;
```

```
        Input_LongMALength.Name = "Long MovAvg Length";
        Input_LongMALength.SetInt(10);
        Input_LongMALength.SetIntLimits(1,MAX_STUDY_LENGTH);
```

```
        Input_ShortMALength.Name = "Short MovAvg Length";
        Input_ShortMALength.SetInt(3);
        Input_ShortMALength.SetIntLimits(1,MAX_STUDY_LENGTH);
```

```
        Input_Divisor.Name = "Divisor";
        Input_Divisor.SetFloat(1);
        Input_Divisor.SetFloatLimits(.00001f, static_cast<float>(MAX_STUDY_LENGTH));
```

```
        Input_OverboughtValue.Name = "Overbought";
        Input_OverboughtValue.SetInt(0);
```

```
        Input_OversoldValue.Name = "Oversold";
        Input_OversoldValue.SetInt(0);
```

```
        sc.AutoLoop = 1;
```

```
        return;
```

```
    }
```

```
    sc.DataStartIndex = max(Input_LongMALength.GetInt(),Input_ShortMALength.GetInt()) - 1;
```

```
    sc.AccumulationDistribution(sc.BaseDataIn, Subgraph_Oscillator.Arrays[0], sc.Index);
    sc.ExponentialMovAvg(Subgraph_Oscillator.Arrays[0], Subgraph_Oscillator.Arrays[1], sc.Index,
```

```

Input_LongMALength.GetInt());
    sc.ExponentialMovAvg(Subgraph_Oscillator.Arrays[0], Subgraph_Oscillator.Arrays[2], sc.Index,
Input_ShortMALength.GetInt());

    Subgraph_Oscillator[sc.Index] = (Subgraph_Oscillator.Arrays[2][sc.Index] - Subgraph_Oscillator.Arrays[1][sc.Index]) /
Input_Divisor.GetInt();
    Subgraph_Overbought[sc.Index] = static_cast<float>(Input_OverboughtValue.GetInt());
    Subgraph_Oversold[sc.Index] = static_cast<float>(Input_OversoldValue.GetInt());
}

/*=====*/
SCSFExport scsf_HistoricalVolatilityRatio(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_HVR = sc.Subgraph[0];
    SCFloatArrayRef Array_TempLog = sc.Subgraph[0].Arrays[0];
    SCFloatArrayRef Array_TempShort = sc.Subgraph[0].Arrays[1];
    SCFloatArrayRef Array_TempLong = sc.Subgraph[0].Arrays[2];

    SCInputRef Input_ShortLength = sc.Input[2];
    SCInputRef Input_LongLength = sc.Input[3];

    SCFloatArrayRef Array_Close = sc.Close;

    // Configuration
    if (sc.SetDefaults)
    {
        sc.GraphName = "Historical Volatility Ratio";

        Subgraph_HVR.Name = "HVR";
        Subgraph_HVR.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_HVR.PrimaryColor = RGB(0,255,0);

        Input_ShortLength.Name = "Short Length";
        Input_ShortLength.SetInt(6);
        Input_ShortLength.SetIntLimits(1, INT_MAX);

        Input_LongLength.Name = "Long Length";
        Input_LongLength.SetInt(100);
        Input_LongLength.SetIntLimits(1, INT_MAX);

        sc.AutoLoop = 1;

        return;
    }

    // Data Processing

    sc.DataStartIndex = max(Input_LongLength.GetInt(), Input_ShortLength.GetInt());

    if (sc.Index == 0)
        return;

    if (Array_Close[sc.Index - 1] != 0.0)
        Array_TempLog[sc.Index] = logf(Array_Close[sc.Index] / Array_Close[sc.Index - 1]);
    else
        Array_TempLog[sc.Index] = 0;

    sc.StdDeviation(Array_TempLog, Array_TempShort, sc.Index, Input_ShortLength.GetInt());
    sc.StdDeviation(Array_TempLog, Array_TempLong, sc.Index, Input_LongLength.GetInt());

    if (Array_TempLog[sc.Index] != 0)

```

```

        Subgraph_HVR[sc.Index] = Array_TempShort[sc.Index] / Array_TempLong[sc.Index];
    else
        Subgraph_HVR[sc.Index] = Subgraph_HVR[sc.Index-1];
}

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

    SCInputRef Input_NumberOfBarsBack = sc.Input[0];
    SCInputRef Input_InputData = sc.Input[1];
    SCInputRef Input_Version = sc.Input[2];
    SCInputRef Input_DisplayCPLOnBars = sc.Input[3];
    SCInputRef Input_ProjectForward = sc.Input[4];
    SCInputRef Input_UseLastFromCurrentQuoteData = sc.Input [5];
    SCInputRef Input_NumberOfForwardProjectionBars = sc.Input[6];
    SCInputRef Input_NumberOfBarsBackToReference = sc.Input[7];

    if(sc.SetDefaults)
    {
        sc.GraphName = "Current Price Line";

        Subgraph_CurrentPriceLine.Name = "CPL";
        Subgraph_CurrentPriceLine.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_CurrentPriceLine.PrimaryColor = RGB(0,255,0);
        Subgraph_CurrentPriceLine.DrawZeros = false;
        Subgraph_CurrentPriceLine.LineStyle = LINESTYLE_DOT;

        sc.ScaleRangeType = SCALE_SAMEASREGION;
        sc.GraphRegion = 0;

        Input_NumberOfBarsBack.Name = "Number of Bars to Display On. 0 = Current Day";
        Input_NumberOfBarsBack.SetInt(0);
        Input_NumberOfBarsBack.SetIntLimits(0,MAX_STUDY_LENGTH);

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

        Input_DisplayCPLOnBars.Name = "Display Current Price Line on Chart Bars";
        Input_DisplayCPLOnBars.SetYesNo(true);

        Input_ProjectForward.Name = "Forward Project Current Price Line";
        Input_ProjectForward.SetYesNo(false);

        Input_UseLastFromCurrentQuoteData.Name = "Use Last Trade Price From Current Quote Data";
        Input_UseLastFromCurrentQuoteData.SetYesNo(false);

        Input_NumberOfForwardProjectionBars.Name = "Number of Forward Projection Bars";
        Input_NumberOfForwardProjectionBars.SetInt(10);

        Input_NumberOfBarsBackToReference.Name = "Number of Bars Back to Reference";
        Input_NumberOfBarsBackToReference.SetInt(0);
        Input_NumberOfBarsBackToReference.SetIntLimits(0, MAX_STUDY_LENGTH);

        Input_Version.SetInt(2);

        sc.AutoLoop = 0; //Manual looping
        return;
    }

    if (Input_NumberOfForwardProjectionBars.GetInt() == 0)
        Input_NumberOfForwardProjectionBars.SetInt(10);

    SCFloatArrayRef InputDataArray = sc.BaseData[Input_InputData.GetInputDataIndex()];

```

```

float LineValue = 0;

if(Input_UseLastFromCurrentQuoteData.GetYesNo())
    LineValue = sc.LastTradePrice;
else
{
    LineValue = InputDataArray[sc.ArraySize - 1 - Input_NumberOfBarsBackToReference.GetInt()];
}

int &r_PriorLineStartIndex = sc.GetPersistentInt(1);
float &r_PriorValue = sc.GetPersistentFloat(2);
int &r_PriorArraySize = sc.GetPersistentInt(3);

if (sc.IsFullRecalculation)
{
    r_PriorValue = 0;
    r_PriorArraySize = 0;
}

bool LineNeedsUpdating = false;
if (LineValue != r_PriorValue || sc.ArraySize != r_PriorArraySize)
    LineNeedsUpdating = true;

r_PriorValue = LineValue;
r_PriorArraySize = sc.ArraySize;

if (!LineNeedsUpdating)//for efficiency
    return;

if(Input_DisplayCPLOnBars.GetYesNo())
{
    int BarsBack = Input_NumberOfBarsBack.GetInt();

    if (BarsBack == 0)// Current Day
    {
        SCDatetime TradingDayStartDateTime =
sc.GetTradingDayStartDateTimeOfBar(sc.BaseDateTimeIn[sc.ArraySize - 1]);

        int BarIndex = 0;
        for (BarIndex = sc.ArraySize - 1;
            sc.BaseDateTimeIn[BarIndex] >= TradingDayStartDateTime && BarIndex >= 0; BarIndex--)
        {
            Subgraph_CurrentPriceLine[BarIndex] = LineValue;

            sc.EarliestUpdateSubgraphDataArrayIndex = BarIndex;
        }
    }
    else
    {
        int StartingIndex = max(0, sc.ArraySize - BarsBack);

        sc.EarliestUpdateSubgraphDataArrayIndex = sc.UpdateStartIndex;

        for (int Index = max(0, sc.UpdateStartIndex - BarsBack); Index < sc.ArraySize; ++Index)
        {
            if(Index >= StartingIndex)
                break;

            Subgraph_CurrentPriceLine[Index] = 0;

            if (Index < sc.EarliestUpdateSubgraphDataArrayIndex)
                sc.EarliestUpdateSubgraphDataArrayIndex = Index;
        }
    }
}

```

```

    for (int Index = sc.ArraySize - 1; BarsBack > 0 && Index >= 0; Index--)
    {
        Subgraph_CurrentPriceLine[Index] = LineValue;
        BarsBack--;

        if (Index < sc.EarliestUpdateSubgraphDataArrayIndex)
            sc.EarliestUpdateSubgraphDataArrayIndex = Index;
    }
}

else if (r_PriorLineStartIndex != 0)
{
    //Clear prior array elements for when projecting forward
    for(int Index = r_PriorLineStartIndex; Index < sc.ArraySize; Index++)
        Subgraph_CurrentPriceLine[Index] = 0.0;
}

// Zero out prior values
int StartClearIndex = 0;
if (Input_DisplayCPLOnBars.GetYesNo())
    StartClearIndex = sc.EarliestUpdateSubgraphDataArrayIndex - 1;
else
    StartClearIndex = sc.ArraySize - 1;

for (int BarIndex = StartClearIndex; BarIndex >= 0; --BarIndex)
{
    if (Subgraph_CurrentPriceLine[BarIndex] == 0.0f)
        break;

    Subgraph_CurrentPriceLine[BarIndex] = 0.0f;
    sc.EarliestUpdateSubgraphDataArrayIndex = BarIndex;
}

if (Input_ProjectForward.GetYesNo())
{
    int NumberOfBars = Input_NumberOfForwardProjectionBars.GetInt();

    Subgraph_CurrentPriceLine.ExtendedArrayElementsToGraph = NumberOfBars;

    for(int Index = 0; Index < NumberOfBars; Index++)
    {
        Subgraph_CurrentPriceLine[sc.ArraySize + Index] = LineValue;
    }

    r_PriorLineStartIndex = sc.ArraySize;
}
else
    Subgraph_CurrentPriceLine.ExtendedArrayElementsToGraph = 0;
}

/*=====*/
SCSFExport scsf_CurrentBidAskLines(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_CurrentBidLine = sc.Subgraph[0];
    SCSubgraphRef Subgraph_CurrentAskLine = sc.Subgraph[1];

    SCInputRef Input_NumberOfBarsBack = sc.Input[0];
    SCInputRef Input_DisplayLinesOnBars = sc.Input[1];
    SCInputRef Input_ProjectForward = sc.Input[2];
    SCInputRef Input_NumberOfForwardProjectionBars = sc.Input[3];

    if (sc.SetDefaults)

```

```

{
    sc.GraphName = "Current Bid Ask Lines";

    Subgraph_CurrentBidLine.Name = "Bid";
    Subgraph_CurrentBidLine.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_CurrentBidLine.PrimaryColor = RGB(0, 255, 0);
    Subgraph_CurrentBidLine.DrawZeros = false;
    Subgraph_CurrentBidLine.LineStyle = LINESTYLE_SOLID;
    Subgraph_CurrentBidLine.LineWidth = 2;

    Subgraph_CurrentAskLine.Name = "Ask";
    Subgraph_CurrentAskLine.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_CurrentAskLine.PrimaryColor = RGB(255, 0, 0);
    Subgraph_CurrentAskLine.DrawZeros = false;
    Subgraph_CurrentAskLine.LineStyle = LINESTYLE_SOLID;
    Subgraph_CurrentAskLine.LineWidth = 2;

    sc.ScaleRangeType = SCALE_SAMEASREGION;
    sc.GraphRegion = 0;

    Input_NumberOfBarsBack.Name = "Number of Bars to Display On. 0=Current Day";
    Input_NumberOfBarsBack.SetInt(0);
    Input_NumberOfBarsBack.SetIntLimits(0, MAX_STUDY_LENGTH);

    Input_DisplayLinesOnBars.Name = "Display Current Lines on Chart Bars";
    Input_DisplayLinesOnBars.SetYesNo(true);

    Input_ProjectForward.Name = "Forward Project Bid and Ask Lines";
    Input_ProjectForward.SetYesNo(false);

    Input_NumberOfForwardProjectionBars.Name = "Number of Forward Projection Bars";
    Input_NumberOfForwardProjectionBars.SetInt(10);

    sc.AutoLoop = 0;

    return;
}

if (Input_NumberOfForwardProjectionBars.GetInt() == 0)
    Input_NumberOfForwardProjectionBars.SetInt(10);

float BidLineValue = sc.Bid;
float AskLineValue = sc.Ask;

int &r_PriorLineStartIndex = sc.GetPersistentInt(1);

int EarliestUpdateSubgraphDataArrayIndex = sc.ArraySize;

if (Input_DisplayLinesOnBars.GetYesNo())
{
    const int BarsBack = Input_NumberOfBarsBack.GetInt();

    if (BarsBack == 0)
    {
        SCDateTime TradingDayStartDateTime =
sc.GetTradingDayStartDateTimeOfBar(sc.BaseDateTimeIn[sc.ArraySize - 1]);

        int BarIndex = 0;
        for (BarIndex = sc.ArraySize - 1;
            sc.BaseDateTimeIn[BarIndex] >= TradingDayStartDateTime && BarIndex >= 0; BarIndex--)
        {
            Subgraph_CurrentBidLine[BarIndex] = BidLineValue;
            Subgraph_CurrentAskLine[BarIndex] = AskLineValue;

            EarliestUpdateSubgraphDataArrayIndex = BarIndex;
        }
    }
}

```

```

    }
}
else
{
    int StartingIndex = sc.ArraySize - BarsBack;

    for (int Index = sc.ArraySize - 1; Index >= StartingIndex; Index--)
    {
        Subgraph_CurrentBidLine[Index] = BidLineValue;
        Subgraph_CurrentAskLine[Index] = AskLineValue;

        EarliestUpdateSubgraphDataArrayIndex = Index;
    }
}

}

if (Input_ProjectForward.GetYesNo())
{
    int NumberOfBars = Input_NumberOfForwardProjectionBars.GetInt();
    Subgraph_CurrentBidLine.ExtendedArrayElementsToGraph = NumberOfBars;
    Subgraph_CurrentAskLine.ExtendedArrayElementsToGraph = NumberOfBars;

    for (int Index = 0; Index < NumberOfBars; Index++)
    {
        Subgraph_CurrentBidLine[sc.ArraySize + Index] = BidLineValue;
        Subgraph_CurrentAskLine[sc.ArraySize + Index] = AskLineValue;
    }
}
else
{
    Subgraph_CurrentBidLine.ExtendedArrayElementsToGraph = 0;
    Subgraph_CurrentAskLine.ExtendedArrayElementsToGraph = 0;
}

for (int Index = r_PriorLineStartIndex; Index < EarliestUpdateSubgraphDataArrayIndex; ++Index)
{
    Subgraph_CurrentBidLine[Index] = 0.0;
    Subgraph_CurrentAskLine[Index] = 0.0;
}

//if (r_PriorLineStartIndex != 0
// && r_PriorLineStartIndex < EarliestUpdateSubgraphDataArrayIndex)
//{
// EarliestUpdateSubgraphDataArrayIndex = r_PriorLineStartIndex;
//}
//else
r_PriorLineStartIndex = EarliestUpdateSubgraphDataArrayIndex;

if (EarliestUpdateSubgraphDataArrayIndex < sc.EarliestUpdateSubgraphDataArrayIndex)
    sc.EarliestUpdateSubgraphDataArrayIndex = EarliestUpdateSubgraphDataArrayIndex;
}

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

    if(sc.SetDefaults)
    {
        sc.GraphName = "Out of Order Timestamps Detector";

        Subgraph_OutOfOrderTimestamp.Name = "Out of Order Timestamp";
        Subgraph_OutOfOrderTimestamp.PrimaryColor = RGB(255,64,32);
    }
}

```

```

Subgraph_OutOfOrderTimestamp.DrawStyle = DRAWSTYLE_BACKGROUND;
Subgraph_OutOfOrderTimestamp.LineWidth = 5;
Subgraph_OutOfOrderTimestamp.DrawZeros = false;

sc.GraphRegion = 0;

sc.DisplayStudyInputValues= false;

sc.ScaleRangeType = SCALE_INDEPENDENT;

sc.DrawStudyUnderneathMainPriceGraph;

sc.AutoLoop = 1;

return;
}

sc.DataStartIndex = 0;

if (sc.Index <= 0)
return;

SCDateTime PreviousBarDateTime = sc.BaseDateTimeIn[sc.Index-1];
SCDateTime CurrentBarDateTime = sc.BaseDateTimeIn[sc.Index];

if(CurrentBarDateTime < PreviousBarDateTime)
    Subgraph_OutOfOrderTimestamp.Data[sc.Index] = 1;
else
    Subgraph_OutOfOrderTimestamp.Data[sc.Index] = 0;

return;
}

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

    if (sc.SetDefaults)
    {
        sc.GraphName = "Duplicate Timestamps Detector";

        Subgraph_DuplicateTimestamp.Name = "DuplicateTimestamp";
        Subgraph_DuplicateTimestamp.PrimaryColor = RGB(255, 64, 32);
        Subgraph_DuplicateTimestamp.DrawStyle = DRAWSTYLE_BACKGROUND;
        Subgraph_DuplicateTimestamp.LineWidth = 5;
        Subgraph_DuplicateTimestamp.DrawZeros = false;

        sc.GraphRegion = 0;

        sc.DisplayStudyInputValues = false;

        sc.ScaleRangeType = SCALE_INDEPENDENT;

        sc.DrawStudyUnderneathMainPriceGraph;

        sc.AutoLoop = 0;

        return;
    }

    sc.DataStartIndex = 1;

    int FirstValidIndex = 1;

```

```

for(int BarIndex = max(FirstValidIndex, sc.UpdateStartIndex); BarIndex < sc.ArraySize; BarIndex++)
{
    SCDateTimeMS PreviousBarDateTime = sc.BaseDateTimeIn[BarIndex - 1];
    SCDateTimeMS CurrentBarDateTime = sc.BaseDateTimeIn[BarIndex];

    if (CurrentBarDateTime == PreviousBarDateTime)
        Subgraph_DuplicateTimestamp.Data[BarIndex] = 1;
    else
        Subgraph_DuplicateTimestamp.Data[BarIndex] = 0;
}

}

/*****
* MoneyFlowIndex
*
* typical price: (high + low + close )/3
* money flow: typical price * volume
* money ratio: positive money flow / negative money flow
*****/
SCSFExport scsf_MoneyFlowIndex(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_MFI = sc.Subgraph[0];

    SCInputRef Input_Length = sc.Input[2];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Money Flow Index";
        sc.GraphRegion = 1;
        sc.ValueFormat = 3;

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

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

        sc.AutoLoop = 1;

        return;
    }

    const double BIAS_DIVISION_DOUBLE = static_cast<double>(10e-20);

    double typ_price = 0.0, prev_typ_price = 0.0, money_flow = 0.0, volume = 0.0, N_pos_mon_flow = 0.0,
    N_neg_mon_flow = 0.0, MoneyFlowIndex = 0.0, money_ratio = 0.0;

    sc.DataStartIndex = Input_Length.GetInt();

    if ((sc.Index - Input_Length.GetInt() + 1) != 0)
        prev_typ_price = sc.HLCAvg[sc.Index - Input_Length.GetInt()];

    for(int IndexOffset= 0; IndexOffset < Input_Length.GetInt(); IndexOffset++)
    {
        volume = sc.Volume[sc.Index - Input_Length.GetInt() + 1 + IndexOffset];
        typ_price = sc.HLCAvg[sc.Index - Input_Length.GetInt() + 1 + IndexOffset];
        money_flow = typ_price * volume;
        N_pos_mon_flow += typ_price > prev_typ_price ? money_flow : 0;
        N_neg_mon_flow += typ_price < prev_typ_price ? money_flow : 0;
    }

```

```

    prev_typ_price = typ_price;
}

money_ratio = N_pos_mon_flow / (N_neg_mon_flow + BIAS_DIVISION_DOUBLE);

MoneyFlowIndex = 100.0 - (100.0 / (1.0 + money_ratio));

Subgraph_MFI[sc.Index] = static_cast<float>(MoneyFlowIndex);
}

/*=====*/
SCSFExport scsf_VolumeWeightedAveragePrice(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_VWAPSubgraph = sc.Subgraph[0];
    SCFloatArrayRef Array_BarSumPriceTimesVolume = Subgraph_VWAPSubgraph.Arrays[0];
    SCFloatArrayRef Array_BarSumPriceSquaredTimesVolume = Subgraph_VWAPSubgraph.Arrays[1];
    SCFloatArrayRef Array_StandardDeviationArray = Subgraph_VWAPSubgraph.Arrays[2];
    SCFloatArrayRef Array_BarSumPriceVWAPDifferenceSquaredTimesVolume = Subgraph_VWAPSubgraph.Arrays[3];
    SCFloatArrayRef Array_CalculatedBarVolume = Subgraph_VWAPSubgraph.Arrays[4];

    SCSubgraphRef Subgraph_SD1Top = sc.Subgraph[1];
    SCSubgraphRef Subgraph_SD1Bottom = sc.Subgraph[2];
    SCSubgraphRef Subgraph_SD2Top = sc.Subgraph[3];
    SCSubgraphRef Subgraph_SD2Bottom = sc.Subgraph[4];
    SCSubgraphRef Subgraph_SD3Top = sc.Subgraph[5];
    SCSubgraphRef Subgraph_SD3Bottom = sc.Subgraph[6];
    SCSubgraphRef Subgraph_SD4Top = sc.Subgraph[7];
    SCSubgraphRef Subgraph_SD4Bottom = sc.Subgraph[8];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_BandCalculationMethod = sc.Input[1];
    SCInputRef Input_VolumeTypeToUse = sc.Input[2];
    SCInputRef Input_TimePeriodType = sc.Input[3];
    SCInputRef Input_TimePeriodLength = sc.Input[4];
    SCInputRef Input_Version = sc.Input[5];
    SCInputRef Input_DistanceMultiplier_V3 = sc.Input[6];
    SCInputRef Input_IgnoreTimePeriodTypeAndLength = sc.Input[7];
    SCInputRef Input_BaseOnUnderlyingData = sc.Input[9];
    SCInputRef Input_UseFixedOffset = sc.Input[10];
    SCInputRef Input_StartDateTime = sc.Input[11];
    SCInputRef Input_Band1OffsetMultiplier = sc.Input[12];
    SCInputRef Input_Band2OffsetMultiplier = sc.Input[13];
    SCInputRef Input_Band3OffsetMultiplier = sc.Input[14];
    SCInputRef Input_Band4OffsetMultiplier = sc.Input[15];

    if(sc.SetDefaults)
    {
        sc.GraphName = "Volume Weighted Average Price";
        sc.GraphRegion = 0;
        sc.ValueFormat = VALUEFORMAT_INHERITED;
        sc.AutoLoop = 1;
        sc.ScaleRangeType=SCALE_SAMEASREGION;

        Subgraph_VWAPSubgraph.Name = "VWAP";
        Subgraph_VWAPSubgraph.DrawStyle = DRAWSTYLE_DASH;
        Subgraph_VWAPSubgraph.LineWidth = 2;
        Subgraph_VWAPSubgraph.PrimaryColor = RGB (237, 13, 255);
        Subgraph_VWAPSubgraph.DrawZeros = false;

        Subgraph_SD1Top.Name = "Top Band 1";
        Subgraph_SD1Top.DrawStyle = DRAWSTYLE_DASH;
        Subgraph_SD1Top.LineWidth = 2;
        Subgraph_SD1Top.PrimaryColor = RGB (192, 192, 192);
    }
}

```

Subgraph_SD1Top.DrawZeros = false;

Subgraph_SD1Bottom.Name = "Bottom Band 1";
Subgraph_SD1Bottom.DrawStyle = DRAWSTYLE_DASH;
Subgraph_SD1Bottom.LineWidth = 2;
Subgraph_SD1Bottom.PrimaryColor = RGB (192, 192, 192);
Subgraph_SD1Bottom.DrawZeros = false;

Subgraph_SD2Top.Name = "Top Band 2";
Subgraph_SD2Top.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_SD2Top.LineWidth = 2;
Subgraph_SD2Top.PrimaryColor = RGB (192, 192, 192);
Subgraph_SD2Top.DrawZeros = false;

Subgraph_SD2Bottom.Name = "Bottom Band 2";
Subgraph_SD2Bottom.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_SD2Bottom.LineWidth = 2;
Subgraph_SD2Bottom.PrimaryColor = RGB (192, 192, 192);
Subgraph_SD2Bottom.DrawZeros = false;

Subgraph_SD3Top.Name = "Top Band 3";
Subgraph_SD3Top.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_SD3Top.LineWidth = 2;
Subgraph_SD3Top.PrimaryColor = RGB (192, 192, 192);
Subgraph_SD3Top.DrawZeros = false;

Subgraph_SD3Bottom.Name = "Bottom Band 3";
Subgraph_SD3Bottom.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_SD3Bottom.LineWidth = 2;
Subgraph_SD3Bottom.PrimaryColor = RGB (192, 192, 192);
Subgraph_SD3Bottom.DrawZeros = false;

Subgraph_SD4Top.Name = "Top Band 4";
Subgraph_SD4Top.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_SD4Top.LineWidth = 2;
Subgraph_SD4Top.PrimaryColor = RGB (192, 192, 192);
Subgraph_SD4Top.DrawZeros = false;

Subgraph_SD4Bottom.Name = "Bottom Band 4";
Subgraph_SD4Bottom.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_SD4Bottom.LineWidth = 2;
Subgraph_SD4Bottom.PrimaryColor = RGB (192, 192, 192);
Subgraph_SD4Bottom.DrawZeros = false;

int DisplayOrder = 1;

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

Input_VolumeTypeToUse.Name = "Volume Type to Use";
Input_VolumeTypeToUse.SetCustomInputStrings("Total Volume;Bid Volume;Ask Volume");
Input_VolumeTypeToUse.SetCustomInputIndex(0);
Input_VolumeTypeToUse.DisplayOrder = DisplayOrder++;

Input_BaseOnUnderlyingData.Name = "Base On Underlying Data";
Input_BaseOnUnderlyingData.SetYesNo(0);
Input_BaseOnUnderlyingData.DisplayOrder = DisplayOrder++;

Input_TimePeriodType.Name = "Time Period Type";
Input_TimePeriodType.SetTimePeriodType(TIME_PERIOD_LENGTH_UNIT_DAYS);
Input_TimePeriodType.DisplayOrder = DisplayOrder++;

Input_TimePeriodLength.Name = "Time Period Length";
Input_TimePeriodLength.SetInt(1);

```

Input_TimePeriodLength.DisplayOrder = DisplayOrder++;

Input_StartDateTime.Name = "Start Date-Time";
Input_StartDateTime.SetDateTime(0.0);
Input_StartDateTime.DisplayOrder = DisplayOrder++;

Input_IgnoreTimePeriodTypeAndLength.Name = "Ignore Time Period Type And Length";
Input_IgnoreTimePeriodTypeAndLength.SetYesNo(false);
Input_IgnoreTimePeriodTypeAndLength.DisplayOrder = DisplayOrder++;

Input_DistanceMultiplier_V3.SetFloat(0.5);

//UseFixedOffset.Name = "Use Fixed Offset Instead of Std. Deviation";
Input_UseFixedOffset.SetYesNo(0);
//UseFixedOffset.DisplayOrder = DisplayOrder++;

Input_BandCalculationMethod.Name = "Std. Deviation Band Calculation Method";
Input_BandCalculationMethod.SetCustomInputStrings("VWAP Variance;Fixed Offset;Standard
Deviation;Percentage");
Input_BandCalculationMethod.SetCustomInputIndex(0);
Input_BandCalculationMethod.DisplayOrder = DisplayOrder++;

Input_Band1OffsetMultiplier.Name = "Band 1 Std. Deviation Multiplier/Fixed Offset";
Input_Band1OffsetMultiplier.SetFloat(0.5);
Input_Band1OffsetMultiplier.DisplayOrder = DisplayOrder++;

Input_Band2OffsetMultiplier.Name = "Band 2 Std. Deviation Multiplier/Fixed Offset";
Input_Band2OffsetMultiplier.SetFloat(1.0);
Input_Band2OffsetMultiplier.DisplayOrder = DisplayOrder++;

Input_Band3OffsetMultiplier.Name = "Band 3 Std. Deviation Multiplier/Fixed Offset";
Input_Band3OffsetMultiplier.SetFloat(1.5);
Input_Band3OffsetMultiplier.DisplayOrder = DisplayOrder++;

Input_Band4OffsetMultiplier.Name = "Band 4 Std. Deviation Multiplier/Fixed Offset";
Input_Band4OffsetMultiplier.SetFloat(2.0);
Input_Band4OffsetMultiplier.DisplayOrder = DisplayOrder++;

Input_Version.SetInt(6);

return;
}

if(Input_Version.GetInt() < 2)
{
    Input_TimePeriodType.SetTimePeriodType(TIME_PERIOD_LENGTH_UNIT_DAYS);
    Input_TimePeriodLength.SetInt(1);

    Input_Version.SetInt(2);
}

if(Input_Version.GetInt() < 3)
{
    Input_DistanceMultiplier_V3.SetFloat(0.5);

    Input_Version.SetInt(3);
}

if (Input_Version.GetInt() < 4)
{
    Input_Band1OffsetMultiplier.SetFloat(Input_DistanceMultiplier_V3.GetFloat());
    Input_Band2OffsetMultiplier.SetFloat(Input_DistanceMultiplier_V3.GetFloat()*2.0f);
    Input_Band3OffsetMultiplier.SetFloat(Input_DistanceMultiplier_V3.GetFloat()*3.0f);
    Input_Band4OffsetMultiplier.SetFloat(Input_DistanceMultiplier_V3.GetFloat()*4.0f);
}

```

```

    Input_Version.SetInt(4);
}

if (Input_Version.GetInt() < 5)
{
    Input_BandCalculationMethod.SetCustomInputIndex(Input_UseFixedOffset.GetYesNo() ? 1 : 0);
    Input_Version.SetInt(5);
}

if (Input_Version.GetInt() < 6)
{
    Input_IgnoreTimePeriodTypeAndLength.SetYesNo(false);
    Input_Version.SetInt(6);
}

sc.DataStartIndex = 1;

int& r_LastIndexProcessed = sc.GetPersistentInt(1);
int& r_StartIndexOfCurrentPeriod = sc.GetPersistentInt(3);

double& r_PriorSumPriceTimesVolume = sc.GetPersistentDouble(1);
double& r_PriorSumPriceSquaredTimesVolume = sc.GetPersistentDouble(2);
double& r_PriorCumulativeVolume = sc.GetPersistentDouble(3);

if (sc.Index == 0)
{
    Array_BarSumPriceTimesVolume[sc.Index] = 0;
    Array_BarSumPriceSquaredTimesVolume[sc.Index] = 0;

    if (Input_BaseOnUnderlyingData.GetYesNo())
    {
        if (sc.MaintainVolumeAtPriceData == 0)
        {
            sc.MaintainVolumeAtPriceData = 1;
            sc.FlagToReloadChartData = 1;
        }
    }
    else
        sc.MaintainVolumeAtPriceData = 0;
}

if (Input_StartDateTime.GetDate() != 0
    && sc.BaseDateTimeIn[sc.Index] < Input_StartDateTime.GetDateTime())
    return;

bool PriorBarIndexExcludedBasedOnDateTime = sc.Index > 0
    && Input_StartDateTime.GetDate() != 0
    && sc.BaseDateTimeIn[sc.Index - 1] < Input_StartDateTime.GetDateTime();

SCDateTime PriorCurrentPeriodStartDateTime;

SCDateTime CurrentPeriodStartDateTime;

if (!Input_IgnoreTimePeriodTypeAndLength.GetYesNo())
{
    CurrentPeriodStartDateTime = sc.GetStartOfPeriodForDateTime(sc.BaseDateTimeIn[sc.Index],
Input_TimePeriodType.GetTimePeriodType(), Input_TimePeriodLength.GetInt(), 0);
    PriorCurrentPeriodStartDateTime = sc.GetStartOfPeriodForDateTime(sc.BaseDateTimeIn[sc.Index - 1],
Input_TimePeriodType.GetTimePeriodType(), Input_TimePeriodLength.GetInt(), 0);
}

bool IsStartOfNewPeriod = false;

//Reset at start of new period
if (sc.Index == 0

```

```

|| CurrentPeriodStartDateTime != PriorCurrentPeriodStartDateTime
|| PriorBarIndexExcludedBasedOnDateTime)
{

    r_PriorSumPriceTimesVolume = 0;
    r_PriorSumPriceSquaredTimesVolume = 0;
    r_PriorCumulativeVolume = 0;

    r_LastIndexProcessed = sc.Index;

    r_StartIndexOfCurrentPeriod = sc.Index;
    IsStartOfNewPeriod = true;
}
else if (sc.Index != r_LastIndexProcessed)
{
    // add in last values
    for (; r_LastIndexProcessed < sc.Index; ++r_LastIndexProcessed)
    {
        r_PriorSumPriceTimesVolume += Array_BarSumPriceTimesVolume[r_LastIndexProcessed];
        r_PriorSumPriceSquaredTimesVolume += Array_BarSumPriceSquaredTimesVolume[r_LastIndexProcessed];
        r_PriorCumulativeVolume += Array_CalculatedBarVolume[r_LastIndexProcessed];
    }
}

double BarVolume = 0;

float CurrentPrice = sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index];

if (Input_BaseOnUnderlyingData.GetYesNo())
{
    Array_BarSumPriceTimesVolume[sc.Index] = 0;
    Array_BarSumPriceSquaredTimesVolume[sc.Index] = 0;

    if (static_cast<int>(sc.VolumeAtPriceForBars->GetNumberOfBars()) < sc.ArraySize)
        return;

    const s_VolumeAtPriceV2 *p_VolumeAtPrice = nullptr;

    const int NumVAPElementsAtBarIndex = sc.VolumeAtPriceForBars->GetSizeAtBarIndex(sc.Index);
    for (int VAPIndex = 0; VAPIndex < NumVAPElementsAtBarIndex; VAPIndex++)
    {
        sc.VolumeAtPriceForBars->GetVAPElementAtIndex(sc.Index, VAPIndex, &p_VolumeAtPrice);

        float Price = p_VolumeAtPrice->PriceInTicks * sc.TickSize;

        uint32_t VolumeToUse = 0;
        if (Input_VolumeTypeToUse.GetIndex() == 1)
            VolumeToUse = p_VolumeAtPrice->BidVolume;
        else if (Input_VolumeTypeToUse.GetIndex() == 2)
            VolumeToUse = p_VolumeAtPrice->AskVolume;
        else
            VolumeToUse = p_VolumeAtPrice->Volume;

        Array_BarSumPriceTimesVolume[sc.Index] += (Price * VolumeToUse);
        Array_BarSumPriceSquaredTimesVolume[sc.Index] += (Price * Price * VolumeToUse);
        BarVolume += VolumeToUse;
    }
}
else
{
    Array_BarSumPriceTimesVolume[sc.Index] = CurrentPrice * sc.Volume[sc.Index];
    Array_BarSumPriceSquaredTimesVolume[sc.Index] = CurrentPrice * CurrentPrice * sc.Volume[sc.Index];
    BarVolume = sc.Volume[sc.Index];
}
}

```

```

Array_CalculatedBarVolume[sc.Index] = static_cast<float> (BarVolume);

double PriceTimesVolume = r_PriorSumPriceTimesVolume + Array_BarSumPriceTimesVolume[sc.Index];
double PriceSquaredTimesVolume = r_PriorSumPriceSquaredTimesVolume +
Array_BarSumPriceSquaredTimesVolume[sc.Index];
double CumulativeVolume = r_PriorCumulativeVolume + BarVolume;

double VolumeWeightedAveragePrice = 0;
if (CumulativeVolume != 0)
    VolumeWeightedAveragePrice = PriceTimesVolume / CumulativeVolume;

Subgraph_VWAPSubgraph[sc.Index] = static_cast<float>(VolumeWeightedAveragePrice);

float BandDistance = 1.0f;
if (Input_BandCalculationMethod.GetIndex() == 1)//Fixed offset
{
    BandDistance = 1.0f;
}
else if (Input_BandCalculationMethod.GetIndex() == 0)//VWAP variance
{
    double PriceVWAPDifferenceSquaredTimesVolume = (static_cast<double>(CurrentPrice) -
Subgraph_VWAPSubgraph[sc.Index]) * (static_cast<double>(CurrentPrice) - Subgraph_VWAPSubgraph[sc.Index]) *
sc.Volume[sc.Index];
    if (IsStartOfNewPeriod)
    {
        Array_BarSumPriceVWAPDifferenceSquaredTimesVolume[sc.Index] = static_cast<float>
(PPriceVWAPDifferenceSquaredTimesVolume);
    }
    else
    {
        Array_BarSumPriceVWAPDifferenceSquaredTimesVolume[sc.Index] =
static_cast<float>(Array_BarSumPriceVWAPDifferenceSquaredTimesVolume[sc.Index - 1] +
PriceVWAPDifferenceSquaredTimesVolume);
    }

    if (CumulativeVolume != 0)
        BandDistance = static_cast<float>
(sqrt(Array_BarSumPriceVWAPDifferenceSquaredTimesVolume[sc.Index]/CumulativeVolume));
    else
        BandDistance = 1.0f;
}
else if (Input_BandCalculationMethod.GetIndex() == 2)//VWAP variance//Standard deviation
{
    sc.StdDeviation(Subgraph_VWAPSubgraph, Array_StandardDeviationArray, sc.Index - r_StartIndexOfCurrentPeriod
+ 1);
    BandDistance = Array_StandardDeviationArray[sc.Index];
}
else //Percentage
{
    BandDistance = static_cast<float>(VolumeWeightedAveragePrice / 100.0f);
}

Subgraph_SD1Top[sc.Index] = Subgraph_VWAPSubgraph[sc.Index] + Input_Band1OffsetMultiplier.GetFloat() *
BandDistance;
Subgraph_SD1Bottom[sc.Index] = Subgraph_VWAPSubgraph[sc.Index] - Input_Band1OffsetMultiplier.GetFloat() *
BandDistance;

Subgraph_SD2Top[sc.Index] = Subgraph_VWAPSubgraph[sc.Index] + Input_Band2OffsetMultiplier.GetFloat() *
BandDistance;
Subgraph_SD2Bottom[sc.Index] = Subgraph_VWAPSubgraph[sc.Index] - Input_Band2OffsetMultiplier.GetFloat() *
BandDistance;

```

```

    Subgraph_SD3Top[sc.Index] = Subgraph_VWAPSubgraph[sc.Index] + Input_Band3OffsetMultiplier.GetFloat() *
BandDistance;
    Subgraph_SD3Bottom[sc.Index] = Subgraph_VWAPSubgraph[sc.Index] - Input_Band3OffsetMultiplier.GetFloat() *
BandDistance;

    Subgraph_SD4Top[sc.Index] = Subgraph_VWAPSubgraph[sc.Index] + Input_Band4OffsetMultiplier.GetFloat() *
BandDistance;
    Subgraph_SD4Bottom[sc.Index] = Subgraph_VWAPSubgraph[sc.Index] - Input_Band4OffsetMultiplier.GetFloat() *
BandDistance;
}

/*=====*/
void GetIndexForStartofDayPeriodForDaysBack
( SCStudyInterfaceRef sc
, const SCDatetime InitialStartDateTime
, const int NumDaysToInclude
, const bool ExcludeWeekends
, const int MinimumRequiredTimePeriodInDayAsPercent
, int& r_InitialBarIndex
)
{
    int DaysBack = -1;

    for (int DaysCount = 0; DaysCount < NumDaysToInclude; ++DaysCount)
    {
        for (int Iterations = 10; Iterations > 0; --Iterations)
        {
            DaysBack++;

            SCDatetime PriorPeriodStartDateTime = InitialStartDateTime - SCDatetime::DAYS(DaysBack);
            SCDatetime PriorPeriodEndDateTime = PriorPeriodStartDateTime + SCDatetime::DAYS(1) -
SCDatetime::SECONDS(1);

            r_InitialBarIndex = sc.GetNearestMatchForSCDatetime(sc.ChartNumber,
PriorPeriodStartDateTime.GetAsDouble());

            const SCDatetime TradingDayDate = sc.GetTradingDayDate(PriorPeriodStartDateTime);

            if (ExcludeWeekends && TradingDayDate.IsWeekend())
                continue;

            // for the current day, do not do any check to see if there is sufficient data because there may not be if the day is
not complete.
            if (NumDaysToInclude == 1)
                break;

            const int PriorPeriodLastIndex = sc.GetNearestMatchForSCDatetime(sc.ChartNumber,
PriorPeriodEndDateTime.GetAsDouble());

            const SCDatetime ActualTimeSpan(sc.CalculateTimeSpanAcrossChartBars
(r_InitialBarIndex, PriorPeriodLastIndex));

            //If there is a sufficient amount of data in this time period
            if (ActualTimeSpan > SCDatetime::SECONDS_Fraction(MinimumRequiredTimePeriodInDayAsPercent * 0.01 *
SECONDS_PER_DAY))
            {
                break;
            }
        }
    }
}

/*=====*/
SCSFExport scsf_RollingVolumeWeightedAveragePrice(SCStudyInterfaceRef sc)
{

```

```

SCSubgraphRef Subgraph_VWAPSubgraph = sc.Subgraph[0];
SCFloatArrayRef Array_BarSumPriceTimesVolume = Subgraph_VWAPSubgraph.Arrays[0];
SCFloatArrayRef Array_StandardDeviationArray = Subgraph_VWAPSubgraph.Arrays[1];
SCFloatArrayRef Array_PriceTimesVolumeArray = Subgraph_VWAPSubgraph.Arrays[2];
SCSubgraphRef Subgraph_BackgroundVWAPForCurrentPeriod = sc.Subgraph[9];
SCSubgraphRef Subgraph_PriceVWAPDifferenceSquaredTimesVolume = sc.Subgraph[10];
SCSubgraphRef Subgraph_BarSumPriceVWAPDifferenceSquaredTimesVolumeArray = sc.Subgraph[11];
SCSubgraphRef Subgraph_CumulativeVolumeForPeriodArray = sc.Subgraph[12];
SCSubgraphRef Subgraph_BandDistance = sc.Subgraph[13];

```

```

SCSubgraphRef Subgraph_SD1Top = sc.Subgraph[1];
SCSubgraphRef Subgraph_SD1Bottom = sc.Subgraph[2];
SCSubgraphRef Subgraph_SD2Top = sc.Subgraph[3];
SCSubgraphRef Subgraph_SD2Bottom = sc.Subgraph[4];
SCSubgraphRef Subgraph_SD3Top = sc.Subgraph[5];
SCSubgraphRef Subgraph_SD3Bottom = sc.Subgraph[6];
SCSubgraphRef Subgraph_SD4Top = sc.Subgraph[7];
SCSubgraphRef Subgraph_SD4Bottom = sc.Subgraph[8];

```

```

SCInputRef Input_InputData = sc.Input[0];
SCInputRef Input_BandCalculationMethod = sc.Input[1];
SCInputRef Input_MinimumRequiredTimePeriodAsPercentForSkipDays = sc.Input[2];
SCInputRef Input_TimePeriodType = sc.Input[3];
SCInputRef Input_TimePeriodLength = sc.Input[4];
SCInputRef Input_Version = sc.Input[5];
SCInputRef Input_DistanceMultiplier_V3 = sc.Input[6];
SCInputRef Input_NumberOfDaysToCalculate = sc.Input[7];
SCInputRef Input_BaseOnUnderlyingData = sc.Input[9];
SCInputRef Input_UseFixedOffset = sc.Input[10];
SCInputRef Input_Band1OffsetMultiplier = sc.Input[12];
SCInputRef Input_Band2OffsetMultiplier = sc.Input[13];
SCInputRef Input_Band3OffsetMultiplier = sc.Input[14];
SCInputRef Input_Band4OffsetMultiplier = sc.Input[15];
SCInputRef Input_ExcludeWeekendInDateCalc = sc.Input[16];
SCInputRef Input_VolumeTypeToUse = sc.Input[17];

```

```

if (sc.SetDefaults)
{
    sc.GraphName = "Volume Weighted Average Price - Rolling";
    sc.GraphRegion = 0;
    sc.ValueFormat = VALUEFORMAT_INHERITED;
    sc.AutoLoop = 0;//Manual looping
    sc.MaintainVolumeAtPriceData = 1;

    sc.ScaleRangeType = SCALE_SAMEASREGION;

    Subgraph_VWAPSubgraph.Name = "VWAP";
    Subgraph_VWAPSubgraph.DrawStyle = DRAWSTYLE_DASH;
    Subgraph_VWAPSubgraph.LineWidth = 2;
    Subgraph_VWAPSubgraph.PrimaryColor = RGB(237, 13, 255);
    Subgraph_VWAPSubgraph.DrawZeros = false;

    Subgraph_SD1Top.Name = "Top Band 1";
    Subgraph_SD1Top.DrawStyle = DRAWSTYLE_DASH;
    Subgraph_SD1Top.LineWidth = 2;
    Subgraph_SD1Top.PrimaryColor = RGB(192, 192, 192);
    Subgraph_SD1Top.DrawZeros = false;

    Subgraph_SD1Bottom.Name = "Bottom Band 1";
    Subgraph_SD1Bottom.DrawStyle = DRAWSTYLE_DASH;
    Subgraph_SD1Bottom.LineWidth = 2;
    Subgraph_SD1Bottom.PrimaryColor = RGB(192, 192, 192);
    Subgraph_SD1Bottom.DrawZeros = false;

    Subgraph_SD2Top.Name = "Top Band 2";

```

```
Subgraph_SD2Top.DrawStyle = DRAWSTYLE_IGNORE;  
Subgraph_SD2Top.LineWidth = 2;  
Subgraph_SD2Top.PrimaryColor = RGB(192, 192, 192);  
Subgraph_SD2Top.DrawZeros = false;
```

```
Subgraph_SD2Bottom.Name = "Bottom Band 2";  
Subgraph_SD2Bottom.DrawStyle = DRAWSTYLE_IGNORE;  
Subgraph_SD2Bottom.LineWidth = 2;  
Subgraph_SD2Bottom.PrimaryColor = RGB(192, 192, 192);  
Subgraph_SD2Bottom.DrawZeros = false;
```

```
Subgraph_SD3Top.Name = "Top Band 3";  
Subgraph_SD3Top.DrawStyle = DRAWSTYLE_IGNORE;  
Subgraph_SD3Top.LineWidth = 2;  
Subgraph_SD3Top.PrimaryColor = RGB(192, 192, 192);  
Subgraph_SD3Top.DrawZeros = false;
```

```
Subgraph_SD3Bottom.Name = "Bottom Band 3";  
Subgraph_SD3Bottom.DrawStyle = DRAWSTYLE_IGNORE;  
Subgraph_SD3Bottom.LineWidth = 2;  
Subgraph_SD3Bottom.PrimaryColor = RGB(192, 192, 192);  
Subgraph_SD3Bottom.DrawZeros = false;
```

```
Subgraph_SD4Top.Name = "Top Band 4";  
Subgraph_SD4Top.DrawStyle = DRAWSTYLE_IGNORE;  
Subgraph_SD4Top.LineWidth = 2;  
Subgraph_SD4Top.PrimaryColor = RGB(192, 192, 192);  
Subgraph_SD4Top.DrawZeros = false;
```

```
Subgraph_SD4Bottom.Name = "Bottom Band 4";  
Subgraph_SD4Bottom.DrawStyle = DRAWSTYLE_IGNORE;  
Subgraph_SD4Bottom.LineWidth = 2;  
Subgraph_SD4Bottom.PrimaryColor = RGB(192, 192, 192);  
Subgraph_SD4Bottom.DrawZeros = false;
```

```
Subgraph_BackgroundVWAPForCurrentPeriod.Name = "BackgroundVWAPForCurrentPeriod";  
Subgraph_BackgroundVWAPForCurrentPeriod.DrawStyle = DRAWSTYLE_IGNORE;  
Subgraph_BackgroundVWAPForCurrentPeriod.DisplayNameValueInWindowsFlags = 1;
```

```
Subgraph_PriceVWAPDifferenceSquaredTimesVolume.Name = "PriceVWAPDifferenceSquaredTimesVolume";  
Subgraph_PriceVWAPDifferenceSquaredTimesVolume.DrawStyle = DRAWSTYLE_IGNORE;  
Subgraph_PriceVWAPDifferenceSquaredTimesVolume.DisplayNameValueInWindowsFlags = 1;
```

```
Subgraph_BarSumPriceVWAPDifferenceSquaredTimesVolumeArray.Name =  
"BarSumPriceVWAPDifferenceSquaredTimesVolumeArray";  
Subgraph_BarSumPriceVWAPDifferenceSquaredTimesVolumeArray.DrawStyle = DRAWSTYLE_IGNORE;  
Subgraph_BarSumPriceVWAPDifferenceSquaredTimesVolumeArray.DisplayNameValueInWindowsFlags = 1;
```

```
Subgraph_CumulativeVolumeForPeriodArray.Name = "CumulativeVolumeForPeriodArray";  
Subgraph_CumulativeVolumeForPeriodArray.DrawStyle = DRAWSTYLE_IGNORE;  
Subgraph_CumulativeVolumeForPeriodArray.DisplayNameValueInWindowsFlags = 1;
```

```
Subgraph_BandDistance.Name = "BandDistance";  
Subgraph_BandDistance.DrawStyle = DRAWSTYLE_IGNORE;  
Subgraph_BandDistance.DisplayNameValueInWindowsFlags = 1;
```

```
int DisplayOrder = 1;
```

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

```
Input_VolumeTypeToUse.Name = "Volume Type to Use";  
Input_VolumeTypeToUse.SetCustomInputStrings("Total Volume;Bid Volume;Ask Volume");  
Input_VolumeTypeToUse.SetCustomInputIndex(0);
```

```

Input_VolumeTypeToUse.DisplayOrder = DisplayOrder++;

Input_BaseOnUnderlyingData.Name = "Base On Underlying Data";
Input_BaseOnUnderlyingData.SetYesNo(0);
Input_BaseOnUnderlyingData.DisplayOrder = DisplayOrder++;

Input_TimePeriodType.Name = "Time Period Type";
Input_TimePeriodType.SetCustomInputStrings("Days - Trading Days;Days - 24 Hour Period;Minutes;Bars");
Input_TimePeriodType.SetCustomInputIndex(0);
Input_TimePeriodType.DisplayOrder = DisplayOrder++;

Input_TimePeriodLength.Name = "Time Period Length";
Input_TimePeriodLength.SetInt(1);
Input_TimePeriodLength.DisplayOrder = DisplayOrder++;

Input_NumberOfDaysToCalculate.Name = "Number of Days to Calculate";
Input_NumberOfDaysToCalculate.SetInt(10);
Input_NumberOfDaysToCalculate.DisplayOrder = DisplayOrder++;

Input_DistanceMultiplier_V3.SetFloat(0.5);

Input_ExcludeWeekendInDateCalc.Name = "Exclude Weekends In Date Look Back";
Input_ExcludeWeekendInDateCalc.SetYesNo(1);
Input_ExcludeWeekendInDateCalc.DisplayOrder = DisplayOrder++;

Input_MinimumRequiredTimePeriodAsPercentForSkipDays.Name = "Minimum Required Time Period as Percent for
Skip Days";
Input_MinimumRequiredTimePeriodAsPercentForSkipDays.SetInt(0);
Input_MinimumRequiredTimePeriodAsPercentForSkipDays.DisplayOrder = DisplayOrder++;

//UseFixedOffset.Name = "Use Fixed Offset Instead of Std. Deviation";
Input_UseFixedOffset.SetYesNo(0);
//UseFixedOffset.DisplayOrder = DisplayOrder++;

Input_BandCalculationMethod.Name = "Std. Deviation Band Calculation Method";
Input_BandCalculationMethod.SetCustomInputStrings("VWAP Variance;Fixed Offset;Standard
Deviation;Percentage");
Input_BandCalculationMethod.SetCustomInputIndex(0);
Input_BandCalculationMethod.DisplayOrder = DisplayOrder++;

Input_Band1OffsetMultiplier.Name = "Band 1 Std. Deviation Multiplier/Fixed Offset";
Input_Band1OffsetMultiplier.SetFloat(0.5);
Input_Band1OffsetMultiplier.DisplayOrder = DisplayOrder++;

Input_Band2OffsetMultiplier.Name = "Band 2 Std. Deviation Multiplier/Fixed Offset";
Input_Band2OffsetMultiplier.SetFloat(1.0);
Input_Band2OffsetMultiplier.DisplayOrder = DisplayOrder++;

Input_Band3OffsetMultiplier.Name = "Band 3 Std. Deviation Multiplier/Fixed Offset";
Input_Band3OffsetMultiplier.SetFloat(1.5);
Input_Band3OffsetMultiplier.DisplayOrder = DisplayOrder++;

Input_Band4OffsetMultiplier.Name = "Band 4 Std. Deviation Multiplier/Fixed Offset";
Input_Band4OffsetMultiplier.SetFloat(2.0);
Input_Band4OffsetMultiplier.DisplayOrder = DisplayOrder++;

Input_Version.SetInt(8);

return;
}

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

if (Input_NumberOfDaysToCalculate.GetInt() == 0)

```

```

Input_NumberOfDaysToCalculate.SetInt(10);

// calculate the period start index
const int PeriodType = Input_TimePeriodType.GetIndex();

if (PeriodType == 0 || PeriodType == 1)
{
    int DaysRequired = Input_TimePeriodLength.GetInt();

    if (Input_NumberOfDaysToCalculate.GetInt() < DaysRequired)
        Input_NumberOfDaysToCalculate.SetInt(DaysRequired);
}

int& r_LastIndexProcessed = sc.GetPersistentInt(1);
int& r_FirstIndexIncluded = sc.GetPersistentInt(2);
int& r_PriorFirstIndexIncluded = sc.GetPersistentInt(3);

int& r_LastExcludedIndex = sc.GetPersistentInt(4);

SCDateTime& r_PriorTradeDate = sc.GetPersistentSCDateTime(1);

if (sc.IsFullRecalculation)//reset
{
    r_LastIndexProcessed = -1;
    r_FirstIndexIncluded = 0;
    r_PriorFirstIndexIncluded = -1;
    r_LastExcludedIndex = -1;

    if (Input_BaseOnUnderlyingData.GetYesNo())
    {
        if (sc.MaintainVolumeAtPriceData == 0)
        {
            sc.MaintainVolumeAtPriceData = 1;
            sc.FlagToReloadChartData = 1;
        }
    }
    else
    {
        sc.MaintainVolumeAtPriceData = 0;
    }
}

if (Input_BaseOnUnderlyingData.GetYesNo() &&
    static_cast<int>(sc.VolumeAtPriceForBars->GetNumberOfBars()) < sc.ArraySize)
{
    return;
}

const SCDateTime TradingDayDateForLastBar = sc.GetTradingDayDate(sc.BaseDateTimeIn[sc.ArraySize - 1]);

for (int Index = sc.UpdateStartIndex; Index < sc.ArraySize; ++Index)
{
    bool NewBar = Index != r_LastIndexProcessed;

    r_LastIndexProcessed = Index;

    Array_BarSumPriceTimesVolume[Index] = 0;

    const SCDateTime TradingDayDateForBar = sc.GetTradingDayDate(sc.BaseDateTimeIn[Index]);

    if ((TradingDayDateForLastBar.GetDate() - TradingDayDateForBar.GetDate() + 1)
        > Input_NumberOfDaysToCalculate.GetInt())

```

```

{
    r_LastExcludedIndex = Index;
    continue;
}

const int VolumeTypeToUse = Input_VolumeTypeToUse.GetIndex();

const int InputDataIndex = Input_InputData.GetInputDataIndex();

float CurrentPrice = sc.BaseDataIn[InputDataIndex][Index];

// Calculate bar values
if (Input_BaseOnUnderlyingData.GetYesNo())
{
    const s_VolumeAtPriceV2 *p_VolumeAtPrice = nullptr;
    int NumberOfPrices = sc.VolumeAtPriceForBars->GetSizeAtBarIndex(Index);

    for (int VAPIndex = 0; VAPIndex < NumberOfPrices; VAPIndex++)
    {
        sc.VolumeAtPriceForBars->GetVAPElementAtIndex(Index, VAPIndex, &p_VolumeAtPrice);

        float Price = p_VolumeAtPrice->PriceInTicks * sc.TickSize;
        uint32_t VolumeToUse = 0;

        if (VolumeTypeToUse == 1)
            VolumeToUse = p_VolumeAtPrice->BidVolume;
        else if (VolumeTypeToUse == 2)
            VolumeToUse = p_VolumeAtPrice->AskVolume;
        else
            VolumeToUse = p_VolumeAtPrice->Volume;

        const float PriceTimesVolume = Price * VolumeToUse;

        Array_BarSumPriceTimesVolume[Index] += PriceTimesVolume;
    }
}
else
{
    Array_BarSumPriceTimesVolume[Index] = CurrentPrice * sc.Volume[Index];
}

if (NewBar)
{
    if (PeriodType == 0) // Days - trading days
    {
        int TradeDate = sc.GetTradingDayDate(sc.BaseDateTimeIn[Index]);

        SCDateTime CurrentDayStartDateTime = sc.GetStartDateTimeForTradingDate(TradeDate);

        r_PriorTradeDate = TradeDate;

        r_FirstIndexIncluded = sc.GetContainingIndexForSCDateTime(sc.ChartNumber,
CurrentDayStartDateTime.GetAsDouble());

        GetIndexForStartOfDayPeriodForDaysBack
        (sc
        , CurrentDayStartDateTime
        , Input_TimePeriodLength.GetInt()
        , Input_ExcludeWeekendInDateCalc.GetYesNo() != 0
        , Input_MinimumRequiredTimePeriodAsPercentForSkipDays.GetInt()
        , r_FirstIndexIncluded
        );
    }
    else if (PeriodType == 1) // Days - 24 hour periods

```

```

{
    SCDateTime CurrentPeriodStartDateTime = sc.BaseDateTimeIn[Index] - SCDateTime::DAYS(1) +
    SCDateTime::SECONDS(1);
    r_FirstIndexIncluded = sc.GetContainingIndexForSCDateTime(sc.ChartNumber,
    CurrentPeriodStartDateTime.GetAsDouble());

    GetIndexForStartOfDayPeriodForDaysBack
    (sc
        , CurrentPeriodStartDateTime
        , Input_TimePeriodLength.GetInt()
        , Input_ExcludeWeekendInDateCalc.GetYesNo() != 0
        , Input_MinimumRequiredTimePeriodAsPercentForSkipDays.GetInt()
        , r_FirstIndexIncluded
    );
}
else if (PeriodType == 2) // Minutes
{
    SCDateTime RollingStartDateTime = 0;

    if (BarPeriod.IntradayChartBarPeriodType == IBPT_DAYS_MINS_SECS && sc.SecondsPerBar != 0)
    {
        RollingStartDateTime = sc.BaseDateTimeIn[Index] + SCDateTime::SECONDS(sc.SecondsPerBar - 1);
        RollingStartDateTime.SubtractMinutes(Input_TimePeriodLength.GetInt());
    }
    else
    {
        if (Index == sc.ArraySize - 1)
            RollingStartDateTime = sc.LatestDateTimeForLastBar;
        else
            RollingStartDateTime = sc.BaseDateTimeIn[Index + 1] - SCDateTime::SECONDS(1);

        RollingStartDateTime.SubtractMinutes(Input_TimePeriodLength.GetInt());
    }

    for (; r_FirstIndexIncluded <= Index; ++r_FirstIndexIncluded)
    {
        if (sc.BaseDateTimeIn[r_FirstIndexIncluded] < RollingStartDateTime)
            continue;

        break;
    }
}
else // Bars
{
    r_FirstIndexIncluded = max(0, Index - Input_TimePeriodLength.GetInt() + 1);
}

if (r_FirstIndexIncluded <= r_LastExcludedIndex)
    r_FirstIndexIncluded = r_LastExcludedIndex + 1;

int CalculationStartIndex = Index;

bool ResetCalculation = false;

if (r_PriorFirstIndexIncluded != r_FirstIndexIncluded
    || r_FirstIndexIncluded == Index)
{
    CalculationStartIndex = r_FirstIndexIncluded;
    ResetCalculation = true;
}

r_PriorFirstIndexIncluded = r_FirstIndexIncluded;

bool TempResetCalculation = ResetCalculation;

```

```

for (int AvgIndex = CalculationStartIndex; AvgIndex <= Index; ++AvgIndex)
{
    if (TempResetCalculation)
    {
        TempResetCalculation = false;

        Array_PriceTimesVolumeArray[AvgIndex] = Array_BarSumPriceTimesVolume[AvgIndex];

        if (VolumeTypeToUse == 1)
        {
            Subgraph_CumulativeVolumeForPeriodArray[AvgIndex] = sc.BidVolume[AvgIndex];
        }
        else if (VolumeTypeToUse == 2)
        {
            Subgraph_CumulativeVolumeForPeriodArray[AvgIndex] = sc.AskVolume[AvgIndex];
        }
        else
        {
            Subgraph_CumulativeVolumeForPeriodArray[AvgIndex] = sc.Volume[AvgIndex];
        }
    }
    else
    {
        Array_PriceTimesVolumeArray[AvgIndex] = Array_PriceTimesVolumeArray[AvgIndex - 1] +
        Array_BarSumPriceTimesVolume[AvgIndex];

        if (VolumeTypeToUse == 1)
        {
            Subgraph_CumulativeVolumeForPeriodArray[AvgIndex] =
            Subgraph_CumulativeVolumeForPeriodArray[AvgIndex - 1] + sc.BidVolume[AvgIndex];
        }
        else if (VolumeTypeToUse == 2)
        {
            Subgraph_CumulativeVolumeForPeriodArray[AvgIndex] =
            Subgraph_CumulativeVolumeForPeriodArray[AvgIndex - 1] + sc.AskVolume[AvgIndex];
        }
        else//Total volume
        {
            Subgraph_CumulativeVolumeForPeriodArray[AvgIndex] =
            Subgraph_CumulativeVolumeForPeriodArray[AvgIndex - 1] + sc.Volume[AvgIndex];
        }
    }

    if (Subgraph_CumulativeVolumeForPeriodArray[AvgIndex] != 0)
    {
        Subgraph_BackgroundVWAPForCurrentPeriod[AvgIndex] = Array_PriceTimesVolumeArray[AvgIndex] /
        Subgraph_CumulativeVolumeForPeriodArray[AvgIndex];
    }
    else
    {
        Subgraph_BackgroundVWAPForCurrentPeriod[AvgIndex] = 0;
    }
}

if (Subgraph_CumulativeVolumeForPeriodArray[Index] != 0)
{
    Subgraph_VWAPSubgraph[Index] = Array_PriceTimesVolumeArray[Index] /
    Subgraph_CumulativeVolumeForPeriodArray[Index];
}

if (Input_BandCalculationMethod.GetIndex() == 1)//Fixed offset
{

```

```

    Subgraph_BandDistance[Index] = 1.0f;
}
else if (Input_BandCalculationMethod.GetIndex() == 0)//VWAP variance
{
    for (int BarIndex = CalculationStartIndex; BarIndex <= Index; ++BarIndex)
    {
        float VolumeToUseInCalculation = 0;
        if (VolumeTypeToUse == 1)
        {
            VolumeToUseInCalculation = sc.BidVolume[BarIndex];
        }
        else if (VolumeTypeToUse == 2)
        {
            VolumeToUseInCalculation = sc.AskVolume[BarIndex];
        }
        else
        {
            VolumeToUseInCalculation = sc.Volume[BarIndex];
        }

        Subgraph_PriceVWAPDifferenceSquaredTimesVolume[BarIndex] =
            (sc.BaseDataIn[InputDataIndex][BarIndex]
             - Subgraph_BackgroundVWAPForCurrentPeriod[BarIndex])
            * (sc.BaseDataIn[InputDataIndex][BarIndex]
              - Subgraph_BackgroundVWAPForCurrentPeriod[BarIndex])
            * VolumeToUseInCalculation;

        if (ResetCalculation)
        {
            ResetCalculation = false;

            Subgraph_BarSumPriceVWAPDifferenceSquaredTimesVolumeArray[BarIndex] =
Subgraph_PriceVWAPDifferenceSquaredTimesVolume[BarIndex];
        }
        else
        {
            Subgraph_BarSumPriceVWAPDifferenceSquaredTimesVolumeArray[BarIndex] =
Subgraph_BarSumPriceVWAPDifferenceSquaredTimesVolumeArray[BarIndex - 1] +
Subgraph_PriceVWAPDifferenceSquaredTimesVolume[BarIndex];

        }
    }

    if (Subgraph_CumulativeVolumeForPeriodArray[Index] != 0)
    {
        Subgraph_BandDistance[Index] = static_cast<float>
(sqrt(Subgraph_BarSumPriceVWAPDifferenceSquaredTimesVolumeArray[Index] /
Subgraph_CumulativeVolumeForPeriodArray[Index]));
    }
    else
        Subgraph_BandDistance[Index] = 1.0f;

}
else if (Input_BandCalculationMethod.GetIndex() == 2) //Standard deviation
{
    sc.StdDeviation(Subgraph_VWAPSubgraph, Array_StandardDeviationArray, Index, Index - r_FirstIndexIncluded +
1);
    Subgraph_BandDistance[Index] = Array_StandardDeviationArray[Index];
}
else //Percentage
{
    Subgraph_BandDistance[Index] = Subgraph_VWAPSubgraph[Index] / 100.0f;
}

Subgraph_SD1Top[Index] = Subgraph_VWAPSubgraph[Index] + Input_Band1OffsetMultiplier.GetFloat() *

```

```

Subgraph_BandDistance[Index];

    Subgraph_SD1Bottom[Index] = Subgraph_VWAPSubgraph[Index] - Input_Band1OffsetMultiplier.GetFloat() *
Subgraph_BandDistance[Index];

    Subgraph_SD2Top[Index] = Subgraph_VWAPSubgraph[Index] + Input_Band2OffsetMultiplier.GetFloat() *
Subgraph_BandDistance[Index];

    Subgraph_SD2Bottom[Index] = Subgraph_VWAPSubgraph[Index] - Input_Band2OffsetMultiplier.GetFloat() *
Subgraph_BandDistance[Index];

    Subgraph_SD3Top[Index] = Subgraph_VWAPSubgraph[Index] + Input_Band3OffsetMultiplier.GetFloat() *
Subgraph_BandDistance[Index];

    Subgraph_SD3Bottom[Index] = Subgraph_VWAPSubgraph[Index] - Input_Band3OffsetMultiplier.GetFloat() *
Subgraph_BandDistance[Index];

    Subgraph_SD4Top[Index] = Subgraph_VWAPSubgraph[Index] + Input_Band4OffsetMultiplier.GetFloat() *
Subgraph_BandDistance[Index];

    Subgraph_SD4Bottom[Index] = Subgraph_VWAPSubgraph[Index] - Input_Band4OffsetMultiplier.GetFloat() *
Subgraph_BandDistance[Index];
}
}

/*=====*/
SCSFExport scsf_RollingHighAccuracyMovingAverage(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_MovingAverage = sc.Subgraph[0];
    SCFloatArrayRef Array_PriceSum = Subgraph_MovingAverage.Arrays[0];
    SCFloatArrayRef Array_PriceCount = Subgraph_MovingAverage.Arrays[1];
    SCFloatArrayRef Array_StandardDeviation = Subgraph_MovingAverage.Arrays[2];

    SCSubgraphRef Subgraph_SD1Top = sc.Subgraph[1];
    SCSubgraphRef Subgraph_SD1Bottom = sc.Subgraph[2];
    SCSubgraphRef Subgraph_SD2Top = sc.Subgraph[3];
    SCSubgraphRef Subgraph_SD2Bottom = sc.Subgraph[4];
    SCSubgraphRef Subgraph_SD3Top = sc.Subgraph[5];
    SCSubgraphRef Subgraph_SD3Bottom = sc.Subgraph[6];
    SCSubgraphRef Subgraph_SD4Top = sc.Subgraph[7];
    SCSubgraphRef Subgraph_SD4Bottom = sc.Subgraph[8];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_BandCalculationMethod = sc.Input[1];
    SCInputRef Input_MinimumRequiredTimePeriodAsPercentForSkipDays = sc.Input[2];
    SCInputRef Input_TimePeriodType = sc.Input[3];
    SCInputRef Input_TimePeriodLength = sc.Input[4];
    SCInputRef Input_Version = sc.Input[5];
    SCInputRef Input_NumberOfDaysToCalculate = sc.Input[7];
    SCInputRef Input_BaseOnUnderlyingData = sc.Input[8];
    SCInputRef Input_Band1OffsetMultiplier = sc.Input[9];
    SCInputRef Input_Band2OffsetMultiplier = sc.Input[10];
    SCInputRef Input_Band3OffsetMultiplier = sc.Input[11];
    SCInputRef Input_Band4OffsetMultiplier = sc.Input[12];
    SCInputRef Input_ExcludeWeekendInDateCalc = sc.Input[13];

    if(sc.SetDefaults)
    {
        sc.GraphName = "Moving Average - Rolling High Accuracy";
        sc.GraphRegion = 0;
        sc.ValueFormat = VALUEFORMAT_INHERITED;
        sc.AutoLoop = 0;//Manual looping
        sc.MaintainVolumeAtPriceData = 1;
    }
}

```

```
Subgraph_MovingAverage.Name = "Moving Average";
Subgraph_MovingAverage.DrawStyle = DRAWSTYLE_LINE;
Subgraph_MovingAverage.LineWidth = 2;
Subgraph_MovingAverage.PrimaryColor = RGB(237, 13, 255);
Subgraph_MovingAverage.DrawZeros = false;
```

```
Subgraph_SD1Top.Name = "Top Band 1";
Subgraph_SD1Top.DrawStyle = DRAWSTYLE_DASH;
Subgraph_SD1Top.LineWidth = 2;
Subgraph_SD1Top.PrimaryColor = RGB (192, 192, 192);
Subgraph_SD1Top.DrawZeros = false;
```

```
Subgraph_SD1Bottom.Name = "Bottom Band 1";
Subgraph_SD1Bottom.DrawStyle = DRAWSTYLE_DASH;
Subgraph_SD1Bottom.LineWidth = 2;
Subgraph_SD1Bottom.PrimaryColor = RGB (192, 192, 192);
Subgraph_SD1Bottom.DrawZeros = false;
```

```
Subgraph_SD2Top.Name = "Top Band 2";
Subgraph_SD2Top.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_SD2Top.LineWidth = 2;
Subgraph_SD2Top.PrimaryColor = RGB (192, 192, 192);
Subgraph_SD2Top.DrawZeros = false;
```

```
Subgraph_SD2Bottom.Name = "Bottom Band 2";
Subgraph_SD2Bottom.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_SD2Bottom.LineWidth = 2;
Subgraph_SD2Bottom.PrimaryColor = RGB (192, 192, 192);
Subgraph_SD2Bottom.DrawZeros = false;
```

```
Subgraph_SD3Top.Name = "Top Band 3";
Subgraph_SD3Top.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_SD3Top.LineWidth = 2;
Subgraph_SD3Top.PrimaryColor = RGB (192, 192, 192);
Subgraph_SD3Top.DrawZeros = false;
```

```
Subgraph_SD3Bottom.Name = "Bottom Band 3";
Subgraph_SD3Bottom.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_SD3Bottom.LineWidth = 2;
Subgraph_SD3Bottom.PrimaryColor = RGB (192, 192, 192);
Subgraph_SD3Bottom.DrawZeros = false;
```

```
Subgraph_SD4Top.Name = "Top Band 4";
Subgraph_SD4Top.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_SD4Top.LineWidth = 2;
Subgraph_SD4Top.PrimaryColor = RGB (192, 192, 192);
Subgraph_SD4Top.DrawZeros = false;
```

```
Subgraph_SD4Bottom.Name = "Bottom Band 4";
Subgraph_SD4Bottom.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_SD4Bottom.LineWidth = 2;
Subgraph_SD4Bottom.PrimaryColor = RGB (192, 192, 192);
Subgraph_SD4Bottom.DrawZeros = false;
```

```
int DisplayOrder = 1;
```

```
Input_InputData.Name = "Input Data";
Input_InputData.SetCustomInputStrings("Price;Volume;Number of Trades;AskVolume-BidVolume");
Input_InputData.SetCustomInputIndex(0);
Input_InputData.DisplayOrder = DisplayOrder++;
```

```
Input_TimePeriodType.Name = "Time Period Type";
Input_TimePeriodType.SetCustomInputStrings("Days - Trading Days;Days - 24 Hour Period;Minutes;Bars");
Input_TimePeriodType.SetCustomInputIndex(0);
Input_TimePeriodType.DisplayOrder = DisplayOrder++;
```

```

Input_TimePeriodLength.Name = "Time Period Length";
Input_TimePeriodLength.SetInt(1);
Input_TimePeriodLength.DisplayOrder = DisplayOrder++;

Input_NumberOfDaysToCalculate.Name = "Number of Days to Calculate";
Input_NumberOfDaysToCalculate.SetInt(10);
Input_NumberOfDaysToCalculate.DisplayOrder = DisplayOrder++;

Input_ExcludeWeekendInDateCalc.Name = "Exclude Weekends In Date Look Back";
Input_ExcludeWeekendInDateCalc.SetYesNo(1);
Input_ExcludeWeekendInDateCalc.DisplayOrder = DisplayOrder++;

Input_MinimumRequiredTimePeriodAsPercentForSkipDays.Name = "Minimum Required Time Period as Percent for
Skip Days";
Input_MinimumRequiredTimePeriodAsPercentForSkipDays.SetInt(25);
Input_MinimumRequiredTimePeriodAsPercentForSkipDays.DisplayOrder = DisplayOrder++;

Input_BandCalculationMethod.Name = "Std. Deviation Band Calculation Method";
Input_BandCalculationMethod.SetCustomInputStrings("Standard Deviation;Fixed Offset");
Input_BandCalculationMethod.SetCustomInputIndex(1);
Input_BandCalculationMethod.DisplayOrder = DisplayOrder++;

Input_Band1OffsetMultiplier.Name = "Band 1 Std. Deviation Multiplier/Fixed Offset";
Input_Band1OffsetMultiplier.SetFloat(0.5);
Input_Band1OffsetMultiplier.DisplayOrder = DisplayOrder++;

Input_Band2OffsetMultiplier.Name = "Band 2 Std. Deviation Multiplier/Fixed Offset";
Input_Band2OffsetMultiplier.SetFloat(1.0);
Input_Band2OffsetMultiplier.DisplayOrder = DisplayOrder++;

Input_Band3OffsetMultiplier.Name = "Band 3 Std. Deviation Multiplier/Fixed Offset";
Input_Band3OffsetMultiplier.SetFloat(1.5);
Input_Band3OffsetMultiplier.DisplayOrder = DisplayOrder++;

Input_Band4OffsetMultiplier.Name = "Band 4 Std. Deviation Multiplier/Fixed Offset";
Input_Band4OffsetMultiplier.SetFloat(2.0);
Input_Band4OffsetMultiplier.DisplayOrder = DisplayOrder++;

Input_Version.SetInt(2);

return;
}

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

if(Input_Version.GetInt() < 2)
{
    Input_Version.SetInt(2);
    Input_InputData.SetCustomInputIndex(0);
}

if (static_cast<int>(sc.VolumeAtPriceForBars->GetNumberOfBars()) < sc.ArraySize)
    return;

if (Input_NumberOfDaysToCalculate.GetInt() == 0)
    Input_NumberOfDaysToCalculate.SetInt(10);

// calculate the period start index
int PeriodType = Input_TimePeriodType.GetIndex();

```

```

if (PeriodType == 0 || PeriodType == 1)
{
    int DaysRequired = Input_TimePeriodLength.GetInt();

    if (Input_NumberOfDaysToCalculate.GetInt() < DaysRequired)
        Input_NumberOfDaysToCalculate.SetInt(DaysRequired);
}

int& LastIndexProcessed = sc.GetPersistentInt(1);
int& FirstIndexIncluded = sc.GetPersistentInt(2);
int& LastExcludedIndex = sc.GetPersistentInt(3);

if (sc.IsFullRecalculation)//reset
{
    LastIndexProcessed = -1;
    FirstIndexIncluded = 0;
    LastExcludedIndex = -1;

    if (sc.MaintainVolumeAtPriceData == 0)
    {
        sc.MaintainVolumeAtPriceData = 1;
        sc.FlagToReloadChartData = 1;
    }
}

for (int Index = sc.UpdateStartIndex; Index < sc.ArraySize; ++Index)
{
    bool NewBar = Index != LastIndexProcessed;

    LastIndexProcessed = Index;

    SCDateTime TradingDayDateForBar = sc.GetTradingDayDate( sc.BaseDateTimeIn[Index]);
    SCDateTime TradingDayDateForLastBar = sc.GetTradingDayDate( sc.BaseDateTimeIn[sc.ArraySize - 1]);

    if ((TradingDayDateForLastBar.GetDate() - TradingDayDateForBar.GetDate() + 1)
    > Input_NumberOfDaysToCalculate.GetInt())
    {
        LastExcludedIndex = Index;
        continue;
    }

    Array_PriceSum[Index] = 0;
    Array_PriceCount[Index] = 0;

    // calculate bar summation

    const s_VolumeAtPriceV2* p_VolumeAtPrice = NULL;

    int NumVAPElementsAtBarIndex = sc.VolumeAtPriceForBars->GetSizeAtBarIndex(Index);
    for (int VAPIndex = 0; VAPIndex < NumVAPElementsAtBarIndex; VAPIndex++)
    {
        sc.VolumeAtPriceForBars->GetVAPElementAtIndex(Index, VAPIndex, &p_VolumeAtPrice);

        int InputDataIndex = Input_InputData.GetIndex();
        if (InputDataIndex == 0)
        {
            float Price = p_VolumeAtPrice->PriceInTicks * sc.TickSize;
            float PriceTimeTrades = Price * p_VolumeAtPrice->NumberOfTrades;

            Array_PriceSum[Index] += PriceTimeTrades;
            Array_PriceCount[Index] += p_VolumeAtPrice->NumberOfTrades;
        }
    }
}

```

```

}
else if (InputDataIndex == 1)
{
    Array_PriceSum[Index] += p_VolumeAtPrice->Volume;
    Array_PriceCount[Index] += 1;
}
else if (InputDataIndex == 2)
{
    Array_PriceSum[Index] += p_VolumeAtPrice->NumberOfTrades;
    Array_PriceCount[Index] += 1;
}
else if (InputDataIndex == 3)
{
    Array_PriceSum[Index] += labs(p_VolumeAtPrice->AskVolume - p_VolumeAtPrice->BidVolume);
    Array_PriceCount[Index] += 1;
}
}

if (NewBar)
{
    if (PeriodType == 0) // Days - trading days
    {
        int TradeDate = sc.GetTradingDayDate(sc.BaseDateTimeIn[Index]);

        SCDateTime CurrentDayStartDateTime = sc.GetStartDateTimeForTradingDate(TradeDate);
        FirstIndexIncluded = sc.GetContainingIndexForSCDateTime(sc.ChartNumber,
CurrentDayStartDateTime.GetAsDouble());

        GetIndexForStartOfDayPeriodForDaysBack
        ( sc
        , CurrentDayStartDateTime
        , Input_TimePeriodLength.GetInt()
        , Input_ExcludeWeekendInDateCalc.GetYesNo() != 0
        , Input_MinimumRequiredTimePeriodAsPercentForSkipDays.GetInt()
        , FirstIndexIncluded
        );
    }
    else if (PeriodType == 1) // Days - 24 hour periods
    {
        SCDateTime CurrentPeriodStartDateTime = sc.BaseDateTimeIn[Index] - SCDateTime::DAYS(1) +
SCDateTime::SECONDS(1);
        FirstIndexIncluded = sc.GetContainingIndexForSCDateTime(sc.ChartNumber,
CurrentPeriodStartDateTime.GetAsDouble());

        GetIndexForStartOfDayPeriodForDaysBack
        ( sc
        , CurrentPeriodStartDateTime
        , Input_TimePeriodLength.GetInt()
        , Input_ExcludeWeekendInDateCalc.GetYesNo() != 0
        , Input_MinimumRequiredTimePeriodAsPercentForSkipDays.GetInt()
        , FirstIndexIncluded
        );
    }
    else if (PeriodType == 2) // Minutes
    {
        SCDateTime RollingStartDateTime = 0;

        if (BarPeriod.IntradayChartBarPeriodType == IBPT_DAYS_MINS_SECS &&
BarPeriod.IntradayChartBarPeriodParameter1 != 0)
        {
            RollingStartDateTime = sc.BaseDateTimeIn[Index] + SCDateTime::SECONDS(sc.SecondsPerBar - 1);
            RollingStartDateTime.SubtractMinutes(Input_TimePeriodLength.GetInt());
        }
    }
}

```

```

else
{
    if (Index == sc.ArraySize - 1)
        RollingStartDateTime = sc.LatestDateTimeForLastBar;
    else
        RollingStartDateTime = sc.BaseDateTimeIn[Index+1] - SCDateTime::SECONDS(1);

    RollingStartDateTime.SubtractMinutes(Input_TimePeriodLength.GetInt());
}

for (; FirstIndexIncluded <= Index; ++FirstIndexIncluded)
{
    if (sc.BaseDateTimeIn[FirstIndexIncluded] < RollingStartDateTime)
        continue;

    break;
}
}
else // Bars
{
    FirstIndexIncluded = max(0, Index - Input_TimePeriodLength.GetInt()+1);
}

}

if (FirstIndexIncluded <= LastExcludedIndex)
    FirstIndexIncluded = LastExcludedIndex + 1;

// calculate average
double CumulativePriceSum = 0;
double CumulativePriceCount = 0;

for (int AvgIndex=FirstIndexIncluded; AvgIndex <= Index; ++AvgIndex)
{
    CumulativePriceSum += Array_PriceSum[AvgIndex];
    CumulativePriceCount += Array_PriceCount[AvgIndex];
}

if (CumulativePriceCount != 0)
    Subgraph_MovingAverage[Index] = static_cast<float>(CumulativePriceSum / CumulativePriceCount);
else
    Subgraph_MovingAverage[Index] = Subgraph_MovingAverage[Index - 1];

float BandDistance = 1.0f;

//Standard deviation
if (Input_BandCalculationMethod.GetInt() == 0)
{
    const int StandardDeviationLength = Index - FirstIndexIncluded + 1;
    if (Index - StandardDeviationLength + 1 > LastExcludedIndex)
    {
        sc.StdDeviation(Subgraph_MovingAverage, Array_StandardDeviation, Index, StandardDeviationLength);
        BandDistance = Array_StandardDeviation[Index];
    }
}

Subgraph_SD1Top[Index] = Subgraph_MovingAverage[Index] + Input_Band1OffsetMultiplier.GetFloat() *
BandDistance;
Subgraph_SD1Bottom[Index] = Subgraph_MovingAverage[Index] - Input_Band1OffsetMultiplier.GetFloat() *
BandDistance;

```

```

        Subgraph_SD2Top[Index] = Subgraph_MovingAverage[Index] + Input_Band2OffsetMultiplier.GetFloat() *
BandDistance;
        Subgraph_SD2Bottom[Index] = Subgraph_MovingAverage[Index] - Input_Band2OffsetMultiplier.GetFloat() *
BandDistance;

        Subgraph_SD3Top[Index] = Subgraph_MovingAverage[Index] + Input_Band3OffsetMultiplier.GetFloat() *
BandDistance;
        Subgraph_SD3Bottom[Index] = Subgraph_MovingAverage[Index] - Input_Band3OffsetMultiplier.GetFloat() *
BandDistance;

        Subgraph_SD4Top[Index] = Subgraph_MovingAverage[Index] + Input_Band4OffsetMultiplier.GetFloat() *
BandDistance;
        Subgraph_SD4Bottom[Index] = Subgraph_MovingAverage[Index] - Input_Band4OffsetMultiplier.GetFloat() *
BandDistance;
    }
}

```

/*=====*/

```

SCSFExport scsf_TrueBarAverage(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Average = sc.Subgraph[0];

    if(sc.SetDefaults)
    {
        sc.ScaleRangeType = SCALE_SAMEASREGION;

        sc.GraphName = "True Bar Average";
        sc.GraphRegion = 0;
        sc.ValueFormat = VALUEFORMAT_INHERITED;
        sc.AutoLoop = 1;
        sc.MaintainVolumeAtPriceData = 1;

        Subgraph_Average.Name = "Average";
        Subgraph_Average.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Average.LineWidth = 2;
        Subgraph_Average.PrimaryColor = RGB(237, 13, 255);
        Subgraph_Average.DrawZeros = false;

        return;
    }

    int PriceCount = 0;
    float Sum = 0.0;

    const s_VolumeAtPriceV2 *p_VolumeAtPrice = NULL;

    int NumVAPElementsAtBarIndex = sc.VolumeAtPriceForBars->GetSizeAtBarIndex(sc.Index);
    for (int VAPIndex = 0; VAPIndex < NumVAPElementsAtBarIndex; VAPIndex++)
    {
        sc.VolumeAtPriceForBars->GetVAPElementAtIndex(sc.Index, VAPIndex, &p_VolumeAtPrice);

        Sum += p_VolumeAtPrice->PriceInTicks * sc.TickSize;
        ++PriceCount;
    }

    if (PriceCount != 0)
        Subgraph_Average[sc.Index] = Sum / PriceCount;
}

```

```

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

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_Length = sc.Input[2];

    if(sc.SetDefaults)
    {
        sc.GraphName = "Moving Linear Regression";

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

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

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

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

        return;
    }

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

    sc.LinearRegressionIndicator(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_MLR,
    Input_Length.GetInt());
}
/*=====*/

SCSFExport scsf_MACD(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_MACD = sc.Subgraph[0];
    SCSubgraphRef Subgraph_MovAvgOfMACD = sc.Subgraph[1];
    SCSubgraphRef Subgraph_MACDDiff = sc.Subgraph[2];
    SCSubgraphRef Subgraph_RefLine = sc.Subgraph[3];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_FastLength = sc.Input[3];
    SCInputRef Input_SlowLength = sc.Input[4];
    SCInputRef Input_MACDLength = sc.Input[5];
    SCInputRef Input_MovingAverageType = sc.Input[6];

    if(sc.SetDefaults)
    {
        sc.GraphName = "MACD";
        sc.AutoLoop = 1;

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

        Subgraph_MACD.Name = "MACD";

```

```

Subgraph_MACD.DrawStyle = DRAWSTYLE_LINE;
Subgraph_MACD.DrawZeros = true;
Subgraph_MACD.PrimaryColor = RGB(0,255,0);

Subgraph_MovAvgOfMACD.Name = "MA of MACD";
Subgraph_MovAvgOfMACD.DrawStyle = DRAWSTYLE_LINE;
Subgraph_MovAvgOfMACD.DrawZeros = true;
Subgraph_MovAvgOfMACD.PrimaryColor = RGB(255,0,255);

Subgraph_MACDDiff.Name = "MACD Diff";
Subgraph_MACDDiff.DrawStyle = DRAWSTYLE_BAR;
Subgraph_MACDDiff.DrawZeros = true;
Subgraph_MACDDiff.PrimaryColor = RGB(255,255,0);

Subgraph_RefLine.Name = "Line";
Subgraph_RefLine.DrawStyle = DRAWSTYLE_LINE;
Subgraph_RefLine.DrawZeros = true;
Subgraph_RefLine.PrimaryColor = RGB(255,127,0);

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

Input_FastLength.Name = "Fast Moving Average Length";
Input_FastLength.SetInt(12);
Input_FastLength.SetIntLimits(1,MAX_STUDY_LENGTH);

Input_SlowLength.Name = "Slow Moving Average Length";
Input_SlowLength.SetInt(26);
Input_SlowLength.SetIntLimits(1,MAX_STUDY_LENGTH);

Input_MACDLength.Name = "MACD Moving Average Length";
Input_MACDLength.SetInt(9);
Input_MACDLength.SetIntLimits(1,MAX_STUDY_LENGTH);

Input_MovingAverageType.Name = "Moving Average Type";
Input_MovingAverageType.SetMovAvgType(MOVAVGTYPE_EXPONENTIAL);

return;
}

if (Input_MovingAverageType.GetMovAvgType()==MOVAVGTYPE_EXPONENTIAL)
    sc.DataStartIndex = 2;
else
    sc.DataStartIndex = Input_MACDLength.GetInt() + max(Input_FastLength.GetInt(), Input_SlowLength.GetInt());

sc.MACD(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_MACD, sc.Index,
Input_FastLength.GetInt(), Input_SlowLength.GetInt(), Input_MACDLength.GetInt(), Input_MovingAverageType.GetInt());

Subgraph_MovAvgOfMACD[sc.Index] = Subgraph_MACD.Arrays[2][sc.Index];
Subgraph_MACDDiff[sc.Index] = Subgraph_MACD.Arrays[3][sc.Index];
Subgraph_RefLine[sc.Index] = 0;
}

/*=====*/

SCSFExport scsf_3_10_Oscillator(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_MACD = sc.Subgraph[0];
    SCSubgraphRef Subgraph_MovAvgOfMACD = sc.Subgraph[1];
    SCSubgraphRef Subgraph_MACDDiff = sc.Subgraph[2];
    SCSubgraphRef Subgraph_RefLine = sc.Subgraph[3];

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

```

```

SCInputRef Input_SlowLength = sc.Input[2];
SCInputRef Input_MACDLength = sc.Input[3];

if (sc.SetDefaults)
{
    sc.GraphName = "3/10 Oscillator";
    sc.AutoLoop = 1;

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

    Subgraph_MACD.Name = "3/10";
    Subgraph_MACD.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_MACD.DrawZeros = true;
    Subgraph_MACD.PrimaryColor = RGB(0, 255, 0);

    Subgraph_MovAvgOfMACD.Name = "Mov Avg 3/10";
    Subgraph_MovAvgOfMACD.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_MovAvgOfMACD.DrawZeros = true;
    Subgraph_MovAvgOfMACD.PrimaryColor = RGB(255, 0, 255);

    Subgraph_MACDDiff.Name = "3/10 Diff";
    Subgraph_MACDDiff.DrawStyle = DRAWSTYLE_BAR;
    Subgraph_MACDDiff.DrawZeros = true;
    Subgraph_MACDDiff.PrimaryColor = RGB(255, 255, 0);

    Subgraph_RefLine.Name = "Line";
    Subgraph_RefLine.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_RefLine.DrawZeros = true;
    Subgraph_RefLine.PrimaryColor = RGB(255, 127, 0);

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

    Input_FastLength.Name = "Fast Moving Average Length";
    Input_FastLength.SetInt(3);
    Input_FastLength.SetIntLimits(1, MAX_STUDY_LENGTH);

    Input_SlowLength.Name = "Slow Moving Average Length";
    Input_SlowLength.SetInt(10);
    Input_SlowLength.SetIntLimits(1, MAX_STUDY_LENGTH);

    Input_MACDLength.Name = "3/10 Moving Average Length";
    Input_MACDLength.SetInt(16);
    Input_MACDLength.SetIntLimits(1, MAX_STUDY_LENGTH);

    return;
}

sc.DataStartIndex = Input_MACDLength.GetInt() + max(Input_FastLength.GetInt(), Input_SlowLength.GetInt());

sc.MACD(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_MACD, sc.Index,
Input_FastLength.GetInt(), Input_SlowLength.GetInt(), Input_MACDLength.GetInt(), MOVAVGTYPE_SIMPLE);

Subgraph_MovAvgOfMACD[sc.Index] = Subgraph_MACD.Arrays[2][sc.Index];
Subgraph_MACDDiff[sc.Index] = Subgraph_MACD.Arrays[3][sc.Index];
Subgraph_RefLine[sc.Index] = 0;
}

/*=====*/

SCSFExport scsf_MACDLeader(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_MACD = sc.Subgraph[0];

```

```

SCFloatArrayRef Array_FastExponential = sc.Subgraph[0].Arrays[0];
SCFloatArrayRef Array_SlowExponential = sc.Subgraph[0].Arrays[1];
SCFloatArrayRef Array_InputDataMinusFastAverage = sc.Subgraph[0].Arrays[2];
SCFloatArrayRef Array_InputDataMinusSlowAverage = sc.Subgraph[0].Arrays[3];

SCFloatArrayRef Array_AvgInputDataMinusFastAverage = sc.Subgraph[0].Arrays[4];
SCFloatArrayRef Array_AvgInputDataMinusSlowAverage = sc.Subgraph[0].Arrays[5];

SCSubgraphRef Subgraph_MovAvgOfMACD = sc.Subgraph[1];
SCSubgraphRef Subgraph_MACDLeader = sc.Subgraph[2];
SCSubgraphRef Subgraph_RefLine = sc.Subgraph[3];

SCInputRef Input_InputData = sc.Input[0];
SCInputRef Input_FastLength = sc.Input[1];
SCInputRef Input_SlowLength = sc.Input[2];
SCInputRef Input_MACDMovingAverageLength = sc.Input[3];
SCInputRef Input_MovingAverageType = sc.Input[4];
SCInputRef Input_MACDMovingAverageType = sc.Input[5];

if (sc.SetDefaults)
{
    sc.GraphName = "MACD Leader";
    sc.AutoLoop = 1;

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

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

    Subgraph_MovAvgOfMACD.Name = "MA of MACD";
    Subgraph_MovAvgOfMACD.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_MovAvgOfMACD.DrawZeros = true;
    Subgraph_MovAvgOfMACD.PrimaryColor = RGB(255, 0, 255);

    Subgraph_MACDLeader.Name = "MACD Leader";
    Subgraph_MACDLeader.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_MACDLeader.LineWidth = 2;
    Subgraph_MACDLeader.DrawZeros = true;
    Subgraph_MACDLeader.PrimaryColor = RGB(255, 128, 0);

    Subgraph_RefLine.Name = "Line";
    Subgraph_RefLine.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_RefLine.DrawZeros = true;
    Subgraph_RefLine.PrimaryColor = RGB(255, 127, 0);

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

    Input_FastLength.Name = "Fast Moving Average Length";
    Input_FastLength.SetInt(12);
    Input_FastLength.SetIntLimits(1, MAX_STUDY_LENGTH);

    Input_SlowLength.Name = "Slow Moving Average Length";
    Input_SlowLength.SetInt(26);
    Input_SlowLength.SetIntLimits(1, MAX_STUDY_LENGTH);

    Input_MACDMovingAverageLength.Name = "MACD Moving Average Length";
    Input_MACDMovingAverageLength.SetInt(9);
    Input_MACDMovingAverageLength.SetIntLimits(1, MAX_STUDY_LENGTH);

    Input_MovingAverageType.Name = "Moving Average Type";

```

```

Input_MovingAverageType.SetMovAvgType(MOVAVGTYPE_EXPONENTIAL);

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

return;
}

sc.DataStartIndex = Input_MACDMovingAverageLength.GetInt() + max(Input_FastLength.GetInt(),
Input_SlowLength.GetInt());

int InputData = Input_InputData.GetInputDataIndex();
sc.MovingAverage(sc.BaseDataIn[InputData], Array_FastExponential, Input_MovingAverageType.GetInt(),
Input_FastLength.GetInt() );
sc.MovingAverage(sc.BaseDataIn[InputData], Array_SlowExponential, Input_MovingAverageType.GetInt(),
Input_SlowLength.GetInt());

Subgraph_MACD[sc.Index] = Array_FastExponential[sc.Index] - Array_SlowExponential[sc.Index];

sc.MovingAverage(Subgraph_MACD, Subgraph_MovAvgOfMACD, Input_MACDMovingAverageType.GetInt(),
Input_MACDMovingAverageLength.GetInt());

Array_InputDataMinusFastAverage[sc.Index] = sc.BaseDataIn[InputData][sc.Index] - Array_FastExponential[sc.Index];

Array_InputDataMinusSlowAverage[sc.Index] = sc.BaseDataIn[InputData][sc.Index] - Array_SlowExponential[sc.Index];

sc.MovingAverage(Array_InputDataMinusFastAverage, Array_AvgInputDataMinusFastAverage,
Input_MovingAverageType.GetInt(), Input_FastLength.GetInt());
sc.MovingAverage(Array_InputDataMinusSlowAverage, Array_AvgInputDataMinusSlowAverage,
Input_MovingAverageType.GetInt(), Input_SlowLength.GetInt());

Subgraph_MACDLeader[sc.Index] = Subgraph_MACD[sc.Index] + Array_AvgInputDataMinusFastAverage[sc.Index] -
Array_AvgInputDataMinusSlowAverage[sc.Index];

Subgraph_RefLine[sc.Index] = 0;
}

/*=====*/

SCSFExport scsf_StandardErrorBands(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_UpperBand = sc.Subgraph[0];
    SCSubgraphRef Subgraph_MidBand = sc.Subgraph[1];
    SCSubgraphRef Subgraph_LowerBand = sc.Subgraph[2];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_Length = sc.Input[2];
    SCInputRef Input_StandardDeviations = sc.Input[3];

    if(sc.SetDefaults)
    {
        sc.GraphName = "Standard Error Bands";

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

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

        Subgraph_LowerBand.Name = "Bottom Band";
        Subgraph_LowerBand.DrawStyle = DRAWSTYLE_LINE;

```

```

Subgraph_LowerBand.DrawZeros = false;
Subgraph_LowerBand.PrimaryColor = RGB(255,255,0);

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

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

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

Input_StandardDeviations.Name = "Standard Deviations";
Input_StandardDeviations.SetFloat(2.0f);
Input_StandardDeviations.SetFloatLimits(.001f, static_cast<float>(MAX_STUDY_LENGTH));

return;
}

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

sc.LinearRegressionIndicator(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_MidBand,
Input_Length.GetInt());

double StandardErrorValue = sc.GetStandardError(sc.BaseDataIn[Input_InputData.GetInputDataIndex()],
Input_Length.GetInt());

Subgraph_UpperBand[sc.Index] = static_cast<float>(Subgraph_MidBand[sc.Index] + StandardErrorValue *
Input_StandardDeviations.GetFloat());

Subgraph_LowerBand[sc.Index] = static_cast<float>(Subgraph_MidBand[sc.Index] - StandardErrorValue *
Input_StandardDeviations.GetFloat());
}

/*=====*/
SCSFExport scsf_BollingerBands(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_UpperBand = sc.Subgraph[0];
    SCSubgraphRef Subgraph_MidBand = sc.Subgraph[1];
    SCSubgraphRef Subgraph_LowerBand = sc.Subgraph[2];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_Length = sc.Input[3];
    SCInputRef Input_StandardDeviations = sc.Input[4];
    SCInputRef Input_Version = sc.Input[5];
    SCInputRef Input_MAType = sc.Input[6];

    if(sc.SetDefaults)
    {
        sc.GraphName = "Bollinger Bands";
        sc.GraphRegion = 0;
        sc.ValueFormat = 3;
        sc.AutoLoop = true;

        Subgraph_MidBand.Name = "Middle Band";
        Subgraph_MidBand.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_MidBand.PrimaryColor = RGB(255,0,255);

        Subgraph_UpperBand.Name = "Top Band";
        Subgraph_UpperBand.DrawStyle = DRAWSTYLE_LINE;

```

```

Subgraph_UpperBand.PrimaryColor = RGB(0,255,0);

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

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

Input_Length.Name = "Length";
Input_Length.SetInt(10);
Input_Length.SetIntLimits(1,MAX_STUDY_LENGTH);

Input_StandardDeviations.Name = "Standard Deviations";
Input_StandardDeviations.SetFloat(2.0f);
Input_StandardDeviations.SetFloatLimits(.0001f, static_cast<float>(MAX_STUDY_LENGTH));

Input_Version.SetInt(2);

Input_MAType.Name = "Moving Average Type";
Input_MAType.SetMovAvgType(MOAVGTYPE_SIMPLE);

return;
}

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

sc.BollingerBands(sc.BaseDataIn[Input_InputData.GetInputDataIndex()],Subgraph_MidBand,Input_Length.GetInt(),Input_Si

Subgraph_UpperBand[sc.Index] = Subgraph_MidBand.Arrays[0][sc.Index];
Subgraph_LowerBand[sc.Index] = Subgraph_MidBand.Arrays[1][sc.Index];
}
/*=====*/

SCSFExport scsf_BollingerBands_StandardDeviationOfAverage(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_UpperBand = sc.Subgraph[0];
    SCSubgraphRef Subgraph_MidBand = sc.Subgraph[1];
    SCSubgraphRef Subgraph_LowerBand = sc.Subgraph[2];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_Length = sc.Input[1];
    SCInputRef Input_StandardDeviations = sc.Input[2];
    SCInputRef Input_MAType = sc.Input[3];

    if(sc.SetDefaults)
    {
        sc.GraphName = "Bollinger Bands - StandardDeviationOfAverage";
        sc.GraphRegion = 0;
        sc.ValueFormat = VALUEFORMAT_INHERITED;
        sc.AutoLoop = true;

        Subgraph_MidBand.Name = "Middle Band";
        Subgraph_MidBand.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_MidBand.PrimaryColor = RGB(255,0,255);

        Subgraph_UpperBand.Name = "Top Band";
        Subgraph_UpperBand.DrawStyle = DRAWSTYLE_LINE;

```

```

Subgraph_UpperBand.PrimaryColor = RGB(0,255,0);

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

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

Input_Length.Name = "Length";
Input_Length.SetInt(10);
Input_Length.SetIntLimits(1,MAX_STUDY_LENGTH);

Input_StandardDeviations.Name = "Standard Deviations";
Input_StandardDeviations.SetFloat(2.0f);
Input_StandardDeviations.SetFloatLimits(.0001f, static_cast<float>(MAX_STUDY_LENGTH));

Input_MAType.Name = "Moving Average Type";
Input_MAType.SetMovAvgType(MOAVGTYPE_SIMPLE);

return;
}

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

BollingerBands_StandardDeviationOfAverage_S(sc.BaseDataIn[Input_InputData.GetInputDataIndex()],
Subgraph_MidBand,
Subgraph_UpperBand,
Subgraph_LowerBand,
Subgraph_MidBand.Arrays[0],
sc.Index,
Input_Length.GetInt(),
Input_StandardDeviations.GetFloat(),
Input_MAType.GetMovAvgType());
}

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

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_InitialValue = sc.Input[2];

    if(sc.SetDefaults)
    {
        sc.GraphName = "Positive Volume Index";
        sc.GraphRegion = 1;
        sc.ValueFormat = 3;
        sc.AutoLoop = 1; // true

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

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

```

```

Input_InitialValue.Name = "Initial Value";
Input_InitialValue.SetFloat(2.5f);
Input_InitialValue.SetFloatLimits(.01f, static_cast<float>(MAX_STUDY_LENGTH));

return;
}

float prev_PVI;
float prev_data;
float data;
float PVIval;
float volume;
float prev_volume;

sc.DataStartIndex = 0;

int pos = sc.Index;
if(pos == 0) // We are starting a new PVI study
{
    Subgraph_PVI[0] = Input_InitialValue.GetFloat();
    pos++;
}

volume = sc.Volume[pos];
prev_volume = sc.Volume[pos-1];
prev_PVI = Subgraph_PVI[pos-1];
if(volume > prev_volume) // current volume less than equal to previous volume.
{
    data = sc.BaseDataIn[Input_InputData.GetInputDataIndex()][pos];
    prev_data = sc.BaseDataIn[Input_InputData.GetInputDataIndex()][pos-1];
    PVIval = prev_PVI + ((data-prev_data) / prev_data * prev_PVI );
}
else
{
    PVIval = prev_PVI;
}

Subgraph_PVI[pos] = PVIval;
}
/*****
SCSFExport scsf_NegativeVolumeIndex(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_NVI = sc.Subgraph[0];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_InitialValue = sc.Input[2];

    if(sc.SetDefaults)
    {
        sc.GraphName = "Negative Volume Index";

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

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

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

```

```

    Input_InitialValue.Name = "Initial Value";
    Input_InitialValue.SetFloat(2.5f);
    Input_InitialValue.SetFloatLimits(0.01f, static_cast<float>(MAX_STUDY_LENGTH));

    return;
}

sc.DataStartIndex = 0;

float prev_NVI;
float prev_data;
float data;
float NVIval;
float volume;
float prev_volume;

int pos = sc.Index;
if(pos == 0) // We are starting a new NVI study
{
    Subgraph_NVI[0] = Input_InitialValue.GetFloat();
    pos++;
}

volume = sc.Volume[pos];
prev_volume = sc.Volume[pos-1];
prev_NVI = Subgraph_NVI[pos-1];
if(volume < prev_volume) // current volume greater then equal to previous volume.
{
    data = sc.BaseDataIn[Input_InputData.GetInputDataIndex()][pos];
    prev_data = sc.BaseDataIn[Input_InputData.GetInputDataIndex()][pos-1];
    NVIval = prev_NVI + ((data-prev_data) / prev_data * prev_NVI);
}
else
{
    NVIval = prev_NVI;
}

Subgraph_NVI[pos] = NVIval;
}
/*****
SCSFExport scsf_DoubleStochasticNew(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_DS = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Line1 = sc.Subgraph[5];
    SCSubgraphRef Subgraph_Line2 = sc.Subgraph[6];

    SCInputRef Input_Length = sc.Input[0];
    SCInputRef Input_EMALength = sc.Input[2];
    SCInputRef Input_MovAvgType = sc.Input[3];
    SCInputRef Input_Line1Value = sc.Input[4];
    SCInputRef Input_Line2Value = sc.Input[5];

    if(sc.SetDefaults)
    {
        sc.GraphName = "Double Stochastic";
        sc.GraphRegion = 1;
        sc.ValueFormat = 3;
        sc.AutoLoop = true;

        Subgraph_DS.Name = "DS";
        Subgraph_DS.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_DS.PrimaryColor = RGB(0,255,0);
    }
}
*****/

```

```

Subgraph_DS.DrawZeros = false;

Subgraph_Line1.Name = "Line 1";
Subgraph_Line1.DrawStyle = DRAWSTYLE_LINE;
Subgraph_Line1.PrimaryColor = RGB(0,255,0);
Subgraph_Line1.DrawZeros = false;

Subgraph_Line2.Name = "Line 2";
Subgraph_Line2.DrawStyle = DRAWSTYLE_LINE;
Subgraph_Line2.PrimaryColor = RGB(0,255,0);
Subgraph_Line2.DrawZeros = false;

Input_Length.Name = "Length";
Input_Length.SetInt(10);
Input_Length.SetIntLimits(1,MAX_STUDY_LENGTH);

Input_EMALength.Name = "Moving Average Length";
Input_EMALength.SetInt(3);
Input_EMALength.SetIntLimits(1,MAX_STUDY_LENGTH);

Input_MovAvgType.Name = "Moving Average Type";
Input_MovAvgType.SetMovAvgType(MOVAVGTYPE_EXPONENTIAL);

Input_Line1Value.Name = "Line 1";
Input_Line1Value.SetInt(10);

Input_Line2Value.Name = "Line 2";
Input_Line2Value.SetInt(85);

return;
}

sc.DoubleStochastic(sc.BaseDataIn, Subgraph_DS, Input_Length.GetInt(), Input_EMALength.GetInt(),
Input_MovAvgType.GetMovAvgType());

Subgraph_Line1[sc.Index] = static_cast<float>(Input_Line1Value.GetInt());
Subgraph_Line2[sc.Index] = static_cast<float>(Input_Line2Value.GetInt());
}

/*=====*/
SCSFExport scsf_CandlesNoTails(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Open = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Last = sc.Subgraph[3];
    SCSubgraphRef Subgraph_Volume = sc.Subgraph[4];
    SCSubgraphRef Subgraph_NumTrades = sc.Subgraph[5];
    SCSubgraphRef Subgraph_OHLCAvg = sc.Subgraph[6];
    SCSubgraphRef Subgraph_HLCAvg = sc.Subgraph[7];
    SCSubgraphRef Subgraph_HLCAvg = sc.Subgraph[7];
    SCSubgraphRef Subgraph_HLCAvg = sc.Subgraph[8];
    SCSubgraphRef Subgraph_HLCAvg = sc.Subgraph[8];
    SCSubgraphRef Subgraph_BidVol = sc.Subgraph[9];
    SCSubgraphRef Subgraph_AskVol = sc.Subgraph[10];

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

        sc.GraphName = "Candles without Tails";

        sc.GraphDrawType = GDT_CANDLESTICK;
        sc.StandardChartHeader = 1;
        sc.DisplayAsMainPriceGraph = 1;
        sc.GraphRegion = 0;
    }
}

```

```

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

Subgraph_Last.Name = "Last";
Subgraph_Last.DrawStyle = DRAWSTYLE_LINE;
Subgraph_Last.PrimaryColor = RGB(255,127,0);
Subgraph_Last.DrawZeros = false;

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

Subgraph_NumTrades.Name = "# of Trades / OI";
Subgraph_NumTrades.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_NumTrades.PrimaryColor = RGB(0,0,255);
Subgraph_NumTrades.DrawZeros = false;

Subgraph_OHLCAvg.Name = "OHLC Avg";
Subgraph_OHLCAvg.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_OHLCAvg.PrimaryColor = RGB(127,0,255);
Subgraph_OHLCAvg.DrawZeros = false;

Subgraph_HLCAvg.Name = "HLC Avg";
Subgraph_HLCAvg.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_HLCAvg.PrimaryColor = RGB(0,255,255);
Subgraph_HLCAvg.DrawZeros = false;

Subgraph_HLAvg.Name = "HL Avg";
Subgraph_HLAvg.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_HLAvg.PrimaryColor = RGB(0,127,255);
Subgraph_HLAvg.DrawZeros = false;

Subgraph_BidVol.Name = "Bid Vol";
Subgraph_BidVol.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_BidVol.PrimaryColor = RGB(0,255,0);
Subgraph_BidVol.DrawZeros = false;

Subgraph_AskVol.Name = "Ask Vol";
Subgraph_AskVol.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_AskVol.PrimaryColor = RGB(0,255,0);
Subgraph_AskVol.DrawZeros = false;

sc.AutoLoop = 1;

return;
}
//create a new name that includes the base graph name.
sc.GraphName.Format("%s No Tails", sc.GetStudyName(0).GetChars());

// Do data processing

Subgraph_Open[sc.Index] = sc.Open[sc.Index];
Subgraph_Last[sc.Index] = sc.Close[sc.Index];
Subgraph_Volume[sc.Index] = sc.Volume[sc.Index];
Subgraph_NumTrades[sc.Index] = sc.NumberOfTrades[sc.Index];
}

```

```

/*****

```

```

SCSFExport scsf_KiwisTrailingStop(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_TSUp = sc.Subgraph[0];
    SCSubgraphRef Subgraph_TSDown = sc.Subgraph[1];

    SCSubgraphRef Subgraph_direction = sc.Subgraph[3];
    SCSubgraphRef Subgraph_hiclose = sc.Subgraph[4];
    SCSubgraphRef Subgraph_loclose = sc.Subgraph[5];

    SCFloatArrayRef Array_e1 = sc.Subgraph[6];
    SCFloatArrayRef Array_e2 = sc.Subgraph[7];
    SCFloatArrayRef Array_e3 = sc.Subgraph[8];
    SCFloatArrayRef Array_e4 = sc.Subgraph[9];
    SCFloatArrayRef Array_e5 = sc.Subgraph[10];
    SCFloatArrayRef Array_e6 = sc.Subgraph[11];

    SCSubgraphRef Subgraph_TempRange = sc.Subgraph[6];
    SCSubgraphRef Subgraph_TempAvgRange = sc.Subgraph[7];
    SCSubgraphRef Subgraph_diff2 = sc.Subgraph[8];
    SCSubgraphRef Subgraph_atrmod = sc.Subgraph[9];

    SCInputRef Input_Use2Subgraphs = sc.Input[0];
    SCInputRef Input_ATRLength = sc.Input[2];
    SCInputRef Input_ATRFactor = sc.Input[3];
    SCInputRef Input_Vervoort = sc.Input[4];
    SCInputRef Input_ATRMax = sc.Input[5];

    if(sc.SetDefaults)
    {
        sc.GraphName="Kiwi's Trailing Stop";
        sc.StudyDescription = "Kiwi's Trailing Stop (Chandelier) plus Sylvain Vervoort variation";

        Subgraph_TSUp.Name="TS";
        Subgraph_TSUp.DrawStyle = DRAWSTYLE_LINE_SKIP_ZEROS;
        Subgraph_TSUp.SecondaryColorUsed = 1;
        Subgraph_TSUp.PrimaryColor = RGB(0,255,0);
        Subgraph_TSUp.SecondaryColor = RGB(255,0,0);
        Subgraph_TSUp.DrawZeros = false;

        Subgraph_TSDown.Name = "TS";
        Subgraph_TSDown.DrawStyle = DRAWSTYLE_LINE_SKIP_ZEROS;
        Subgraph_TSDown.PrimaryColor = RGB(255,0,0);
        Subgraph_TSDown.DrawZeros = false;

        Subgraph_TempRange.Name = "TR";
        Subgraph_TempRange.DrawStyle = DRAWSTYLE_IGNORE;
        Subgraph_TempRange.PrimaryColor = RGB(127,0,255);
        Subgraph_TempRange.DrawZeros = false;

        Subgraph_TempAvgRange.Name = "TAR";
        Subgraph_TempAvgRange.DrawStyle = DRAWSTYLE_IGNORE;
        Subgraph_TempAvgRange.PrimaryColor = RGB(0,255,255);
        Subgraph_TempAvgRange.DrawZeros = false;

        Subgraph_diff2.Name = "diff2";
        Subgraph_diff2.DrawStyle = DRAWSTYLE_IGNORE;
        Subgraph_diff2.PrimaryColor = RGB(0,127,255);
        Subgraph_diff2.DrawZeros = false;

        Subgraph_atrmod.Name = "atrmod";
        Subgraph_atrmod.DrawStyle = DRAWSTYLE_IGNORE;
        Subgraph_atrmod.PrimaryColor = RGB(0,255,0);
        Subgraph_atrmod.DrawZeros = false;
    }
}

```

```

sc.AutoLoop = 0;

Input_ATRLength.Name="ATR Length";
Input_ATRLength.SetFloat(9.0);

Input_ATRFactor.Name="ATR Factor";
Input_ATRFactor.SetFloat(2.5);

Input_Vervoort.Name="Vervoort Variation?";
Input_Vervoort.SetYesNo(false);

Input_ATRMax.Name="Vervoort Maximum Range";
Input_ATRMax.SetFloat(1.5);

Input_Use2Subgraphs.Name="Use 2 subgraphs";
Input_Use2Subgraphs.SetYesNo(true);

sc.GraphRegion = 0;

return;
}

int pos;
float Ave;
float b,b2,b3,c1,c2,c3,c4,w1=0,w2=0;

sc.DataStartIndex=10;

if(Input_Use2Subgraphs.GetYesNo())
{
    Subgraph_TSUp.Name = "TS Up";
    Subgraph_TSDown.Name = "TS Down";

    Subgraph_TSUp.SecondaryColorUsed = false;
}
else
{
    Subgraph_TSUp.Name="TS";

    Subgraph_TSUp.SecondaryColorUsed = 1;
}

if (sc.UpdateStartIndex == 0)
{
    Subgraph_direction[sc.UpdateStartIndex] = 1;
    Subgraph_loclose[sc.UpdateStartIndex] = sc.Close[sc.UpdateStartIndex];
    Subgraph_hiclose[sc.UpdateStartIndex] = Subgraph_loclose[sc.UpdateStartIndex];

    Array_e1[sc.UpdateStartIndex]=sc.High[sc.UpdateStartIndex]-sc.Low[sc.UpdateStartIndex];
    Array_e2[sc.UpdateStartIndex]=Array_e1[sc.UpdateStartIndex];
    Array_e3[sc.UpdateStartIndex]=Array_e1[sc.UpdateStartIndex];
    Array_e4[sc.UpdateStartIndex]=Array_e1[sc.UpdateStartIndex];
    Array_e5[sc.UpdateStartIndex]=Array_e1[sc.UpdateStartIndex];
    Array_e6[sc.UpdateStartIndex]=Array_e1[sc.UpdateStartIndex];
}

if(!Input_Vervoort.GetYesNo())
{
    b=0.5;           // 0.5
    b2=(b*b);       // 0.25
    b3=(b2*b);      // 0.125

    c1=-b3;         // - 0.125

```

```

c2=(3*(b2+b3)); // 0.45
c3=(-3*(2*b2+b+b3));
c4=(1+3*b+b3+3*b2);
w1 = 2/(Input_ATRLength.GetFloat()+1);
w2 = 1-w1;

for (pos=max(sc.UpdateStartIndex, 1); pos < sc.ArraySize; pos++)
{
    float temp = max(sc.High[pos]-sc.Low[pos], sc.High[pos]-sc.Close[pos-1]);
    float P = max(temp,sc.Close[pos-1]-sc.Low[pos]);

    Array_e1[pos] = w1*P + w2*Array_e1[pos - 1];
    Array_e2[pos] = w1*Array_e1[pos] + w2*Array_e2[pos - 1];
    Array_e3[pos] = w1*Array_e2[pos] + w2*Array_e3[pos - 1];
    Array_e4[pos] = w1*Array_e3[pos] + w2*Array_e4[pos - 1];
    Array_e5[pos] = w1*Array_e4[pos] + w2*Array_e5[pos - 1];
    Array_e6[pos] = w1*Array_e5[pos] + w2*Array_e6[pos - 1];

    Ave = c1*Array_e6[pos] + c2*Array_e5[pos] + c3*Array_e4[pos] + c4*Array_e3[pos];

    if((Subgraph_direction[pos-1]==1 && sc.Close[pos]<(Subgraph_loclose[pos-1]))
        ||
        (Subgraph_direction[pos-1]==-1 && sc.Close[pos]>(Subgraph_hiclose[pos-1])))
    {
        if(Subgraph_direction[pos-1]==1)
        {
            Subgraph_direction[pos] = -1; //reverse short
            Subgraph_hiclose[pos] = sc.Close[pos]+(Ave*Input_ATRFactor.GetFloat());
            Subgraph_loclose[pos] = 0;
        }
        else
        {
            Subgraph_direction[pos] = 1; //reverse long
            Subgraph_loclose[pos] = sc.Close[pos]-(Ave*Input_ATRFactor.GetFloat());
            Subgraph_hiclose[pos] = 0;
        }
    }
    else
    {
        if(Subgraph_direction[pos-1]==1)
        {
            if(sc.Close[pos]-(Ave*Input_ATRFactor.GetFloat())>Subgraph_loclose[pos-1])
            {
                Subgraph_loclose[pos] = sc.Close[pos]-(Ave*Input_ATRFactor.GetFloat());
                Subgraph_hiclose[pos] = Subgraph_hiclose[pos-1];
            }
            else
            {
                Subgraph_loclose[pos] = Subgraph_loclose[pos-1];
                Subgraph_hiclose[pos] = Subgraph_hiclose[pos-1];
            }
        }
        else
        {
            if(sc.Close[pos]+(Ave*Input_ATRFactor.GetFloat())<Subgraph_hiclose[pos-1])
            {
                Subgraph_loclose[pos] = Subgraph_loclose[pos-1];
                Subgraph_hiclose[pos] = sc.Close[pos]+(Ave*Input_ATRFactor.GetFloat());
            }
            else
            {
                Subgraph_loclose[pos] = Subgraph_loclose[pos-1];
                Subgraph_hiclose[pos] = Subgraph_hiclose[pos-1];
            }
        }
    }
};

```

```

    Subgraph_direction[pos] = Subgraph_direction[pos-1]; // no change
};

if(Subgraph_direction[pos]==1)
{
    if(Input_Use2Subgraphs.GetYesNo() == false)
    {
        Subgraph_TSUp[pos] = (Subgraph_loclose[pos]);
        Subgraph_TSUp.DataColor[pos] = Subgraph_TSUp.PrimaryColor;
    }
    else
    {
        Subgraph_TSUp[pos] = (Subgraph_loclose[pos]);
        Subgraph_TSDown[pos] = 0;
    }
}
else
{
    if(Input_Use2Subgraphs.GetYesNo() == false)
    {
        Subgraph_TSUp[pos] = (Subgraph_hiclose[pos]);
        Subgraph_TSUp.DataColor[pos] = Subgraph_TSUp.SecondaryColor;
    }
    else
    {
        Subgraph_TSDown[pos] = (Subgraph_hiclose[pos]);
        Subgraph_TSUp[pos] = 0;
    }
}
}
} else
{

/*
HiLo:=If(H-L<1.5*Mov(H-L,period,S),H-L, 1.5*Mov
(H-L,period,S));
Href:=If(L<=Ref(H,-1),H-Ref(C,-1),(H-Ref(C,-1))-(L-Ref(H,-1))/2);
Lref:=If(H>=Ref(L,-1),Ref(C,-1)-L,(Ref(C,-1)-L)-(Ref(L,-1)-H)/2);
diff1:=Max(HiLo,Href);
diff2:=Max(diff1,Lref);
atrmod:=Wilders(diff2,period);
loss:=atrmod*atrmod;
trail:=
If(C>PREV AND Ref(C,-1)>PREV,
Max(PREV,C-loss),
If(C<PREV AND Ref(C,-1)<PREV,
Min(PREV,C+loss),
If(C>PREV,C-loss,C+loss)));
*/

for (pos=max(sc.UpdateStartIndex, 1); pos < sc.ArraySize; pos++)
{
    float HiLo,Href,Lref;
    int period = static_cast<int>(Input_ATRLength.GetFloat());
    Subgraph_TempRange[pos] = sc.High[pos]-sc.Low[pos];
    sc.MovingAverage(Subgraph_TempRange, Subgraph_TempAvgRange, MOVAVGTYPE_SIMPLE, pos, period);

    HiLo=min(Subgraph_TempRange[pos],Input_ATRMax.GetFloat()*Subgraph_TempAvgRange[pos]);
    if(sc.Low[pos]<=sc.High[pos-1]) Href=sc.High[pos]-sc.Close[pos-1]; else Href=(sc.High[pos]-sc.Close[pos-1])-
(sc.Low[pos]-sc.High[pos-1])/2;
    if(sc.High[pos]>=sc.Low[pos-1]) Lref=sc.Close[pos-1]-sc.Low[pos]; else Lref=(sc.Close[pos-1]-sc.Low[pos])-
(sc.Low[pos-1]-sc.High[pos])/2;

```

```

Subgraph_diff2[pos] = max(Lref,max(HiLo,Href));
sc.MovingAverage(Subgraph_diff2, Subgraph_atrmod, MOVAVGTYPE_WILDERS, pos, period);

Ave = Subgraph_atrmod[pos];

if((Subgraph_direction[pos-1]==1 && sc.Close[pos]<(Subgraph_loclose[pos-1]))
||
(Subgraph_direction[pos-1]==-1 && sc.Close[pos]>(Subgraph_hiclose[pos-1])))
{
    if(Subgraph_direction[pos-1]==1)
    {
        Subgraph_direction[pos] = -1; //reverse short
        Subgraph_hiclose[pos] = sc.Close[pos]+(Ave*Input_ATRFactor.GetFloat());
        Subgraph_loclose[pos] = 0;
    }
    else
    {
        Subgraph_direction[pos] = 1; //reverse long
        Subgraph_loclose[pos] = sc.Close[pos]-(Ave*Input_ATRFactor.GetFloat());
        Subgraph_hiclose[pos] = 0;
    }
}
else
{
    if(Subgraph_direction[pos-1]==1)
    {
        if(sc.Close[pos]-(Ave*Input_ATRFactor.GetFloat())>Subgraph_loclose[pos-1])
        {
            Subgraph_loclose[pos] = sc.Close[pos]-(Ave*Input_ATRFactor.GetFloat());
            Subgraph_hiclose[pos] = Subgraph_hiclose[pos-1];
        }
        else
        {
            Subgraph_loclose[pos] = Subgraph_loclose[pos-1];
            Subgraph_hiclose[pos] = Subgraph_hiclose[pos-1];
        }
    }
    else
    {
        if(sc.Close[pos]+(Ave*Input_ATRFactor.GetFloat())<Subgraph_hiclose[pos-1])
        {
            Subgraph_loclose[pos] = Subgraph_loclose[pos-1];
            Subgraph_hiclose[pos] = sc.Close[pos]+(Ave*Input_ATRFactor.GetFloat());
        }
        else
        {
            Subgraph_loclose[pos] = Subgraph_loclose[pos-1];
            Subgraph_hiclose[pos] = Subgraph_hiclose[pos-1];
        }
    }
};

Subgraph_direction[pos] = Subgraph_direction[pos-1]; // no change
};

if(Subgraph_direction[pos]==1)
{
    if(Input_Use2Subgraphs.GetYesNo() == false)
    {
        Subgraph_TSUp[pos] = (Subgraph_loclose[pos]);
        Subgraph_TSUp.DataColor[pos] = Subgraph_TSUp.PrimaryColor;
    }
    else
    {
        Subgraph_TSUp[pos] = (Subgraph_loclose[pos]);
        Subgraph_TSDown[pos] = 0;
    }
}

```

```

    }
}
else
{
    if(Input_Use2Subgraphs.GetYesNo() == false)
    {
        Subgraph_TSUp[pos] = (Subgraph_hiclose[pos]);
        Subgraph_TSUp.DataColor[pos] = Subgraph_TSUp.SecondaryColor;
    }
    else
    {
        Subgraph_TSDown[pos] = (Subgraph_hiclose[pos]);
        Subgraph_TSUp[pos] = 0;
    }
}
}
}
}

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

    SCInputRef Input_HorizontalPosition = sc.Input[0];
    SCInputRef Input_VerticalPosition = sc.Input[1];
    SCInputRef Input_StudySubgraphReference = sc.Input[3];
    SCInputRef Input_DisplaySubgraphName = sc.Input[4];
    SCInputRef Input_SubgraphOffset = sc.Input[5];
    SCInputRef Input_UseTextForNonZeroValue = sc.Input[6];
    SCInputRef Input_NonZeroValueText = sc.Input[7];

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

        sc.GraphName = "Text Display For Study from Chart";

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

        Subgraph_TextDisplay.Name = "Text Display";
        Subgraph_TextDisplay.LineWidth = 20;
        Subgraph_TextDisplay.DrawStyle = DRAWSTYLE_CUSTOM_TEXT;
        Subgraph_TextDisplay.PrimaryColor = RGB(0, 0, 0); //black
        Subgraph_TextDisplay.SecondaryColor = RGB(128, 255, 255);
        Subgraph_TextDisplay.SecondaryColorUsed = true;
        Subgraph_TextDisplay.DisplayNameValueInWindowsFlags = 0;

        Input_HorizontalPosition.Name.Format("Horizontal Position From Left (1-%d)", static_cast<int>
(CHART_DRAWING_MAX_HORIZONTAL_AXIS_RELATIVE_POSITION));
        Input_HorizontalPosition.SetInt(20);
        Input_HorizontalPosition.SetIntLimits(1, static_cast<int>
(CHART_DRAWING_MAX_HORIZONTAL_AXIS_RELATIVE_POSITION));

        Input_VerticalPosition.Name.Format("Vertical Position From Bottom (1-%d)", static_cast<int>
(CHART_DRAWING_MAX_VERTICAL_AXIS_RELATIVE_POSITION));
        Input_VerticalPosition.SetInt(90);
        Input_VerticalPosition.SetIntLimits(1, static_cast<int>
(CHART_DRAWING_MAX_VERTICAL_AXIS_RELATIVE_POSITION));

        Input_StudySubgraphReference.Name = "Study And Subgraph To Display";
        Input_StudySubgraphReference.SetChartStudySubgraphValues(1, 1, 0);

        Input_DisplaySubgraphName.Name = "Display Subgraph Name";

```

```

Input_DisplaySubgraphName.SetYesNo(false);

Input_SubgraphOffset.Name = "Subgraph Columns Back";
Input_SubgraphOffset.SetInt(0);

Input_UseTextForNonZeroValue.Name = "Use Text for Non-Zero Value";
Input_UseTextForNonZeroValue.SetYesNo(false);

Input_NonZeroValueText.Name = "Non-Zero Value Text";
Input_NonZeroValueText.SetString("");

sc.TextInputName = "Prefix Text for Display Value";

return;
}

// Do data processing

SCFloatArray StudyReference;

sc.GetStudyArrayFromChartUsingID(Input_StudySubgraphReference.GetChartNumber(),
Input_StudySubgraphReference.GetStudyID(), Input_StudySubgraphReference.GetSubgraphIndex(), StudyReference);

SCString ValueText;
if (!sc.TextInput.IsEmpty())
{
    ValueText += sc.TextInput;
    ValueText += " ";
}

if (Input_DisplaySubgraphName.GetYesNo())
{
    SCString SubgraphNameFromChartStudy;
    sc.GetStudySubgraphNameFromChart(Input_StudySubgraphReference.GetChartNumber(),
Input_StudySubgraphReference.GetStudyID(), Input_StudySubgraphReference.GetSubgraphIndex(),
SubgraphNameFromChartStudy);
    ValueText += SubgraphNameFromChartStudy;
    ValueText += " ";
}

int ColumnsBack = Input_SubgraphOffset.GetInt();
float StudyValue = StudyReference[StudyReference.GetArraySize() - 1 - ColumnsBack];

//Patch for when referencing data from the Numbers Bars Calculated Values study. This value could be used.
if (StudyValue == -FLT_MAX)
    StudyValue = 0;

if (Input_UseTextForNonZeroValue.GetYesNo())
{
    if (StudyValue != 0)
        ValueText += Input_NonZeroValueText.GetString();
    else
        ValueText += " ";
}
else
    ValueText += sc.FormatGraphValue(StudyValue, sc.GetValueFormat());

sc.AddAndManageSingleTextDrawingForStudy
(sc
, false
, Input_HorizontalPosition.GetInt()
, Input_VerticalPosition.GetInt()

```

```

    , Subgraph_TextDisplay
    , false
    , ValueText
    , true
);
}

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

    SCInputRef Input_HorizontalPosition = sc.Input[0];
    SCInputRef Input_VerticalPosition = sc.Input[1];
    SCInputRef Input_StudySubgraphReference = sc.Input[3];
    SCInputRef Input_DisplaySubgraphName = sc.Input[4];
    SCInputRef Input_SubgraphOffset = sc.Input[5];
    SCInputRef Input_PrefixTextForDisplayValue = sc.Input[6];
    SCInputRef Input_SuffixTextForDisplayValue = sc.Input[7];
    SCInputRef Input_UseTextForNonZeroValue = sc.Input[8];
    SCInputRef Input_NonZeroValueText = sc.Input[9];

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

        sc.GraphName = "Text Display For Study";

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

        Subgraph_TextDisplay.Name = "Text Display";
        Subgraph_TextDisplay.LineWidth = 20;
        Subgraph_TextDisplay.DrawStyle = DRAWSTYLE_CUSTOM_TEXT;
        Subgraph_TextDisplay.PrimaryColor = RGB(0, 0, 0); //black
        Subgraph_TextDisplay.SecondaryColor = RGB(128, 255, 255);
        Subgraph_TextDisplay.SecondaryColorUsed = true;
        Subgraph_TextDisplay.DisplayNameValueInWindowsFlags = 0;

        Input_HorizontalPosition.Name.Format("Horizontal Position From Left (1-%d)", static_cast<int>
(CHART_DRAWING_MAX_HORIZONTAL_AXIS_RELATIVE_POSITION));
        Input_HorizontalPosition.SetInt(20);
        Input_HorizontalPosition.SetIntLimits(1, static_cast<int>
(CHART_DRAWING_MAX_HORIZONTAL_AXIS_RELATIVE_POSITION));

        Input_VerticalPosition.Name.Format("Vertical Position From Bottom (1-%d)", static_cast<int>
(CHART_DRAWING_MAX_VERTICAL_AXIS_RELATIVE_POSITION));
        Input_VerticalPosition.SetInt(90);
        Input_VerticalPosition.SetIntLimits(1, static_cast<int>
(CHART_DRAWING_MAX_VERTICAL_AXIS_RELATIVE_POSITION));

        Input_StudySubgraphReference.Name = "Study and Subgraph to Display";
        Input_StudySubgraphReference.SetStudySubgraphValues(1, 0);

        Input_DisplaySubgraphName.Name = "Display Subgraph Name";
        Input_DisplaySubgraphName.SetYesNo(false);

        Input_SubgraphOffset.Name = "Subgraph Columns Back";
        Input_SubgraphOffset.SetInt(0);

        Input_PrefixTextForDisplayValue.Name = "Prefix Text for Display Value";
        Input_PrefixTextForDisplayValue.SetString("");

        Input_SuffixTextForDisplayValue.Name = "Suffix Text for Display Value";

```

```

Input_SuffixTextForDisplayValue.SetString("");

Input_UseTextForNonZeroValue.Name = "Use Text for Non-Zero Value";
Input_UseTextForNonZeroValue.SetYesNo(false);

Input_NonZeroValueText.Name = "Non-Zero Value Text";
Input_NonZeroValueText.SetString("");

return;
}

// Do data processing

SCFloatArray StudyReference;
sc.GetStudyArrayUsingID(Input_StudySubgraphReference.GetStudyID(),
Input_StudySubgraphReference.GetSubgraphIndex(), StudyReference);

int ColumnsBack = Input_SubgraphOffset.GetInt();
SCString ValueText;
if (Input_PrefixTextForDisplayValue.GetString()[0] != '\0')
{
    ValueText += Input_PrefixTextForDisplayValue.GetString();
    ValueText += " ";
}

if (Input_DisplaySubgraphName.GetYesNo())
{
    const char* SubgraphName = sc.GetStudySubgraphName(Input_StudySubgraphReference.GetStudyID(),
Input_StudySubgraphReference.GetSubgraphIndex());

    if (SubgraphName != NULL)
    {
        ValueText += SubgraphName;
        ValueText += ": ";
    }
}

float StudyValue = StudyReference[StudyReference.GetArraySize() - 1 - ColumnsBack];

if (StudyValue == -FLT_MAX)//Patch for when referencing data from the Numbers Bars Calculated Values study. This
value could be used.
    StudyValue = 0;

if(Input_UseTextForNonZeroValue.GetYesNo() )
{
    if (StudyValue != 0)
        ValueText += Input_NonZeroValueText.GetString();
    else
        ValueText += " ";
}
else
    ValueText += sc.FormatGraphValue(StudyValue, sc.GetValueFormat());

ValueText += Input_SuffixTextForDisplayValue.GetString();

sc.AddAndManageSingleTextDrawingForStudy
( sc
, false
, Input_HorizontalPosition.GetInt()
, Input_VerticalPosition.GetInt()
, Subgraph_TextDisplay
, false
, ValueText

```

```

    , sc.DrawStudyUnderneathMainPriceGraph? 0 : 1
);

}

/*===== */
SCSFExport scsf_ProfitLossTextStudy(SCStudyInterfaceRef sc)
{
    SCInputRef Input_HorizontalPosition = sc.Input[0];
    SCInputRef Input_VerticalPosition = sc.Input[1];
    SCInputRef Input_DisplayInFillSpace = sc.Input[2];
    SCInputRef Input_ProfitLossFormat = sc.Input[3];
    SCInputRef Input_RoundTurnCommission = sc.Input[4];

    SCInputRef Input_ShowOpenPL = sc.Input[5];
    SCInputRef Input_ShowClosedPL = sc.Input[6];
    SCInputRef Input_ShowDailyPL = sc.Input[7];
    SCInputRef Input_IncludeOpenPLInDailyPL = sc.Input[8];

    SCInputRef Input_TextSize = sc.Input[10];
    SCInputRef Input_TextColor = sc.Input[11];
    SCInputRef Input_TextBackColor = sc.Input[12];
    SCInputRef Input_ShowTotalTrades = sc.Input[13];
    SCInputRef Input_ShowTotalQuantityFilled = sc.Input[14];
    SCInputRef Input_ShowDailyTotalTrades = sc.Input[15];
    SCInputRef Input_ShowDailyTotalQuantityFilled = sc.Input[16];
    SCInputRef Input_ShowMaximumOpenPositionProfit = sc.Input[17];
    SCInputRef Input_ShowMaximumOpenPositionLoss = sc.Input[18];
    SCInputRef Input_ShowFlatToFlatClosedProfitLoss = sc.Input[19];
    SCInputRef Input_ShowCurrentTradeTimeDuration = sc.Input[20];
    SCInputRef Input_ShowNetProfitLossForAllSymbolsForTradeAccount = sc.Input[21];

    if (sc.SetDefaults)
    {
        // Set the configuration and defaults
        sc.GraphName = "Trading: Profit/Loss Text";

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

        Input_HorizontalPosition.Name.Format("Horizontal Position From Left (1-%d)", static_cast<int>
(CHART_DRAWING_MAX_HORIZONTAL_AXIS_RELATIVE_POSITION));
        Input_HorizontalPosition.SetInt(20);
        Input_HorizontalPosition.SetIntLimits(1, static_cast<int>
(CHART_DRAWING_MAX_HORIZONTAL_AXIS_RELATIVE_POSITION));

        Input_VerticalPosition.Name.Format("Vertical Position From Bottom (1-%d)", static_cast<int>
(CHART_DRAWING_MAX_VERTICAL_AXIS_RELATIVE_POSITION));
        Input_VerticalPosition.SetInt(90);
        Input_VerticalPosition.SetIntLimits(1, static_cast<int>
(CHART_DRAWING_MAX_VERTICAL_AXIS_RELATIVE_POSITION));

        Input_DisplayInFillSpace.Name = "Display in Fill Space";
        Input_DisplayInFillSpace.SetYesNo(false);

        Input_ShowOpenPL.Name = "Show Open Profit/Loss";
        Input_ShowOpenPL.SetYesNo(1);

        Input_ShowClosedPL.Name = "Show Closed Profit/Loss";
        Input_ShowClosedPL.SetYesNo(1);

        Input_ShowDailyPL.Name = "Show Daily Profit/Loss";
        Input_ShowDailyPL.SetYesNo(1);
    }
}

```

```

Input_IncludeOpenPLInDailyPL.Name = "Include Open Profit/Loss in Daily Profit/Loss";
Input_IncludeOpenPLInDailyPL.SetYesNo(0);

Input_TextSize.Name = "Text Size";
Input_TextSize.SetInt(14);
Input_TextSize.SetIntLimits(3, 50);

Input_TextColor.Name = "Text Color";
Input_TextColor.SetColor(0,0,0); // black

Input_TextBackColor.Name = "Text Background Color";
Input_TextBackColor.SetColor(128,255,255);

Input_ShowTotalTrades.Name = "Show Total Trades";
Input_ShowTotalTrades.SetYesNo(0);

Input_ShowTotalQuantityFilled.Name = "Show Total Quantity Filled";
Input_ShowTotalQuantityFilled.SetYesNo(0);

Input_ShowDailyTotalTrades.Name = "Show Daily Total Trades";
Input_ShowDailyTotalTrades.SetYesNo(0);

Input_ShowDailyTotalQuantityFilled.Name = "Show Daily Total Quantity Filled";
Input_ShowDailyTotalQuantityFilled.SetYesNo(0);

Input_ShowMaximumOpenPositionProfit.Name = "Show Maximum Open Position Profit";
Input_ShowMaximumOpenPositionProfit.SetYesNo(0);

Input_ShowMaximumOpenPositionLoss.Name = "Show Maximum Open Position Loss";
Input_ShowMaximumOpenPositionLoss.SetYesNo(0);

Input_ShowFlatToFlatClosedProfitLoss.Name = "Show Flat to Flat Closed Profit Loss";
Input_ShowFlatToFlatClosedProfitLoss.SetYesNo(0);

Input_ProfitLossFormat.Name = "Profit/Loss Display Format";
Input_ProfitLossFormat.SetCustomInputStrings("Currency Value;Points (P);Points - Ignore Quantity (p);Ticks
(T);Ticks - Ignore Quantity (t);Currency Value & Points (P);Currency Value & Points - Ignore Quantity (p);Currency Value
& Ticks (T);Currency Value & Ticks - Ignore Quantity (t)");
Input_ProfitLossFormat.SetCustomInputIndex(0);

Input_RoundTurnCommission.Name = "Round Turn Commission To Include";
Input_RoundTurnCommission.SetFloat(0);

Input_ShowCurrentTradeTimeDuration.Name = "Show Current Trade Time Duration";
Input_ShowCurrentTradeTimeDuration.SetYesNo(0);

Input_ShowNetProfitLossForAllSymbolsForTradeAccount.Name = "Show Net Profit/Loss for All Symbols for Trade
Account";
Input_ShowNetProfitLossForAllSymbolsForTradeAccount.SetYesNo(0);

return;
}

// This is to ensure that we access the trade data that matches the simulation mode setting.
sc.SendOrdersToTradeService = !sc.GlobalTradeSimulationIsOn;

if (sc.HideStudy)
return;

int& r_TextDrawingLineNumber = sc.GetPersistentInt(1);

// Do data processing
s_UseTool Tool;

Tool.Clear(); // Reset tool structure for our next use. Unnecessary in this case, but good practice.

```

```

Tool.DrawingType = DRAWING_TEXT;
Tool.ChartNumber = sc.ChartNumber;
Tool.Region = sc.GraphRegion;

if (r_TextDrawingLineNumber != 0)
    Tool.LineNumber = r_TextDrawingLineNumber;

Tool.AddMethod = UTAM_ADD_OR_ADJUST;

if(Input_DisplayInFillSpace.GetYesNo())
    Tool.BeginDateTime = -3;
else
    Tool.BeginDateTime = Input_HorizontalPosition.GetInt();

Tool.BeginValue = static_cast<float>(Input_VerticalPosition.GetInt());
Tool.UseRelativeVerticalValues = true;
Tool.Color = Input_TextColor.GetColor();
Tool.FontFace = sc.ChartTextFont();
Tool.FontBackColor = Input_TextBackColor.GetColor();
Tool.FontSize = Input_TextSize.GetInt();
Tool.FontBold = true;
Tool.ReverseTextColor = false;
Tool.MultiLineLabel = true;
Tool.TextAlignment = DT_LEFT;

ProfitLossDisplayFormatEnum ProfitLossDisplayFormat = static_cast<ProfitLossDisplayFormatEnum>
(Input_ProfitLossFormat.GetIndex()+1);

s_SCPositionData PositionData;
sc.GetTradePosition(PositionData);

const double RoundTurnCommissionRate = sc.CurrencyValuePerTick == 0 ? 0 :
Input_RoundTurnCommission.GetFloat();

SCString Text;

n_ACSIL::s_TradeStatistics AllStats;
n_ACSIL::s_TradeStatistics DailyStats;

sc.GetTradeStatisticsForSymbolV2(n_ACSIL::STATS_TYPE_ALL_TRADES, AllStats);
sc.GetTradeStatisticsForSymbolV2(n_ACSIL::STATS_TYPE_DAILY_ALL_TRADES, DailyStats);

if (Input_ShowOpenPL.GetYesNo())
{
    SCString PLString;
    double Commission = RoundTurnCommissionRate * fabs(PositionData.PositionQuantity) / 2.0;
    sc.CreateProfitLossDisplayString(PositionData.OpenProfitLoss - Commission, PositionData.PositionQuantity,
ProfitLossDisplayFormat, PLString);

    Text.Format("Open PL: %s", PLString.GetChars());
}

if (Input_ShowClosedPL.GetYesNo())
{
    if (!Text.IsEmpty())
        Text.Append("\n");

    SCString PLString;
    double Commission = RoundTurnCommissionRate * fabs(AllStats.TotalQuantity);

    sc.CreateProfitLossDisplayString(AllStats.ClosedTradesProfitLoss - Commission, AllStats.TotalQuantity,
ProfitLossDisplayFormat, PLString);

    SCString ClosedPLText;
    ClosedPLText.Format("Closed PL: %s", PLString.GetChars());
}

```

```

    Text.Append(ClosedPLText);
}

if (Input_ShowDailyPL.GetYesNo())
{
    if (!Text.IsEmpty())
        Text.Append("\n");

    double DailyPL = DailyStats.ClosedTradesProfitLoss;
    double DailyClosedQuantity = DailyStats.TotalQuantity;
    double Commission = RoundTurnCommissionRate * fabs(DailyClosedQuantity);

    if (Input_IncludeOpenPLInDailyPL.GetYesNo())
    {
        DailyPL += PositionData.OpenProfitLoss;
        const double OpenQuantity = abs(PositionData.PositionQuantity);
        DailyClosedQuantity += OpenQuantity;
        Commission += (RoundTurnCommissionRate * OpenQuantity / 2.0);
        Text.Append("Daily Net PL: ");
    }
    else
        Text.Append("Daily PL: ");

    SCString PLString;
    sc.CreateProfitLossDisplayString(DailyPL - Commission, DailyClosedQuantity, ProfitLossDisplayFormat, PLString);

    Text.Append(PLString);
}

if (Input_ShowTotalTrades.GetYesNo())
{
    if (!Text.IsEmpty())
        Text.Append("\n");

    SCString TotalTradesText;
    TotalTradesText.Format("Total Trades: %i", AllStats.TotalTrades);

    Text.Append(TotalTradesText);
}

if (Input_ShowTotalQuantityFilled.GetYesNo())
{
    if (!Text.IsEmpty())
        Text.Append("\n");

    SCString TotalFilledText("Total Filled Qty: ");
    SCString QuantityString;
    sc.OrderQuantityToString(AllStats.TotalFilledQuantity, QuantityString);
    TotalFilledText += QuantityString;

    Text.Append(TotalFilledText);
}

if (Input_ShowDailyTotalTrades.GetYesNo())
{
    if (!Text.IsEmpty())
        Text.Append("\n");

    SCString TotalTradesText;
    TotalTradesText.Format("Daily Trades: %i", DailyStats.TotalTrades);

    Text.Append(TotalTradesText);
}

```

```

if (Input_ShowDailyTotalQuantityFilled.GetYesNo())
{
    if (!Text.IsEmpty())
        Text.Append("\n");

    SCString TotalFilledText("Daily Filled Qty: ");
    SCString QuantityString;
    sc.OrderQuantityToString(DailyStats.TotalFilledQuantity, QuantityString);
    TotalFilledText += QuantityString;

    Text.Append(TotalFilledText);
}

if (Input_ShowMaximumOpenPositionProfit.GetYesNo())
{
    if (!Text.IsEmpty())
        Text.Append("\n");

    SCString PLString;
    sc.CreateProfitLossDisplayString(PositionData.MaximumOpenPositionProfit, PositionData.PositionQuantity,
ProfitLossDisplayFormat, PLString);
    Text.AppendFormat("Max Open Profit: %s", PLString.GetChars());
}

if (Input_ShowMaximumOpenPositionLoss.GetYesNo())
{
    if (!Text.IsEmpty())
        Text.Append("\n");

    SCString PLString;
    sc.CreateProfitLossDisplayString(PositionData.MaximumOpenPositionLoss, PositionData.PositionQuantity,
ProfitLossDisplayFormat, PLString);
    Text.AppendFormat("Max Open Loss: %s", PLString.GetChars());
}

if (Input_ShowFlatToFlatClosedProfitLoss.GetYesNo())
{
    if (!Text.IsEmpty())
        Text.Append("\n");

    SCString PLString;
    sc.CreateProfitLossDisplayString(DailyStats.ClosedFlatToFlatTradesProfitLoss, 0, ProfitLossDisplayFormat,
PLString);
    Text.AppendFormat("FTF P/L: %s", PLString.GetChars());
}

if (Input_ShowCurrentTradeTimeDuration.GetYesNo())
{
    if (!Text.IsEmpty())
        Text.Append("\n");

    s_ACSTrade TradeEntry;
    int32_t TradesSize = sc.GetFlatToFlatTradeListSize();
    if (TradesSize >= 1)
    {
        const int Result = sc.GetFlatToFlatTradeListEntry(TradesSize - 1, TradeEntry);

        if (TradeEntry.CloseDateTime.IsUnset())
        {
            SCDateTime TimeDuration = sc.GetCurrentDateTime() - TradeEntry.OpenDateTime;
            SCString DurationString;
            DurationString = sc.DateTimeToString(TimeDuration.GetAsDouble(),
FLAG_DT_DATE_TIME_AS_TIME_DURATION_FORMAT);
            Text += DurationString;
        }
    }
}

```

```

    }
}

if (Input_ShowNetProfitLossForAllSymbolsForTradeAccount.GetYesNo())
{
    if (!Text.IsEmpty())
        Text.Append("\n");

    SCString PLString;
    double NetProfitLoss = sc.GetTotalNetProfitLossForAllSymbols(1);
    sc.CreateProfitLossDisplayString(NetProfitLoss, 0, ProfitLossDisplayFormat, PLString);
    Text.AppendFormat("AllSymbols Daily Net P/L: %s", PLString.GetChars());
}

Tool.Text = Text;

sc.UseTool(Tool);

r_TextDrawingLineNumber = Tool.LineNumber;
}

/*=====*/
SCSFExport scsf_UpDownVolumeRatio(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_UpDownVolumeRatio = sc.Subgraph[0];
    SCSubgraphRef Subgraph_ZeroLine = sc.Subgraph[1];

    SCInputRef Input_MovingAverageLength = sc.Input[0];
    SCInputRef Input_VolumeOrTrades = sc.Input[1];
    SCInputRef Input_MovingAverageType = sc.Input [2];

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

        sc.GraphName = "Up/Down Volume Ratio";
        sc.MaintainAdditionalChartDataArrays = 1;

        sc.GraphRegion = 1;

        Subgraph_UpDownVolumeRatio.Name = "VR";
        Subgraph_UpDownVolumeRatio.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_UpDownVolumeRatio.PrimaryColor = RGB(0, 255, 0);
        Subgraph_UpDownVolumeRatio.SecondaryColor = RGB(255, 0, 0);
        Subgraph_UpDownVolumeRatio.LineWidth = 3;
        Subgraph_UpDownVolumeRatio.AutoColoring = AUTOCOLOR_POSNEG;
        Subgraph_UpDownVolumeRatio.DrawZeros = false; // false

        Subgraph_ZeroLine.Name = "Zero";
        Subgraph_ZeroLine.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_ZeroLine.PrimaryColor = RGB(128, 128, 128);
        Subgraph_ZeroLine.LineWidth = 2;
        Subgraph_ZeroLine.DrawZeros = true; // true

        Input_MovingAverageLength.Name = "Moving Average Length";
        Input_MovingAverageLength.SetInt(14);
        Input_MovingAverageLength.SetIntLimits(1, INT_MAX);

        Input_VolumeOrTrades.Name = "Calculation Based On";
        Input_VolumeOrTrades.SetCustomInputStrings("Up/Down Volume;Ask/Bid Volume;Up/Down Trades");
        Input_VolumeOrTrades.SetCustomInputIndex(0);
    }
}

```

```

Input_MovingAverageType.Name = "Moving Average Type";
Input_MovingAverageType.SetMovAvgType(MOVAVGTYPE_EXPONENTIAL);

sc.AutoLoop = 1;

return;
}

// Do data processing
int Up = SC_UPVOL;
int Down = SC_DOWNVOL;

if(Input_VolumeOrTrades.GetIndex() == 0)
{
    Up = SC_UPVOL;
    Down = SC_DOWNVOL;
}
else if(Input_VolumeOrTrades.GetIndex() == 1)
{
    Up = SC_ASKVOL;
    Down = SC_BIDVOL;
}
else
{
    Up = SC_ASKNT;
    Down = SC_BIDNT;
}

float TotalTrades = sc.BaseData[Up][sc.Index] + sc.BaseData[Down][sc.Index];

if (TotalTrades > 0)
    Subgraph_UpDownVolumeRatio.Arrays[0][sc.Index] = 100 * (sc.BaseData[Up][sc.Index] - sc.BaseData[Down]
[sc.Index]) / TotalTrades;
else
    Subgraph_UpDownVolumeRatio.Arrays[0][sc.Index] = 0.0f;

sc.MovingAverage(Subgraph_UpDownVolumeRatio.Arrays[0], Subgraph_UpDownVolumeRatio,
Input_MovingAverageType.GetMovAvgType(), Input_MovingAverageLength.GetInt());
}

/*=====*/

SCSFExport scsf_MovAvgCross(SCStudyInterfaceRef sc)
{
    SCInputRef Input_MA1Price = sc.Input[0];
    SCInputRef Input_MA1Type = sc.Input[1];
    SCInputRef Input_MA1Length = sc.Input[2];
    SCInputRef Input_MA2Price = sc.Input[3];
    SCInputRef Input_MA2Type = sc.Input[4];
    SCInputRef Input_MA2Length = sc.Input[5];
    SCInputRef Input_DotsSpacing = sc.Input[6];

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

        sc.GraphName = "Moving Average Crossover";

        sc.AutoLoop = 1;

        sc.GraphRegion = 0;

        sc.Subgraph[0].Name = "Moving Average 1";

```

```

sc.Subgraph[0].DrawStyle = DRAWSTYLE_LINE;
sc.Subgraph[0].PrimaryColor = RGB(0,255,0);

sc.Subgraph[1].Name = "Moving Average 2";
sc.Subgraph[1].DrawStyle = DRAWSTYLE_LINE;
sc.Subgraph[1].PrimaryColor = RGB(255,0,255);

sc.Subgraph[2].Name = "Buy/Sell";
sc.Subgraph[2].SecondaryColorUsed = 1;
sc.Subgraph[2].PrimaryColor = RGB(0,255,0);
sc.Subgraph[2].SecondaryColor = RGB(255,0,0);
sc.Subgraph[2].DrawStyle = DRAWSTYLE_POINT;
sc.Subgraph[2].LineWidth = 4;

Input_MA1Price.Name = "Moving Average 1 Input Data";
Input_MA1Price.SetInputDataIndex(SC_LAST);
Input_MA1Type.Name = "MA1.Type";
Input_MA1Type.SetMovAvgType(MOVAVGTYPE_EXPONENTIAL);
Input_MA1Length.Name = "MA1.Length";
Input_MA1Length.SetInt(5);
Input_MA1Length.SetIntLimits(1,MAX_STUDY_LENGTH);

Input_MA2Price.Name = "Moving Average 2 Input Data";
Input_MA2Price.SetInputDataIndex(SC_OPEN);
Input_MA2Type.Name = "MA2.Type";
Input_MA2Type.SetMovAvgType(MOVAVGTYPE_WEIGHTED);
Input_MA2Length.Name = "MA2.Length";
Input_MA2Length.SetInt(6);
Input_MA2Length.SetIntLimits(1,MAX_STUDY_LENGTH);

return;
}

// Do data processing
sc.MovingAverage(sc.BaseData[Input_MA1Price.GetInputDataIndex()],
sc.Subgraph[0],Input_MA1Type.GetMovAvgType(),Input_MA1Length.GetInt());

sc.MovingAverage(sc.BaseData[Input_MA2Price.GetInputDataIndex()],
sc.Subgraph[1],Input_MA2Type.GetMovAvgType(),Input_MA2Length.GetInt());

int direction = sc.CrossOver(sc.Subgraph[0],sc.Subgraph[1]);
if (direction == CROSS_FROM_TOP)
{
    sc.Subgraph[2][sc.Index] = sc.BaseData[SC_HIGH][sc.Index];
    sc.Subgraph[2].DataColor[sc.Index] = sc.Subgraph[2].SecondaryColor;
}
else if (direction == CROSS_FROM_BOTTOM)
{
    sc.Subgraph[2][sc.Index] = sc.BaseData[SC_LOW][sc.Index];
    sc.Subgraph[2].DataColor[sc.Index] = sc.Subgraph[2].PrimaryColor;
}
else
{
    sc.Subgraph[2][sc.Index] = 0;
}
}

/*=====*/
SCSFExport scsf_BillWilliamsAC(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Up = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Down = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Flat = sc.Subgraph[2];

```

```

SCSubgraphRef Subgraph_All = sc.Subgraph[3];
SCSubgraphRef Subgraph_Temp4 = sc.Subgraph[4];
SCSubgraphRef Subgraph_Sum1 = sc.Subgraph[5];
SCSubgraphRef Subgraph_Avg1 = sc.Subgraph[6];
SCSubgraphRef Subgraph_Sum2 = sc.Subgraph[7];
SCSubgraphRef Subgraph_Avg2 = sc.Subgraph[8];
SCSubgraphRef Subgraph_Sum4 = sc.Subgraph[9];
SCSubgraphRef Subgraph_Avg4 = sc.Subgraph[10];

SCInputRef Input_LongMaLength = sc.Input[0];
SCInputRef Input_ShortMaLength = sc.Input[1];
SCInputRef Input_SignalMaLength = sc.Input[2];
SCInputRef Input_MaType = sc.Input[3];

if (sc.SetDefaults)
{
    sc.GraphName = "Bill Williams AC";

    sc.StudyDescription = "";

    Subgraph_Up.Name = "Up";
    Subgraph_Up.DrawStyle = DRAWSTYLE_DASH;
    Subgraph_Up.PrimaryColor = RGB( 0, 255, 0);
    Subgraph_Up.DrawZeros = false;

    Subgraph_Down.Name = "Down";
    Subgraph_Down.DrawStyle = DRAWSTYLE_DASH;
    Subgraph_Down.PrimaryColor = RGB( 255, 0, 255);
    Subgraph_Down.DrawZeros = false;

    Subgraph_Flat.Name = "Flat";
    Subgraph_Flat.DrawStyle = DRAWSTYLE_DASH;
    Subgraph_Flat.PrimaryColor = RGB( 192, 192, 192);
    Subgraph_Flat.DrawZeros = false;

    Subgraph_All.Name = "All";
    Subgraph_All.DrawStyle = DRAWSTYLE_IGNORE;
    Subgraph_All.PrimaryColor = RGB( 255, 127, 0);
    Subgraph_All.DrawZeros = false;

    Input_LongMaLength.Name = "Long Moving Average Length";
    Input_LongMaLength.SetInt(34);
    Input_LongMaLength.SetIntLimits(1, INT_MAX);

    Input_ShortMaLength.Name = "Short Moving Average Length";
    Input_ShortMaLength.SetInt(5);
    Input_ShortMaLength.SetIntLimits(1, INT_MAX);

    Input_SignalMaLength.Name = "Signal Moving Average Length";
    Input_SignalMaLength.SetInt(5);
    Input_SignalMaLength.SetIntLimits(1, INT_MAX);

    Input_MaType.Name = "Moving Average Type";
    Input_MaType.SetCustomInputStrings("Bill Williams Smoothed;Exponential;Simple");
    Input_MaType.SetCustomInputIndex(0);

    sc.AutoLoop = 1;

    return;
}

// Do data processing

```

```

const float cfactor1 = 2.0f / (Input_LongMaLength.GetInt() + 1);    const float cinvfactor1 = 1.0f - cfactor1;
const float cfactor2 = 2.0f / (Input_ShortMaLength.GetInt() + 1);  const float cinvfactor2 = 1.0f - cfactor2;
const float cfactor4 = 2.0f / (Input_SignalMaLength.GetInt() + 1); const float cinvfactor4 = 1.0f - cfactor4;

sc.DataStartIndex = 50;

int pos = sc.Index;
if (pos < 50)
    return;

int j;
float avg3, avg4;

switch (Input_MaType.GetInt())
{
case 0:
{
    Subgraph_Sum1[pos] = Subgraph_Sum1[pos - 1] + sc.Close[pos];
    if (pos >= 50 + Input_LongMaLength.GetInt())
        Subgraph_Sum1[pos] = Subgraph_Sum1[pos] - Subgraph_Avg1[pos - 1];
    Subgraph_Avg1[pos] = Subgraph_Sum1[pos] / Input_LongMaLength.GetInt();

    Subgraph_Sum2[pos] = Subgraph_Sum2[pos - 1] + sc.Close[pos];
    if (pos >= 50 + Input_ShortMaLength.GetInt())
        Subgraph_Sum2[pos] = Subgraph_Sum2[pos] - Subgraph_Avg2[pos - 1];
    Subgraph_Avg2[pos] = Subgraph_Sum2[pos] / Input_ShortMaLength.GetInt();

    avg3 = Subgraph_Avg2[pos] - Subgraph_Avg1[pos];

    Subgraph_Temp4.Data[pos] = avg3;
    Subgraph_Sum4[pos] = Subgraph_Sum4[pos - 1] + avg3;
    avg4 = Subgraph_Sum4[pos] / Input_SignalMaLength.GetInt();
    j = pos - Input_SignalMaLength.GetInt() + 1;
    if (pos > 50 + Input_SignalMaLength.GetInt() - 2)
        Subgraph_Sum4[pos] = Subgraph_Sum4[pos] - Subgraph_Temp4.Data[j];
}
break;

case 1:
{
    Subgraph_Avg1[pos] = Subgraph_Avg1[pos - 1] * cinvfactor1 + sc.Close[pos] * cfactor1;
    Subgraph_Avg2[pos] = Subgraph_Avg2[pos - 1] * cinvfactor2 + sc.Close[pos] * cfactor2;
    avg3 = Subgraph_Avg2[pos] - Subgraph_Avg1[pos];
    //Temp4.Data[pos] = avg3;
    avg4 = Subgraph_Avg4[pos] = Subgraph_Avg4[pos - 1] * cinvfactor4 + avg3 * cfactor4;
}
break;

case 2:
default:
{
    Subgraph_Sum1[pos] = Subgraph_Sum1[pos - 1] + sc.Close[pos];
    float avg1 = Subgraph_Sum1[pos] / Input_LongMaLength.GetInt();
    j = pos - Input_LongMaLength.GetInt() + 1;
    if (pos > 50 + Input_LongMaLength.GetInt() - 2)
        Subgraph_Sum1[pos] = Subgraph_Sum1[pos] - sc.Close[j];

    Subgraph_Sum2[pos] = Subgraph_Sum2[pos - 1] + sc.Close[pos];
    float avg2 = Subgraph_Sum2[pos] / Input_ShortMaLength.GetInt();
    j = pos - Input_ShortMaLength.GetInt() + 1;
    if (pos > 50 + Input_ShortMaLength.GetInt() - 2)
        Subgraph_Sum2[pos] = Subgraph_Sum2[pos] - sc.Close[j];

    avg3 = avg2 - avg1;

```

```

    Subgraph_Temp4.Data[pos] = avg3;
    Subgraph_Sum4[pos] = Subgraph_Sum4[pos - 1] + avg3;
    avg4 = Subgraph_Sum4[pos] / Input_SignalMaLength.GetInt();
    j = pos - Input_SignalMaLength.GetInt() + 1;
    if (pos > 50 + Input_SignalMaLength.GetInt() - 2)
        Subgraph_Sum4[pos] = Subgraph_Sum4[pos] - Subgraph_Temp4.Data[j];
}
break;
}

```

```

Subgraph_All.Data[pos] = avg3 - avg4;

```

```

if (Subgraph_All.Data[pos] > Subgraph_All.Data[pos - 1])
{
    Subgraph_Up.Data[pos] = Subgraph_All.Data[pos];
    Subgraph_Down.Data[pos] = 0;
    Subgraph_Flat.Data[pos] = 0;
}
else if (Subgraph_All.Data[pos] < Subgraph_All.Data[pos - 1])
{
    Subgraph_Up.Data[pos] = 0;
    Subgraph_Down.Data[pos] = Subgraph_All.Data[pos];
    Subgraph_Flat.Data[pos] = 0;
}
else
{
    Subgraph_Up.Data[pos] = 0;
    Subgraph_Down.Data[pos] = 0;
    Subgraph_Flat.Data[pos] = Subgraph_All.Data[pos];
}
}

```

```

/*=====*/

```

```

SCSFExport scsf_BillWilliamsAO(SCStudyInterfaceRef sc)

```

```

{
    SCSubgraphRef Subgraph_AO = sc.Subgraph[0];
    SCSubgraphRef Subgraph_ZeroLine = sc.Subgraph[1];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_MA1Length = sc.Input[2];
    SCInputRef Input_MA2Length = sc.Input[3];
    SCInputRef Input_MAType = sc.Input[4];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Bill Williams Awesome Oscillator";

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

        Subgraph_AO.Name = "AO";
        Subgraph_AO.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_AO.LineWidth = 2;
        Subgraph_AO.SecondaryColor = RGB(255,0,0);
        Subgraph_AO.AutoColoring = AUTOCOLOR_SLOPE;
        Subgraph_AO.DrawZeros = false;

        Subgraph_ZeroLine.Name = "Zero Line";
        Subgraph_ZeroLine.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_ZeroLine.LineWidth = 1;
        Subgraph_ZeroLine.PrimaryColor = RGB(255,0,255);
        Subgraph_ZeroLine.DrawZeros = true;
    }
}

```

```

Input_InputData.SetInputDataIndex(SC_HL_AVG);

Input_MA1Length.Name = "Moving Average 1 Length";
Input_MA1Length.SetInt(34);
Input_MA1Length.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_MA2Length.Name = "Moving Average 2 Length";
Input_MA2Length.SetInt(5);
Input_MA2Length.SetIntLimits(1, MAX_STUDY_LENGTH);

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

return;
}

sc.DataStartIndex = max(Input_MA1Length.GetInt(), Input_MA2Length.GetInt());

sc.MovingAverage(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_AO.Arrays[0],
Input_MAType.GetMovAvgType(), Input_MA1Length.GetInt());
sc.MovingAverage(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_AO.Arrays[1],
Input_MAType.GetMovAvgType(), Input_MA2Length.GetInt());

Subgraph_AO.Data[sc.Index] = Subgraph_AO.Arrays[1][sc.Index] - Subgraph_AO.Arrays[0][sc.Index];
}

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

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_MovingAverageLength = sc.Input[2];
    SCInputRef Input_MovingAverageType = sc.Input[4];
    SCInputRef Input_Version = sc.Input[5];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Bill Williams Moving Average";

        sc.StudyDescription = "To offset the moving average, use the Displacement setting for the MovAvg Subgraph.";
        sc.GraphRegion= 0;

        Subgraph_MovAvg.Name = "MovAvg";
        Subgraph_MovAvg.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_MovAvg.PrimaryColor = RGB(0,255,0);

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

        Input_MovingAverageLength.Name = "MovAvg Length";
        Input_MovingAverageLength.SetInt(34);
        Input_MovingAverageLength.SetIntLimits(1, INT_MAX);

        Input_MovingAverageType.Name = "Moving Average Type";
        Input_MovingAverageType.SetCustomInputStrings(";Bill Willams EMA;Smoothed Moving Average");
        Input_MovingAverageType.SetCustomInputIndex(2);

        Input_Version.SetInt(1);

```

```

    sc.AutoLoop = 1;

    return;
}

// Do data processing

sc.DataStartIndex = 0;

if (Input_Version.GetInt() < 1)
{
    Input_InputData.SetInputDataIndex(SC_LAST);
    Input_Version.SetInt(1);
}

if (Input_MovingAverageType.GetIndex() == 0)
    Input_MovingAverageType.SetCustomInputIndex(1);

uint32_t InputIndex = Input_InputData.GetInputDataIndex();

if (Input_MovingAverageType.GetIndex() == 1)
{
    if (sc.Index == 0)
    {
        Subgraph_MovAvg[0] = sc.BaseData[InputIndex][0];
        Subgraph_MovAvg.Arrays[4][sc.Index] = sc.BaseData[InputIndex][0]*Input_MovingAverageLength.GetInt();
    }
    else
    {
        Subgraph_MovAvg.Arrays[4][sc.Index] = Subgraph_MovAvg.Arrays[4][sc.Index - 1] + sc.BaseData[InputIndex]
[sc.Index];
        Subgraph_MovAvg.Arrays[4][sc.Index] -= Subgraph_MovAvg[sc.Index - 1];
        Subgraph_MovAvg[sc.Index] = Subgraph_MovAvg.Arrays[4][sc.Index] / Input_MovingAverageLength.GetInt();
    }
}
else
{
    sc.SmoothedMovingAverage(sc.BaseData[InputIndex], Subgraph_MovAvg, sc.Index,
Input_MovingAverageLength.GetInt());
}
}

/*=====*/
SCSFExport scsf_BillWilliamsAlligator(SCStudyInterfaceRef sc)
{
    const int jaw = 0;
    const int teeth = 1;
    const int lips = 2;

    SCSubgraphRef Subgraph_Jaw = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Teeth = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Lips = sc.Subgraph[2];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_JawLength = sc.Input[1];
    SCInputRef Input_TeethLength = sc.Input[3];
    SCInputRef Input_LipsLength = sc.Input[5];
    SCInputRef Input_VersionUpdate1 = sc.Input[7];
    SCInputRef Input_MovAvgType = sc.Input[8];
    SCInputRef Input_VersionUpdate2 = sc.Input[9];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Bill Williams Alligator";
    }
}

```

```

sc.StudyDescription = "";

// Set the region to draw the graph in. Region zero is the main
// price graph region.
sc.GraphRegion = 0;

// Set the name of the first subgraph
// Set the color and style of the graph line. If these are not set
// the default will be used.
Subgraph_Jaw.Name = "Jaw";
Subgraph_Jaw.PrimaryColor = RGB(0,0,255); // Blue
Subgraph_Jaw.DrawStyle = DRAWSTYLE_LINE;
Subgraph_Jaw.GraphicalDisplacement = 8;
Subgraph_Jaw.DrawZeros = false;

// Set the color and style of the graph line. If these are not set
// the default will be used.
Subgraph_Teeth.Name = "Teeth";
Subgraph_Teeth.PrimaryColor = RGB(255,0,0); // Red
Subgraph_Teeth.DrawStyle = DRAWSTYLE_LINE;
Subgraph_Teeth.GraphicalDisplacement = 5;
Subgraph_Teeth.DrawZeros = false;

// Set the color and style of the graph line. If these are not set
// the default will be used.
Subgraph_Lips.Name = "Lips";
Subgraph_Lips.PrimaryColor = RGB(0, 255, 0); // Red
Subgraph_Lips.DrawStyle = DRAWSTYLE_LINE;
Subgraph_Lips.GraphicalDisplacement = 3;
Subgraph_Lips.DrawZeros = false;

sc.AutoLoop = 1;

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

Input_JawLength.Name = "Jaw Length";
Input_JawLength.SetIntLimits(1,MAX_STUDY_LENGTH);
Input_JawLength.SetInt(13);

Input_TeethLength.Name = "Teeth Length";
Input_TeethLength.SetIntLimits(1,MAX_STUDY_LENGTH);
Input_TeethLength.SetInt(8);

Input_LipsLength.Name = "Lips Length";
Input_LipsLength.SetIntLimits(1,MAX_STUDY_LENGTH);
Input_LipsLength.SetInt(5);

Input_VersionUpdate1.SetYesNo(true); //Used For Settings Updating

Input_MovAvgType.Name = "Moving Average Type";
Input_MovAvgType.SetMovAvgType(MOVAVGTYPE_SMOOTHED);

Input_VersionUpdate2.SetYesNo(true);
return;
}

// data processing

if (Input_VersionUpdate2.GetYesNo() == false)
    Input_MovAvgType.SetMovAvgType(MOVAVGTYPE_SMOOTHED);

if (Input_VersionUpdate1.GetYesNo() == false)
{

```

```

    Subgraph_Jaw.GraphicalDisplacement = 8;
    Subgraph_Teeth.GraphicalDisplacement = 5;
    Subgraph_Lips.GraphicalDisplacement = 3;
}

// calculate jaw, teeth and lips

    sc.MovingAverage(sc.BaseData[Input_InputData.GetInputDataIndex()], Subgraph_Jaw,
Input_MovAvgType.GetMovAvgType(), Input_JawLength.GetInt());
    sc.MovingAverage(sc.BaseData[Input_InputData.GetInputDataIndex()], Subgraph_Teeth,
Input_MovAvgType.GetMovAvgType(), Input_TeethLength.GetInt());
    sc.MovingAverage(sc.BaseData[Input_InputData.GetInputDataIndex()], Subgraph_Lips,
Input_MovAvgType.GetMovAvgType(), Input_LipsLength.GetInt());
}

/*=====*/

SCSFExport scsf_AverageOfAverage(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Avg = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Temp1 = sc.Subgraph[1];

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

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

        sc.GraphName = "Average of Average";

        sc.StudyDescription = "";

        sc.AutoLoop = 1; // true

        Subgraph_Avg.Name = "Avg";
        Subgraph_Avg.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Avg.PrimaryColor = RGB(0,255,0);

        Input_Length.Name = "Length";
        Input_Length.SetInt(10);
        Input_Length.SetIntLimits(1, INT_MAX);

        Input_EmaLength.Name = "ExpMA Length";
        Input_EmaLength.SetInt(10);
        Input_EmaLength.SetIntLimits(1, INT_MAX);

        return;
    }

    // Do data processing

    /* This calculates the Exponential Moving Average of a Linear Regression Indicator
    by passing the output array from the Linear Regression Indicator function to the
    Input data parameter of Exponential Moving Average function.*/

    sc.LinearRegressionIndicator(sc.Close, Subgraph_Temp1, sc.Index, Input_Length.GetInt());
    sc.ExponentialMovAvg(Subgraph_Temp1, Subgraph_Avg, sc.Index, Input_EmaLength.GetInt());
}

```

```
/*=====*/
```

```
SCSFExport scsf_WoodiesEMA(SCStudyInterfaceRef sc)
```

```
{  
    SCSubgraphRef Subgraph_EMA = sc.Subgraph[0];  
    SCSubgraphRef Subgraph_UpDownColors = sc.Subgraph[1];  
    SCSubgraphRef Subgraph_EMAOutput = sc.Subgraph[2];  
  
    SCInputRef Input_NumberOfBars = sc.Input[0];  
    SCInputRef Input_PriceChange = sc.Input[1];  
    SCInputRef Input_EMALength = sc.Input[2];  
    SCInputRef Input_FixedValue = sc.Input[4];  
    SCInputRef Input_InputData = sc.Input[6];  
  
    if(sc.SetDefaults)  
    {  
        sc.GraphName="Woodies EMA";  
  
        sc.AutoLoop = 1;  
  
        sc.GraphRegion = 0;  
  
        Subgraph_EMA.Name = "EMA";  
        Subgraph_EMA.DrawStyle = DRAWSTYLE_LINE;  
        Subgraph_EMA.PrimaryColor = RGB(0,255,0);  
        Subgraph_EMA.DrawZeros = false;  
  
        Subgraph_UpDownColors.Name = "Up / Down Color";  
        Subgraph_UpDownColors.DrawStyle = DRAWSTYLE_IGNORE;  
        Subgraph_UpDownColors.PrimaryColor = RGB(0,255,0);  
        Subgraph_UpDownColors.SecondaryColor = RGB(255,0,0);  
        Subgraph_UpDownColors.SecondaryColorUsed = 1;  
        Subgraph_UpDownColors.DrawZeros = false;  
  
        Input_InputData.Name = "Input Data";  
        Input_InputData.SetInputDataIndex(SC_HLC_AVG);  
  
        Input_NumberOfBars.Name = "Number of Bars for Angle";  
        Input_NumberOfBars.SetInt(4);  
        Input_NumberOfBars.SetIntLimits(1,MAX_STUDY_LENGTH);  
  
        Input_PriceChange.Name = "Price Change for Angle";  
        Input_PriceChange.SetFloat(0.000001f);  
  
        Input_EMALength.Name = "EMA Length";  
        Input_EMALength.SetInt(34);  
        Input_EMALength.SetIntLimits(1,MAX_STUDY_LENGTH);  
  
        Input_FixedValue.Name = "Fixed Value to Use";  
        Input_FixedValue.SetFloat(0);  
  
        return;  
    }  
}
```

```
sc.DataStartIndex = Input_EMALength.GetInt() + Input_NumberOfBars.GetInt();
```

```
sc.ExponentialMovAvg(sc.BaseData[Input_InputData.GetInputDataIndex()], Subgraph_EMAOutput,  
Input_EMALength.GetInt());
```

```
if (sc.Index < Input_NumberOfBars.GetInt())  
    return;
```

```
int BarIndex = sc.Index;
```

```

if(Input_FixedValue.GetFloat() == 0.0f)
    Subgraph_EMA[BarIndex] = Subgraph_EMAOutput[BarIndex];
else
    Subgraph_EMA[BarIndex] = Input_FixedValue.GetFloat();

if (Subgraph_EMAOutput[BarIndex] - Subgraph_EMAOutput[BarIndex-Input_NumberOfBars.GetInt()] >=
Input_PriceChange.GetFloat())
{
    Subgraph_EMA.DataColor[BarIndex] = Subgraph_UpDownColors.PrimaryColor;
}
else if (Subgraph_EMAOutput[BarIndex] - Subgraph_EMAOutput[BarIndex-Input_NumberOfBars.GetInt()] <= -
Input_PriceChange.GetFloat())
{
    Subgraph_EMA.DataColor[BarIndex] = Subgraph_UpDownColors.SecondaryColor;
}
else
{
    Subgraph_EMA.DataColor[BarIndex] = Subgraph_EMA.PrimaryColor;
}
}

/*=====*/
SCSFExport scsf_WoodieCCITrend(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Above = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Below = sc.Subgraph[1];
    SCSubgraphRef Subgraph_AboveColorValues = sc.Subgraph[2];
    SCSubgraphRef Subgraph_DoNotDraw1 = sc.Subgraph[3];
    SCSubgraphRef Subgraph_DoNotDraw2 = sc.Subgraph[4];
    SCSubgraphRef Subgraph_CCI = sc.Subgraph[5];

    SCInputRef Input_WarningBar = sc.Input[0];
    SCInputRef Input_AlternateColor = sc.Input[1];
    SCInputRef Input_CCILength = sc.Input[2];
    SCInputRef Input_NumberOfBars = sc.Input[3];
    SCInputRef Input_BarsIntertrendState = sc.Input[4];
    SCInputRef Input_BarsFixPos = sc.Input[5];

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

        sc.AutoLoop = 0;

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

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

        Subgraph_AboveColorValues.Name = "Color Value Output";
        Subgraph_AboveColorValues.PrimaryColor = RGB(255, 255, 255);
        Subgraph_AboveColorValues.DrawStyle = DRAWSTYLE_IGNORE;
        Subgraph_AboveColorValues.DrawZeros = false;

        Subgraph_DoNotDraw1.DrawStyle = DRAWSTYLE_HIDDEN;
        Subgraph_DoNotDraw1.PrimaryColor = RGB(255,255,0);
        Subgraph_DoNotDraw1.DrawZeros = false;
    }
}

```

```

Subgraph_DoNotDraw2.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_DoNotDraw2.PrimaryColor = RGB(98,98,98);
Subgraph_DoNotDraw2.DrawZeros = false;

Subgraph_CCI.Name = "CCI";
Subgraph_CCI.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_CCI.PrimaryColor = RGB(0,0,255);
Subgraph_CCI.DrawZeros = false;

Input_WarningBar.Name = "Use a Warning Bar";
Input_WarningBar.SetYesNo(false);

Input_AlternateColor.Name = "Alternate Colour for uncertain trend states 0, 1, 2 or 4";
Input_AlternateColor.SetInt(0);
Input_AlternateColor.SetIntLimits(0,4);

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_BarsIntertrendState.Name = "# of Bars for Intertrend State";
Input_BarsIntertrendState.SetInt(1);
Input_BarsIntertrendState.SetIntLimits(1,MAX_STUDY_LENGTH);

Input_BarsFixPos.Name = "0 = Bars or n = Fixed Position (n/1000)";
Input_BarsFixPos.SetFloat(0);

return;
}

Subgraph_AboveColorValues.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_DoNotDraw1.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_DoNotDraw2.DrawStyle = DRAWSTYLE_HIDDEN;

uint32_t Color0 = Subgraph_Above.PrimaryColor;
uint32_t Color1 = Subgraph_Below.PrimaryColor;
uint32_t Color2 = Subgraph_DoNotDraw1.PrimaryColor;
uint32_t Color3 = Subgraph_DoNotDraw2.PrimaryColor;

int ArraySize = sc.ArraySize;

switch (Input_AlternateColor.GetInt())
{
case 4:
case 2:
    Subgraph_DoNotDraw1.Name = "Change on Next Bar";
    Subgraph_DoNotDraw2.Name = "Trend & Position Confused";
    break;
case 1:
    Subgraph_DoNotDraw1.Name = "Trend Up but below Zero";
    Subgraph_DoNotDraw2.Name = "Trend Dn but above Zero";
    break;
case 0:
default:
    Subgraph_DoNotDraw1.Name = "Change on Next Bar";
}

int ud = -2, goingpos = 0, goingneg = 0, posbar = 0, negbar = 0, up = 0,
dn = 0, next = 0;

```

```

for(int i = sc.UpdateStartIndex; i < ArraySize;i++)
{
    sc.CCI(sc.HLCAvg, Subgraph_CCI.Arrays[0], Subgraph_CCI, i, Input_CCILength.GetInt(), 0.015f);
}

int Start = max (sc.UpdateStartIndex, Input_CCILength.GetInt()*2 - 1);

float cci0;
for (int i = Start-1; i >= Input_CCILength.GetInt()*2; i--)
{
    cci0 = sc.Subgraph[5][i];
    if (cci0 > 0)
    {
        dn = 0;
        up++;
    }

    else if (cci0 < 0)
    {
        up = 0;
        dn++;
    }

    if(up >= Input_NumberOfBars.GetInt())
    {
        Start = i + Input_NumberOfBars.GetInt();
        ud=2;
        break;
    }
    else if(dn >= Input_NumberOfBars.GetInt())
    {
        ud=-2;
        Start = i + Input_NumberOfBars.GetInt();
        break;
    }
}

sc.DataStartIndex = Input_CCILength.GetInt()*2 - 1;

for (int pos=Start; pos < ArraySize; pos++)
{
    cci0 = Subgraph_CCI[pos];

    if (cci0>0)
    {
        if(Input_AlternateColor.GetInt() == 4 && Input_BarsIntertrendState.GetInt() <= up+1)
            dn = 0;

        if(ud <= 0)
        {
            if (goingpos == 0)
            {
                goingpos = 1;
                up = 1;
                dn = 0;
                negbar = 0;
                if(Input_AlternateColor.GetInt() == 4 && Input_BarsIntertrendState.GetInt() == 1)
                    ud=0;
            }
            else
            {
                up++;
                if(up == Input_NumberOfBars.GetInt())

```

```

        dn = 0;

        if(Input_AlternateColor.GetInt() == 4 && Input_BarsIntertrendState.GetInt() <= up)
            ud = 0;
    }
}
else if(ud >= 2)
{
    if (goingneg == 1)
    {
        posbar++;
        if(posbar >= 2)
        {
            goingneg = 0;
            dn = 0;
        }
    }
    else
        up++;
}
}

if (cci0<0)
{
    if(Input_AlternateColor.GetInt()==4 && Input_BarsIntertrendState.GetInt()<=dn+1)
        up=0;

    if(ud>= 0)
    {
        if (goingneg==0)
        {
            goingneg=1;
            dn=1;
            up=0;
            posbar=0;
            if(Input_AlternateColor.GetInt()==4 && Input_BarsIntertrendState.GetInt()==1)
                ud=0;
        }
        else
        {
            dn++;
            if(dn==Input_NumberOfBars.GetInt())
                up=0;

            if(Input_AlternateColor.GetInt()==4 && Input_BarsIntertrendState.GetInt()<=dn)
                ud=0;
        }
    }
}
else if(ud <= -2)
{
    if (goingpos == 1)
    {
        negbar++;
        if(negbar>=2)
        {
            goingpos=0;
            up=0;
        }
    }
    else
        dn++;
}
};

```

```

next=0;
if(up==(Input_NumberOfBars.GetInt()-1) && (ud<=-2 || (ud==0 && Input_AlternateColor.GetInt()==4)) &&
Input_WarningBar.GetYesNo()!=0)
    next=1;

else if(dn==(Input_NumberOfBars.GetInt()-1) && (ud>= 2 || (ud==0 && Input_AlternateColor.GetInt()==4)) &&
Input_WarningBar.GetYesNo()!=0)
    next=1;

if(up>=Input_NumberOfBars.GetInt())
    ud=2;

if(dn>=Input_NumberOfBars.GetInt())
    ud=-2;

if (Input_BarsFixPos.GetFloat())
    Subgraph_Above[pos] = Input_BarsFixPos.GetFloat()/1000;
else
    Subgraph_Above[pos] = cci0;

if (next==1)
{
    if(Input_AlternateColor.GetInt()==0 || Input_AlternateColor.GetInt()==2 || Input_AlternateColor.GetInt()==4)
    {
        Subgraph_Above.DataColor[pos] = Color2;
    }
    else if(Input_AlternateColor.GetInt()==1)
    {
        if(Input_BarsFixPos.GetFloat()==0)
        {
            Subgraph_Above.DataColor[pos] = Color0;
            Subgraph_Below[pos]=cci0/2;
        };
    };
}
else
{
    if (ud==0 && Input_AlternateColor.GetInt()==4)
    {
        Subgraph_Above.DataColor[pos] = Color3;
    }
    else if (ud>=2)
    {
        if(Input_AlternateColor.GetInt()==0 || cci0>=0)
        {
            Subgraph_Above.DataColor[pos] = Color0;
        }
        else if(Input_AlternateColor.GetInt()==1)
        {
            Subgraph_Above.DataColor[pos] = Color2;
        }
        else if(Input_AlternateColor.GetInt()==2 || Input_AlternateColor.GetInt()==4)
        {
            Subgraph_Above.DataColor[pos] = Color3;
        }
    }
    else if (ud<=-2)
    {
        if(Input_AlternateColor.GetInt()==0 || cci0<=0)
        {
            Subgraph_Above.DataColor[pos] = Color1;
        }
        else
        {
            Subgraph_Above.DataColor[pos] = Color3;
        }
    }
}

```

```

    }
}

if (Subgraph_Above.DataColor[pos] == Color0)
{
    Subgraph_AboveColorValues[pos] = 1;
}
else if (Subgraph_Above.DataColor[pos] == Color1)
{
    Subgraph_AboveColorValues[pos] = 2;
}
else if (Subgraph_Above.DataColor[pos] == Color2)
{
    Subgraph_AboveColorValues[pos] = 3;
}
else if (Subgraph_Above.DataColor[pos] == Color3)
{
    Subgraph_AboveColorValues[pos] = 4;
}
}
}

/*=====*/
SCSFExport scsf_Sidewinder(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_SwTop = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SwBottom = sc.Subgraph[1];
    SCSubgraphRef Subgraph_FlatColor = sc.Subgraph[2];
    SCSubgraphRef Subgraph_OutWorksheet = sc.Subgraph[3];
    SCSubgraphRef Subgraph_Lsma = sc.Subgraph[4];
    SCSubgraphRef Subgraph_LsmaAngle = sc.Subgraph[5];
    SCSubgraphRef Subgraph_Ema = sc.Subgraph[6];
    SCSubgraphRef Subgraph_EmaAngle = sc.Subgraph[7];

    SCInputRef Input_LsmaLength = sc.Input[0];
    SCInputRef Input_EmaLength = sc.Input[1];
    SCInputRef Input_TrendingThreshold = sc.Input[3];
    SCInputRef Input_NormalThreshold = sc.Input[4];
    SCInputRef Input_LineValue = sc.Input[5];
    SCInputRef Input_TickValueOverride = sc.Input[6];

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

        sc.AutoLoop = 0;

        Subgraph_SwTop.Name = "SW Top";
        Subgraph_SwTop.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_SwTop.LineWidth = 2;
        Subgraph_SwTop.PrimaryColor = RGB(255,255,0);
        Subgraph_SwTop.SecondaryColor = RGB(0,255,0);
        Subgraph_SwTop.SecondaryColorUsed = 1; // true
        Subgraph_SwTop.DrawZeros = false;

        Subgraph_SwBottom.Name = "SW Bottom";
        Subgraph_SwBottom.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_SwBottom.LineWidth = 2;
        Subgraph_SwBottom.PrimaryColor = RGB(255,255,0);
        Subgraph_SwBottom.SecondaryColor = RGB(0,255,0);
        Subgraph_SwBottom.SecondaryColorUsed = 1; // true
        Subgraph_SwBottom.DrawZeros = false;
    }
}

```

```

Subgraph_FlatColor.Name = "Flat Color";
Subgraph_FlatColor.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_FlatColor.PrimaryColor = RGB(255,0,0);
Subgraph_FlatColor.DrawZeros = false;

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

Subgraph_Lsma.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_Lsma.DrawZeros = false;

Subgraph_LsmaAngle.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_LsmaAngle.DrawZeros = false;

Subgraph_Ema.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_Ema.DrawZeros = false;

Subgraph_EmaAngle.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_EmaAngle.DrawZeros = false;

Input_LsmaLength.Name = "LSMA Length";
Input_LsmaLength.SetInt(25);
Input_LsmaLength.SetIntLimits(1,MAX_STUDY_LENGTH);

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

Input_TrendingThreshold.Name = "Trending Threshold";
Input_TrendingThreshold.SetFloat(100.0f);

Input_NormalThreshold.Name = "Normal Threshold";
Input_NormalThreshold.SetFloat(30.0f);

Input_LineValue.Name = "Line Value";
Input_LineValue.SetFloat(200.0f);

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

sc.AutoLoop = 1;

return;
}

Subgraph_OutWorksheet.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_OutWorksheet.PrimaryColor = RGB(255,255,255);

const float RAD_TO_DEG = (180.0f/3.14159265358979f);

// Colors
uint32_t colorFlat = Subgraph_FlatColor.PrimaryColor;
uint32_t colorNormalUp = Subgraph_SwTop.PrimaryColor;
uint32_t colorTrendingUp = Subgraph_SwTop.SecondaryColor;
uint32_t colorNormalDown = Subgraph_SwBottom.PrimaryColor;
uint32_t colorTrendingDown = Subgraph_SwBottom.SecondaryColor;

if (Input_TickValueOverride.GetFloat() == 1)
    Input_TickValueOverride.SetFloat(0);

float inTickSize = sc.TickSize;

if(Input_TickValueOverride.GetFloat() != 0)

```

```

inTickSize = Input_TickValueOverride.GetFloat();

float &PriorTickSize = sc.GetPersistentFloat(1);

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

float InverseTickSize = 1/inTickSize;

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

// Determine the input array element to begin calculation at
int Pos = sc.Index;
if (Pos < 2)
    return;

// Calculate the EMA
sc.ExponentialMovAvg(sc.Close, Subgraph_Ema, Pos, Input_EmaLength.GetInt());

// Calculate the LSMA
sc.LinearRegressionIndicator(sc.Close, Subgraph_Lsma, Pos, Input_LsmaLength.GetInt());

// Calculate the LSMA angle
float lsma_Angle = static_cast<float>(sc.SlopeToAngleInDegrees((Subgraph_Lsma[Pos]-
((Subgraph_Lsma[Pos-1]+Subgraph_Lsma[Pos-2])/2.0f))*InverseTickSize));

Subgraph_LsmaAngle[Pos] = lsma_Angle;

// Calculate the EMA angle
float ema_Angle = static_cast<float>(sc.SlopeToAngleInDegrees((Subgraph_Ema[Pos]-
((Subgraph_Ema[Pos-1]+Subgraph_Ema[Pos-2])/2.0f))*InverseTickSize));

Subgraph_EmaAngle[Pos] = ema_Angle;

// Calculate SW
uint32_t color = 0;
float worksheetValue = 0;

if (fabs(Subgraph_EmaAngle[Pos]) >= 15.0f
    && (Subgraph_EmaAngle[Pos] + Subgraph_LsmaAngle[Pos]) >= Input_TrendingThreshold.GetFloat())
{
    color = colorTrendingUp;
    worksheetValue = 2;
}
else if (fabs(Subgraph_EmaAngle[Pos]) >= 15.0f
    && (Subgraph_EmaAngle[Pos] + Subgraph_LsmaAngle[Pos]) <= -Input_TrendingThreshold.GetFloat())
{
    color = colorTrendingDown;
    worksheetValue = -2;
}
else if (Subgraph_EmaAngle[Pos] > 5.0f && Subgraph_LsmaAngle[Pos] > 0.0f
    && (Subgraph_EmaAngle[Pos] + Subgraph_LsmaAngle[Pos]) >= Input_NormalThreshold.GetFloat())
{
    color = colorNormalUp;
    worksheetValue = 1;
}
else if (Subgraph_EmaAngle[Pos] < 5.0f && Subgraph_LsmaAngle[Pos] < 0.0f
    && (Subgraph_EmaAngle[Pos] + Subgraph_LsmaAngle[Pos]) <= -Input_NormalThreshold.GetFloat())
{
    color = colorNormalDown;

```

```

        worksheetValue = -1;
    }
    else
    {
        color = colorFlat;
        worksheetValue = 0;
    }

    Subgraph_SwTop[Pos] = Input_LineValue.GetFloat();
    Subgraph_SwBottom[Pos] = -Input_LineValue.GetFloat();

    Subgraph_SwTop.DataColor[Pos] = color;
    Subgraph_SwBottom.DataColor[Pos] = color;

    Subgraph_OutWorksheet[Pos] = worksheetValue;
}

/*=====*/
SCSFExport scsf_LSMAAboveBelowCCI(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_LsmaCCI = sc.Subgraph[0];
    SCSubgraphRef Subgraph_DoNotDraw1 = sc.Subgraph[1];
    SCSubgraphRef Subgraph_DoNotDraw2 = sc.Subgraph[2];
    SCSubgraphRef Subgraph_DoNotDraw3 = sc.Subgraph[3];
    SCSubgraphRef Subgraph_DoNotDraw4 = sc.Subgraph[4];

    SCInputRef Input_LsmaLength = sc.Input[0];
    SCInputRef Input_CCILength = sc.Input[2];
    SCInputRef Input_BarsPointsLsma = sc.Input[3];

    if (sc.SetDefaults)
    {
        // Set the configuration and defaults
        sc.GraphName = "LSMA Above/Below CCI";
        sc.GraphRegion = 1;

        Subgraph_LsmaCCI.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_LsmaCCI.Name = "LSMA Above/Below CCI";
        Subgraph_LsmaCCI.PrimaryColor = RGB(0, 255, 0);
        Subgraph_LsmaCCI.SecondaryColorUsed = 1;
        Subgraph_LsmaCCI.SecondaryColor = RGB(255, 0, 255);
        Subgraph_LsmaCCI.DrawZeros = false;

        Subgraph_DoNotDraw4.DrawStyle = DRAWSTYLE_IGNORE;
        Subgraph_DoNotDraw4.Name = "Spreadsheet Output";
        Subgraph_DoNotDraw4.PrimaryColor = RGB(255,0,0);
        Subgraph_DoNotDraw4.DrawZeros = false;

        Input_LsmaLength.Name = "MA Length";
        Input_LsmaLength.SetInt(25);
        Input_LsmaLength.SetIntLimits(1,MAX_STUDY_LENGTH);

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

        Input_BarsPointsLsma.Name = "Bars (0) or Points (n/1000) or LSMA (9999)";
        Input_BarsPointsLsma.SetInt(0);
        Input_BarsPointsLsma.SetIntLimits(0, 9999);

        sc.AutoLoop = 1;

        return;
    }
}

```

```

}

uint32_t Color0 = Subgraph_LsmaCCI.PrimaryColor;
uint32_t Color1 = Subgraph_LsmaCCI.SecondaryColor;

sc.DataStartIndex = max(Input_LsmaLength.GetInt(), Input_CCILength.GetInt());

int MAType = MOVAVGTYPE_LINEARREGRESSION;

sc.MovingAverage(sc.Close, Subgraph_DoNotDraw1, MAType, sc.Index, Input_LsmaLength.GetInt());
sc.CCI(sc.HLCAvg, Subgraph_DoNotDraw2, Subgraph_DoNotDraw3, sc.Index, Input_CCILength.GetInt(), 0.015f);

float fLSMA = Subgraph_DoNotDraw1[sc.Index];
float fCCI = Subgraph_DoNotDraw3[sc.Index];

if (Input_BarsPointsLsma.GetInt() == 0)
    Subgraph_LsmaCCI[sc.Index] = fCCI;
else if (Input_BarsPointsLsma.GetInt() == 9999)
    Subgraph_LsmaCCI[sc.Index] = fLSMA;
else
    Subgraph_LsmaCCI[sc.Index] = static_cast<float>(Input_BarsPointsLsma.GetInt()) / 1000.0f;

if (sc.Close[sc.Index] > fLSMA)
{
    Subgraph_LsmaCCI.DataColor[sc.Index] = Color0;
    Subgraph_DoNotDraw4[sc.Index] = 1;
}
else if (sc.Close[sc.Index] < fLSMA)
{
    Subgraph_LsmaCCI.DataColor[sc.Index] = Color1;
    Subgraph_DoNotDraw4[sc.Index] = -1;
}
}

/*=====*/
SCSFExport scsf_LSMAAboveBelow(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Lsma = sc.Subgraph[0];
    SCSubgraphRef Subgraph_DoNotDraw1 = sc.Subgraph[1];
    SCSubgraphRef Subgraph_WorksheetOutput = sc.Subgraph[2];

    SCInputRef Input_LsmaLength = sc.Input[0];

    if (sc.SetDefaults)
    {
        // Set the configuration and defaults
        sc.GraphName = "LSMA Above/Below Last";
        sc.GraphRegion = 1;

        Subgraph_Lsma.DrawStyle = DRAWSTYLE_SQUARE;
        Subgraph_Lsma.LineWidth = 2;
        Subgraph_Lsma.Name = "Above/Below";
        Subgraph_Lsma.PrimaryColor = RGB(0, 255, 0);
        Subgraph_Lsma.SecondaryColorUsed = 1;
        Subgraph_Lsma.SecondaryColor = RGB(255, 0, 0);
        Subgraph_Lsma.DrawZeros = true;

        Subgraph_WorksheetOutput.Name = "Spreadsheet Output";
        Subgraph_WorksheetOutput.DrawStyle = DRAWSTYLE_IGNORE;
        Subgraph_WorksheetOutput.PrimaryColor = RGB(255, 255, 0);
        Subgraph_WorksheetOutput.DrawZeros = true;

        Input_LsmaLength.Name = "MovAvg Length";
        Input_LsmaLength.SetInt(25);
    }
}

```

```

    Input_LsmaLength.SetIntLimits(1,MAX_STUDY_LENGTH);

    sc.AutoLoop = 1;

    return;
}

uint32_t color0 = Subgraph_Lsma.PrimaryColor;
uint32_t color1 = Subgraph_Lsma.SecondaryColor;

sc.DataStartIndex = Input_LsmaLength.GetInt();

int index = sc.Index;

    sc.MovingAverage(sc.Close, Subgraph_DoNotDraw1, MOVAVGTYPE_LINEARREGRESSION, index,
Input_LsmaLength.GetInt());

    float fLSMA = Subgraph_DoNotDraw1[index];

    if (sc.Close[index] > fLSMA)
    {
        Subgraph_Lsma.DataColor[index] = color0;
        Subgraph_WorksheetOutput[index] = 1;

    }
    else if (sc.Close[index] < fLSMA)
    {
        Subgraph_Lsma.DataColor[index] = color1;
        Subgraph_WorksheetOutput[index] = -1;

    }
    else
    {
        Subgraph_Lsma.DataColor[index] = 0;
        Subgraph_WorksheetOutput[index] = 0;

    }
}
/*=====*/
SCSFExport scsf_EMAAboveBelow(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_ExponentialMovingAverage = sc.Subgraph[0];
    SCSubgraphRef Subgraph_DoNotDraw1 = sc.Subgraph[1];
    SCSubgraphRef Subgraph_WorksheetOutput = sc.Subgraph[2];

    SCInputRef Subgraph_MovingAverageLength = sc.Input[0];

    if (sc.SetDefaults)
    {
        // Set the configuration and defaults
        sc.GraphName = "EMA Above/Below Last";
        sc.GraphRegion = 1;

        Subgraph_ExponentialMovingAverage.DrawStyle = DRAWSTYLE_SQUARE;
        Subgraph_ExponentialMovingAverage.LineWidth = 2;
        Subgraph_ExponentialMovingAverage.Name = "Above/Below";
        Subgraph_ExponentialMovingAverage.PrimaryColor = RGB(0, 255, 0);
        Subgraph_ExponentialMovingAverage.SecondaryColorUsed = 1;
        Subgraph_ExponentialMovingAverage.SecondaryColor = RGB(255, 0, 0);
        Subgraph_ExponentialMovingAverage.DrawZeros = true;

        Subgraph_WorksheetOutput.Name = "Spreadsheet Output";
        Subgraph_WorksheetOutput.DrawStyle= DRAWSTYLE_IGNORE;
        Subgraph_WorksheetOutput.PrimaryColor = RGB(255, 255, 0);
        Subgraph_WorksheetOutput.DrawZeros = true;
    }
}

```

```

Subgraph_MovingAverageLength.Name = "Moving Average Length";
Subgraph_MovingAverageLength.SetInt(25);
Subgraph_MovingAverageLength.SetIntLimits(1,MAX_STUDY_LENGTH);

sc.AutoLoop = 1;

return;
}

uint32_t color0 = Subgraph_ExponentialMovingAverage.PrimaryColor;
uint32_t color1 = Subgraph_ExponentialMovingAverage.SecondaryColor;

sc.DataStartIndex = Subgraph_MovingAverageLength.GetInt();

sc.MovingAverage(sc.Close, Subgraph_DoNotDraw1, MOVAVGTYPE_EXPONENTIAL, sc.Index,
Subgraph_MovingAverageLength.GetInt());

if (sc.Close[sc.Index] > Subgraph_DoNotDraw1[sc.Index])
{
    Subgraph_ExponentialMovingAverage.DataColor[sc.Index] = color0;
    Subgraph_WorksheetOutput[sc.Index] = 1;
}
else if (sc.Close[sc.Index] < Subgraph_DoNotDraw1[sc.Index])
{
    Subgraph_ExponentialMovingAverage.DataColor[sc.Index] = color1;
    Subgraph_WorksheetOutput[sc.Index] = -1;
}
else
{
    Subgraph_ExponentialMovingAverage.DataColor[sc.Index] = 0;
    Subgraph_WorksheetOutput[sc.Index] = 0;
}
}

/*=====*/
SCSFExport scsf_MurreyMath(SCStudyInterfaceRef sc)
{
    const int NumberOfVisibleLines = 25;

    SCInputRef Input_SquareWidth = sc.Input[0];
    SCInputRef Input_UseLastBarAsEndDateTime = sc.Input[1];
    SCInputRef Input_EndDateTime = sc.Input[2];
    SCInputRef Input_SquareWidthNew = sc.Input[3];
    SCInputRef Input_Version = sc.Input[4];
    SCInputRef Input_SquareWidthMultiplier = sc.Input[5];
    SCInputRef Input_RoundToTickSize = sc.Input[6];
    SCInputRef Input_CustomSquareWidth = sc.Input[7];

    // Configuration
    if (sc.SetDefaults)
    {
        sc.GraphName = "Murrey Math";
        sc.AutoLoop = 0;
        sc.GraphRegion = 0;
        sc.ScaleRangeType = SCALE_SAMEASREGION;

        for (int SubgraphIndex = 0; SubgraphIndex < NumberOfVisibleLines; SubgraphIndex++)
        {
            SCString SubgraphString;

            if(SubgraphIndex <= 16)
                SubgraphString.Format("%d/8", SubgraphIndex-8);
        }
    }
}

```

```

else
    SubgraphString.Format("+%d/8", SubgraphIndex-16);

sc.Subgraph[SubgraphIndex].Name = SubgraphString;
sc.Subgraph[SubgraphIndex].DrawStyle = DRAWSTYLE_DASH;
sc.Subgraph[SubgraphIndex].LineWidth = 1;

sc.Subgraph[SubgraphIndex].PrimaryColor = RGB(0, 255, 0);
sc.Subgraph[SubgraphIndex].LineLabel |= LL_NAME_ALIGN_LEFT | LL_DISPLAY_NAME |
LL_NAME_ALIGN_CENTER;

}

//SquareWidth.Name = "Square Width";
Input_SquareWidth.SetInt(64);
Input_SquareWidth.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_SquareWidthNew.Name = "Square Width";
Input_SquareWidthNew.SetCustomInputStrings("4;8;16;32;64;128;256;512");
Input_SquareWidthNew.SetCustomInputIndex(0);

Input_UseLastBarAsEndDateTime.Name = "Use Last Bar as End Date-Time";
Input_UseLastBarAsEndDateTime.SetYesNo(true);

Input_EndDateTime.Name = "End Date-Time";
Input_EndDateTime.SetDateTime(0);

Input_Version.SetInt(1);

Input_SquareWidthMultiplier.Name = "Square Width Multiplier";
Input_SquareWidthMultiplier.SetFloat(1.5);
Input_SquareWidthMultiplier.SetFloatLimits(1.0, 100);

Input_RoundToTickSize.Name = "Round to Tick Size";
Input_RoundToTickSize.SetYesNo(false);

Input_CustomSquareWidth.Name = "Custom Square Width";
Input_CustomSquareWidth.SetInt(0);
Input_CustomSquareWidth.SetIntLimits(0, MAX_STUDY_LENGTH);

return;
}

if(Input_Version.GetInt() < 1)
{
    Input_Version.SetInt(1);
    int OldSquareWidth = Input_SquareWidth.GetInt();

    switch(OldSquareWidth)
    {
        case 4:
            Input_SquareWidthNew.SetCustomInputIndex(0);
            break;

        case 8:
            Input_SquareWidthNew.SetCustomInputIndex(1);
            break;

        case 16:
            Input_SquareWidthNew.SetCustomInputIndex(2);
            break;

        case 32:

```

```

        Input_SquareWidthNew.SetCustomInputIndex(3);
    break;

    default:
    case 64:
        Input_SquareWidthNew.SetCustomInputIndex(4);
    break;

    case 128:
        Input_SquareWidthNew.SetCustomInputIndex(5);
    break;

    case 256:
        Input_SquareWidthNew.SetCustomInputIndex(6);
    break;

    case 512:
        Input_SquareWidthNew.SetCustomInputIndex(7);
    break;
}
}

int SquareWidthNumber = atoi(Input_SquareWidthNew.GetSelectedCustomString());

if (Input_CustomSquareWidth.GetInt() > 0)
    SquareWidthNumber = Input_CustomSquareWidth.GetInt();

int BarsBackToReference = sc.Round(SquareWidthNumber * Input_SquareWidthMultiplier.GetFloat());

int &PreviousStartOutputBarIndex = sc.GetPersistentInt(1);

//Clear previously used elements
for(int BarIndex = PreviousStartOutputBarIndex; BarIndex < sc.ArraySize; BarIndex++)
{
    for (int LineIndex = 0; LineIndex < NumberOfVisibleLines; LineIndex++)
    {
        sc.Subgraph[LineIndex][BarIndex] = 0;
    }
}

//Determine the starting index
if (Input_UseLastBarAsEndDateTime.GetYesNo())
    PreviousStartOutputBarIndex = sc.ArraySize - BarsBackToReference;
else
{
    PreviousStartOutputBarIndex = sc.GetNearestMatchForSCDateTime(sc.ChartNumber,
Input_EndDateTime.GetDateTime().GetAsDouble()) - BarsBackToReference + 1;
}

double HighestHighOfSquareWidthLength = static_cast<double>(sc.GetHighest(sc.High, PreviousStartOutputBarIndex
+ BarsBackToReference - 1, BarsBackToReference));

double LowestLowOfSquareWidthLength = static_cast<double>(sc.GetLowest(sc.Low, PreviousStartOutputBarIndex +
BarsBackToReference - 1, BarsBackToReference));

double PriceRange = HighestHighOfSquareWidthLength - LowestLowOfSquareWidthLength;
double SR = 0.0, OctaveCount = 0.0, Si = 0.0, M = 0.0, B = 0.0, TT = 0.0;

int I = 0;

if(HighestHighOfSquareWidthLength > 25)
{
    if(log(0.4*HighestHighOfSquareWidthLength)/log(10.0)-int(log(0.4*HighestHighOfSquareWidthLength)/log(10.0))>0)

```

```

        SR = exp(log(10.0)*(int(log(0.4*HighestHighOfSquareWidthLength)/log(10.0))+1));
    else
        SR = exp(log(10.0)*(int(log(0.4*HighestHighOfSquareWidthLength)/log(10.0))));
    }
    else
    {
        SR = 100*exp(log(8.0)*(int(log(0.005*HighestHighOfSquareWidthLength)/log(8.0))));
    }

    double RangeMMI = log(SR/PriceRange)/log(8.0);
    if(RangeMMI<=0)
    {
        OctaveCount = 0;
    }
    else
    {
        if(log(SR/PriceRange)/log(8.0)- fmod(log(SR/PriceRange),log(8.0))== 0)
        {
            OctaveCount = int(RangeMMI);
        }
        else
        {
            OctaveCount = int(RangeMMI)+1.0;
        }
    }
}

Si = SR *exp(-OctaveCount * log(8.0));

if(PriceRange == 0)
    M = 0.0f;
else
    M = int(((1 / log(2.0)) * log(fabs(PriceRange / Si))) + .00001);

double ValueToRound = (((HighestHighOfSquareWidthLength + LowestLowOfSquareWidthLength) * .5)/(Si*
exp((M-1)*log(2.0))));
I = sc.Round(static_cast<float>(ValueToRound));

B =(I-1) * Si * exp((M-1)* log(2.0));
TT =(I+1) * Si * exp((M-1) * log(2.0));

for (int LineIndex = 0; LineIndex < NumberOfVisibleLines; LineIndex++)
{
    for(int BarIndex = PreviousStartOutputBarIndex; BarIndex < sc.ArraySize; BarIndex++)
    {
        float MurreyMathValue = static_cast<float>(B + ((LineIndex-8) * 0.125 * (TT -B)));

        if (Input_RoundToTickSize.GetYesNo())
            MurreyMathValue = sc.RoundToTickSize(MurreyMathValue);

        sc.Subgraph[LineIndex][BarIndex] = MurreyMathValue;
    }
}
}

/*=====*/
SCSFExport scsf_PreviousCloseLine(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_PreviousCloseLine = sc.Subgraph[0];
    SCInputRef Input_UsePreviousCloseFromQuoteData = sc.Input[0];

    if(sc.SetDefaults)

```

```

{
    sc.GraphName = "Previous Close Line";

    Subgraph_PreviousCloseLine.Name = "PCL";
    Subgraph_PreviousCloseLine.DrawStyle = DRAWSTYLE_LINE;

    Subgraph_PreviousCloseLine.LineStyle = LINESTYLE_DOT;
    Subgraph_PreviousCloseLine.PrimaryColor = RGB(0, 255, 0);
    Subgraph_PreviousCloseLine.LineWidth = 2;
    Subgraph_PreviousCloseLine.DrawZeros = false;

    Input_UsePreviousCloseFromQuoteData.Name = "Use Previous Close From Quote Data";
    Input_UsePreviousCloseFromQuoteData.SetYesNo(0);

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

    return;
}

SCDateTime PriorDayCloseDateTime =
sc.GetStartDateTimeForTradingDate(sc.GetTradingDayDate(sc.BaseDateTimeln[sc.ArraySize - 1]));

PriorDayCloseDateTime.SubtractSeconds(1);

int PriorDayCloseDateTimeIndex = sc.GetContainingIndexForSCDateTime(sc.ChartNumber,
PriorDayCloseDateTime.GetAsDouble());

float PriorDayCloseValue = 0.0;
if (Input_UsePreviousCloseFromQuoteData.GetYesNo())
    PriorDayCloseValue = sc.PreviousClose;
else
    PriorDayCloseValue = sc.BaseData[SC_LAST][PriorDayCloseDateTimeIndex];

for (int BarIndex = sc.UpdateStartIndex ; BarIndex < sc.ArraySize; ++BarIndex)
{
    if (BarIndex < PriorDayCloseDateTimeIndex)
        continue;

    Subgraph_PreviousCloseLine[BarIndex] = PriorDayCloseValue;
}

}

/*=====*/
SCSFExport scsf_ValueChartLevels(SCStudyInterfaceRef sc)
{
    SCInputRef Input_ModOBLevel = sc.Input[1];
    SCInputRef Input_SigOBLevel = sc.Input[2];

    SCSubgraphRef Subgraph_PlusModerateOverboughtLevel = sc.Subgraph[0];
    SCSubgraphRef Subgraph_PlusSignificantOverboughtLevel = sc.Subgraph[1];
    SCSubgraphRef Subgraph_ZeroLevel = sc.Subgraph[2];
    SCSubgraphRef Subgraph_MinusModerateOverboughtLevel = sc.Subgraph[3];
    SCSubgraphRef Subgraph_MinusSignificantOverboughtLevel = sc.Subgraph[4];

    if (sc.SetDefaults)
    {
        // Set the configuration and defaults
        sc.GraphName = "Value Chart Levels";

        sc.GraphDrawType = GDT_CUSTOM;
        sc.AutoLoop = 1;
    }
}

```

```

Input_ModOBLevel.Name = "Moderately Overbought Level";
Input_ModOBLevel.SetFloat(4.0f);
Input_ModOBLevel.SetFloatLimits(0,100.0f);

Input_SigOBLevel.Name = "Significantly Overbought Level";
Input_SigOBLevel.SetFloat(8.0f);
Input_SigOBLevel.SetFloatLimits(0,100.0f);

Subgraph_PlusModerateOverboughtLevel.Name = "+ Moderate Overbought Level";
Subgraph_PlusModerateOverboughtLevel.PrimaryColor = RGB(0,255,0);
Subgraph_PlusModerateOverboughtLevel.DrawStyle = DRAWSTYLE_LINE;
Subgraph_PlusModerateOverboughtLevel.DisplayNameValueInWindowsFlags= 0;

Subgraph_PlusSignificantOverboughtLevel.Name = "+ Significant Overbought Level";
Subgraph_PlusSignificantOverboughtLevel.PrimaryColor = RGB(255,0,0);
Subgraph_PlusSignificantOverboughtLevel.DrawStyle = DRAWSTYLE_LINE;
Subgraph_PlusSignificantOverboughtLevel.DisplayNameValueInWindowsFlags= 0;

Subgraph_MinusModerateOverboughtLevel.Name = "- Moderate Overbought Level";
Subgraph_MinusModerateOverboughtLevel.PrimaryColor = RGB(0,255,0);
Subgraph_MinusModerateOverboughtLevel.DrawStyle = DRAWSTYLE_LINE;
Subgraph_MinusModerateOverboughtLevel.DisplayNameValueInWindowsFlags= 0;

Subgraph_MinusSignificantOverboughtLevel.Name = "- Significant Overbought Level";
Subgraph_MinusSignificantOverboughtLevel.PrimaryColor = RGB(255,0,0);
Subgraph_MinusSignificantOverboughtLevel.DrawStyle = DRAWSTYLE_LINE;
Subgraph_MinusSignificantOverboughtLevel.DisplayNameValueInWindowsFlags= 0;

Subgraph_ZeroLevel.Name = "Zero Level";
Subgraph_ZeroLevel.PrimaryColor = RGB(128,128,128);
Subgraph_ZeroLevel.DrawStyle = DRAWSTYLE_LINE;
Subgraph_ZeroLevel.DisplayNameValueInWindowsFlags= 0;
Subgraph_ZeroLevel.DrawZeros = 1;

}

Subgraph_PlusModerateOverboughtLevel[sc.Index] = Input_ModOBLevel.GetFloat();
Subgraph_PlusSignificantOverboughtLevel[sc.Index] = Input_SigOBLevel.GetFloat();

Subgraph_MinusModerateOverboughtLevel[sc.Index] = Input_ModOBLevel.GetFloat() * -1.0f;
Subgraph_MinusSignificantOverboughtLevel[sc.Index] = Input_SigOBLevel.GetFloat() * -1.0f;

}

/*=====*/
SCSFExport scsf_ValueChart(SCStudyInterfaceRef sc)
{
    SCInputRef Input_Length    = sc.Input[0];

    SCSubgraphRef Subgraph_VcOpen  = sc.Subgraph[0];
    SCSubgraphRef Subgraph_VcHigh  = sc.Subgraph[1];
    SCSubgraphRef Subgraph_VcLow   = sc.Subgraph[2];
    SCSubgraphRef Subgraph_VcClose = sc.Subgraph[3];

    SCFloatArrayRef Array_VcHLMA = Subgraph_VcOpen.Arrays[1];
    SCFloatArrayRef Array_TR      = Subgraph_VcOpen.Arrays[2];
    SCFloatArrayRef Array_ATR     = Subgraph_VcOpen.Arrays[3];
    SCFloatArrayRef Array_HH      = Subgraph_VcOpen.Arrays[4];
    SCFloatArrayRef Array_LL      = Subgraph_VcOpen.Arrays[5];

```

```

if (sc.SetDefaults)
{
    // Set the configuration and defaults
    sc.GraphName = "Value Chart";

    sc.GraphDrawType = GDT_CANDLESTICK;
    sc.AutoLoop = 1;

    // Inputs
    Input_Length.Name = "Length";
    Input_Length.SetInt(5);
    Input_Length.SetIntLimits(2,10000);

    // Subgraphs
    Subgraph_VcOpen.Name = "VC Open";
    Subgraph_VcOpen.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_VcOpen.PrimaryColor = RGB(0,255,0);
    Subgraph_VcOpen.SecondaryColorUsed = true;
    Subgraph_VcOpen.SecondaryColor = RGB(0,255,0);
    Subgraph_VcOpen.DrawZeros = 1;

    Subgraph_VcHigh.Name = "VC High";
    Subgraph_VcHigh.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_VcHigh.PrimaryColor = RGB(128,255,128);
    Subgraph_VcHigh.DrawZeros = 1;

    Subgraph_VcLow.Name = "VC Low";
    Subgraph_VcLow.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_VcLow.PrimaryColor = RGB(255,0,0);
    Subgraph_VcLow.SecondaryColor = RGB(255,0,0);
    Subgraph_VcLow.SecondaryColorUsed = true;
    Subgraph_VcLow.DrawZeros = 1;

    Subgraph_VcClose.Name = "VC Close";
    Subgraph_VcClose.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_VcClose.PrimaryColor = RGB(255,128,128);
    Subgraph_VcClose.DrawZeros = 1;

    return;
}

int NumBars = Input_Length.GetInt();
float LRange = 1;

if (NumBars > 7)
{
    int VarNumBars = sc.Round(static_cast<float>(NumBars) / 5.0f);

    sc.Highest(sc.High, Array_HH, VarNumBars);
    sc.Lowest(sc.Low, Array_LL, VarNumBars);

    float R1 = Array_HH[sc.Index] - Array_LL[sc.Index];
    if (R1 == 0 && VarNumBars == 1)
        R1 = abs(sc.Close[sc.Index] - sc.Close[sc.Index-VarNumBars]);

    float R2 = Array_HH[sc.Index-VarNumBars-1] - Array_LL[sc.Index-VarNumBars];
    if (R2 == 0 && VarNumBars == 1)
        R2 = abs(sc.Close[sc.Index-VarNumBars] - sc.Close[sc.Index-(VarNumBars*2)]);

    float R3 = Array_HH[sc.Index-(VarNumBars*2)] - Array_LL[sc.Index-(VarNumBars*2)];
    if (R3 == 0 && VarNumBars == 1)

```

```

R3 = abs(sc.Close[sc.Index-(VarNumBars*2)] - sc.Close[sc.Index-(VarNumBars*3)]);

float R4 = Array_HH[sc.Index-(VarNumBars*3)] - Array_LL[sc.Index-(VarNumBars*3)];
if (R4 == 0 && VarNumBars == 1)
    R4 = abs(sc.Close[sc.Index-(VarNumBars*3)] - sc.Close[sc.Index-(VarNumBars*4)]);

float R5 = Array_HH[sc.Index-(VarNumBars*4)] - Array_LL[sc.Index-(VarNumBars*4)];
if (R5 == 0 && VarNumBars == 1)
    R5 = abs(sc.Close[sc.Index-(VarNumBars*4)] - sc.Close[sc.Index-(VarNumBars*5)]);

LRange = (R1 + R2 + R3 + R4 + R5) / 5 / 5;
}
else
{
    sc.ATR(sc.BaseDataIn, Array_TR, Array_ATR, 5, MOVAVGTYPE_SIMPLE);
    LRange = Array_ATR[sc.Index] / 5;
}

sc.MovingAverage(sc.HLAv, Array_VcHLMA, MOVAVGTYPE_SIMPLE, Input_Length.GetInt());
float AvgVcHL = Array_VcHLMA[sc.Index];

Subgraph_VcOpen[sc.Index] = (sc.Open[sc.Index] - AvgVcHL) / LRange;
Subgraph_VcHigh[sc.Index] = (sc.High[sc.Index] - AvgVcHL) / LRange;
Subgraph_VcLow[sc.Index] = (sc.Low[sc.Index] - AvgVcHL) / LRange;
Subgraph_VcClose[sc.Index] = (sc.Close[sc.Index] - AvgVcHL) / LRange;
}

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

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_Length = sc.Input[1];
    SCInputRef Input_Multiplier = sc.Input[2];
    SCInputRef Input_GDEMACount = sc.Input[3];

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

        sc.GraphName = "Moving Average - Generalized Double Exponential";
        sc.GraphRegion = 0;

        sc.StudyDescription = "Moving Average - Generalized Double Moving Exponential";

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

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

        Input_Length.Name = "Length";
        Input_Length.SetInt(10);
        Input_Length.SetIntLimits(1, MAX_STUDY_LENGTH);

        Input_Multiplier.Name = "Multiplier";
        Input_Multiplier.SetFloat(1.0f);

        Input_GDEMACount.Name = "GDEMA Count";
        Input_GDEMACount.SetInt(1);
        Input_GDEMACount.SetIntLimits(1, SC_SUBGRAPHS_AVAILABLE - 1);
    }
}

```

```

    sc.AutoLoop = 1;

    return;
}

int Length = Input_Length.GetInt();
float Multiplier = Input_Multiplier.GetFloat();
int Count = Input_GDEMACount.GetInt();

sc.DataStartIndex = Length * Count;

if (sc.Index == 0)
{
    for (int SubgraphIndex = 1; SubgraphIndex <= Count; SubgraphIndex++)
    {
        sc.Subgraph[SubgraphIndex].Name.Format("%d", SubgraphIndex);
        sc.Subgraph[SubgraphIndex].DrawStyle = DRAWSTYLE_IGNORE;
        sc.Subgraph[SubgraphIndex].DisplayNameValueInWindowsFlags = 0;
    }
}

sc.ExponentialMovAvg(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], sc.Subgraph[1].Arrays[0], Length);
sc.ExponentialMovAvg(sc.Subgraph[1].Arrays[0], sc.Subgraph[1].Arrays[1], Length);
sc.Subgraph[1][sc.Index] = (1 + Multiplier) * sc.Subgraph[1].Arrays[0][sc.Index] - Multiplier * sc.Subgraph[1].Arrays[1][sc.Index];

for (int SubgraphIndex = 2; SubgraphIndex <= Count; SubgraphIndex++)
{
    sc.ExponentialMovAvg(sc.Subgraph[SubgraphIndex - 1], sc.Subgraph[SubgraphIndex].Arrays[0], Length);
    sc.ExponentialMovAvg(sc.Subgraph[SubgraphIndex].Arrays[0], sc.Subgraph[SubgraphIndex].Arrays[1], Length);
    sc.Subgraph[SubgraphIndex][sc.Index] = (1 + Multiplier) * sc.Subgraph[SubgraphIndex].Arrays[0][sc.Index] -
Multiplier * sc.Subgraph[SubgraphIndex].Arrays[1][sc.Index];
}

Subgraph_GDEMA[sc.Index] = sc.Subgraph[Count][sc.Index];
}

/*=====*/
SCSFExport scsf_SchaffTrendCycle(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_SchaffTrendCycle = sc.Subgraph[0];
    SCSubgraphRef Subgraph_MACD = sc.Subgraph[1];
    SCSubgraphRef Subgraph_FastKMACD = sc.Subgraph[2];
    SCSubgraphRef Subgraph_FastDMACD = sc.Subgraph[3];
    SCSubgraphRef Subgraph_FastKFastDMACD = sc.Subgraph[4];
    SCSubgraphRef Subgraph_Overbought = sc.Subgraph[5];
    SCSubgraphRef Subgraph_Oversold = sc.Subgraph[6];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_FastLength = sc.Input[2];
    SCInputRef Input_SlowLength = sc.Input[3];
    SCInputRef Input_MovingAverageType = sc.Input[4];
    SCInputRef Input_Length = sc.Input[5];
    SCInputRef Input_Multiplier = sc.Input[6];
    SCInputRef Input_Overbought = sc.Input[7];
    SCInputRef Input_Oversold = sc.Input[8];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Schaff Trend Cycle";
        sc.AutoLoop = 1;
        sc.GraphRegion = 1;

        Subgraph_SchaffTrendCycle.Name = "Schaff Trend Cycle";
    }
}

```

```

Subgraph_SchaffTrendCycle.DrawStyle = DRAWSTYLE_LINE;
Subgraph_SchaffTrendCycle.DrawZeros = true;
Subgraph_SchaffTrendCycle.PrimaryColor = RGB(0, 0, 255);

Subgraph_MACD.Name = "MACD";
Subgraph_MACD.DrawStyle = DRAWSTYLE_IGNORE;

Subgraph_FastKMACD.Name = "Fast %K of MACD";
Subgraph_FastKMACD.DrawStyle = DRAWSTYLE_IGNORE;

Subgraph_FastDMACD.Name = "Fast %D of MACD";
Subgraph_FastDMACD.DrawStyle = DRAWSTYLE_IGNORE;

Subgraph_FastKFastDMACD.Name = "Fast %K of Fast %D of MACD";
Subgraph_FastKFastDMACD.DrawStyle = DRAWSTYLE_IGNORE;

Subgraph_Overbought.Name = "Overbought Line";
Subgraph_Overbought.DrawStyle = DRAWSTYLE_LINE;
Subgraph_Overbought.DrawZeros = true;
Subgraph_Overbought.PrimaryColor = RGB(0, 255, 0);

Subgraph_Oversold.Name = "Oversold Line";
Subgraph_Oversold.DrawStyle = DRAWSTYLE_LINE;
Subgraph_Oversold.DrawZeros = true;
Subgraph_Oversold.PrimaryColor = RGB(255, 0, 0);

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

Input_FastLength.Name = "Fast Moving Average Length";
Input_FastLength.SetInt(23);
Input_FastLength.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_SlowLength.Name = "Slow Moving Average Length";
Input_SlowLength.SetInt(50);
Input_SlowLength.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_MovingAverageType.Name = "Moving Average Type";
Input_MovingAverageType.SetMovAvgType(MOVAVGTYPE_EXPONENTIAL);

Input_Length.Name = "Length";
Input_Length.SetInt(10);
Input_Length.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_Multiplier.Name = "Multiplier";
Input_Multiplier.SetFloat(0.5f);

Input_Overbought.Name = "Overbought Level";
Input_Overbought.SetFloat(75.0f);

Input_Oversold.Name = "Oversold Level";
Input_Oversold.SetFloat(25.0f);

return;
}

// MACD of Price Data
sc.MACD(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_MACD, sc.Index,
Input_FastLength.GetInt(), Input_SlowLength.GetInt(), Input_Length.GetInt(), Input_MovingAverageType.GetInt());

// Fast %K of MACD
float MaxMACD = sc.GetHighest(Subgraph_MACD, sc.Index, Input_Length.GetInt());
float MinMACD = sc.GetLowest(Subgraph_MACD, sc.Index, Input_Length.GetInt());
float RangeMACD = MaxMACD - MinMACD;

```

```

if (RangeMACD > 0.0f)
    Subgraph_FastKMACD[sc.Index] = 100.0f * (Subgraph_MACD[sc.Index] - MinMACD) / RangeMACD;
else
    Subgraph_FastKMACD[sc.Index] = Subgraph_FastKMACD[sc.Index - 1];

// Fast %D of MACD
if (sc.Index == 0)
    Subgraph_FastDMACD[sc.Index] = Subgraph_FastKMACD[sc.Index];
else
    Subgraph_FastDMACD[sc.Index] = Subgraph_FastDMACD[sc.Index - 1] + Input_Multiplier.GetFloat() *
(Subgraph_FastKMACD[sc.Index] - Subgraph_FastDMACD[sc.Index - 1]);

// Fast %K of Fast %D of MACD
float MaxFastDMACD = sc.GetHighest(Subgraph_FastDMACD, sc.Index, Input_Length.GetInt());
float MinFastDMACD = sc.GetLowest(Subgraph_FastDMACD, sc.Index, Input_Length.GetInt());
float RangeFastDMACD = MaxFastDMACD - MinFastDMACD;

if (RangeFastDMACD > 0.0f)
    Subgraph_FastKFastDMACD[sc.Index] = 100.0f * (Subgraph_FastDMACD[sc.Index] - MinFastDMACD) /
RangeFastDMACD;
else
    Subgraph_FastKFastDMACD[sc.Index] = Subgraph_FastKFastDMACD[sc.Index - 1];

// Schaff Trend Cycle
if (sc.Index == 0)
    Subgraph_SchaffTrendCycle[sc.Index] = Subgraph_FastKFastDMACD[sc.Index];
else
    Subgraph_SchaffTrendCycle[sc.Index] = Subgraph_SchaffTrendCycle[sc.Index - 1] + Input_Multiplier.GetFloat() *
(Subgraph_FastKFastDMACD[sc.Index] - Subgraph_SchaffTrendCycle[sc.Index - 1]);

// Overbought and Oversold Lines
Subgraph_Overbought[sc.Index] = Input_Overbought.GetFloat();
Subgraph_Oversold[sc.Index] = Input_Oversold.GetFloat();
}

/*=====*/
SCSFExport scsf_PremierStochasticOscillator(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_FastK = sc.Subgraph[0];
    SCSubgraphRef Subgraph_FastD = sc.Subgraph[1];
    SCSubgraphRef Subgraph_SlowD = sc.Subgraph[2];
    SCSubgraphRef Subgraph_PSO = sc.Subgraph[3];
    SCSubgraphRef Subgraph_Line1 = sc.Subgraph[4];
    SCSubgraphRef Subgraph_Line2 = sc.Subgraph[5];
    SCSubgraphRef Subgraph_Line3 = sc.Subgraph[6];
    SCSubgraphRef Subgraph_Line4 = sc.Subgraph[7];
    SCSubgraphRef Subgraph_CenterLine = sc.Subgraph[8];

    SCFloatArrayRef Array_Stochastic = sc.Subgraph[3].Arrays[0];
    SCFloatArrayRef Array_EMAStochastic = sc.Subgraph[3].Arrays[1];
    SCFloatArrayRef Array_EMAEMAStochastic = sc.Subgraph[3].Arrays[2];
    SCFloatArrayRef Array_ExponentialValue = sc.Subgraph[3].Arrays[3];

    SCInputRef Input_InputDataHigh = sc.Input[0];
    SCInputRef Input_InputDataLow = sc.Input[1];
    SCInputRef Input_InputDataLast = sc.Input[2];
    SCInputRef Input_StochasticType = sc.Input[3];
    SCInputRef Input_FastKLength = sc.Input[4];
    SCInputRef Input_FastDLength = sc.Input[5];
    SCInputRef Input_SlowDLength = sc.Input[6];
    SCInputRef Input_PercentDMovAvgType = sc.Input[7];
    SCInputRef Input_PSOLength = sc.Input[8];
    SCInputRef Input_PSOMovAvgType = sc.Input[9];
    SCInputRef Input_Line1Value = sc.Input[10];
    SCInputRef Input_Line2Value = sc.Input[11];

```

```

SCInputRef Input_Line3Value = sc.Input[12];
SCInputRef Input_Line4Value = sc.Input[13];

if (sc.SetDefaults)
{
    sc.GraphName = "Premier Stochastic Oscillator";

    sc.ValueFormat = 2;

    Subgraph_FastK.Name = "Fast %K";
    Subgraph_FastK.DrawStyle = DRAWSTYLE_IGNORE;

    Subgraph_FastD.Name = "Fast %D (Slow %K)";
    Subgraph_FastD.DrawStyle = DRAWSTYLE_IGNORE;

    Subgraph_SlowD.Name = "Slow %D";
    Subgraph_SlowD.DrawStyle = DRAWSTYLE_IGNORE;

    Subgraph_PSO.Name = "Premier Stochastic Oscillator";
    Subgraph_PSO.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_PSO.PrimaryColor = RGB(0, 255, 0);
    Subgraph_PSO.DrawZeros = true;

    Subgraph_Line1.Name = "Line 1";
    Subgraph_Line1.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_Line1.PrimaryColor = RGB(0, 0, 255);
    Subgraph_Line1.DrawZeros = true;

    Subgraph_Line2.Name = "Line 2";
    Subgraph_Line2.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_Line2.PrimaryColor = RGB(0, 0, 255);
    Subgraph_Line2.DrawZeros = true;

    Subgraph_Line3.Name = "Line 3";
    Subgraph_Line3.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_Line3.PrimaryColor = RGB(0, 0, 255);
    Subgraph_Line3.DrawZeros = true;

    Subgraph_Line4.Name = "Line 4";
    Subgraph_Line4.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_Line4.PrimaryColor = RGB(0, 0, 255);
    Subgraph_Line4.DrawZeros = true;

    Subgraph_CenterLine.Name = "Center Line";
    Subgraph_CenterLine.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_CenterLine.PrimaryColor = RGB(128, 0, 128);
    Subgraph_CenterLine.DrawZeros = true;

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

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

    Input_InputDataLast.Name = "Input Data for Last";
    Input_InputDataLast.SetInputDataIndex(SC_LAST);

    Input_StochasticType.Name = "Stochastic Type";
    Input_StochasticType.SetCustomInputStrings
("Fast %K"
";Fast %D (Slow %K)"
";Slow %D"
);
    Input_StochasticType.SetCustomInputIndex(0);

```

```

Input_FastKLength.Name = "Fast %K Length";
Input_FastKLength.SetInt(8);
Input_FastKLength.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_FastDLength.Name = "Fast %D (Slow %K) Length";
Input_FastDLength.SetInt(8);
Input_FastDLength.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_SlowDLength.Name = "Slow %D Length";
Input_SlowDLength.SetInt(8);
Input_SlowDLength.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_PercentDMovAvgType.Name = "Fast/Slow %D Moving Average Type";
Input_PercentDMovAvgType.SetMovAvgType(MOVAVGTYPE_SIMPLE);

Input_PSOLength.Name = "PSO Length";
Input_PSOLength.SetInt(5);
Input_PSOLength.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_PSOMovAvgType.Name = "PSO Moving Average Type";
Input_PSOMovAvgType.SetMovAvgType(MOVAVGTYPE_EXPONENTIAL);

Input_Line1Value.Name = "Line 1 Value";
Input_Line1Value.SetFloat(0.5f);
Input_Line1Value.SetFloatLimits(-1.0f, 1.0f);

Input_Line2Value.Name = "Line 2 Value";
Input_Line2Value.SetFloat(0.25f);
Input_Line2Value.SetFloatLimits(-1.0f, 1.0f);

Input_Line3Value.Name = "Line 3 Value";
Input_Line3Value.SetFloat(-0.25f);
Input_Line3Value.SetFloatLimits(-1.0f, 1.0f);

Input_Line4Value.Name = "Line 4 Value";
Input_Line4Value.SetFloat(-0.5f);
Input_Line4Value.SetFloatLimits(-1.0f, 1.0f);

sc.AutoLoop = true;
return;
}

sc.Stochastic(
    sc.BaseData[Input_InputDataHigh.GetInputDataIndex()],
    sc.BaseData[Input_InputDataLow.GetInputDataIndex()],
    sc.BaseData[Input_InputDataLast.GetInputDataIndex()],
    Subgraph_FastK, // Data member is Fast %K
    Input_FastKLength.GetInt(),
    Input_FastDLength.GetInt(),
    Input_SlowDLength.GetInt(),
    Input_PercentDMovAvgType.GetMovAvgType());

Subgraph_FastD[sc.Index] = Subgraph_FastK.Arrays[0][sc.Index];
Subgraph_SlowD[sc.Index] = Subgraph_FastK.Arrays[1][sc.Index];

const int SelectedIndex = Input_StochasticType.GetIndex();

switch (SelectedIndex)
{
case 0:
{
    sc.DataStartIndex = Input_FastKLength.GetInt() + 2 * Input_PSOMovAvgType.GetInt() - 1;

    Array_Stochastic[sc.Index] = 0.1f * (Subgraph_FastK[sc.Index] - 50.0f);
}
}

```

```

break;

case 1:
{
    sc.DataStartIndex = Input_FastKLength.GetInt() + Input_FastDLength.GetInt() + 2 *
Input_PSOMovAvgType.GetInt() - 1;

    Array_Stochastic[sc.Index] = 0.1f * (Subgraph_FastD[sc.Index] - 50.0f);
}
break;

case 2:
{
    sc.DataStartIndex = Input_FastKLength.GetInt() + Input_FastDLength.GetInt() + Input_SlowDLength.GetInt() + 2 *
Input_PSOMovAvgType.GetInt() - 1;

    Array_Stochastic[sc.Index] = 0.1f * (Subgraph_SlowD[sc.Index] - 50.0f);
}
break;
}

sc.MovingAverage(Array_Stochastic, Array_EMAStochastic, Input_PSOMovAvgType.GetMovAvgType(),
Input_PSOLength.GetInt());
sc.MovingAverage(Array_EMAStochastic, Array_EMAEMAStochastic, Input_PSOMovAvgType.GetMovAvgType(),
Input_PSOLength.GetInt());

Array_ExponentialValue[sc.Index] = exp(Array_EMAEMAStochastic[sc.Index]);

Subgraph_PSO[sc.Index] = (Array_ExponentialValue[sc.Index] - 1) / (Array_ExponentialValue[sc.Index] + 1);

Subgraph_Line1[sc.Index] = Input_Line1Value.GetFloat();
Subgraph_Line2[sc.Index] = Input_Line2Value.GetFloat();
Subgraph_Line3[sc.Index] = Input_Line3Value.GetFloat();
Subgraph_Line4[sc.Index] = Input_Line4Value.GetFloat();

Subgraph_CenterLine[sc.Index] = 0.0f;
}

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

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_Length = sc.Input[1];
    SCInputRef Input_Poles = sc.Input[2];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Super Smoother Filter";

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

        Subgraph_SuperSmootherFilter.Name = "Super Smoother Filter";
        Subgraph_SuperSmootherFilter.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_SuperSmootherFilter.PrimaryColor = RGB(0, 255, 0);

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

        Input_Length.Name = "Length";
        Input_Length.SetInt(15);
        Input_Length.SetIntLimits(1, MAX_STUDY_LENGTH);
    }
}

```

```

    Input_Poles.Name = "Number of Poles";
    Input_Poles.SetCustomInputStrings("2;3");
    Input_Poles.SetCustomInputIndex(0);

    return;
}

const int SelectedIndex = Input_Poles.GetIndex();
switch (SelectedIndex)
{
case 0:
{
    sc.SuperSmoother2Pole(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_SuperSmootherFilter,
sc.Index, Input_Length.GetInt());
}
break;

case 1:
{
    sc.SuperSmoother3Pole(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_SuperSmootherFilter,
sc.Index, Input_Length.GetInt());
}
break;
}
}

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

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_Length = sc.Input[1];
    SCInputRef Input_Poles = sc.Input[2];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Butterworth Filter";

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

        Subgraph_ButterworthFilter.Name = "Butterworth Filter";
        Subgraph_ButterworthFilter.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_ButterworthFilter.PrimaryColor = RGB(0, 255, 0);

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

        Input_Length.Name = "Length";
        Input_Length.SetInt(15);
        Input_Length.SetIntLimits(1, MAX_STUDY_LENGTH);

        Input_Poles.Name = "Number of Poles";
        Input_Poles.SetCustomInputStrings("2;3");
        Input_Poles.SetCustomInputIndex(0);

        return;
    }

    const int SelectedIndex = Input_Poles.GetIndex();
    switch (SelectedIndex)
    {
case 0:

```

```

    {
        sc.Butterworth2Pole(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_ButterworthFilter, sc.Index,
Input_Length.GetInt());
    }
    break;

    case 1:
    {
        sc.Butterworth3Pole(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_ButterworthFilter, sc.Index,
Input_Length.GetInt());
    }
    break;
}
}

```

```

/*=====*/

```

```

SCSFExport scsf_ConnorsRSI(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_RSI = sc.Subgraph[0];
    SCSubgraphRef Subgraph_UpDownLength = sc.Subgraph[1];
    SCSubgraphRef Subgraph_RSISupDownLength = sc.Subgraph[2];
    SCSubgraphRef Subgraph_RateOfChange = sc.Subgraph[3];
    SCSubgraphRef Subgraph_ConnorsRSI = sc.Subgraph[4];
    SCSubgraphRef Subgraph_Line1 = sc.Subgraph[5];
    SCSubgraphRef Subgraph_Line2 = sc.Subgraph[6];

    SCFloatArrayRef Array_PriceChange = sc.Subgraph[3].Arrays[0];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_RSILength = sc.Input[1];
    SCInputRef Input_RSIAvgType = sc.Input[2];
    SCInputRef Input_RateOfChangeLength = sc.Input[3];
    SCInputRef Input_Line1Value = sc.Input[4];
    SCInputRef Input_Line2Value = sc.Input[5];

    if (sc.SetDefaults)
    {
        sc.GraphName = "RSI - Connors ";

        sc.GraphRegion = 1;
        sc.ValueFormat = 2;
        sc.AutoLoop = 1;

        Subgraph_RSI.Name = "RSI";
        Subgraph_RSI.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_UpDownLength.Name = "Up-Down Streak Length";
        Subgraph_UpDownLength.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_RSISupDownLength.Name = "RSI of Up-Down Streak Length";
        Subgraph_RSISupDownLength.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_RateOfChange.Name = "Rate of Change";
        Subgraph_RateOfChange.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_ConnorsRSI.Name = "Connors RSI";
        Subgraph_ConnorsRSI.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_ConnorsRSI.PrimaryColor = RGB(0, 0, 255);
        Subgraph_ConnorsRSI.DrawZeros = true;

        Subgraph_Line1.Name = "Line 1";
        Subgraph_Line1.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Line1.PrimaryColor = RGB(255, 0, 255);
        Subgraph_Line1.DrawZeros = true;
    }
}

```

```

Subgraph_Line2.Name = "Line 2";
Subgraph_Line2.DrawStyle = DRAWSTYLE_LINE;
Subgraph_Line2.PrimaryColor = RGB(255, 255, 0);
Subgraph_Line2.DrawZeros = true;

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

Input_RSILength.Name = "RSI Length";
Input_RSILength.SetInt(3);
Input_RSILength.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_RSIAvgType.Name = "RSI Average Type";
Input_RSIAvgType.SetMovAvgType(MOVAVGTYPE_WILDERS);

Input_RateOfChangeLength.Name = "Rate of Change Length";
Input_RateOfChangeLength.SetInt(100);
Input_RateOfChangeLength.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_Line1Value.Name = "Line 1 Value";
Input_Line1Value.SetFloat(70.0f);

Input_Line2Value.Name = "Line 2 Value";
Input_Line2Value.SetFloat(30.0f);

return;
}

//Compute RSI
sc.RSI(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_RSI, Input_RSIAvgType.GetMovAvgType(),
Input_RSILength.GetInt());

//Compute RSI of Up-Down Length
if (sc.Index == 0)
{
    Subgraph_UpDownLength[sc.Index] = 0;
    Subgraph_RateOfChange[sc.Index] = 0;
}
else
{
    int i = 0;
    int j = 0;
    float UpDownStreak = 0.0f;

    while (sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - i - 1] <
sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - i])
    {
        UpDownStreak += 1.0f;

        i++;
    }

    while (sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - j - 1] >
sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - j])
    {
        UpDownStreak -= 1.0f;

        j++;
    }

    Subgraph_UpDownLength[sc.Index] = UpDownStreak;
}

```

```

sc.RSI(Subgraph_UpDownLength, Subgraph_RSIUpDownLength, Input_RSIAvgType.GetMovAvgType(), 2);

//Compute Rate of Change
Array_PriceChange[sc.Index] = sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index] -
sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - 1];

float ROCStreak = 0.0f;

for (int i = 1; i <= Input_RateOfChangeLength.GetInt(); i++)
{
    if (Array_PriceChange[sc.Index - i] < Array_PriceChange[sc.Index])
        ROCStreak += 1;
}

float ROCLength = static_cast<float>(Input_RateOfChangeLength.GetInt());

Subgraph_RateOfChange[sc.Index] = 100.0f * ROCStreak / ROCLength;

//Compute Connors RSI
Subgraph_ConnorsRSI[sc.Index] = (Subgraph_RSI[sc.Index] + Subgraph_RSIUpDownLength[sc.Index] +
Subgraph_RateOfChange[sc.Index]) / 3.0f;

Subgraph_Line1[sc.Index] = Input_Line1Value.GetFloat();
Subgraph_Line2[sc.Index] = Input_Line2Value.GetFloat();
}

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

    SCFloatArrayRef Array_AvgVol = sc.Subgraph[0].Arrays[0];
    SCFloatArrayRef Array_StdDevVol = sc.Subgraph[0].Arrays[1];

    SCInputRef Input_Length = sc.Input[0];
    SCInputRef Input_NumStDevs = sc.Input[1];
    SCInputRef Input_MovAvgType = sc.Input[2];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Relative Volume Standard Deviation";

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

        Subgraph_RelVol.Name = "Relative Volume Standard Deviation";
        Subgraph_RelVol.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_RelVol.PrimaryColor = RGB(0, 255, 0);
        Subgraph_RelVol.SecondaryColor = RGB(255, 0, 0);

        Input_Length.Name = "Length";
        Input_Length.SetInt(60);
        Input_Length.SetIntLimits(1, MAX_STUDY_LENGTH);

        Input_NumStDevs.Name = "Number of Standard Deviations";
        Input_NumStDevs.SetInt(2);
        Input_NumStDevs.SetIntLimits(1, MAX_STUDY_LENGTH);

        Input_MovAvgType.Name = "Moving Average Type";
        Input_MovAvgType.SetMovAvgType(MOAVGTYPE_SIMPLE);

        return;
    }

    sc.MovingAverage(sc.Volume, Array_AvgVol, Input_MovAvgType.GetMovAvgType(), Input_Length.GetInt());

```

```

sc.StdDeviation(sc.Volume, Array_StdDevVol, Input_Length.GetInt());

if (Array_StdDevVol[sc.Index] != 0.0f)
    Subgraph_RelVol[sc.Index] = (sc.Volume[sc.Index] - Array_AvgVol[sc.Index]) / Array_StdDevVol[sc.Index];
else
    Subgraph_RelVol[sc.Index] = 0.0f;

if (Subgraph_RelVol[sc.Index] > Input_NumStDevs.GetInt())
    Subgraph_RelVol.DataColor[sc.Index] = Subgraph_RelVol.PrimaryColor;
else
    Subgraph_RelVol.DataColor[sc.Index] = Subgraph_RelVol.SecondaryColor;
}

/*=====*/
SCSFExport scsf_FreedomOfMovement(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_NormMove = sc.Subgraph[0];
    SCSubgraphRef Subgraph_NormRelVol = sc.Subgraph[1];
    SCSubgraphRef Subgraph_FOM = sc.Subgraph[2];

    SCFloatArrayRef Array_Move = sc.Subgraph[0].Arrays[0];
    SCFloatArrayRef Array_HighMove = sc.Subgraph[0].Arrays[1];
    SCFloatArrayRef Array_LowMove = sc.Subgraph[0].Arrays[2];
    SCFloatArrayRef Array_RelVol = sc.Subgraph[1].Arrays[0];
    SCFloatArrayRef Array_AvgVol = sc.Subgraph[1].Arrays[1];
    SCFloatArrayRef Array_StdDevVol = sc.Subgraph[1].Arrays[2];
    SCFloatArrayRef Array_HighRelVol = sc.Subgraph[1].Arrays[3];
    SCFloatArrayRef Array_LowRelVol = sc.Subgraph[1].Arrays[4];
    SCFloatArrayRef Array_RelVolByMove = sc.Subgraph[2].Arrays[0];
    SCFloatArrayRef Array_AvgRelVolByMove = sc.Subgraph[2].Arrays[1];
    SCFloatArrayRef Array_SDRelVolByMove = sc.Subgraph[2].Arrays[2];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_Length = sc.Input[1];
    SCInputRef Input_NumStDevs = sc.Input[2];
    SCInputRef Input_MovAvgType = sc.Input[3];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Freedom of Movement";

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

        Subgraph_NormMove.Name = "Normalized Move";
        Subgraph_NormMove.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_NormRelVol.Name = "Normalized Relative Volume Standard Deviation";
        Subgraph_NormRelVol.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_FOM.Name = "Freedom of Movement";
        Subgraph_FOM.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_FOM.PrimaryColor = RGB(0, 255, 0);
        Subgraph_FOM.SecondaryColor = RGB(255, 0, 0);

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

        Input_Length.Name = "Length";
        Input_Length.SetInt(60);
        Input_Length.SetIntLimits(1, MAX_STUDY_LENGTH);

        Input_NumStDevs.Name = "Number of Standard Deviations";
        Input_NumStDevs.SetInt(2);
    }
}

```

```

Input_NumStDevs.SetIntLimits(1, MAX_STUDY_LENGTH);

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

return;
}

// Compute Normalized Move
if (sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - 1] != 0.0f)
    Array_Move[sc.Index] = (sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index] -
sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - 1]) / sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - 1];
else
    Array_Move[sc.Index] = 0.0f;

sc.Highest(Array_Move, Array_HighMove, Input_Length.GetInt());
float HighMove = Array_HighMove[sc.Index];

sc.Lowest(Array_Move, Array_LowMove, Input_Length.GetInt());
float LowMove = Array_LowMove[sc.Index];

if (HighMove - LowMove != 0.0f)
    Subgraph_NormMove[sc.Index] = 1.0f + 9.0f * (Array_Move[sc.Index] - LowMove) / (HighMove - LowMove);
else
    Subgraph_NormMove[sc.Index] = 0.0f;

// Compute Normalized Relative Volume
sc.MovingAverage(sc.Volume, Array_AvgVol, Input_MovAvgType.GetMovAvgType(), Input_Length.GetInt());

sc.StdDeviation(sc.Volume, Array_StdDevVol, Input_Length.GetInt());

if (Array_StdDevVol[sc.Index] != 0.0f)
    Array_RelVol[sc.Index] = (sc.Volume[sc.Index] - Array_AvgVol[sc.Index]) / Array_StdDevVol[sc.Index];
else
    Array_RelVol[sc.Index] = 0.0f;

sc.MovingAverage(sc.Volume, Array_AvgVol, Input_MovAvgType.GetMovAvgType(), Input_Length.GetInt());

sc.StdDeviation(sc.Volume, Array_StdDevVol, Input_Length.GetInt());

if (Array_StdDevVol[sc.Index] != 0.0f)
    Array_RelVol[sc.Index] = (sc.Volume[sc.Index] - Array_AvgVol[sc.Index]) / Array_StdDevVol[sc.Index];
else
    Array_RelVol[sc.Index] = 0.0f;

sc.Highest(Array_RelVol, Array_HighRelVol, Input_Length.GetInt());
float HighRelVol = Array_HighRelVol[sc.Index];

sc.Lowest(Array_RelVol, Array_LowRelVol, Input_Length.GetInt());
float LowRelVol = Array_LowRelVol[sc.Index];

if (HighRelVol - LowRelVol != 0.0f)
    Subgraph_NormRelVol[sc.Index] = 1.0f + 9.0f * (Array_RelVol[sc.Index] - LowRelVol) / (HighRelVol - LowRelVol);
else
    Subgraph_NormRelVol[sc.Index] = 0.0f;

// Compute Freedom of Movement
if (Subgraph_NormMove[sc.Index] != 0.0f)
    Array_RelVolByMove[sc.Index] = Subgraph_NormRelVol[sc.Index] / Subgraph_NormMove[sc.Index];
else
    Array_RelVolByMove[sc.Index] = 0.0f;

sc.MovingAverage(Array_RelVolByMove, Array_AvgRelVolByMove, Input_MovAvgType.GetMovAvgType(),
Input_Length.GetInt());

```

```
sc.StdDeviation(Array_RelVolByMove, Array_SDRelVolByMove, Input_Length.GetInt());

if (Array_SDRelVolByMove[sc.Index] != 0.0f)
    Subgraph_FOM[sc.Index] = (Array_RelVolByMove[sc.Index] - Array_AvgRelVolByMove[sc.Index]) /
Array_SDRelVolByMove[sc.Index];
else
    Subgraph_FOM[sc.Index] = 0.0f;

if (Subgraph_FOM[sc.Index] > Input_NumStDevs.GetInt())
    Subgraph_FOM.DataColor[sc.Index] = Subgraph_FOM.PrimaryColor;
else
    Subgraph_FOM.DataColor[sc.Index] = Subgraph_FOM.SecondaryColor;
}
```