

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

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

```
SCSFExport scsf_Ergodic(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Erg = sc.Subgraph[0];
    SCSubgraphRef Subgraph_ErgSignalLine = sc.Subgraph[1];
    SCSubgraphRef Subgraph_ErgOsc = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Temp3 = sc.Subgraph[3];
    SCSubgraphRef Subgraph_Temp4 = sc.Subgraph[4];
    SCSubgraphRef Subgraph_Temp5 = sc.Subgraph[5];
    SCSubgraphRef Subgraph_Temp6 = sc.Subgraph[6];
    SCSubgraphRef Subgraph_Temp7 = sc.Subgraph[7];
    SCSubgraphRef Subgraph_Temp8 = sc.Subgraph[8];

    SCInputRef Input_Data      = sc.Input[0];
    SCInputRef Input_MAType     = sc.Input[1];
    SCInputRef Input_LongMALength = sc.Input[3];
    SCInputRef Input_ShortMALength = sc.Input[4];
    SCInputRef Input_Multiplier = sc.Input[5];
    SCInputRef SignalLineMALength = sc.Input[6];

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

        sc.GraphRegion = 1;
        sc.ValueFormat = 2;
        sc.AutoLoop = 1;

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

        Subgraph_ErgSignalLine.Name = "Erg Signal Line";
        Subgraph_ErgSignalLine.DrawStyle = DRAWSTYLE_IGNORE;
        Subgraph_ErgSignalLine.PrimaryColor = RGB(255,0,255);
        Subgraph_ErgSignalLine.DrawZeros = true;

        Subgraph_ErgOsc.Name = "Erg Osc";
        Subgraph_ErgOsc.DrawStyle = DRAWSTYLE_IGNORE;
        Subgraph_ErgOsc.PrimaryColor = RGB(255,255,0);
        Subgraph_ErgOsc.DrawZeros = true;

        // inputs
        Input_Data.Name = "Input Data";
        Input_Data.SetInputDataIndex(SC_LAST);

        Input_MAType.Name = "MovAvg Type";
        Input_MAType.SetMovAvgType(MOVAVGTYPE_EXPONENTIAL);

        Input_LongMALength.Name = "Long MovAvg Length";
        Input_LongMALength.SetInt(15);
        Input_LongMALength.SetIntLimits(1, MAX_STUDY_LENGTH);

        Input_ShortMALength.Name = "Short MovAvg Length";
        Input_ShortMALength.SetInt(5);
        Input_ShortMALength.SetIntLimits(1, MAX_STUDY_LENGTH);

        Input_Multiplier.Name = "Multiplier";
        Input_Multiplier.SetInt(1);
    }
}
```

```

SignalLineMALength.Name = "Signal Line MovAvg Length";
SignalLineMALength.SetInt(10);
SignalLineMALength.SetIntLimits(1, MAX_STUDY_LENGTH);

return;
}

// Subgraphs 3-7 are used for calculation only
sc.DataStartIndex = ((Input_LongMALength.GetInt() ) + (Input_ShortMALength.GetInt()) +
SignalLineMALength.GetInt()) ;

if (sc.Index < 1) // not large enough
return;

int LongStart = sc.Index;
if (LongStart < Input_LongMALength.GetInt() - 1) //2
LongStart = Input_LongMALength.GetInt() - 1; //2

int ShortStart = sc.Index;
if (ShortStart < Input_LongMALength.GetInt() - 1 + Input_ShortMALength.GetInt() - 1) //3
ShortStart = Input_LongMALength.GetInt() - 1 + Input_ShortMALength.GetInt() - 1; //3

int SignalStart = sc.Index;
if (SignalStart < Input_ShortMALength.GetInt() - 1 + SignalLineMALength.GetInt() - 1 + Input_LongMALength.GetInt() - 1)
//4
SignalStart = Input_ShortMALength.GetInt() - 1 + SignalLineMALength.GetInt() - 1 + Input_LongMALength.GetInt()
- 1; //4

int DataStart = sc.Index;
if (DataStart < 1)
DataStart = 1;

if(sc.Index >= DataStart)
Subgraph_Temp3[sc.Index] = sc.BaseDataIn[Input_Data.GetInputDataIndex()][sc.Index] -
sc.BaseDataIn[Input_Data.GetInputDataIndex()][sc.Index-1];

// Calculate the Numerator into SubGraphData[0]
if(sc.Index >= LongStart)
sc.MovingAverage(Subgraph_Temp3, Subgraph_Temp4, Input_MAType.GetMovAvgType(),
Input_LongMALength.GetInt());

if(sc.Index >= ShortStart)
sc.MovingAverage(Subgraph_Temp4, Subgraph_Temp5, Input_MAType.GetMovAvgType(),
Input_ShortMALength.GetInt());

// Calculate the Denominator into SubGraphData[1]
if(sc.Index >= DataStart)
Subgraph_Temp6[sc.Index] = fabs(Subgraph_Temp3[sc.Index]);

if(sc.Index >= LongStart)
sc.MovingAverage(Subgraph_Temp6, Subgraph_Temp7, Input_MAType.GetMovAvgType(),
Input_LongMALength.GetInt());

if(sc.Index >= ShortStart)
{
sc.MovingAverage(Subgraph_Temp7, Subgraph_Temp8, Input_MAType.GetMovAvgType(),
Input_ShortMALength.GetInt());

// Store the TSI (Numerator / Denominator)
if (Subgraph_Temp8[sc.Index] != 0)
Subgraph_Erg[sc.Index] = Input_Multiplier.GetInt() * Subgraph_Temp5[sc.Index]/Subgraph_Temp8[sc.Index];
else
Subgraph_Erg[sc.Index] = 0;
}

```

```

// Calculate the Signal Line ( EMA[TSI] )
if(sc.Index >= SignalStart)
    sc.MovingAverage(Subgraph_Erg, Subgraph_ErgSignalLine, Input_MAType.GetMovAvgType(),
SignalLineMALength.GetInt());

// Calculate the Oscillator (TSI - EMA[TSI])
if(sc.Index >= sc.DataStartIndex)
    Subgraph_ErgOsc[sc.Index] = Subgraph_Erg[sc.Index] - Subgraph_ErgSignalLine[sc.Index];
}

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

    SCInputRef Input_Data = sc.Input[0];
    SCInputRef Input_HullMALength = sc.Input[1];

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

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

        Subgraph_HullMA.Name = "Hull MA";
        Subgraph_HullMA.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_HullMA.PrimaryColor = RGB(0,255,0);
        Subgraph_HullMA.DrawZeros = true;

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

        Input_HullMALength.Name = "Moving Average Length";
        Input_HullMALength.SetInt(25);
        Input_HullMALength.SetIntLimits(1, MAX_STUDY_LENGTH);

        return;
    }

    int SqrtLength = static_cast<int>(sc.Round(sqrt(static_cast<float>(Input_HullMALength.GetInt()))));
    sc.DataStartIndex = Input_HullMALength.GetInt() + SqrtLength - 1;

    sc.HullMovingAverage(sc.BaseDataIn[Input_Data.GetInputDataIndex()], Subgraph_HullMA,
Input_HullMALength.GetInt());
}

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

    SCInputRef Input_VersionCheckInput = sc.Input[6];
    SCInputRef Input_StudySubgraph1 = sc.Input[7];
    SCInputRef Input_StudySubgraph2 = sc.Input[8];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Study Subgraphs Average";

        sc.ValueFormat = 3;
        sc.AutoLoop = 0;//manual looping

        Subgraph_Average.Name = "Average";

```

```

Subgraph_Average.DrawStyle = DRAWSTYLE_LINE;
Subgraph_Average.PrimaryColor = RGB(0, 255, 0);
Subgraph_Average.DrawZeros = false;

Input_StudySubgraph1.Name = "Study Subgraph Reference 1";
Input_StudySubgraph1.SetStudySubgraphValues(0,0);

Input_StudySubgraph2.Name = "Study Subgraph Reference 2";
Input_StudySubgraph2.SetStudySubgraphValues(0,0);

Input_VersionCheckInput.SetInt(2);

sc.CalculationPrecedence = LOW_PREC_LEVEL;

return;
}

SCFloatArray Study1Array;
sc.GetStudyArrayUsingID(Input_StudySubgraph1.GetStudyID(), Input_StudySubgraph1.GetSubgraphIndex(),
Study1Array);

SCFloatArray Study2Array;
sc.GetStudyArrayUsingID(Input_StudySubgraph2.GetStudyID(), Input_StudySubgraph2.GetSubgraphIndex(),
Study2Array);

for (int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize; BarIndex++)
{
    Subgraph_Average[BarIndex] = (Study1Array[BarIndex] + Study2Array[BarIndex]) / 2.0f;
}
}

/*=====*/
SCSFExport scsf_StudySubgraphsAverageMultiple(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Average = sc.Subgraph[0];
    SCFloatArrayRef Array_SubgraphCount = sc.Subgraph[0].Arrays[0];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Study Subgraphs Average - Multiple";

        sc.ValueFormat = 3;
        sc.AutoLoop = 0;//manual looping

        sc.CalculationPrecedence = LOW_PREC_LEVEL;

        Subgraph_Average.Name = "Average";
        Subgraph_Average.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Average.LineWidth = 2;
        Subgraph_Average.PrimaryColor = RGB(0, 255, 0);
        Subgraph_Average.DrawZeros = false;

        int StudyReferenceIndex = 0;

        for (int InputIndex = 0; InputIndex < 40; InputIndex += 2)
        {
            StudyReferenceIndex++;
            sc.Input[InputIndex].Name.Format("Use Study Subgraph Reference %d", StudyReferenceIndex);
            sc.Input[InputIndex].SetYesNo(false);

            sc.Input[InputIndex + 1].Name.Format("Study Subgraph Reference %d", StudyReferenceIndex);
            sc.Input[InputIndex + 1].SetStudySubgraphValues(0, 0);
        }
    }
}

```

```

    }

    return;
}

int NumberUsedSubgraphs = 0;

int CalculationStartIndex = sc.GetCalculationStartIndexForStudy();
sc.EarliestUpdateSubgraphDataArrayIndex = CalculationStartIndex;

for (int SubgraphIndex = 0; SubgraphIndex < 20; SubgraphIndex++)
{
    if (!sc.Input[SubgraphIndex * 2].GetYesNo())
        continue;

    SCFloatArray StudyArray;
    sc.GetStudyArrayUsingID(sc.Input[SubgraphIndex * 2 + 1].GetStudyID(), sc.Input[SubgraphIndex * 2 +
1].GetSubgraphIndex(), StudyArray);

    if (StudyArray.GetArraySize() == 0)
        continue;

    NumberUsedSubgraphs++;

    for (int BarIndex = CalculationStartIndex; BarIndex < sc.ArraySize; BarIndex++)
    {
        if (NumberUsedSubgraphs == 1)
        {
            Subgraph_Average[BarIndex] = 0;
            Array_SubgraphCount[BarIndex] = 0;
        }

        //If the input subgraph is zero at this bar index, then do not include it in the average.
        if (StudyArray[BarIndex] == 0)
            continue;

        Subgraph_Average[BarIndex] += StudyArray[BarIndex];
        Array_SubgraphCount[BarIndex] += 1;
    }
}

for (int BarIndex = CalculationStartIndex; BarIndex < sc.ArraySize; BarIndex++)
{
    if (Array_SubgraphCount[BarIndex] != 0)
        Subgraph_Average[BarIndex] /= Array_SubgraphCount[BarIndex];
}
}

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

    SCFloatArrayRef Array_TR = sc.Subgraph[0].Arrays[0];
    SCFloatArrayRef Array_ATR = sc.Subgraph[0].Arrays[1];

    SCInputRef Input_LookBackLength_NRn = sc.Input[0];
    SCInputRef Input_LookBackLength_xBarNR = sc.Input[1];
    SCInputRef Input_LookBackType = sc.Input[2];
    SCInputRef Input_UseOpenToCloseRange_NRn = sc.Input[3];
    SCInputRef Input_GroupSize = sc.Input[4];
    SCInputRef Input_LookBackLength_ATR = sc.Input[5];
    SCInputRef Input_ATRMAType = sc.Input[6];

```

```

if (sc.SetDefaults)
{
    sc.GraphName = "Narrow Range Bar";

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

    int InputDisplayOrder = 0;

    Subgraph_NarrowRange.Name = "Narrow Range";
    Subgraph_NarrowRange.DrawStyle = DRAWSTYLE_COLOR_BAR;
    Subgraph_NarrowRange.PrimaryColor = RGB(0, 0, 255);
    Subgraph_NarrowRange.SecondaryColorUsed = true;
    Subgraph_NarrowRange.SecondaryColor = RGB(127, 0, 127);
    Subgraph_NarrowRange.DrawZeros = false;

    Input_LookBackType.Name = "Lookback Type";
    Input_LookBackType.SetCustomInputStrings
("NR n;xBar NR;NR vs ATR");
    Input_LookBackType.SetCustomInputIndex(0);
    Input_LookBackType.DisplayOrder = ++InputDisplayOrder;

    Input_LookBackLength_NRn.Name = "Lookback Length - NR n";
    Input_LookBackLength_NRn.SetInt(5);
    Input_LookBackLength_NRn.SetIntLimits(0, MAX_STUDY_LENGTH);
    Input_LookBackLength_NRn.DisplayOrder = ++InputDisplayOrder;

    Input_UseOpenToCloseRange_NRn.Name = "Use Open To Close Range for NR n Lookback Type";
    Input_UseOpenToCloseRange_NRn.SetYesNo(false);
    Input_UseOpenToCloseRange_NRn.DisplayOrder = ++InputDisplayOrder;

    Input_LookBackLength_xBarNR.Name = "Lookback Length - xBar NR";
    Input_LookBackLength_xBarNR.SetInt(20);
    Input_LookBackLength_xBarNR.SetIntLimits(Input_GroupSize.GetInt(), MAX_STUDY_LENGTH);
    Input_LookBackLength_xBarNR.DisplayOrder = ++InputDisplayOrder;

    Input_GroupSize.Name = "Number of Bars in Comparison Group";
    Input_GroupSize.SetInt(3);
    Input_GroupSize.SetIntLimits(0, Input_LookBackLength_xBarNR.GetInt());
    Input_GroupSize.DisplayOrder = ++InputDisplayOrder;

    Input_LookBackLength_ATR.Name = "Lookback Length - NR vs ATR";
    Input_LookBackLength_ATR.SetInt(10);
    Input_LookBackLength_ATR.SetIntLimits(0, MAX_STUDY_LENGTH);
    Input_LookBackLength_ATR.DisplayOrder = ++InputDisplayOrder;

    Input_ATRMAType.Name = "ATR Moving Average Type";
    Input_ATRMAType.SetMovAvgType(MOVAVGTYPE_SIMPLE);
    Input_ATRMAType.DisplayOrder = ++InputDisplayOrder;

    return;
}

const int SelectedIndex = Input_LookBackType.GetIndex();

switch (Input_LookBackType.GetIndex())
{
case 0:
{
    if (sc.Index < Input_LookBackLength_NRn.GetInt())
        return;

    float isLowest = 1.0f;

```

```

/* Find range of current bar. */
float RangeOfCurrentBar;

if (Input_UseOpenToCloseRange_NRn.GetYesNo())
    RangeOfCurrentBar = fabs(sc.Open[sc.Index] - sc.Close[sc.Index]);
else
    RangeOfCurrentBar = sc.High[sc.Index] - sc.Low[sc.Index];

/* Find the minimum range in the lookback period. */
int LookBackLength = Input_LookBackLength_NRn.GetInt();
float PriorRange;

for (int i = 1; i <= LookBackLength; i++)
{
    if (Input_UseOpenToCloseRange_NRn.GetYesNo())
        PriorRange = fabs(sc.Open[sc.Index - i] - sc.Close[sc.Index - i]);
    else
        PriorRange = sc.High[sc.Index - i] - sc.Low[sc.Index - i];

    if (!sc.FormattedEvaluate(PriorRange, sc.BaseGraphValueFormat, GREATER_EQUAL_OPERATOR,
RangeOfCurrentBar, sc.BaseGraphValueFormat))
    {
        isLowest = 0.0f;
        break;
    }
}

Subgraph_NarrowRange.DataColor[sc.Index] = Subgraph_NarrowRange.PrimaryColor;
Subgraph_NarrowRange[sc.Index] = isLowest;
}
break;

case 1:
{
    if (sc.Index < Input_LookBackLength_xBarNR.GetInt())
        return;

    int GroupSize = Input_GroupSize.GetInt();
    float isLowest = 1.0f;

    /* Find the range of the current group of bars. */
    float RangeOfCurrentBarGroup;
    float CurrentHighestHigh = sc.High[sc.Index];
    float CurrentLowestLow = sc.Low[sc.Index];

    for (int i = 1; i < GroupSize; i++)
    {
        if (sc.High[sc.Index - i] > CurrentHighestHigh)
            CurrentHighestHigh = sc.High[sc.Index - i];

        if (sc.Low[sc.Index - i] < CurrentLowestLow)
            CurrentLowestLow = sc.Low[sc.Index - i];
    }

    RangeOfCurrentBarGroup = CurrentHighestHigh - CurrentLowestLow;

    /* Find the minimum x-bar range in the lookback period. */
    int LookBackLength = Input_LookBackLength_xBarNR.GetInt();
    float PriorLowestGroupRange = FLT_MAX;

    /* Find the group of x bars with the lowest range. */
    for (int i = 1; i <= LookBackLength - GroupSize + 1; i++)
    {
        float GroupHighestHigh = sc.High[sc.Index - i];

```

```

float GroupLowestLow = sc.Low[sc.Index - i];
float GroupRange;

/* Find range of single group within lookback period. */
for (int j = i + 1; j < i + GroupSize; j++)
{
    if (sc.High[sc.Index - j] > GroupHighestHigh)
        GroupHighestHigh = sc.High[sc.Index - j];
    if (sc.Low[sc.Index - j] < GroupLowestLow)
        GroupLowestLow = sc.Low[sc.Index - j];
}

GroupRange = GroupHighestHigh - GroupLowestLow;

if (GroupRange < PriorLowestGroupRange)
    PriorLowestGroupRange = GroupRange;

if (!sc.FormattedEvaluate(PriorLowestGroupRange, sc.BaseGraphValueFormat,
GREATER_EQUAL_OPERATOR, RangeOfCurrentBarGroup, sc.BaseGraphValueFormat))
{
    isLowest = 0.0f;
    break;
}

Subgraph_NarrowRange.DataColor[sc.Index] = Subgraph_NarrowRange.PrimaryColor;
Subgraph_NarrowRange[sc.Index] = isLowest;

if (isLowest == 1.0f)
{
    for (int i = 1; i < GroupSize; i++)
    {
        Subgraph_NarrowRange.DataColor[sc.Index - i] = Subgraph_NarrowRange.SecondaryColor;
        Subgraph_NarrowRange[sc.Index - i] = isLowest;
    }
}
break;

case 2:
{
    sc.ATR(sc.BaseDataIn, Array_TR, Array_ATR, Input_LookBackLength_ATR.GetInt(),
Input_ATRMAType.GetMovAvgType());

    float CurrentBarRange = sc.High[sc.Index] - sc.Low[sc.Index];

    if (CurrentBarRange < Array_ATR[sc.Index - 1] && sc.Index >= Input_LookBackLength_ATR.GetInt())
    {
        Subgraph_NarrowRange[sc.Index] = 1.0f;
        Subgraph_NarrowRange.DataColor[sc.Index] = Subgraph_NarrowRange.PrimaryColor;
    }
    else
    {
        Subgraph_NarrowRange[sc.Index] = 0.0f;
    }
}
break;
}

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

```



```

SCFloatArrayRef Array_TR = sc.Subgraph[0].Arrays[0];
SCFloatArrayRef Array_ATR = sc.Subgraph[0].Arrays[1];

SCInputRef Input_LookBackLength_WRn = sc.Input[0];
SCInputRef Input_UseOpenToCloseRange_WRn = sc.Input[1];
SCInputRef Input_LookBackLength_xBarWR = sc.Input[2];
SCInputRef Input_LookBackType = sc.Input[3];
SCInputRef Input_GroupSize = sc.Input[4];
SCInputRef Input_LookBackLength_ATR = sc.Input[5];
SCInputRef Input_ATRMAType = sc.Input[6];

if (sc.SetDefaults)
{
    sc.GraphName = "Wide Range Bar";

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

    Subgraph_WideRange.Name = "Wide Range";
    Subgraph_WideRange.DrawStyle = DRAWSTYLE_COLOR_BAR;
    Subgraph_WideRange.PrimaryColor = RGB(0, 0, 255);
    Subgraph_WideRange.SecondaryColorUsed = true;
    Subgraph_WideRange.SecondaryColor = RGB(127, 0, 127);
    Subgraph_WideRange.DrawZeros = false;

    Input_LookBackType.Name = "Lookback Type";
    Input_LookBackType.SetCustomInputStrings
("WR n;xBar WR;WR vs ATR");
    Input_LookBackType.SetCustomInputIndex(0);
    Input_LookBackType.DisplayOrder = ++InputDisplayOrder;

    Input_LookBackLength_WRn.Name = "Lookback Length - WR n";
    Input_LookBackLength_WRn.SetInt(5);
    Input_LookBackLength_WRn.SetIntLimits(0, MAX_STUDY_LENGTH);
    Input_LookBackLength_WRn.DisplayOrder = ++InputDisplayOrder;

    Input_UseOpenToCloseRange_WRn.Name = "Use Open To Close Range for WR n Lookback Type";
    Input_UseOpenToCloseRange_WRn.SetYesNo(false);
    Input_UseOpenToCloseRange_WRn.DisplayOrder = ++InputDisplayOrder;

    Input_LookBackLength_xBarWR.Name = "Lookback Length - xBar WR";
    Input_LookBackLength_xBarWR.SetInt(20);
    Input_LookBackLength_xBarWR.SetIntLimits(Input_GroupSize.GetInt(), MAX_STUDY_LENGTH);
    Input_LookBackLength_xBarWR.DisplayOrder = ++InputDisplayOrder;

    Input_GroupSize.Name = "Number of Bars in Comparison Group";
    Input_GroupSize.SetInt(3);
    Input_GroupSize.SetIntLimits(0, Input_LookBackLength_xBarWR.GetInt());
    Input_GroupSize.DisplayOrder = ++InputDisplayOrder;

    Input_LookBackLength_ATR.Name = "Lookback Length - WR vs ATR";
    Input_LookBackLength_ATR.SetInt(10);
    Input_LookBackLength_ATR.SetIntLimits(0, MAX_STUDY_LENGTH);
    Input_LookBackLength_ATR.DisplayOrder = ++InputDisplayOrder;

    Input_ATRMAType.Name = "ATR Moving Average Type";
    Input_ATRMAType.SetMovAvgType(MOAVGTYPE_SIMPLE);
    Input_ATRMAType.DisplayOrder = ++InputDisplayOrder;

    return;
}

const int SelectedIndex = Input_LookBackType.GetIndex();

```

```

switch (Input_LookBackType.GetIndex())
{
case 0:
{
    if (sc.Index < Input_LookBackLength_WRn.GetInt())
        return;

    float isHighest = 1.0f;

    /* Find range of current bar. */
    float RangeOfCurrentBar;

    if (Input_UseOpenToCloseRange_WRn.GetYesNo())
        RangeOfCurrentBar = fabs(sc.Open[sc.Index] - sc.Close[sc.Index]);
    else
        RangeOfCurrentBar = sc.High[sc.Index] - sc.Low[sc.Index];

    /* Find the maximum range in the lookback period. */
    int LookBackLength = Input_LookBackLength_WRn.GetInt();
    float PriorRange;

    for (int i = 1; i <= LookBackLength; i++)
    {
        if (Input_UseOpenToCloseRange_WRn.GetYesNo())
            PriorRange = fabs(sc.Open[sc.Index - i] - sc.Close[sc.Index - i]);
        else
            PriorRange = sc.High[sc.Index - i] - sc.Low[sc.Index - i];

        if (!sc.FormattedEvaluate(PriorRange, sc.BaseGraphValueFormat, LESS_EQUAL_OPERATOR,
RangeOfCurrentBar, sc.BaseGraphValueFormat))
        {
            isHighest = 0.0f;
            break;
        }
    }

    Subgraph_WideRange.DataColor[sc.Index] = Subgraph_WideRange.PrimaryColor;
    Subgraph_WideRange[sc.Index] = isHighest;
}
break;

case 1:
{
    if (sc.Index < Input_LookBackLength_xBarWR.GetInt())
        return;

    int GroupSize = Input_GroupSize.GetInt();
    float isHighest = 1.0f;

    /* Find the range of the current group of bars. */
    float RangeOfCurrentBarGroup;
    float CurrentHighestHigh = sc.High[sc.Index];
    float CurrentLowestLow = sc.Low[sc.Index];

    for (int i = 1; i < GroupSize; i++)
    {
        if (sc.High[sc.Index - i] > CurrentHighestHigh)
            CurrentHighestHigh = sc.High[sc.Index - i];

        if (sc.Low[sc.Index - i] < CurrentLowestLow)
            CurrentLowestLow = sc.Low[sc.Index - i];
    }

    RangeOfCurrentBarGroup = CurrentHighestHigh - CurrentLowestLow;

```

```

/* Find the maximum x-bar range in the lookback period. */
int LookBackLength = Input_LookBackLength_xBarWR.GetInt();
float PriorHighestGroupRange = -1.0f*FLT_MAX;

/* Find the group of x bars with the highest range. */
for (int i = 1; i <= LookBackLength - GroupSize + 1; i++)
{
    float GroupHighestHigh = sc.High[sc.Index - i];
    float GroupLowestLow = sc.Low[sc.Index - i];
    float GroupRange;

    /* Find range of single group within lookback period. */
    for (int j = i + 1; j < i + GroupSize; j++)
    {
        if (sc.High[sc.Index - j] > GroupHighestHigh)
            GroupHighestHigh = sc.High[sc.Index - j];
        if (sc.Low[sc.Index - j] < GroupLowestLow)
            GroupLowestLow = sc.Low[sc.Index - j];
    }

    GroupRange = GroupHighestHigh - GroupLowestLow;

    if (GroupRange > PriorHighestGroupRange)
        PriorHighestGroupRange = GroupRange;

    if (!sc.FormattedEvaluate(PriorHighestGroupRange, sc.BaseGraphValueFormat, LESS_EQUAL_OPERATOR,
RangeOfCurrentBarGroup, sc.BaseGraphValueFormat))
    {
        isHighest = 0.0f;
        break;
    }
}

Subgraph_WideRange.DataColor[sc.Index] = Subgraph_WideRange.PrimaryColor;
Subgraph_WideRange[sc.Index] = isHighest;

if (isHighest == 1.0f)
{
    for (int i = 1; i < GroupSize; i++)
    {
        Subgraph_WideRange.DataColor[sc.Index - i] = Subgraph_WideRange.SecondaryColor;
        Subgraph_WideRange[sc.Index - i] = isHighest;
    }
}
break;

case 2:
{
    sc.ATR(sc.BaseDataIn, Array_TR, Array_ATR, Input_LookBackLength_ATR.GetInt(),
Input_ATRMAType.GetMovAvgType());

    float CurrentBarRange = sc.High[sc.Index] - sc.Low[sc.Index];

    if (CurrentBarRange > Array_ATR[sc.Index - 1] && sc.Index >= Input_LookBackLength_ATR.GetInt())
    {
        Subgraph_WideRange[sc.Index] = 1.0f;
        Subgraph_WideRange.DataColor[sc.Index] = Subgraph_WideRange.PrimaryColor;
    }
    else
    {
        Subgraph_WideRange[sc.Index] = 0.0f;
    }
}
}

```

```

        break;
    }
}

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

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

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

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

        return;
    }

    Subgraph_BidVol[sc.Index] = sc.BidVolume[sc.Index];
}

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

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

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

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

        return;
    }

    Subgraph_AskVol[sc.Index] = sc.AskVolume[sc.Index];
}

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

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

        sc.ValueFormat = 0;
        sc.AutoLoop = 1;
        sc.MaintainAdditionalChartDataArrays = 1;

        Subgraph_UpTickVol.Name = "UpTick Volume";
    }
}

```

```

    Subgraph_UpTickVol.DrawStyle = DRAWSTYLE_BAR;
    Subgraph_UpTickVol.PrimaryColor = RGB(0,255,0);
    Subgraph_UpTickVol.DrawZeros = true;

    return;
}

Subgraph_UpTickVol[sc.Index] = sc.UpTickVolume[sc.Index];
}

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

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

        sc.ValueFormat = 0;
        sc.AutoLoop = 1;
        sc.MaintainAdditionalChartDataArrays = 1;

        Subgraph_DownTickVol.Name = "DownTick Volume";
        Subgraph_DownTickVol.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_DownTickVol.PrimaryColor = RGB(0,255,0);
        Subgraph_DownTickVol.DrawZeros = true;

        return;
    }

    Subgraph_DownTickVol[sc.Index] = sc.DownTickVolume[sc.Index];
}

/*=====*/
SCSFExport scsf_MaximumMinimumUpTickVolumeDowntickVolumeDiff(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_MinDifference = sc.Subgraph[0];
    SCSubgraphRef Subgraph_MaxDifference = sc.Subgraph[1];
    SCSubgraphRef Subgraph_MinBidAskVolumeDifference = sc.Subgraph[2];
    SCSubgraphRef Subgraph_MaxBidAskVolumeDifference = sc.Subgraph[3];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Maximum/Minimum UpTick Volume Downtick Volume Difference";

        sc.ValueFormat = 0;
        sc.AutoLoop = 1;
        sc.MaintainAdditionalChartDataArrays = 1;

        Subgraph_MinDifference.Name = "Minimum UpTick Volume Downtick Volume Difference";
        Subgraph_MinDifference.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_MinDifference.PrimaryColor = RGB(0,255,0);
        Subgraph_MinDifference.DrawZeros = true;

        Subgraph_MaxDifference.Name = "Maximum UpTick Volume Downtick Volume Difference";
        Subgraph_MaxDifference.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_MaxDifference.PrimaryColor = RGB(0,255, 255);
        Subgraph_MaxDifference.DrawZeros = true;

        Subgraph_MinBidAskVolumeDifference.Name = "Minimum Bid Ask Volume Difference";
        Subgraph_MinBidAskVolumeDifference.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_MinBidAskVolumeDifference.PrimaryColor = RGB(255,255,0);
        Subgraph_MinBidAskVolumeDifference.DrawZeros = true;
    }
}

```

```

Subgraph_MaxBidAskVolumeDifference.Name = "Maximum Bid Ask Volume Difference";
Subgraph_MaxBidAskVolumeDifference.DrawStyle = DRAWSTYLE_BAR;
Subgraph_MaxBidAskVolumeDifference.PrimaryColor = RGB(255, 0, 0);
Subgraph_MaxBidAskVolumeDifference.DrawZeros = true;

return;
}

Subgraph_MinDifference[sc.Index] = sc.BaseDataIn[SC_UPDOWN_VOL_DIFF_LOW][sc.Index];
Subgraph_MaxDifference[sc.Index] = sc.BaseDataIn[SC_UPDOWN_VOL_DIFF_HIGH][sc.Index];
Subgraph_MinBidAskVolumeDifference[sc.Index] = sc.BaseDataIn[SC_ASKBID_DIFF_LOW][sc.Index];
Subgraph_MaxBidAskVolumeDifference[sc.Index] = sc.BaseDataIn[SC_ASKBID_DIFF_HIGH][sc.Index];
}

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

    if (sc.SetDefaults)
    {
        sc.GraphName = "Number of Trades - Bid";

        sc.ValueFormat = 0;
        sc.AutoLoop = 1;
        sc.MaintainAdditionalChartDataArrays = 1;

        Subgraph_BidNT.Name = "Bid Trades";
        Subgraph_BidNT.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_BidNT.PrimaryColor = RGB(0,255,0);
        Subgraph_BidNT.DrawZeros = true;

        return;
    }

    Subgraph_BidNT[sc.Index] = sc.NumberOfBidTrades[sc.Index];
}

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

    if (sc.SetDefaults)
    {
        sc.GraphName = "Number of Trades - Ask";

        sc.ValueFormat = 0;
        sc.AutoLoop = 1;
        sc.MaintainAdditionalChartDataArrays = 1;

        Subgraph_AskNT.Name = "Ask Trades";
        Subgraph_AskNT.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_AskNT.PrimaryColor = RGB(0,255,0);
        Subgraph_AskNT.DrawZeros = true;

        return;
    }

    Subgraph_AskNT[sc.Index] = sc.NumberOfAskTrades[sc.Index];
}

/*=====*/
SCSFExport scsf_McClellanOscillator1Chart(SCStudyInterfaceRef sc)

```

```

{
    SCSubgraphRef Subgraph_MO = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Temp2 = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Temp3 = sc.Subgraph[3];

    SCInputRef UseAbsValue = sc.Input[0];

    if (sc.SetDefaults)
    {
        sc.GraphName = "McClellan Oscillator - 1 Chart" ;

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

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

        UseAbsValue.Name = "Use ABS Value.";
        UseAbsValue.SetYesNo(0);

        return;
    }

    sc.DataStartIndex = 1;

    Subgraph_Temp3[0] = Subgraph_Temp2[0] = sc.Close[0];

    if (sc.Index < sc.DataStartIndex)
        return;

    Subgraph_Temp3[sc.Index] = Subgraph_Temp3[sc.Index - 1] * 0.9f + sc.Close[sc.Index] * 0.1f;

    if (UseAbsValue.GetYesNo())
    {
        Subgraph_Temp2[sc.Index] = Subgraph_Temp2[sc.Index - 1] * 0.95f + fabs(sc.Close[sc.Index] * 0.05f);
    }
    else
    {
        Subgraph_Temp2[sc.Index] = Subgraph_Temp2[sc.Index - 1] * 0.95f + sc.Close[sc.Index] * 0.05f;
    }

    Subgraph_MO[sc.Index] = Subgraph_Temp3[sc.Index] - Subgraph_Temp2[sc.Index];
}

/*=====*/
SCSFExport scsf_McClellanSumIndex1Chart(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Sum = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Temp2 = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Temp3 = sc.Subgraph[3];

    if (sc.SetDefaults)
    {
        sc.GraphName = "McClellan Sum. Index - 1 Chart" ;

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

        Subgraph_Sum.Name = "Sum";
        Subgraph_Sum.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Sum.PrimaryColor = RGB(0,255,0);
    }
}

```

```

    Subgraph_Sum.DrawZeros = true;

    return;
}

sc.DataStartIndex = 1;

Subgraph_Temp3[0] = Subgraph_Temp2[0] = sc.Close[0];

if (sc.Index < sc.DataStartIndex)
    return;

if (sc.Close[sc.Index] == 0)
{
    Subgraph_Sum[sc.Index] = Subgraph_Sum[sc.Index - 1];
}
else
{
    Subgraph_Temp3[sc.Index] = Subgraph_Temp3[sc.Index - 1] * 0.9f + sc.Close[sc.Index] * 0.1f;
    Subgraph_Temp2[sc.Index] = Subgraph_Temp2[sc.Index - 1] * 0.95f + sc.Close[sc.Index] * 0.05f;

    Subgraph_Sum[sc.Index] = Subgraph_Sum[sc.Index - 1] + (Subgraph_Temp3[sc.Index] -
Subgraph_Temp2[sc.Index]);
}
}

/*=====*/
SCSFExport scsf_CumulativeSumOfStudy(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Sum = sc.Subgraph[0];
    SCInputRef Input_ResetAtSessionStart = sc.Input[2];
    SCInputRef Input_StudySubgraphReference = sc.Input[3];
    SCInputRef Input_Version = sc.Input[4];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Cumulative Sum of Study";
        sc.AutoLoop = 0;//Manual looping
        sc.GraphRegion = 1;
        sc.CalculationPrecedence = LOW_PREC_LEVEL;

        Subgraph_Sum.Name = "Sum";
        Subgraph_Sum.PrimaryColor = RGB(0,255,0);
        Subgraph_Sum.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Sum.LineWidth = 2;
        Subgraph_Sum.DrawZeros = false;

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

        Input_ResetAtSessionStart.Name = "Reset at Start of Trading Day";
        Input_ResetAtSessionStart.SetYesNo(0);

        Input_Version.SetInt(2);

        return;
    }

    SCFloatArray StudyArray;
    sc.GetStudyArrayUsingID(Input_StudySubgraphReference.GetStudyID(),
Input_StudySubgraphReference.GetSubgraphIndex(), StudyArray);
    if(StudyArray.GetArraySize() == 0)
    {
        sc.AddMessageToLog("Subgraph array being referenced is empty.", 1);
        return;
    }

```



```

}

sc.DataStartIndex = sc.GetStudyDataStartIndexUsingID(Input_StudySubgraphReference.GetStudyID());

if (StudyArray.GetArraySize() != sc.ArraySize)
{
    sc.AddMessageToLog("Subgraph array being referenced is not of the correct array size. Check the settings for the
'Study and Subgraph to Reference' Input.", 1);
    return;
}

SCDateTime NextSessionStart =
SCDateTime(sc.GetTradingDayStartDateTimeOfBar(sc.BaseDateTimeIn[sc.UpdateStartIndex - 1])) + SCDateTime(1, 0);

if (sc.UpdateStartIndex == 0)
{
    Subgraph_Sum[0] = StudyArray[0];
}

for (int Index = max(sc.UpdateStartIndex, 1); Index < sc.ArraySize; Index++)
{
    Subgraph_Sum[Index] = Subgraph_Sum[Index - 1] + StudyArray[Index];

    if (Input_ResetAtSessionStart.GetYesNo() != 0)
    {
        SCDateTime IndexDateTime = sc.BaseDateTimeIn[Index];

        if (IndexDateTime >= NextSessionStart)
        {
            Subgraph_Sum[Index] = StudyArray[Index];

            NextSessionStart = SCDateTime(sc.GetTradingDayStartDateTimeOfBar(IndexDateTime)) +
SCDateTime::DAYS(1);
        }
    }
}

}

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

    if (sc.SetDefaults)
    {
        sc.GraphName = "Cumulative Daily Volume";
        sc.AutoLoop = 0; //Manual looping
        sc.GraphRegion = 1;

        Subgraph_Sum.Name = "Volume";
        Subgraph_Sum.PrimaryColor = RGB(0, 255, 0);
        Subgraph_Sum.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Sum.LineWidth = 2;
        Subgraph_Sum.DrawZeros = false;

        return;
    }

    SCDateTime NextSessionStart =
SCDateTime(sc.GetTradingDayStartDateTimeOfBar(sc.BaseDateTimeIn[sc.UpdateStartIndex - 1])) +
SCDateTime::DAYS(1);

    if (sc.UpdateStartIndex == 0)
    {

```

```

    Subgraph_Sum[0] = sc.Volume[0];
}

for (int Index = max(sc.UpdateStartIndex, 1); Index < sc.ArraySize; Index++)
{
    SCDatetime IndexDateTime = sc.BaseDateTimeIn[Index];

    if (IndexDateTime >= NextSessionStart)
    {
        Subgraph_Sum[Index] = sc.Volume[Index];

        NextSessionStart = SCDatetime(sc.GetTradingDayStartDateTimeOfBar(IndexDateTime)) +
        SCDatetime::DAYS(1);
    }
    else
        Subgraph_Sum[Index] = Subgraph_Sum[Index - 1] + sc.Volume[Index];
}
}

/*=====*/
SCSFExport scsf_AverageContinuous(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_AverageResultSubgraph = sc.Subgraph[0];
    SCFloatArrayRef Array_Summation = sc.Subgraph[0].Arrays[0];
    SCFloatArrayRef Array_BarCount = sc.Subgraph[0].Arrays[1];

    SCInputRef Input_StudySubgraphReference = sc.Input[0];
    SCInputRef Input_ResetAtSessionStart = sc.Input[1];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Average - Continuous";
        sc.AutoLoop = 0; //Manual looping
        sc.GraphRegion = 1;
        sc.CalculationPrecedence = LOW_PREC_LEVEL;

        Subgraph_AverageResultSubgraph.Name = "Average";
        Subgraph_AverageResultSubgraph.PrimaryColor = RGB(0, 255, 0);
        Subgraph_AverageResultSubgraph.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_AverageResultSubgraph.LineWidth = 2;
        Subgraph_AverageResultSubgraph.DrawZeros = false;

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

        Input_ResetAtSessionStart.Name = "Reset at Start of Trading Day";
        Input_ResetAtSessionStart.SetYesNo(0);

        return;
    }

    SCFloatArray StudyArray;
    sc.GetStudyArrayUsingID(Input_StudySubgraphReference.GetStudyID(),
    Input_StudySubgraphReference.GetSubgraphIndex(), StudyArray);
    if (StudyArray.GetArraySize() == 0)
    {
        sc.AddMessageToLog("Subgraph array being referenced is empty.", 1);
        return;
    }

    sc.DataStartIndex = sc.GetStudyDataStartIndexUsingID(Input_StudySubgraphReference.GetStudyID());

```

```

if (StudyArray.GetArraySize() != sc.ArraySize)
{
    sc.AddMessageToLog("Subgraph array being referenced is not of the correct array size. Check the settings for the
'Study and Subgraph to Reference' Input.", 1);
    return;
}

SCDateTime NextSessionStart;
if (Input_ResetAtSessionStart.GetYesNo() != 0)
    NextSessionStart = SCDateTime(sc.GetTradingDayStartDateTimeOfBar(sc.BaseDateTimeIn[max(0,
sc.UpdateStartIndex - 1)])) + SCDateTime::DAYS(1);

if (sc.UpdateStartIndex == 0)
{
    Array_Summation[0] = StudyArray[0];
    Array_BarCount[0] = 1;
}

for (int Index = max(sc.UpdateStartIndex, 1); Index < sc.ArraySize; Index++)
{
    Array_Summation[Index] = Array_Summation[Index - 1] + StudyArray[Index];
    Array_BarCount[Index] = Array_BarCount[Index-1] + 1;

    if (Input_ResetAtSessionStart.GetYesNo() != 0)
    {
        SCDateTime IndexDateTime = sc.BaseDateTimeIn[Index];

        if (IndexDateTime >= NextSessionStart)
        {
            Array_Summation[Index] = StudyArray[Index];
            Array_BarCount[Index] = 1;
            NextSessionStart = SCDateTime(sc.GetTradingDayStartDateTimeOfBar(IndexDateTime)) +
SCDateTime::DAYS(1);
        }
    }

    Subgraph_AverageResultSubgraph [Index] = Array_Summation[Index] / Array_BarCount[Index];
}
}

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

    SCInputRef Input_StartTime = sc.Input[0];
    SCInputRef Input_InputDataForCurrentPrice = sc.Input[1];
    SCInputRef Input_Version = sc.Input[2];
    SCInputRef Input_Multiplier = sc.Input[3];
    SCInputRef Input_InputDataForOpeningBar = sc.Input[4];
    SCInputRef Input_TimePeriodType = sc.Input[5];
    SCInputRef Input_TimePeriodLength = sc.Input[6];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Percent Change Since Open";
        sc.AutoLoop = 0;
        sc.GraphRegion = 1;
        sc.ValueFormat = VALUEFORMAT_PERCENT;

        Subgraph_PercentChange.Name = "Percent Change";
        Subgraph_PercentChange.PrimaryColor = RGB(0,255,0);
    }
}

```

```

Subgraph_PercentChange.DrawStyle = DRAWSTYLE_LINE;
Subgraph_PercentChange.LineWidth = 1;
Subgraph_PercentChange.DrawZeros= true;

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

int DisplayOrder = 1;

Input_StartTime.Name = "Start Time";
Input_StartTime.SetCustomInputStrings("Time Period Type/Length;First Bar In Chart");
Input_StartTime.SetCustomInputIndex(0);
Input_StartTime.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_InputDataForOpeningBar.Name = "Input Data for Opening Bar";
Input_InputDataForOpeningBar.SetInputDataIndex(SC_OPEN);
Input_InputDataForOpeningBar.DisplayOrder = DisplayOrder++;

Input_InputDataForCurrentPrice.Name = "Input Data for Current Price";
Input_InputDataForCurrentPrice.SetInputDataIndex(SC_LAST);
Input_InputDataForCurrentPrice.DisplayOrder = DisplayOrder++;

Input_Multiplier.Name = "Multiplier";
Input_Multiplier.SetFloat(1);
Input_Multiplier.DisplayOrder = DisplayOrder++;

Input_Version.SetInt(4);

return;
}

if (Input_Version.GetInt() < 2)
{
    Input_Version.SetInt(2);
    Input_InputDataForCurrentPrice.SetInputDataIndex(SC_LAST);
    sc.ValueFormat = VALUEFORMAT_PERCENT;
}

if (Input_Version.GetInt() < 3)
{
    Input_Version.SetInt(3);
    Input_Multiplier.SetFloat(1.0);
}

if (Input_Version.GetInt() < 4)
{
    Input_Version.SetInt(4);
    Input_InputDataForOpeningBar.SetInputDataIndex(SC_OPEN);
    Input_TimePeriodType.SetTimePeriodType(TIME_PERIOD_LENGTH_UNIT_DAYS);
}

float& r_CurrentPeriodOpenValue = sc.GetPersistentFloat(1);
SCDateTime& r_PriorCurrentPeriodStartDateTime = sc.GetPersistentSCDateTime(2);

```

```

if (sc.IsFullRecalculation)
    r_PriorCurrentPeriodStartDateTime.Clear();

for (int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize; BarIndex++)
{
    if (BarIndex == 0)
    {
        r_CurrentPeriodOpenValue = sc.BaseData[Input_InputDataForOpeningBar.GetInputDataIndex()][BarIndex];
    }

    if (Input_StartTime.GetIndex() == 0)
    {
        SCDateTime CurrentPeriodBeginDateTime = sc.GetStartOfPeriodForDateTime(sc.BaseDateIn[BarIndex],
Input_TimePeriodType.GetTimePeriodType(), Input_TimePeriodLength.GetInt(), 0);

        if (CurrentPeriodBeginDateTime != r_PriorCurrentPeriodStartDateTime)
        {
            r_CurrentPeriodOpenValue = sc.BaseData[Input_InputDataForOpeningBar.GetInputDataIndex()][BarIndex];
        }

        r_PriorCurrentPeriodStartDateTime = CurrentPeriodBeginDateTime;
    }

    if (r_CurrentPeriodOpenValue == 0)
        Subgraph_PercentChange[BarIndex] = 0;
    else
    {
        float CurrentValue = sc.BaseData[Input_InputDataForCurrentPrice.GetInputDataIndex()][BarIndex];

        Subgraph_PercentChange[BarIndex] = (CurrentValue - r_CurrentPeriodOpenValue) / r_CurrentPeriodOpenValue
* Input_Multiplier.GetFloat();
    }
}

}

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

    SCInputRef Input_Data = sc.Input[1];
    SCInputRef Input_Version = sc.Input[2];
    SCInputRef Input_Multiplier = sc.Input[3];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Percent Change Since Previous Close";
        sc.AutoLoop = 1;
        sc.GraphRegion = 1;
        sc.ValueFormat = VALUEFORMAT_PERCENT;

        Subgraph_PercentChange.Name = "Percent Change";
        Subgraph_PercentChange.PrimaryColor = RGB(0,255,0);
        Subgraph_PercentChange.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_PercentChange.LineWidth = 1;
        Subgraph_PercentChange.DrawZeros= true;

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

```

```

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

Input_Version.SetInt(3);

Input_Multiplier.Name = "Multiplier";
Input_Multiplier.SetFloat(1);

return;
}

if (Input_Version.GetInt() < 2)
{
    Input_Version.SetInt(2);
    Input_Data.SetInputDataIndex(SC_LAST);
    sc.ValueFormat = VALUEFORMAT_PERCENT;
}

if (Input_Version.GetInt() < 3)
{
    Input_Version.SetInt(3);
    Input_Multiplier.SetFloat(1);
}

float& PreviousCloseValue = sc.GetPersistentFloat(1);
SCDateTime& SessionStart = sc.GetPersistentSCDateTime(1);

if (sc.Index == 0)
{
    PreviousCloseValue = sc.BaseData[Input_Data.GetInputDataIndex()][sc.Index];
    SessionStart = sc.GetTradingDayStartDateTimeOfBar(sc.BaseDateTimeIn[sc.Index]);
}

SCDateTime BarSessionStart = sc.GetTradingDayStartDateTimeOfBar(sc.BaseDateTimeIn[sc.Index]);

if (BarSessionStart != SessionStart)
{
    PreviousCloseValue = sc.BaseData[Input_Data.GetInputDataIndex()][sc.Index - 1];

    SessionStart = BarSessionStart;
}

if (PreviousCloseValue == 0)
    Subgraph_PercentChange[sc.Index] = 0;
else
{
    Subgraph_PercentChange[sc.Index] = (sc.BaseData[Input_Data.GetInputDataIndex()][sc.Index] -
PreviousCloseValue) / PreviousCloseValue * Input_Multiplier.GetFloat();
}
}

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

    SCInputRef Input_StartTimeType = sc.Input[0];

    SCInputRef Input_TimePeriodType = sc.Input[1];
    SCInputRef Input_TimePeriodLength = sc.Input[2];
    SCInputRef Input_InputDataForOpeningBar = sc.Input[3];
    SCInputRef Input_InputDataForCurrentPrice = sc.Input[4];
    SCInputRef Input_Multiplier = sc.Input[5];
    SCInputRef Input_Version = sc.Input[6];

```

```

if (sc.SetDefaults)
{
    sc.GraphName = "Price Change Since Open";
    sc.AutoLoop = 0;
    sc.GraphRegion = 1;
    sc.ValueFormat = VALUEFORMAT_INHERITED;

    Subgraph_PriceChange.Name = "Price Change";
    Subgraph_PriceChange.PrimaryColor = RGB(0, 255, 0);
    Subgraph_PriceChange.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_PriceChange.LineWidth = 1;
    Subgraph_PriceChange.DrawZeros = true;

    int DisplayOrder = 1;

    Input_StartTimeType.Name = "Start Time Type";
    Input_StartTimeType.SetCustomInputStrings("Time Period Type/Length;First Bar In Chart");
    Input_StartTimeType.SetCustomInputIndex(0);
    Input_StartTimeType.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_InputDataForOpeningBar.Name = "Input Data for Opening Bar";
    Input_InputDataForOpeningBar.SetInputDataIndex(SC_OPEN);
    Input_InputDataForOpeningBar.DisplayOrder = DisplayOrder++;

    Input_InputDataForCurrentPrice.Name = "Input Data for Current Price";
    Input_InputDataForCurrentPrice.SetInputDataIndex(SC_LAST);
    Input_InputDataForCurrentPrice.DisplayOrder = DisplayOrder++;

    Input_Multiplier.Name = "Multiplier";
    Input_Multiplier.SetFloat(1);
    Input_Multiplier.DisplayOrder = DisplayOrder++;

    Input_Version.SetInt(4);

    return;
}

float& r_CurrentPeriodOpenValue = sc.GetPersistentFloat(1);
SCDateTime& r_PriorCurrentPeriodStartDateTime = sc.GetPersistentSCDateTime(2);

if (sc.IsFullRecalculation)
    r_PriorCurrentPeriodStartDateTime.Clear();

for (int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize; BarIndex++)
{
    if (BarIndex == 0)
    {
        r_CurrentPeriodOpenValue = sc.BaseData[Input_InputDataForOpeningBar.GetInputDataIndex()][BarIndex];
    }

    if (Input_StartTimeType.GetIndex() == 0)
    {
        SCDateTime CurrentPeriodBeginDateTime = sc.GetStartOfPeriodForDateTime(sc.BaseDateTimeIn[BarIndex],
        Input_TimePeriodType.GetTimePeriodType(), Input_TimePeriodLength.GetInt(), 0);

        if (CurrentPeriodBeginDateTime != r_PriorCurrentPeriodStartDateTime)

```

```

    {
        r_CurrentPeriodOpenValue = sc.BaseData[Input_InputDataForOpeningBar.GetInputDataIndex()][BarIndex];
    }

    r_PriorCurrentPeriodStartDateTime = CurrentPeriodBeginDateTime;
}

if (r_CurrentPeriodOpenValue == 0)
    Subgraph_PriceChange[BarIndex] = 0;
else
{
    float CurrentValue = sc.BaseData[Input_InputDataForCurrentPrice.GetInputDataIndex()][BarIndex];

    Subgraph_PriceChange[BarIndex] = (CurrentValue - r_CurrentPeriodOpenValue) * Input_Multiplier.GetFloat();
}
}
}

/*=====*/
double PointAndFigureAddBoxes(SCStudyInterfaceRef sc, double Value, double BoxSize, int Direction, float TickSize)
{
    double Epsilon = 0.5 * TickSize;

    if (Direction == 1) //up
    {
        int NumberOfBoxes = static_cast<int>(floor(Value / BoxSize + Epsilon));
        return (NumberOfBoxes * BoxSize);
    }
    else if (Direction == -1) //down
    {
        int NumberOfBoxes = static_cast<int>(ceil(Value / BoxSize - Epsilon));
        return (NumberOfBoxes * BoxSize);
    }
    else //unknown
        return sc.Round(static_cast<float>(Value / BoxSize)) * BoxSize;

    return 0;
}

/*=====*/
SCSFExport scsf_PointAndFigureChart(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Open      = sc.Subgraph[SC_OPEN];
    SCSubgraphRef Subgraph_High      = sc.Subgraph[SC_HIGH];
    SCSubgraphRef Subgraph_Low       = sc.Subgraph[SC_LOW];
    SCSubgraphRef Subgraph_Last      = sc.Subgraph[SC_LAST];
    SCSubgraphRef Subgraph_Volume    = sc.Subgraph[SC_VOLUME];
    SCSubgraphRef Subgraph_NumTrades = sc.Subgraph[SC_NUM_TRADES];
    SCSubgraphRef Subgraph_OHLCAvg   = sc.Subgraph[SC_OHLC_AVG];
    SCSubgraphRef Subgraph_HLCAvg    = sc.Subgraph[SC_HLC_AVG];
    SCSubgraphRef Subgraph_HLAvG     = sc.Subgraph[SC_HL_AVG];
    SCSubgraphRef Subgraph_BidVol    = sc.Subgraph[SC_BIDVOL];
    SCSubgraphRef Subgraph_AskVol    = sc.Subgraph[SC_ASKVOL];
    SCSubgraphRef Subgraph_UpTickVol = sc.Subgraph[SC_UPVOL];
    SCSubgraphRef Subgraph_DownTickVol = sc.Subgraph[SC_DOWNVOL];
    SCSubgraphRef Subgraph_BidTrades = sc.Subgraph[SC_BIDNT];
    SCSubgraphRef Subgraph_AskTrades = sc.Subgraph[SC_ASKNT];
    SCSubgraphRef Subgraph_BidAskDiffMax = sc.Subgraph[SC_ASKBID_DIFF_HIGH];
    SCSubgraphRef Subgraph_BidAskDiffMin = sc.Subgraph[SC_ASKBID_DIFF_LOW];
    SCSubgraphRef Subgraph_NumberOfBoxes = sc.Subgraph[PF_NUM_BOXES_ARRAY];
    SCSubgraphRef Subgraph_DirectionArray = sc.Subgraph[PF_DIRECTION_ARRAY];
    SCSubgraphRef Subgraph_TrueLast     = sc.Subgraph[PF_TRUELAST_ARRAY];

    const int ArrayCount = PF_TRUELAST_ARRAY + 1;

```



```

SCInputRef Input_BoxSize           = sc.Input[0];
SCInputRef Input_ReversalSize      = sc.Input[1];
SCInputRef Input_AllowOneBoxReversals = sc.Input[2];
SCInputRef Input_NewBarAtStartOfDay = sc.Input[3];
SCInputRef Input_IgnoreLastBarUntilComplete = sc.Input[4];
SCInputRef Input_UsePercentageForBoxSize = sc.Input[5];
SCInputRef Input_BoxSizePercentage = sc.Input[6];
SCInputRef Input_OnlyUseClosePrices = sc.Input[8];
SCInputRef Input_Version           = sc.Input[9];

if (sc.SetDefaults)
{
    sc.GraphName = "Point & Figure Chart";
    sc.AutoLoop = 0;
    sc.GraphRegion = 0;
    sc.StandardChartHeader = 1;
    sc.IsCustomChart = 1;
    sc.GraphDrawType = GDT_POINT_AND_FIGURE_BARS;//GDT_POINT_AND_FIGURE_XO
    sc.MaintainAdditionalChartDataArrays = 1;
    sc.ValueFormat = VALUEFORMAT_INHERITED;

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

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

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

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

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

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

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

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

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

```

```

Subgraph_HLAvg.DrawZeros = false;

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

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

Subgraph_NumberOfBoxes.Name = "Number of Boxes";
Subgraph_NumberOfBoxes.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_NumberOfBoxes.PrimaryColor = RGB(0xFF,0x99,0x00);
Subgraph_NumberOfBoxes.DrawZeros = false;

Subgraph_TrueLast.Name = "True Last";
Subgraph_TrueLast.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_TrueLast.PrimaryColor = RGB(255,255,255);
Subgraph_TrueLast.DrawZeros = false;

Input_BoxSize.Name = "Box Size (in ticks)";
Input_BoxSize.SetFloat(0.0f);
Input_BoxSize.SetFloatLimits(0, FLT_MAX);

Input_ReversalSize.Name = "Reversal Size (num boxes)";
Input_ReversalSize.SetFloat(3.0f);
Input_ReversalSize.SetFloatLimits(FLT_MIN, FLT_MAX);

Input_AllowOneBoxReversals.Name = "Allow One Box Reversals";
Input_AllowOneBoxReversals.SetYesNo(1);

Input_NewBarAtStartOfDay.Name = "New Bar at Start of Day";
Input_NewBarAtStartOfDay.SetYesNo(0);

Input_IgnoreLastBarUntilComplete.Name = "Ignore Last Bar Until Completed";
Input_IgnoreLastBarUntilComplete.SetYesNo(0);

Input_UsePercentageForBoxSize.Name = "Use Percentage of Last Price for Box Size";
Input_UsePercentageForBoxSize.SetYesNo(0);

Input_BoxSizePercentage.Name = "Box Size Percentage (%)";
Input_BoxSizePercentage.SetFloat(1.0f);
Input_BoxSizePercentage.SetFloatLimits(0.0001f, 5000.0f);

Input_OnlyUseClosePrices.Name = "Calculate Only Using Close Prices";
Input_OnlyUseClosePrices.SetYesNo(0);

Input_Version.SetInt(1);

return;
}

if (Input_Version.GetInt() < 1)
{
    Input_Version.SetInt(1);

    // convert box size value to ticks
    if (sc.TickSize != 0)
        Input_BoxSize.SetFloat(Input_BoxSize.GetFloat() / sc.TickSize);
}

const int PersistentPriorOutDataBaseKey = 100;

```

```

int& r_PersistentPriorOutArraySize = sc.GetPersistentInt(1);

SCDateTime& NextSessionStart = sc.GetPersistentSCDateTime(1);

int InputBarIndex = sc.UpdateStartIndex;
int AvgStartIndex = 0;

float& r_CurrentBoxSize = sc.PointAndFigureXOGraphDrawTypeBoxSize;

if (InputBarIndex == 0)
{
    r_CurrentBoxSize = Input_BoxSize.GetFloat() * sc.TickSize;

    if (Input_UsePercentageForBoxSize.GetYesNo())
        r_CurrentBoxSize = sc.BaseData[SC_LAST][sc.ArraySize-1] * Input_BoxSizePercentage.GetFloat() * 0.01f;

    r_CurrentBoxSize = sc.RoundToTickSize(r_CurrentBoxSize, sc.TickSize);

    if (r_CurrentBoxSize == 0.0)
    {
        if (sc.TickSize != 0)
            r_CurrentBoxSize = sc.TickSize;
        else
            r_CurrentBoxSize = 1;

        Input_BoxSize.SetFloat(static_cast<float>(sc.Round(r_CurrentBoxSize / sc.TickSize)));
    }

    sc.ResizeArrays(0);

    AvgStartIndex = 0;

    r_PersistentPriorOutArraySize = 0;
    for (int Index = 0; Index < ArrayCount; ++Index)
    {
        sc.GetPersistentFloat(Index + PersistentPriorOutDataBaseKey) = 0;
    }

    if (!sc.AddElements(1))
        return;

    sc.DateTimeOut[0] = sc.BaseDateTimeln[0];

    Subgraph_Open[0] = sc.Open[0];
    Subgraph_High[0] = static_cast<float>(sc.RoundToTickSize(sc.Close[0], r_CurrentBoxSize)); //round off to the
nearest box size
    Subgraph_Low[0] = static_cast<float>(sc.RoundToTickSize(sc.Close[0], r_CurrentBoxSize)); //round off to the
nearest box size
    Subgraph_Last[0] = sc.Close[0];
    Subgraph_TrueLast[0] = sc.Close[0];

    NextSessionStart = SCDateTime(sc.GetTradingDayStartDateOfBar(sc.BaseDateTimeln[0])) +
SCDateTime::DAYS(1);

    if (Input_OnlyUseClosePrices.GetYesNo())
        Input_IgnoreLastBarUntilComplete.SetYesNo(1);

    if (sc.ChartDataType == DAILY_DATA)
        Subgraph_NumTrades.Name = "Open Interest";
    else
        Subgraph_NumTrades.Name = "# of Trades";

    sc.GraphName.Format("Point & Figure Chart %.6gx%.6g %s",

```

```

    r_CurrentBoxSize,
    Input_ReversalSize.GetFloat(),
    sc.GetStudyNameFromChart(sc.ChartNumber, 0).GetChars());
}
else if (!Input_IgnoreLastBarUntilComplete.GetYesNo())
{
    AvgStartIndex = r_PersistentPriorOutArraySize - 1;

    // Restore array to size just before processing last input bar (sc.BaseDataIn) and restore state of last output bar.
    sc.ResizeArrays(r_PersistentPriorOutArraySize);

    for (int SubgraphIndex = 0; SubgraphIndex < ArrayCount; ++SubgraphIndex)
    {
        sc.Subgraph[SubgraphIndex][AvgStartIndex] = sc.GetPersistentFloat(SubgraphIndex +
PersistentPriorOutDataBaseKey);
    }
}

int OutputIndex = sc.DateTimeOut.GetArraySize() - 1;
if (InputBarIndex == 0)
    InputBarIndex = 1;

SCFloatArrayRef OpenInputData = Input_OnlyUseClosePrices.GetYesNo() != 0 ? sc.Close : sc.Open;
SCFloatArrayRef HighInputData = Input_OnlyUseClosePrices.GetYesNo() != 0 ? sc.Close : sc.High;
SCFloatArrayRef LowInputData = Input_OnlyUseClosePrices.GetYesNo() != 0 ? sc.Close : sc.Low;
SCFloatArrayRef CloseInputData = sc.Close;

for (; InputBarIndex < sc.ArraySize; ++InputBarIndex)
{
    bool NewBar = false;

    if (Input_IgnoreLastBarUntilComplete.GetYesNo() && sc.GetBarHasClosedStatus(InputBarIndex) ==
BHCS_BAR_HAS_NOT_CLOSED)
        return;

    if (Input_NewBarAtStartOfDay.GetYesNo() != 0)
    {
        SCDatetime IndexDateTime = sc.BaseDateTimeIn[InputBarIndex];

        if (IndexDateTime >= NextSessionStart)
        {
            sc.CalculateOHLCAverages(OutputIndex);

            if (!sc.AddElements(1))
                return;

            NewBar = true;
            OutputIndex = sc.DateTimeOut.GetArraySize() - 1;

            sc.DateTimeOut[OutputIndex] = sc.BaseDateTimeIn[InputBarIndex];

            Subgraph_Open[OutputIndex] = OpenInputData[InputBarIndex];
            Subgraph_High[OutputIndex] = static_cast<float>(sc.RoundToTickSize(CloseInputData[InputBarIndex],
r_CurrentBoxSize)); //round off to the nearest box size
            Subgraph_Low[OutputIndex] = static_cast<float>(sc.RoundToTickSize(CloseInputData[InputBarIndex],
r_CurrentBoxSize)); //round off to the nearest box size
            Subgraph_Last[OutputIndex] = CloseInputData[InputBarIndex];
            Subgraph_TrueLast[OutputIndex] = CloseInputData[InputBarIndex];

            NextSessionStart = SCDatetime(sc.GetTradingDayStartDateTimeOfBar(IndexDateTime)) +
SCDatetime::DAYS(1);
        }
    }

    // Remember state of last P&F bar when reach last input bar because last

```

```

// input bar can change due to updating, causing for example a new P&F bar
// to be added when on a later update no new bar is added.
if (!Input_IgnoreLastBarUntilComplete.GetYesNo() && InputBarIndex == sc.ArraySize - 1)
{
    r_PersistentPriorOutArraySize = OutputIndex + 1;

    for (int SubgraphIndex = 0; SubgraphIndex < ArrayCount; ++SubgraphIndex)
    {
        sc.GetPersistentFloat(SubgraphIndex + PersistentPriorOutDataBaseKey) = sc.Subgraph[SubgraphIndex]
[OutputIndex];
    }
}

// Determine our initial direction if not known
if (Subgraph_DirectionArray[OutputIndex] == 0.0f)
{
    if (Subgraph_Last[OutputIndex] > CloseInputData[InputBarIndex])
        Subgraph_DirectionArray[OutputIndex] = -1; // Down

    else if (Subgraph_Last[OutputIndex] < CloseInputData[InputBarIndex])
        Subgraph_DirectionArray[OutputIndex] = 1; // Up
}

if (Subgraph_DirectionArray[OutputIndex] == 0.0f)
    continue; // No direction

bool UseHighFirst = false;

int OpenToHighLow = sc.CompareOpenToHighLow(OpenInputData[InputBarIndex], HighInputData[InputBarIndex],
LowInputData[InputBarIndex], sc.BaseGraphValueFormat);

if (OpenToHighLow == 1)
    UseHighFirst = true;
else if (OpenToHighLow == -1)
    UseHighFirst = false;
else if (sc.FormattedEvaluate(OpenInputData[InputBarIndex], sc.BaseGraphValueFormat,
GREATER_OPERATOR, CloseInputData[InputBarIndex], sc.BaseGraphValueFormat))
    UseHighFirst = true;
else if (sc.FormattedEvaluate(OpenInputData[InputBarIndex], sc.BaseGraphValueFormat, LESS_OPERATOR,
CloseInputData[InputBarIndex], sc.BaseGraphValueFormat))
    UseHighFirst = false;
else if (Subgraph_DirectionArray[OutputIndex] == 1) // Up Bar
    UseHighFirst = true;
else
    UseHighFirst = false;

for (int PassNumber = 0; PassNumber < 2; PassNumber++)
{
    if (Subgraph_DirectionArray[OutputIndex] == 1) // Up bar
    {
        if ((UseHighFirst && PassNumber == 0) || (!UseHighFirst && PassNumber == 1))
        {
            if (sc.FormattedEvaluate(HighInputData[InputBarIndex], sc.BaseGraphValueFormat,
GREATER_EQUAL_OPERATOR, Subgraph_High[OutputIndex] + r_CurrentBoxSize, sc.BaseGraphValueFormat))
            {
                Subgraph_High[OutputIndex] = static_cast<float>(PointAndFigureAddBoxes(sc,
HighInputData[InputBarIndex], r_CurrentBoxSize, 1, sc.TickSize));
            }
        }
    }

    if ((PassNumber == 0) || (UseHighFirst && PassNumber == 1))
    {
        if (sc.FormattedEvaluate(LowInputData[InputBarIndex], sc.BaseGraphValueFormat,
LESS_EQUAL_OPERATOR, Subgraph_High[OutputIndex] - r_CurrentBoxSize * Input_ReversalSize.GetFloat(),

```

```

sc.BaseGraphValueFormat))
{
    Subgraph_Last[OutputIndex] = Subgraph_High[OutputIndex];

    bool SetOpen = false;

    if (Input_AllowOneBoxReversals.GetYesNo() //If one box is allowed upon a reversal
        //Column contains at least two boxes. One box would have a zero range. Two boxes has a range of
r_CurrentBoxSize.
        || sc.FormattedEvaluate((Subgraph_High[OutputIndex] - Subgraph_Low[OutputIndex]),
sc.BaseGraphValueFormat, GREATER_EQUAL_OPERATOR, r_CurrentBoxSize, sc.BaseGraphValueFormat)
        )
    {
        // Calculate the number of boxes for the column
        Subgraph_NumberOfBoxes[OutputIndex] = static_cast<float>
(sc.Round((Subgraph_High[OutputIndex] - Subgraph_Low[OutputIndex]) / r_CurrentBoxSize + 1));

        sc.CalculateOHLCAverages(OutputIndex);

        if (!sc.AddElements(1))
            return;

        NewBar = true;
        OutputIndex++;
        sc.DateTimeOut[OutputIndex] = sc.BaseDateTimeIn[InputBarIndex];

        Subgraph_High[OutputIndex] = Subgraph_High[OutputIndex - 1] - r_CurrentBoxSize;

        SetOpen = true;
    }

    Subgraph_DirectionArray[OutputIndex] = -1;
    Subgraph_Low[OutputIndex] = static_cast<float>(PointAndFigureAddBoxes(sc,
LowInputData[InputBarIndex], r_CurrentBoxSize, -1, sc.TickSize));

    if (SetOpen)
        Subgraph_Open[OutputIndex] = Subgraph_Low[OutputIndex];
}
}

if (UseHighFirst)
    PassNumber++; // Skip next pass
}
else// down column
{
    if ((!UseHighFirst && PassNumber == 0) || (UseHighFirst && PassNumber == 1))
    {
        if (sc.FormattedEvaluate(LowInputData[InputBarIndex] , sc.BaseGraphValueFormat,
LESS_EQUAL_OPERATOR, Subgraph_Low[OutputIndex] - r_CurrentBoxSize, sc.BaseGraphValueFormat))
        {
            Subgraph_Low[OutputIndex] = static_cast<float>(PointAndFigureAddBoxes(sc,
LowInputData[InputBarIndex], r_CurrentBoxSize, -1, sc.TickSize));
        }
    }

    if (PassNumber == 0 || (!UseHighFirst && PassNumber == 1))
    {
        if (sc.FormattedEvaluate(HighInputData[InputBarIndex] , sc.BaseGraphValueFormat,
GREATER_EQUAL_OPERATOR, Subgraph_Low[OutputIndex] + r_CurrentBoxSize * Input_ReversalSize.GetFloat(),
sc.BaseGraphValueFormat))
        {
            Subgraph_Last[OutputIndex] = Subgraph_Low[OutputIndex];

```

```

        bool SetOpen = false;
        if (Input_AllowOneBoxReversals.GetYesNo()
            || sc.FormattedEvaluate((Subgraph_High[OutputIndex] - Subgraph_Low[OutputIndex]),
            sc.BaseGraphValueFormat, GREATER_EQUAL_OPERATOR, r_CurrentBoxSize, sc.BaseGraphValueFormat)
        )
        {
            // Calculate the number of boxes for the column
            Subgraph_NumberOfBoxes[OutputIndex] = static_cast<float>
(sc.Round((Subgraph_High[OutputIndex] - Subgraph_Low[OutputIndex]) / r_CurrentBoxSize + 1));

            sc.CalculateOHLCAverages(OutputIndex);

            if (!sc.AddElements(1))
                return;

            NewBar = true;
            OutputIndex++;
            sc.DateTimeOut[OutputIndex] = sc.BaseDateTimeIn[InputBarIndex];

            Subgraph_Low[OutputIndex] = Subgraph_Low[OutputIndex - 1] + r_CurrentBoxSize;

            SetOpen = true;
        }

        Subgraph_DirectionArray[OutputIndex] = 1;
        Subgraph_High[OutputIndex] = static_cast<float>(PointAndFigureAddBoxes(sc,
HighInputData[InputBarIndex], r_CurrentBoxSize, 1, sc.TickSize));

        if (SetOpen)
            Subgraph_Open[OutputIndex] = Subgraph_High[OutputIndex];
    }
}

if (!UseHighFirst)
    PassNumber++; // Skip next pass

} //else down column
} // PassNumber for loop

Subgraph_TrueLast[OutputIndex] = CloseInputData[InputBarIndex];
Subgraph_Last[OutputIndex] = CloseInputData[InputBarIndex];
sc.CalculateOHLCAverages(OutputIndex);
Subgraph_NumberOfBoxes[OutputIndex] = static_cast<float>(sc.Round((Subgraph_High[OutputIndex] -
Subgraph_Low[OutputIndex]) / r_CurrentBoxSize + 1));

// Add in volume and open interest
Subgraph_Volume[OutputIndex] += sc.Volume[InputBarIndex];
Subgraph_NumTrades[OutputIndex] += sc.OpenInterest[InputBarIndex];
float BidAskDiffHigh = 0;
float BidAskDiffLow = 0;
if (sc.BaseDataIn.GetArraySize() >= SC_ASKBID_DIFF_LOW+1)
{
    BidAskDiffHigh = Subgraph_AskVol[OutputIndex] - Subgraph_BidVol[OutputIndex] +
sc.BaseDataIn[SC_ASKBID_DIFF_HIGH][InputBarIndex];
    BidAskDiffLow = Subgraph_AskVol[OutputIndex] - Subgraph_BidVol[OutputIndex] +
sc.BaseDataIn[SC_ASKBID_DIFF_LOW][InputBarIndex];
}
Subgraph_BidVol[OutputIndex] += sc.BidVolume[InputBarIndex];
Subgraph_AskVol[OutputIndex] += sc.AskVolume[InputBarIndex];
Subgraph_UpTickVol[OutputIndex] += sc.UpTickVolume[InputBarIndex];
Subgraph_DownTickVol[OutputIndex] += sc.DownTickVolume[InputBarIndex];
Subgraph_BidTrades[OutputIndex] += sc.NumberOfBidTrades[InputBarIndex];
Subgraph_AskTrades[OutputIndex] += sc.NumberOfAskTrades[InputBarIndex];

if (NewBar || BidAskDiffHigh > Subgraph_BidAskDiffMax[OutputIndex])

```



```

Subgraph_BidAskDiffMax[OutputIndex] = BidAskDiffHigh;

if (NewBar || BidAskDiffLow < Subgraph_BidAskDiffMin[OutputIndex])
    Subgraph_BidAskDiffMin[OutputIndex] = BidAskDiffLow;
}
}

/*=====*/
SCSFExport scsf_StudyOverlayOHLC(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Open = sc.Subgraph[0];
    SCSubgraphRef Subgraph_High = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Low = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Last = sc.Subgraph[3];
    SCSubgraphRef Subgraph_OHLCAvg = sc.Subgraph[6];
    SCSubgraphRef Subgraph_HLCAvg = sc.Subgraph[7];
    SCSubgraphRef Subgraph_HLAvg = sc.Subgraph[8];

    SCInputRef Input_UseZeroValuesFromSource = sc.Input[0];
    SCInputRef Input_Multiplier = sc.Input[3];
    SCInputRef Input_ChartStudySubgraphReference = sc.Input[4];
    SCInputRef Input_Version = sc.Input[5];
    SCInputRef Input_DrawZeros = sc.Input[6];
    SCInputRef Input_UseStandardOHLCSubgraphsFromSourceStudy = sc.Input[7];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Study Overlay - OHLC";

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

        sc.AutoLoop = 0;

        Subgraph_Open.Name = "Open";
        Subgraph_Open.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Open.PrimaryColor = RGB(0,255,0);
        Subgraph_Open.SecondaryColor = RGB(0,255,0);
        Subgraph_Open.SecondaryColorUsed = 1;
        Subgraph_Open.DrawZeros = false;
        Subgraph_Open.LineWidth = 1;

        Subgraph_High.Name = "High";
        Subgraph_High.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_High.PrimaryColor = RGB(128,255,128);
        Subgraph_High.DrawZeros = false;
        Subgraph_High.LineWidth = 1;

        Subgraph_Low.Name = "Low";
        Subgraph_Low.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Low.PrimaryColor = RGB(255,0,0);
        Subgraph_Low.SecondaryColor = RGB(255,0,0);
        Subgraph_Low.SecondaryColorUsed = 1;
        Subgraph_Low.DrawZeros = false;
        Subgraph_Low.LineWidth = 1;

        Subgraph_Last.Name = "Last";
        Subgraph_Last.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Last.PrimaryColor = RGB(255,128,128);
        Subgraph_Last.DrawZeros = false;
        Subgraph_Last.LineWidth = 1;

        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;

Input_UseZeroValuesFromSource.Name = "Use Zero Values from Source Study";
Input_UseZeroValuesFromSource.SetYesNo(false);

Input_Multiplier.Name = "Multiplier";
Input_Multiplier.SetFloat(1.0f);

Input_ChartStudySubgraphReference.Name = "Chart, Study, Subgraph Reference";
Input_ChartStudySubgraphReference.SetChartStudySubgraphValues(1,1,0);

Input_Version.SetInt(1);

Input_DrawZeros.Name = "Draw Zero Values";
Input_DrawZeros.SetYesNo(false);

Input_UseStandardOHLCSubgraphsFromSourceStudy.Name = "Use Standard OHLC Subgraphs from Source Study";
Input_UseStandardOHLCSubgraphsFromSourceStudy.SetYesNo(false);

return;
}

if (Input_DrawZeros.GetYesNo())
{
    Subgraph_Open.DrawZeros = true;
    Subgraph_High.DrawZeros = true;
    Subgraph_Low.DrawZeros = true;
    Subgraph_Last.DrawZeros = true;
    Subgraph_OHLCAvg.DrawZeros = true;
    Subgraph_HLCAvg.DrawZeros = true;
    Subgraph_HLAvg.DrawZeros = true;
}
else
{
    Subgraph_Open.DrawZeros = false;
    Subgraph_High.DrawZeros = false;
    Subgraph_Low.DrawZeros = false;
    Subgraph_Last.DrawZeros = false;
    Subgraph_OHLCAvg.DrawZeros = false;
    Subgraph_HLCAvg.DrawZeros = false;
    Subgraph_HLAvg.DrawZeros = false;
}

int ChartNumber = Input_ChartStudySubgraphReference.GetChartNumber();
int StudyID = Input_ChartStudySubgraphReference.GetStudyID();
int StudySubgraphNumber = Input_ChartStudySubgraphReference.GetSubgraphIndex();

float MultiplierVal = Input_Multiplier.GetFloat();
if (MultiplierVal == 0.0f)
    MultiplierVal = 1.0f;

SCDateTimeArray ReferenceChartDateTimeArray;

```

```

sc.GetChartDateTimeArray(ChartNumber, ReferenceChartDateTimeArray);
if (ReferenceChartDateTimeArray.GetArraySize() == 0)
    return;

SCGraphData ReferenceArrays;
sc.GetStudyArraysFromChartUsingID(ChartNumber, StudyID, ReferenceArrays);

c_ArrayWrapper<float>* p_ReferenceOpen = NULL;
c_ArrayWrapper<float>* p_ReferenceHigh = NULL;
c_ArrayWrapper<float>* p_ReferenceLow = NULL;
c_ArrayWrapper<float>* p_ReferenceClose = NULL;

if (Input_UseStandardOHLCSubgraphsFromSourceStudy.GetYesNo())
{
    p_ReferenceOpen = &ReferenceArrays[SC_OPEN];
    p_ReferenceHigh = &ReferenceArrays[SC_HIGH];
    p_ReferenceLow = &ReferenceArrays[SC_LOW];
    p_ReferenceClose = &ReferenceArrays[SC_LAST];
}
else
{
    p_ReferenceOpen = &ReferenceArrays[StudySubgraphNumber];
    p_ReferenceHigh = &ReferenceArrays[StudySubgraphNumber];
    p_ReferenceLow = &ReferenceArrays[StudySubgraphNumber];
    p_ReferenceClose = &ReferenceArrays[StudySubgraphNumber];
}

SCFloatArrayRef ReferenceSubgraphArray = ReferenceArrays[StudySubgraphNumber];

if (Input_UseStandardOHLCSubgraphsFromSourceStudy.GetYesNo())
{
    if (ReferenceArrays[SC_OPEN].GetArraySize() == 0
        || ReferenceArrays[SC_HIGH].GetArraySize() == 0
        || ReferenceArrays[SC_LOW].GetArraySize() == 0
        || ReferenceArrays[SC_LAST].GetArraySize() == 0
    )
    {
        sc.AddMessageToLog("Study references a Study Subgraph that does not exist.", 1);
        return;
    }
}
else if (ReferenceSubgraphArray.GetArraySize() == 0)
{
    sc.AddMessageToLog("Study references a Study Subgraph that does not exist.", 1);
    return;
}

if (sc.UpdateStartIndex == 0)
{
    SCString StudyName = sc.GetStudyNameFromChart(ChartNumber, StudyID);
    if (ChartNumber == sc.ChartNumber && StudyID == sc.StudyGraphInstanceID)
        StudyName = "Self";

    sc.GraphName.Format("Overlay OHLC of %s", StudyName.GetChars());

    //Zero all output values at all chart bars
    for (int Index = 0; Index < sc.ArraySize; Index++)
    {
        Subgraph_Open[Index] = 0.0f;
        Subgraph_High[Index] = 0.0f;
        Subgraph_Low[Index] = 0.0f;
        Subgraph_Last[Index] = 0.0f;
        sc.CalculateOHLCAverages(Index);
    }
}

```

```

int RefDataStartIndex = sc.GetStudyDataStartIndexFromChartUsingID(ChartNumber, StudyID);

sc.DataStartIndex =
    sc.GetContainingIndexForSCDateTime(sc.ChartNumber,
ReferenceChartDateTimeArray[RefDataStartIndex].GetAsDouble());

int ArraySize = sc.BaseDateTimeln.GetArraySize();
int RefArraySize = ReferenceChartDateTimeArray.GetArraySize();

int RefIndex = 0;
int StudyStartIndex = max(sc.DataStartIndex, sc.UpdateStartIndex - 1);

// Find the appropriate reference index
RefIndex = sc.GetContainingIndexForSCDateTime(ChartNumber,
sc.BaseDateTimeln[StudyStartIndex].GetAsDouble());

// If no exact match is found, IndexForDateTime returns index right before the specified date and time
if (sc.BaseDateTimeln[StudyStartIndex] > ReferenceChartDateTimeArray[RefIndex])
{
    RefIndex++;
}

if (RefIndex >= RefArraySize || StudyStartIndex >= ArraySize)
    return;

int StartingPoint = StudyStartIndex; // for CalculateAverages afterwards

float OpenVal = (*p_ReferenceOpen)[RefIndex];
float HighVal = (*p_ReferenceHigh)[RefIndex];
float LowVal = (*p_ReferenceLow)[RefIndex];
float CloseVal = (*p_ReferenceClose)[RefIndex];

for (int OutputArrayIndex = StudyStartIndex; OutputArrayIndex < ArraySize; OutputArrayIndex++)
{
    // If DateTime is not contained, then skip. Most likely because referenced graph has a larger duration time period
    if (OutputArrayIndex + 1 < ArraySize && sc.BaseDateTimeln[OutputArrayIndex + 1] <=
ReferenceChartDateTimeArray[RefIndex])
    {
        Subgraph_Open[OutputArrayIndex] = 0;
        Subgraph_Last[OutputArrayIndex] = 0;
        Subgraph_High[OutputArrayIndex] = 0;
        Subgraph_Low[OutputArrayIndex] = 0;
        continue;
    }

    SCDateTime NextIndexDateTime = SCDateTime::GetMaximumDate();

    if (OutputArrayIndex + 1 < ArraySize)
    {
        NextIndexDateTime = sc.BaseDateTimeln[OutputArrayIndex + 1];
    }

    // Increment RefIndex while Next BaseTime is greater than RefTime
    while (NextIndexDateTime > ReferenceChartDateTimeArray[RefIndex])
    {
        if ((Input_UseZeroValuesFromSource.GetYesNo() || (*p_ReferenceHigh)[RefIndex] != 0.0f)
            && HighVal < (*p_ReferenceHigh)[RefIndex])
        {
            HighVal = (*p_ReferenceHigh)[RefIndex];
        }
        else if ((Input_UseZeroValuesFromSource.GetYesNo() || (*p_ReferenceLow)[RefIndex] != 0.0f)
            && LowVal > (*p_ReferenceLow)[RefIndex])
        {
            LowVal = (*p_ReferenceLow)[RefIndex];
        }
    }
}

```

```

    {
        LowVal = (*p_ReferenceLow)[RefIndex];
    }

    if (Input_UseZeroValuesFromSource.GetYesNo() || (*p_ReferenceClose)[RefIndex] != 0.0f)
        CloseVal = (*p_ReferenceClose)[RefIndex];

    RefIndex++;

    if (RefIndex >= RefArraySize)
        break;
}

Subgraph_Open[OutputArrayIndex] = OpenVal * MultiplierVal;
Subgraph_Last[OutputArrayIndex] = CloseVal * MultiplierVal;
Subgraph_High[OutputArrayIndex] = HighVal * MultiplierVal;
Subgraph_Low[OutputArrayIndex] = LowVal * MultiplierVal;

sc.CalculateOHLCAverages(OutputArrayIndex);

if (RefIndex >= RefArraySize)
    break;

OpenVal = (*p_ReferenceOpen)[RefIndex];
HighVal = (*p_ReferenceHigh)[RefIndex];
LowVal = (*p_ReferenceLow)[RefIndex];
CloseVal = (*p_ReferenceClose)[RefIndex];
}
}

/*=====*/
SCSFExport scsf_MarketFacilitationIndexColored(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_MFI = sc.Subgraph[0];
    SCSubgraphRef Subgraph_MfiUpVolumeUp = sc.Subgraph[1];
    SCSubgraphRef Subgraph_MfiDownVolumeDown = sc.Subgraph[2];
    SCSubgraphRef Subgraph_MfiUpVolumeDown = sc.Subgraph[3];
    SCSubgraphRef Subgraph_MfiDownVolumeUp = sc.Subgraph[4];

    SCInputRef Multiplier = sc.Input[0];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Market Facilitation Index Colored";
        sc.StudyDescription = "Market Facilitation Index with Coloring";

        sc.GraphRegion = 1;
        sc.ValueFormat = 2;
        sc.AutoLoop = 1;

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

        Subgraph_MfiUpVolumeUp.Name = "MFI Up Volume Up Color";
        Subgraph_MfiUpVolumeUp.DrawStyle = DRAWSTYLE_IGNORE;
        Subgraph_MfiUpVolumeUp.PrimaryColor = RGB (0, 255, 0);// Green
        Subgraph_MfiUpVolumeUp.DrawZeros = true;

        Subgraph_MfiDownVolumeDown.Name = "MFI Down Volume Down Color";
        Subgraph_MfiDownVolumeDown.DrawStyle = DRAWSTYLE_IGNORE;
        Subgraph_MfiDownVolumeDown.PrimaryColor = RGB (140, 69, 16);// Brown
        Subgraph_MfiDownVolumeDown.DrawZeros = true;
    }
}

```

```

Subgraph_MfiUpVolumeDown.Name = "MFI Up Volume Down Color";
Subgraph_MfiUpVolumeDown.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_MfiUpVolumeDown.PrimaryColor = RGB (0, 0, 255); // Blue
Subgraph_MfiUpVolumeDown.DrawZeros = true;

Subgraph_MfiDownVolumeUp.Name = "MFI Down Volume Up Color";
Subgraph_MfiDownVolumeUp.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_MfiDownVolumeUp.PrimaryColor = RGB (255, 0, 0); // Red
Subgraph_MfiDownVolumeUp.DrawZeros = true;

Multiplier.Name = "Multiplier";
Multiplier.SetFloat(1.0f);

return;
}

Subgraph_MFI[sc.Index] = Multiplier.GetFloat() * (sc.High[sc.Index] - sc.Low[sc.Index]) /
    sc.Volume[sc.Index];

if (sc.Index == 0)
{
    Subgraph_MFI.DataColor[sc.Index] = Subgraph_MfiUpVolumeUp.PrimaryColor;
    return;
}

// Coloring
if (Subgraph_MFI[sc.Index] > Subgraph_MFI[sc.Index - 1] && sc.Volume[sc.Index] > sc.Volume[sc.Index - 1])
{
    Subgraph_MFI.DataColor[sc.Index] = Subgraph_MfiUpVolumeUp.PrimaryColor;
}
else if (Subgraph_MFI[sc.Index] < Subgraph_MFI[sc.Index - 1] && sc.Volume[sc.Index] < sc.Volume[sc.Index - 1])
{
    Subgraph_MFI.DataColor[sc.Index] = Subgraph_MfiDownVolumeDown.PrimaryColor;
}
else if (Subgraph_MFI[sc.Index] > Subgraph_MFI[sc.Index - 1] && sc.Volume[sc.Index] < sc.Volume[sc.Index - 1])
{
    Subgraph_MFI.DataColor[sc.Index] = Subgraph_MfiUpVolumeDown.PrimaryColor;
}
else if (Subgraph_MFI[sc.Index] < Subgraph_MFI[sc.Index - 1] && sc.Volume[sc.Index] > sc.Volume[sc.Index - 1])
{
    Subgraph_MFI.DataColor[sc.Index] = Subgraph_MfiDownVolumeUp.PrimaryColor;
}
}

/*=====*/
SCSFExport scsf_StarcBands(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_KeltnerAverage = sc.Subgraph[0];
    SCSubgraphRef Subgraph_TopBand = sc.Subgraph[1];
    SCSubgraphRef Subgraph_BottomBand = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Temp3 = sc.Subgraph[3];
    SCSubgraphRef Subgraph_Temp4 = sc.Subgraph[4];

    SCInputRef Input_Data = sc.Input[0];
    SCInputRef Input_KeltnerMALength = sc.Input[3];
    SCInputRef Input_TrueRangeAvgLength = sc.Input[4];
    SCInputRef Input_TopBandMult = sc.Input[5];
    SCInputRef Input_BottomBandMult = sc.Input[6];
    SCInputRef Input_KeltnerMAType = sc.Input[7];
    SCInputRef Input_ATR_MAType = sc.Input[8];

    if (sc.SetDefaults)
    {

```

```

sc.GraphName = "Starc Bands";

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

Subgraph_KeltnerAverage.Name = "Starc Average";
Subgraph_KeltnerAverage.DrawStyle = DRAWSTYLE_LINE;
Subgraph_KeltnerAverage.PrimaryColor = RGB(0,255,0);
Subgraph_KeltnerAverage.DrawZeros = true;

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

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

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

Input_KeltnerMALength.Name = "Starc Mov Avg Length";
Input_KeltnerMALength.SetInt(6);
Input_KeltnerMALength.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_TrueRangeAvgLength.Name = "True Range Avg Length";
Input_TrueRangeAvgLength.SetInt(15);
Input_TrueRangeAvgLength.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_TopBandMult.Name = "Top Band Multiplier";
Input_TopBandMult.SetFloat(2.0f);

Input_BottomBandMult.Name="Bottom Band Multiplier";
Input_BottomBandMult.SetFloat(2.0f);

Input_KeltnerMAType.Name = "Starc Mov Avg Type (Center line)";
Input_KeltnerMAType.SetMovAvgType(MOVAVGTYPE_SIMPLE);

Input_ATR_MAType.Name = "ATR Mov Avg Type";
Input_ATR_MAType.SetMovAvgType(MOVAVGTYPE_SIMPLE);

return;
}

sc.DataStartIndex = max(Input_KeltnerMALength.GetInt(), Input_TrueRangeAvgLength.GetInt());

sc.MovingAverage(sc.BaseDataIn[static_cast<int>(Input_Data.GetInputDataIndex())], Subgraph_KeltnerAverage,
Input_KeltnerMAType.GetMovAvgType(), Input_KeltnerMALength.GetInt());

sc.ATR(sc.BaseDataIn, Subgraph_Temp3, Input_TrueRangeAvgLength.GetInt(),
Input_ATR_MAType.GetMovAvgType());

sc.ATR(sc.BaseDataIn, Subgraph_Temp4, Input_TrueRangeAvgLength.GetInt(),
Input_ATR_MAType.GetMovAvgType());

Subgraph_TopBand[sc.Index] = Subgraph_Temp3[sc.Index] * Input_TopBandMult.GetFloat() +
Subgraph_KeltnerAverage[sc.Index];
Subgraph_BottomBand[sc.Index] = Subgraph_KeltnerAverage[sc.Index] - (Subgraph_Temp4[sc.Index] *
Input_BottomBandMult.GetFloat());
}

```

```

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

    SCInputRef Input_Multiplier = sc.Input[0];

    SCInputRef Input_UseMaxDuration = sc.Input[1];

    SCInputRef Input_MaxDuration = sc.Input[2];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Bar Time Duration";
        sc.StudyDescription = "This study calculates and displays the time duration of a bar. For the highest accuracy, this study requires that the Intraday Data Storage Time Unit setting is 1 second or less.";

        Subgraph_TimeDiff.Name = "Duration";
        Subgraph_TimeDiff.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_TimeDiff.PrimaryColor = RGB(0,255,0);

        Input_UseMaxDuration.Name = "Use Maximum Duration";
        Input_UseMaxDuration.SetYesNo(0);

        Input_MaxDuration.Name="Maximum Duration";
        Input_MaxDuration.SetTime(0);

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

        return;
    }

    SCDateTime BarEndTime = sc.GetEndingDateTimeForBarIndex(sc.Index);

    Subgraph_TimeDiff[sc.Index] = static_cast<float>((BarEndTime - sc.BaseDateTimeIn[sc.Index] +
    SCDateTime::MICROSECONDS(1)).GetAsDouble());

    if (Subgraph_TimeDiff[sc.Index] < 0)
        Subgraph_TimeDiff[sc.Index] = 0;

    if(Input_UseMaxDuration.GetYesNo())
    {
        SCDateTime MaxTime(0,Input_MaxDuration.GetTime());
        if (Subgraph_TimeDiff[sc.Index] > MaxTime.GetAsDouble())
            Subgraph_TimeDiff[sc.Index] = static_cast<float>(MaxTime.GetAsDouble());
    }
}

/*=====*/

SCSFExport scsf_ReferenceDataFromAnotherChart(SCStudyInterfaceRef sc)
{
    SCInputRef Input_ChartStudyInput = sc.Input[0];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Reference Data";
        sc.StudyDescription = "This is an example of referencing data from another chart.";
    }
}

```

```

sc.AutoLoop = 1;

Input_ChartStudyInput.Name = "Study Reference";
Input_ChartStudyInput.SetChartStudyValues(1,1);

return;
}

// The following code is for getting the High array
// and corresponding index from another chart.

// Define a graph data object to get all of the base graph data from the specified chart
SCGraphData BaseGraphData;

// Get the base graph data from the specified chart
sc.GetChartBaseData(Input_ChartStudyInput.GetChartNumber(), BaseGraphData);

// Define a reference to the High array
SCFloatArrayRef HighArray = BaseGraphData[SC_HIGH];

// Array is empty. Nothing to do.
if(HighArray.GetArraySize() == 0)
    return;

// Get the index in the specified chart that is
// nearest to current index.
int RefChartIndex = sc.GetNearestMatchForDateTimeIndex(Input_ChartStudyInput.GetChartNumber(), sc.Index);
float NearestRefChartHigh = HighArray[RefChartIndex];

// Get the index in the specified chart that contains
// the DateTime of the bar at the current index.
RefChartIndex = sc.GetContainingIndexForDateTimeIndex(Input_ChartStudyInput.GetChartNumber(), sc.Index);
float ContainingRefChartHigh = HighArray[RefChartIndex];

// Get the index in the specified chart that exactly
// matches the DateTime of the current index.
RefChartIndex = sc.GetExactMatchForSCDateTime(Input_ChartStudyInput.GetChartNumber(),
sc.BaseDateTimeIn[sc.Index]);
if(RefChartIndex != -1)
{
    float ExactMatchRefChartHigh = HighArray[RefChartIndex];
}

// The following code is for getting a study subgraph array
// and corresponding index from another chart.
// For example, this could be a moving average study subgraph.

// Define a graph data object to get all of the study data
SCGraphData StudyData;

// Get the study data from the specified chart
sc.GetStudyArraysFromChartUsingID(Input_ChartStudyInput.GetChartNumber(), Input_ChartStudyInput.GetStudyID(),
StudyData);

//Check if the study has been found. If it has, GetArraySize() will return the number of Subgraphs in the study.
if(StudyData.GetArraySize() == 0)
    return;

// Define a reference to the first subgraph array. Note the type must be 'SCFloatArrayRef' and not 'SCFloatArray'.
SCFloatArrayRef SubgraphArray = StudyData[0];

```



```

// Array is empty. Nothing to do.
if(SubgraphArray.GetSize() == 0)
    return;

// Get the index in the specified chart that is nearest
// to current index.
RefChartIndex = sc.GetNearestMatchForDateTimeIndex(Input_ChartStudyInput.GetChartNumber(), sc.Index);
float NearestSubgraphValue = SubgraphArray[RefChartIndex];

// Get the index in the specified chart that contains
// the DateTime of the bar at the current index.
RefChartIndex = sc.GetContainingIndexForDateTimeIndex(Input_ChartStudyInput.GetChartNumber(), sc.Index);
float ContainingSubgraphValue = SubgraphArray[RefChartIndex];

// Get the index in the specified chart that exactly
// matches the DateTime of the current index.
RefChartIndex = sc.GetExactMatchForSCDateTime(Input_ChartStudyInput.GetChartNumber(),
sc.BaseDateTimeIn[sc.Index]);
if(RefChartIndex != -1)//-1 means that there was not an exact match and therefore we do not have a valid index to work
with
{
    float ExactMatchSubgraphValue = SubgraphArray[RefChartIndex];
}
}

/*****
SCSFExport scsf_ReferenceStudyData(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Average = sc.Subgraph[0];
    SCInputRef Input_Study1 = sc.Input[0];
    SCInputRef Input_Study1Subgraph = sc.Input[1];

    if (sc.SetDefaults)
    {

        sc.GraphName = "Reference Study Data";
        sc.StudyDescription = "This study function is an example of referencing data from other studies on the chart.";

        sc.AutoLoop = 1;

        // We must use a low precedence level to ensure
        // the other studies are already calculated first.
        sc.CalculationPrecedence = LOW_PREC_LEVEL;

        Subgraph_Average.Name = "Average";
        Subgraph_Average.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Average.PrimaryColor = RGB(0,255,0);

        Input_Study1.Name = "Input Study 1";
        Input_Study1.SetStudyID(0);

        Input_Study1Subgraph.Name = "Study 1 Subgraph";
        Input_Study1Subgraph.SetSubgraphIndex(0);

        return;
    }

    // Get the subgraph specified with the Study 1
    // Subgraph input from the study specified with
    // the Input Study 1 input.
    SCFloatArray Study1Array;
    sc.GetStudyArrayUsingID(Input_Study1.GetStudyID(),Input_Study1Subgraph.GetSubgraphIndex(),Study1Array);

```

```

// We are getting the value of the study subgraph
// at sc.Index. For example, this could be
// a moving average value if the study we got in
// the prior step is a moving average.
float RefStudyCurrentValue = Study1Array[sc.Index];

// Here we will add 10 to this value and compute
// an average of it. Since the moving average
// function we will be calling requires an input
// array, we will use one of the internal arrays
// on a subgraph to hold this intermediate
// calculation. This internal array could be
// thought of as a Worksheet column where you
// are performing intermediate calculations.

Subgraph_Average.Arrays[9][sc.Index] = RefStudyCurrentValue + 10;

sc.SimpleMovAvg(Subgraph_Average.Arrays[9],Subgraph_Average,15);
}

/*=====*/
SCSFExport scsf_IntermediateStudyCalculationsUsingArrays(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Average = sc.Subgraph[0];
    SCInputRef Input_Length = sc.Input[0];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Intermediate Study Calculations Using Arrays";
        sc.StudyDescription = "For more information about this study, refer to the Using ACSIL Study Calculation Functions
documentation page.";

        sc.AutoLoop = 1;

        Subgraph_Average.Name = "Average";
        Subgraph_Average.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Average.PrimaryColor = RGB(0,255,0);

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

        return;
    }

    // Here we will add 10 to the sc.BaseData Last value at the current index
    // and compute an average of it. Since the moving average function we
    // will be calling requires an input array and not a single value, we
    // will use one of the internal extra arrays on a Subgraph to hold
    // this intermediate calculation. This internal extra array could be
    // thought of as a Worksheet column where you are performing intermediate
    // calculations. We will use one of the internal extra arrays that is
    // part of the Subgraph we are using to hold the output from the moving
    // average study calculation function we will be calling next. Although
    // any Subgraph internal extra array or even a Subgraph Data array
    // could be used.

    SCFloatArrayRef Last = sc.Close;
    SCFloatArrayRef Array8 = Subgraph_Average.Arrays[8];

```

```

Array8[sc.Index] = Last[sc.Index] + 10;

// In this function call we are passing this internal extra array and
// we also pass in, sc.Subgraph[0], to receive the result at the
// current index.

sc.SimpleMovAvg(Array8, Subgraph_Average, Input_Length.GetInt());
}

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

    SCInputRef Input_Data = sc.Input[0];
    SCInputRef Input_Multiplier = sc.Input[1];
    SCInputRef Input_DrawZeros = sc.Input[2];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Study Subgraph Multiply";
        sc.StudyDescription = "This study multiplies selected study subgraph by the specified Multiplier. To select the study to use, set the Based On study setting to that study and set the Input Data input to the specific study subgraph in that study to multiply by the specified Multiplier.";

        sc.AutoLoop = 0; // Needed when using sc.GetCalculationStartIndexForStudy

        Subgraph_Result.Name = "Result";
        Subgraph_Result.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Result.PrimaryColor = RGB(0, 255, 0);

        Input_Data.Name = "Input Data";
        Input_Data.SetInputDataIndex(0);

        Input_Multiplier.Name = "Multiplier";
        Input_Multiplier.SetFloat(1.0);

        Input_DrawZeros.Name = "Draw Zeros";
        Input_DrawZeros.SetYesNo(false);

        return;
    }

    Subgraph_Result.DrawZeros = Input_DrawZeros.GetYesNo();

    int CalculationStartIndex = sc.GetCalculationStartIndexForStudy();

    for (int Index = CalculationStartIndex; Index < sc.ArraySize; Index++)
    {
        Subgraph_Result[Index] = sc.BaseData[Input_Data.GetInputDataIndex()][Index] * Input_Multiplier.GetFloat();
    }

    sc.EarliestUpdateSubgraphDataArrayIndex = CalculationStartIndex;
}

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

```

```

SCInputRef Input_Data = sc.Input[0];
SCInputRef Input_Divider = sc.Input[1];

SCInputRef DrawZeros = sc.Input[2];

if (sc.SetDefaults)
{
    sc.GraphName = "Study Subgraph Divide";
    sc.StudyDescription = "This study divides selected study subgraph by the specified Divider. To select the study to use, set the Based On study setting to that study and set the Input Data input to the specific study subgraph in that study to divide by the specified Divider.";

    sc.AutoLoop = 0;

    Subgraph_Result.Name = "Result";
    Subgraph_Result.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_Result.PrimaryColor = RGB(0,255,0);

    Input_Data.Name = "Input Data";
    Input_Data.SetInputDataIndex(0);

    Input_Divider.Name = "Divider";
    Input_Divider.SetFloat(1.0);

    DrawZeros.Name = "Draw Zeros";
    DrawZeros.SetYesNo(false);

    return;
}

Subgraph_Result.DrawZeros = DrawZeros.GetYesNo();

int CalculationStartIndex = sc.GetCalculationStartIndexForStudy();

for (int Index = CalculationStartIndex; Index < sc.ArraySize; Index++)
{
    if (Input_Divider.GetFloat() != 0)
    {
        Subgraph_Result[Index] = sc.BaseData[Input_Data.GetInputDataIndex()][Index] / Input_Divider.GetFloat();
    }
    else
        Subgraph_Result[Index] = 0;
}

sc.EarliestUpdateSubgraphDataArrayIndex = CalculationStartIndex;
}

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

    SCInputRef Input_Data = sc.Input[0];
    SCInputRef Input_AmountToAdd = sc.Input[1];
    SCInputRef Input_DrawZeros = sc.Input[2];
    SCInputRef Input_AddToZeroValuesInBasedOnStudy = sc.Input[3];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Study Subgraph Add";

        sc.StudyDescription = "This study adds the specified value to the selected study Subgraph. To select the study to use, set the 'Based On' study setting to that study and set the 'Input Data' input to the specific study Subgraph in that study to Add the value to.";
    }
}

```

```

sc.AutoLoop = 0;

Subgraph_Result.Name = "Result";
Subgraph_Result.DrawStyle = DRAWSTYLE_LINE;
Subgraph_Result.PrimaryColor = RGB(0,255,0);

Input_Data.Name = "Input Data";
Input_Data.SetInputDataIndex(0);

Input_AmountToAdd.Name = "Amount to Add";
Input_AmountToAdd.SetFloat(0);

Input_DrawZeros.Name = "Draw Zeros";
Input_DrawZeros.SetYesNo(false);

Input_AddToZeroValuesInBasedOnStudy.Name = "Add to Zero Values in Based On Study";
Input_AddToZeroValuesInBasedOnStudy.SetYesNo(true);

return;
}

Subgraph_Result.DrawZeros = Input_DrawZeros.GetYesNo();

int CalculationStartIndex = sc.GetCalculationStartIndexForStudy();

for (int Index = CalculationStartIndex; Index < sc.ArraySize; Index++)
{
    float BasedOnStudySubgraphValue = sc.BaseData[Input_Data.GetInputDataIndex()][Index];

    if (Input_AddToZeroValuesInBasedOnStudy.GetYesNo() == 0 && BasedOnStudySubgraphValue == 0.0)
    {
        Subgraph_Result[Index] = 0.0;
    }
    else
    {
        Subgraph_Result[Index] = BasedOnStudySubgraphValue + Input_AmountToAdd.GetFloat();
    }
}

sc.EarliestUpdateSubgraphDataArrayIndex = CalculationStartIndex;

}

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

    SCInputRef Input_Data = sc.Input[0];
    SCInputRef Input_AmountToSubtract = sc.Input[1];
    SCInputRef Input_DrawZeros = sc.Input[2];
    SCInputRef Input_SubtractFromZeroValuesInBasedOnStudy = sc.Input[3];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Study Subgraph Subtract";
        sc.StudyDescription = "This study subtracts the specified value from the selected study subgraph. To select the study to use, set the Based On study setting to that study and set the Input Data input to the specific study subgraph in that study to Subtract the value from.";

        sc.AutoLoop = 0;
    }
}

```

```

Subgraph_Result.Name = "Result";
Subgraph_Result.DrawStyle = DRAWSTYLE_LINE;
Subgraph_Result.PrimaryColor = RGB(0,255,0);

Input_Data.Name = "Input Data";
Input_Data.SetInputDataIndex(0);

Input_AmountToSubtract.Name = "Amount to Subtract";
Input_AmountToSubtract.SetFloat(0);

Input_DrawZeros.Name = "Draw Zeros";
Input_DrawZeros.SetYesNo(false);

Input_SubtractFromZeroValuesInBasedOnStudy.Name = "Subtract From Zero Values in Based On Study";
Input_SubtractFromZeroValuesInBasedOnStudy.SetYesNo(true);

return;
}

Subgraph_Result.DrawZeros= Input_DrawZeros.GetYesNo();

int CalculationStartIndex = sc.GetCalculationStartIndexForStudy();

for (int Index = CalculationStartIndex; Index < sc.ArraySize; Index++)
{
    float BasedOnStudySubgraphValue = sc.BaseData[Input_Data.GetInputDataIndex()][Index];

    if (Input_SubtractFromZeroValuesInBasedOnStudy.GetYesNo() == 0 && BasedOnStudySubgraphValue == 0.0)
    {
        Subgraph_Result[Index] = 0.0;
    }
    else
        Subgraph_Result[Index] = BasedOnStudySubgraphValue - Input_AmountToSubtract.GetFloat();
}

sc.EarliestUpdateSubgraphDataArrayIndex = CalculationStartIndex;
}

/*=====*/
SCSFExport scsf_StudySubgraphStandardDeviation(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_StdDevTop    = sc.Subgraph[0];
    SCSubgraphRef Subgraph_StdDevBottom = sc.Subgraph[1];

    SCFloatArrayRef Array_StdDev = Subgraph_StdDevTop.Arrays[0];

    SCInputRef Input_Data      = sc.Input[0];
    SCInputRef Input_StdDevLength  = sc.Input[1];
    SCInputRef Input_StdDevMultiplier = sc.Input[2];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Study Subgraph Standard Deviation";
        sc.StudyDescription = "This study calculates the standard deviation of the selected study subgraph using the
specified length and multiplier. To select the study to use, set the Based On study setting to that study and set the Input
Data input to the specific study subgraph in that study to calculate the standard deviation on.";

        sc.AutoLoop = 0;
    }
}

```

```

Subgraph_StdDevTop.Name = "Standard Deviation Top";
Subgraph_StdDevTop.DrawStyle = DRAWSTYLE_LINE;
Subgraph_StdDevTop.PrimaryColor = RGB(0,255,0);

Subgraph_StdDevBottom.Name = "Standard Deviation Bottom";
Subgraph_StdDevBottom.DrawStyle = DRAWSTYLE_LINE;
Subgraph_StdDevBottom.PrimaryColor = RGB(255,0,255);

Input_Data.Name = "Input Data";
Input_Data.SetInputDataIndex(0);

Input_StdDevLength.Name = "Standard Deviation Length";
Input_StdDevLength.SetInt(20);
Input_StdDevLength.SetIntLimits(1, INT_MAX);

Input_StdDevMultiplier.Name = "Standard Deviation Multiplier";
Input_StdDevMultiplier.SetFloat(1.0f);

return;
}

int CalculationStartIndex = sc.GetCalculationStartIndexForStudy();

for (int Index = CalculationStartIndex; Index < sc.ArraySize; Index++)
{
    sc.StdDeviation(sc.BaseData[Input_Data.GetInputDataIndex()], Array_StdDev, Index, Input_StdDevLength.GetInt());

    float Value = sc.BaseData[Input_Data.GetInputDataIndex()][Index];
    float ChannelSize = Input_StdDevMultiplier.GetFloat() * Array_StdDev[Index];

    Subgraph_StdDevTop[Index] = Value + ChannelSize;
    Subgraph_StdDevBottom[Index] = Value - ChannelSize;
}

sc.EarliestUpdateSubgraphDataArrayIndex = CalculationStartIndex;

}

/*=====
Volume At Price array test.
-----*/
SCSFExport scsf_VolumeAtPriceArrayTest(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Volume = sc.Subgraph[0];
    SCSubgraphRef Subgraph_VolumeAtPriceVolume = sc.Subgraph[1];

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

        sc.GraphName = "Volume At Price Array Test";

        sc.StudyDescription = "This study tests the sc.VolumeAtPriceForBars array.";

        sc.AutoLoop = 1;

        sc.MaintainVolumeAtPriceData = 1; // true

        Subgraph_Volume.Name = "Volume";

```

```

Subgraph_Volume.DrawStyle = DRAWSTYLE_BAR;
Subgraph_Volume.PrimaryColor = RGB(0,255,0);
Subgraph_Volume.LineWidth = 2;

Subgraph_VolumeAtPriceVolume.Name = "VAP Volume";
Subgraph_VolumeAtPriceVolume.DrawStyle = DRAWSTYLE_DASH;
Subgraph_VolumeAtPriceVolume.LineWidth = 2;

return;
}

// Do data processing

Subgraph_Volume[sc.Index] = sc.Volume[sc.Index]; // For comparison to the subgraph VolumeAtPriceVolume

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

// Get the sum of the volumes from all of the prices at this bar
unsigned int TotalVolume = 0;

const s_VolumeAtPriceV2 *p_VolumeAtPrice=NULL;
int VAPSizeAtBarIndex = sc.VolumeAtPriceForBars->GetSizeAtBarIndex(sc.Index);
for (int VAPIndex = 0; VAPIndex < VAPSizeAtBarIndex; VAPIndex++)
{
    if (!sc.VolumeAtPriceForBars->GetVAPElementAtIndex(sc.Index, VAPIndex, &p_VolumeAtPrice))
        break;

    TotalVolume += p_VolumeAtPrice->Volume;

    //Calculate the price. This requires multiplying p_VolumeAtPrice->PriceInTicks by the tick size
    float Price = p_VolumeAtPrice->PriceInTicks * sc.TickSize;

    //Other members available:
    unsigned int AskVolume = p_VolumeAtPrice->AskVolume;
    unsigned int BidVolume = p_VolumeAtPrice->BidVolume;
    unsigned int NumberOfTrades = p_VolumeAtPrice->NumberOfTrades;
}

Subgraph_VolumeAtPriceVolume[sc.Index] = static_cast<float>(TotalVolume);

// Get the sum of the volumes from all of the prices at this bar by using sc.VolumeAtPriceForBars-
>GetVAPElementAtIndex()
TotalVolume = 0;
s_VolumeAtPriceV2 *p_VolumeAtPriceAtIndex;
int Count = sc.VolumeAtPriceForBars-> GetSizeAtBarIndex(sc.Index);
for (int ElementIndex = 0; ElementIndex < Count; ElementIndex ++)
{
    sc.VolumeAtPriceForBars->GetVAPElementAtIndex(sc.Index, ElementIndex, &p_VolumeAtPriceAtIndex);

    //verify the pointer is not a null, otherwise an exception will occur
    if (p_VolumeAtPriceAtIndex)
        TotalVolume += p_VolumeAtPriceAtIndex->Volume;
}

Subgraph_VolumeAtPriceVolume[sc.Index] = static_cast<float>(TotalVolume);
}

/*=====*/

```



/\*

Here are the other markets (in addition to the e-mini S&P):

\* 30-year Bonds (ZB)

o Tick Setting: 6 ticks (Please adjust the calculation of Ping Pong Entries accordingly)

\* Euro FX (6E)

o Tick Setting: 16 ticks (Please adjust the calculation of Ping Pong Entries accordingly)

\* e-mini Dow (YM)

o Tick Setting: 16 ticks (Please adjust the calculation of Ping Pong Entries accordingly)

\*/

SCSFExport scsf\_RockwellTrading(SCStudyInterfaceRef sc)

{

//-----#1

SCSubgraphRef Subgraph\_BaseColor = sc.Subgraph[0];

SCSubgraphRef Subgraph\_StudySubgraphAddition = sc.Subgraph[1];

SCSubgraphRef Subgraph\_StudySubgraphSubtraction = sc.Subgraph[2];

SCInputRef Input\_Offset = sc.Input[0];

//-----#2

SCSubgraphRef Subgraph\_DailyRangeAverage = sc.Subgraph[3];

SCSubgraphRef Subgraph\_StopLoss = sc.Subgraph[4];

SCSubgraphRef Subgraph\_ProfitTarget = sc.Subgraph[5];

SCInputRef Input\_DailyRangeChartNumber = sc.Input[1];

SCInputRef Input\_DailyRangeAverageStudyNumber = sc.Input[2];

SCInputRef Input\_DailyRangeAverageReference = sc.Input[9];

SCInputRef Input\_DisplayTargetAndStopValues=sc.Input[8];

//-----#3 Coloring the Main Price Graph

SCSubgraphRef Subgraph\_ColorBarSubgraph = sc.Subgraph[6];

SCInputRef Input\_SelectMACD = sc.Input[3];

//-----#4

SCSubgraphRef Subgraph\_PointOnHigh = sc.Subgraph[7];

SCSubgraphRef Subgraph\_PointOnLow = sc.Subgraph[8];

SCInputRef Input\_SelectRSI = sc.Input[4];

//-----

SCInputRef Input\_Version = sc.Input[5];

SCInputRef Input\_StopLossMultiplier = sc.Input[6];

SCInputRef Input\_ProfitTargetMultiplier = sc.Input[7];

if (sc.SetDefaults)

{

// Set the configuration and defaults

sc.GraphName = "Rockwell Trading";

sc.StudyDescription = "The study is for the Rockwell trading method. Notes: The Offset input specifies the amount as an actual value added to the Low of the last bar and subtracted from the High the last bar. These results are displayed as text on the right side of the chart.";

sc.AutoLoop = 1;

```

sc.GraphRegion = 0;

//-----#1
//Subgraphs
Subgraph_BaseColor.Name = "Base Color";
Subgraph_BaseColor.PrimaryColor = RGB(255,255,255);
Subgraph_BaseColor.DrawStyle = DRAWSTYLE_COLOR_BAR;
Subgraph_BaseColor.LineLabel = 0;

Subgraph_StudySubgraphAddition.Name = "Addition";
Subgraph_StudySubgraphAddition.DrawStyle = DRAWSTYLE_TEXT;
Subgraph_StudySubgraphAddition.LineWidth = 8;
Subgraph_StudySubgraphAddition.LineLabel = LL_DISPLAY_VALUE | LL_VALUE_ALIGN_RIGHT |
LL_VALUE_ALIGN_CENTER;
Subgraph_StudySubgraphAddition.PrimaryColor = RGB(0,255,0);

Subgraph_StudySubgraphSubtraction.Name = "Subtraction";
Subgraph_StudySubgraphSubtraction.DrawStyle = DRAWSTYLE_TEXT;
Subgraph_StudySubgraphSubtraction.LineWidth = 8;
Subgraph_StudySubgraphSubtraction.LineLabel = LL_DISPLAY_VALUE | LL_VALUE_ALIGN_RIGHT |
LL_VALUE_ALIGN_CENTER;
Subgraph_StudySubgraphSubtraction.PrimaryColor = RGB(255,0,0);

//Inputs
Input_Offset.Name = "Offset";
Input_Offset.SetFloat(2.25f);

//-----#2
//Subgraphs

Subgraph_DailyRangeAverage.Name = "Daily Range Average";
Subgraph_DailyRangeAverage.DrawStyle = DRAWSTYLE_CUSTOM_TEXT;
Subgraph_DailyRangeAverage.PrimaryColor = RGB(255,255,255);
Subgraph_DailyRangeAverage.LineWidth = 10;

Subgraph_StopLoss.Name = "Stop Loss";
Subgraph_StopLoss.DrawStyle = DRAWSTYLE_CUSTOM_TEXT;
Subgraph_StopLoss.PrimaryColor = RGB(255,0,0);
Subgraph_StopLoss.LineWidth = 10;

Subgraph_ProfitTarget.Name = "Profit Target";
Subgraph_ProfitTarget.DrawStyle = DRAWSTYLE_CUSTOM_TEXT;
Subgraph_ProfitTarget.PrimaryColor = RGB(0,255,0);
Subgraph_ProfitTarget.LineWidth = 10;

//DailyRangeChartNumber.Name = "Daily Chart # Reference";
//DailyRangeChartNumber.SetChartNumber(2);

//DailyRangeAverageStudyNumber.Name = "Daily Range Average Study Number";
//DailyRangeAverageStudyNumber.SetInt(2);

Input_DailyRangeAverageReference.Name = "Daily Range Average Reference";
Input_DailyRangeAverageReference.SetChartStudySubgraphValues(2,2, 0);

//-----#3 Coloring the Main Price Graph
Subgraph_ColorBarSubgraph.Name = "Color Bar";
Subgraph_ColorBarSubgraph.DrawStyle = DRAWSTYLE_COLOR_BAR;
Subgraph_ColorBarSubgraph.SecondaryColorUsed = 1;
Subgraph_ColorBarSubgraph.PrimaryColor = RGB(0,255,0);
Subgraph_ColorBarSubgraph.SecondaryColor = RGB(255,0,0);

Input_SelectMACD.Name = "MACD Study Reference";

```

```

Input_SelectMACD.SetStudyID(2);

//-----#4

Subgraph_PointOnHigh.Name = "RSI Low";
Subgraph_PointOnHigh.DrawStyle = DRAWSTYLE_POINT_ON_HIGH;
Subgraph_PointOnHigh.PrimaryColor = RGB(128,0,0);
Subgraph_PointOnHigh.LineWidth = 5;

Subgraph_PointOnLow.Name = "RSI High";
Subgraph_PointOnLow.DrawStyle = DRAWSTYLE_POINT_ON_LOW;
Subgraph_PointOnLow.PrimaryColor = RGB(0,128,0);
Subgraph_PointOnLow.LineWidth = 5;

Input_SelectRSI.Name = "RSI Study Reference";
Input_SelectRSI.SetStudyID(9);
//-----

Input_DisplayTargetAndStopValues. Name = "Display Target and Stop Values";
Input_DisplayTargetAndStopValues.SetYesNo(true);

Input_Version.SetInt(4);

Input_StopLossMultiplier.Name = "Stop Loss Multiplier";
Input_StopLossMultiplier.SetFloat(0.1f);
Input_ProfitTargetMultiplier.Name = "Profit Target Multiplier";
Input_ProfitTargetMultiplier.SetFloat(0.15f);

sc.CalculationPrecedence = LOW_PREC_LEVEL;

return;
}

//Version update
if(Input_Version.GetInt()<3)
{
    Input_StopLossMultiplier.SetFloat(0.1f);
    Input_ProfitTargetMultiplier.SetFloat(0.15f);
}
else if (Input_Version.GetInt()<4)
{
    Input_DisplayTargetAndStopValues.SetYesNo(true);

    Input_DailyRangeAverageReference.
SetChartStudySubgraphValues(Input_DailyRangeChartNumber.GetChartNumber(),
Input_DailyRangeAverageStudyNumber.GetInt(),0);
}

Input_Version.SetInt(4);

// Do data processing

//-----#1

Subgraph_StudySubgraphAddition[sc.Index] = sc.Low[sc.Index] + Input_Offset.GetFloat();
Subgraph_StudySubgraphSubtraction[sc.Index] = sc.High[sc.Index] - Input_Offset.GetFloat();

//-----#2
if (sc.Index == sc.ArraySize - 1)

```

```

{

    SCFloatArray DailyRangeAverageData;

    sc.GetStudyArrayFromChartUsingID(Input_DailyRangeAverageReference.GetChartStudySubgraphValues() ,
    DailyRangeAverageData);

    if(DailyRangeAverageData.GetArraySize() == 0)
        return;

    float PriorDayDailyRangeAverage = DailyRangeAverageData[DailyRangeAverageData.GetArraySize()-2]; // Get the
prior day's daily range average
    float CurrentStopLoss = static_cast<float>
(sc.RoundToTickSize(PriorDayDailyRangeAverage*Input_StopLossMultiplier.GetFloat(),sc.TickSize));
    float CurrentProfitTarget = static_cast<float>
(sc.RoundToTickSize(PriorDayDailyRangeAverage*Input_ProfitTargetMultiplier.GetFloat(),sc.TickSize));

    s_UseTool Tool_DailyRangeAverage;

    Tool_DailyRangeAverage.Clear();
    Tool_DailyRangeAverage.ChartNumber = sc.ChartNumber;
    Tool_DailyRangeAverage.DrawingType = DRAWING_TEXT;
    Tool_DailyRangeAverage.LineNumber = 72342;
    Tool_DailyRangeAverage.BeginDateTime = 5;
    Tool_DailyRangeAverage.BeginValue = 95;
    Tool_DailyRangeAverage.UseRelativeVerticalValues= true;
    Tool_DailyRangeAverage.Region = sc.GraphRegion;
    Tool_DailyRangeAverage.Color = Subgraph_DailyRangeAverage.PrimaryColor;
    Tool_DailyRangeAverage.FontBold = true;
    Tool_DailyRangeAverage.FontFace = "Lucida Console";
    Tool_DailyRangeAverage.FontSize = Subgraph_DailyRangeAverage.LineWidth;
    Tool_DailyRangeAverage.AddMethod = UTAM_ADD_OR_ADJUST;
    Tool_DailyRangeAverage.ReverseTextColor = false;

    Tool_DailyRangeAverage.Text.Format("DailyRangeAvg:
%s",sc.FormatGraphValue(PriorDayDailyRangeAverage,sc.BaseGraphValueFormat).GetChars());

    sc.UseTool(Tool_DailyRangeAverage);

    if (Input_DisplayTargetAndStopValues.GetYesNo())
    {
        s_UseTool Tool_StopLoss;

        Tool_StopLoss.Clear();
        Tool_StopLoss.ChartNumber = sc.ChartNumber;
        Tool_StopLoss.DrawingType = DRAWING_TEXT;
        Tool_StopLoss.LineNumber = 72343;
        Tool_StopLoss.BeginDateTime = 5;
        Tool_StopLoss.BeginValue = 90;
        Tool_StopLoss.UseRelativeVerticalValues= true;
        Tool_StopLoss.Region = sc.GraphRegion;
        Tool_StopLoss.Color = Subgraph_StopLoss.PrimaryColor;
        Tool_StopLoss.FontBold = true;
        Tool_StopLoss.FontFace = "Lucida Console";
        Tool_StopLoss.FontSize = Subgraph_StopLoss.LineWidth;
        Tool_StopLoss.AddMethod = UTAM_ADD_OR_ADJUST;
        Tool_StopLoss.ReverseTextColor = false;

        Tool_StopLoss.Text.Format("StopLoss: %s
",sc.FormatGraphValue(CurrentStopLoss,sc.BaseGraphValueFormat).GetChars());
    }
}

```

```

    sc.UseTool(Tool_StopLoss);

}

if (Input_DisplayTargetAndStopValues.GetYesNo())
{
    s_UseTool Tool_ProfitTarget;

    Tool_ProfitTarget.Clear();
    Tool_ProfitTarget.ChartNumber = sc.ChartNumber;
    Tool_ProfitTarget.DrawingType = DRAWING_TEXT;
    Tool_ProfitTarget.LineNumber = 72344;
    Tool_ProfitTarget.BeginDateTime = 5;
    Tool_ProfitTarget.BeginValue = 85;
    Tool_ProfitTarget.UseRelativeVerticalValues= true;
    Tool_ProfitTarget.Region = sc.GraphRegion;
    Tool_ProfitTarget.Color = Subgraph_ProfitTarget.PrimaryColor;
    Tool_ProfitTarget.FontBold = true;
    Tool_ProfitTarget.FontFace = "Lucida Console";
    Tool_ProfitTarget.FontSize = Subgraph_ProfitTarget.LineWidth;
    Tool_ProfitTarget.AddMethod = UTAM_ADD_OR_ADJUST;
    Tool_ProfitTarget.ReverseTextColor = false;

    Tool_ProfitTarget.Text.Format("Target: %s
",sc.FormatGraphValue(CurrentProfitTarget,sc.BaseGraphValueFormat).GetChars());

    sc.UseTool(Tool_ProfitTarget);
}

}

//-----#3 Coloring the Main Price Graph

SCFloatArray MACD;
sc.GetStudyArrayUsingID(Input_SelectMACD.GetStudyID(), 0, MACD);

SCFloatArray MACDDiff;
sc.GetStudyArrayUsingID(Input_SelectMACD.GetStudyID(), 2, MACDDiff);

//BaseColor
Subgraph_ColorBarSubgraph[sc.Index] = 1;
Subgraph_ColorBarSubgraph.DataColor[sc.Index] = Subgraph_BaseColor.PrimaryColor;

if(MACD[sc.Index] > 0.0 && MACDDiff[sc.Index] > 0.0)
    Subgraph_ColorBarSubgraph.DataColor[sc.Index] = Subgraph_ColorBarSubgraph.PrimaryColor;

else if(MACD[sc.Index] < 0.0 && MACDDiff[sc.Index] < 0.0)
    Subgraph_ColorBarSubgraph.DataColor[sc.Index] = Subgraph_ColorBarSubgraph.SecondaryColor;

//-----#4
SCFloatArray RSI;
sc.GetStudyArrayUsingID(Input_SelectRSI.GetStudyID(), 0, RSI);

SCFloatArray RSIHigh;
sc.GetStudyArrayUsingID(Input_SelectRSI.GetStudyID(), 1, RSIHigh);

SCFloatArray RSILow;
sc.GetStudyArrayUsingID(Input_SelectRSI.GetStudyID(), 2, RSILow);

if(RSI[sc.Index] < RSILow[sc.Index])
    Subgraph_PointOnHigh[sc.Index] = 1;
else
    Subgraph_PointOnHigh[sc.Index] = 0;

```

```

if(RSI[sc.Index] > RSIHigh[sc.Index])
    Subgraph_PointOnLow[sc.Index] = 1;
else
    Subgraph_PointOnLow[sc.Index] = 0;
}

/*=====*/
SCSFExport scsf_DailyOHLC(SCStudyInterfaceRef sc)
{
    SCInputRef Input_UseThisIntradayChart = sc.Input[0];
    SCInputRef Input_DailyChartNumber = sc.Input[1];
    SCInputRef Input_ReferenceDaysBack = sc.Input[2];
    SCInputRef Input_GraphHighAndLowHistorically = sc.Input[3];
    SCInputRef Input_UseSaturdayData = sc.Input[4];
    SCInputRef Input_NumberOfDaysToCalculate = sc.Input[5];
    SCInputRef Input_RoundToTickSize = sc.Input[6];
    SCInputRef Input_UseDaySessionOnly = sc.Input[7];
    SCInputRef Input_GraphOpenCloseHistorically = sc.Input[8];
    SCInputRef Input_DisplayOnDaySessionOnly = sc.Input[9];
    SCInputRef Input_DisplayTotalDailyVolumeOnly = sc.Input[10];
    SCInputRef Input_DrawZeros = sc.Input[11];
    SCInputRef Input_UsePreviousValuesUntilNewDaySession = sc.Input[12];
    SCInputRef Input_IncludeFridayEveningSessionWithSundayEveningSession = sc.Input[13];
    SCInputRef Input_AlwaysUseSameAsRegionForScaleRange = sc.Input[14];
    SCInputRef Input_UseSundayData = sc.Input[15];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Daily OHLC";

        sc.AutoLoop = false;

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

        sc.Subgraph[SC_OPEN].Name = "Open";
        sc.Subgraph[SC_OPEN].DrawStyle = DRAWSTYLE_DASH;
        sc.Subgraph[SC_OPEN].PrimaryColor = RGB(255,128,0);
        sc.Subgraph[SC_OPEN].DrawZeros = false;

        sc.Subgraph[SC_HIGH].Name = "High";
        sc.Subgraph[SC_HIGH].DrawStyle = DRAWSTYLE_DASH;
        sc.Subgraph[SC_HIGH].PrimaryColor = RGB(0,255,0);
        sc.Subgraph[SC_HIGH].LineWidth = 2;
        sc.Subgraph[SC_HIGH].DrawZeros = false;

        sc.Subgraph[SC_LOW].Name = "Low";
        sc.Subgraph[SC_LOW].DrawStyle = DRAWSTYLE_DASH;
        sc.Subgraph[SC_LOW].PrimaryColor = RGB(255,0,0);
        sc.Subgraph[SC_LOW].LineWidth = 2;
        sc.Subgraph[SC_LOW].DrawZeros = false;

        sc.Subgraph[SC_LAST].Name = "Close";
        sc.Subgraph[SC_LAST].DrawStyle = DRAWSTYLE_DASH;
        sc.Subgraph[SC_LAST].PrimaryColor = RGB(128,0,255);
        sc.Subgraph[SC_LAST].DrawZeros = false;

        sc.Subgraph[SC_OHLC_AVG].Name = "OHLC Avg";
        sc.Subgraph[SC_OHLC_AVG].DrawStyle = DRAWSTYLE_IGNORE;
        sc.Subgraph[SC_OHLC_AVG].PrimaryColor = RGB(128,128,128);
        sc.Subgraph[SC_OHLC_AVG].DrawZeros = false;
    }
}

```

```

sc.Subgraph[SC_HLC_AVG].Name= "HLC Avg";
sc.Subgraph[SC_HLC_AVG].DrawStyle = DRAWSTYLE_IGNORE;
sc.Subgraph[SC_HLC_AVG].PrimaryColor = RGB(128,128,128);
sc.Subgraph[SC_HLC_AVG].DrawZeros = false;

sc.Subgraph[SC_HL_AVG].Name= "HL Avg";
sc.Subgraph[SC_HL_AVG].DrawStyle = DRAWSTYLE_IGNORE;
sc.Subgraph[SC_HL_AVG].PrimaryColor = RGB(128,128,128);
sc.Subgraph[SC_HL_AVG].DrawZeros = false;

sc.Subgraph[SC_VOLUME].Name= "Volume";
sc.Subgraph[SC_VOLUME].DrawStyle = DRAWSTYLE_IGNORE;
sc.Subgraph[SC_VOLUME].PrimaryColor = RGB(128,128,128);
sc.Subgraph[SC_VOLUME].DrawZeros = false;

Input_UseThisIntradayChart.Name = "Use this Intraday Chart";
Input_UseThisIntradayChart.SetYesNo(1);

Input_DailyChartNumber.Name = "Daily Chart Number";
Input_DailyChartNumber.SetChartNumber(1);

Input_ReferenceDaysBack.Name = "Reference Days Back";
Input_ReferenceDaysBack.SetInt(0);
Input_ReferenceDaysBack.SetIntLimits(0, MAX_STUDY_LENGTH);

Input_GraphHighAndLowHistorically.Name = "Graph High and Low Historically (Reference Days Back=0)";
Input_GraphHighAndLowHistorically.SetYesNo(0);

Input_UseSaturdayData.Name = "Use Saturday Data";
Input_UseSaturdayData.SetYesNo(0);

Input_NumberOfDaysToCalculate.Name = "Number of Days To Calculate";
Input_NumberOfDaysToCalculate.SetInt(50);
Input_NumberOfDaysToCalculate.SetIntLimits(1,MAX_STUDY_LENGTH);

Input_RoundToTickSize.Name = "Round to Tick Size";
Input_RoundToTickSize.SetYesNo(0);

Input_UseDaySessionOnly.Name = "Use Day Session Only";
Input_UseDaySessionOnly.SetYesNo(0);

Input_GraphOpenCloseHistorically.Name = "Graph Open and Close Historically (Reference Days Back=0)";
Input_GraphOpenCloseHistorically.SetYesNo(0);

Input_DisplayOnDaySessionOnly.Name = "Display on Day Session Only";
Input_DisplayOnDaySessionOnly.SetYesNo(false);

Input_DisplayTotalDailyVolumeOnly.Name = "Display Total Daily Volume Only";
Input_DisplayTotalDailyVolumeOnly.SetYesNo(false);

Input_DrawZeros.Name = "Draw Zeros";
Input_DrawZeros.SetYesNo(false);

Input_UsePreviousValuesUntilNewDaySession.Name = "Use Previous Values Until New Day Session When Based
on Day Session Only";
Input_UsePreviousValuesUntilNewDaySession.SetYesNo(false);

Input_IncludeFridayEveningSessionWithSundayEveningSession.Name = "Include Friday Evening with Sunday
Evening Session";
Input_IncludeFridayEveningSessionWithSundayEveningSession.SetYesNo(false);

Input_AlwaysUseSameAsRegionForScaleRange.Name = "Always Use Same As Region For Scale Range";
Input_AlwaysUseSameAsRegionForScaleRange.SetYesNo(true);

```



```

    Input_UseSundayData.Name = "Use Sunday Data";
    Input_UseSundayData.SetYesNo(false);

    return;
}

int InUseThisIntradayChart = Input_UseThisIntradayChart.GetYesNo();
int InDailyChartNumber = Input_DailyChartNumber.GetInt();
int InNumberOfDaysBack = Input_ReferenceDaysBack.GetInt();
int InGraphHighLowHistorically = Input_GraphHighAndLowHistorically.GetYesNo();

if (InNumberOfDaysBack > 0)
    InGraphHighLowHistorically = 0;

const int InUseSaturdayData = Input_UseSaturdayData.GetYesNo();
const int InUseSundayData = Input_UseSundayData.GetYesNo();
const int InNumberOfDaysToCalculate = Input_NumberOfDaysToCalculate.GetInt();
const int InRoundToTickSize = Input_RoundToTickSize.GetYesNo();

int InIncludeFridayEveningSessionWithSundayEveningSession =
Input_IncludeFridayEveningSessionWithSundayEveningSession.GetYesNo();

if (Input_AlwaysUseSameAsRegionForScaleRange.GetYesNo())
    sc.ScaleRangeType = SCALE_SAMEASREGION;

if (Input_DisplayTotalDailyVolumeOnly.GetYesNo()
    && sc.Subgraph[SC_OPEN].DrawStyle != DRAWSTYLE_IGNORE)
{
    sc.Subgraph[SC_OPEN].DrawStyle = DRAWSTYLE_IGNORE;
    sc.Subgraph[SC_HIGH].DrawStyle = DRAWSTYLE_IGNORE;
    sc.Subgraph[SC_LOW].DrawStyle = DRAWSTYLE_IGNORE;
    sc.Subgraph[SC_LAST].DrawStyle = DRAWSTYLE_IGNORE;
    sc.Subgraph[SC_OHLC_AVG].DrawStyle = DRAWSTYLE_IGNORE;
    sc.Subgraph[SC_HLC_AVG].DrawStyle = DRAWSTYLE_IGNORE;
    sc.Subgraph[SC_HL_AVG].DrawStyle = DRAWSTYLE_IGNORE;
    sc.Subgraph[SC_VOLUME].DrawStyle = DRAWSTYLE_DASH;
}
else if (!Input_DisplayTotalDailyVolumeOnly.GetYesNo()
    && sc.Subgraph[SC_OPEN].DrawStyle == DRAWSTYLE_IGNORE
    && sc.Subgraph[SC_VOLUME].DrawStyle != DRAWSTYLE_IGNORE)
{
    sc.Subgraph[SC_OPEN].DrawStyle = DRAWSTYLE_DASH;
    sc.Subgraph[SC_HIGH].DrawStyle = DRAWSTYLE_DASH;
    sc.Subgraph[SC_LOW].DrawStyle = DRAWSTYLE_DASH;
    sc.Subgraph[SC_LAST].DrawStyle = DRAWSTYLE_DASH;
    sc.Subgraph[SC_VOLUME].DrawStyle = DRAWSTYLE_IGNORE;
}

for (int Index = SC_OPEN; Index <= SC_HL_AVG; ++Index)
    sc.Subgraph[Index].DrawZeros = Input_DrawZeros.GetYesNo();

float Open = 0.0f, High = 0.0f, Low = 0.0f, Close = 0.0f, Volume = 0.0f;

int IsValid = 1;

// we get chart data only once for speed
SCGraphData DailyChartData;
SCDateTimeArray DailyChartDateTimes;
if (!InUseThisIntradayChart)
{
    sc.GetChartData(InDailyChartNumber, DailyChartData);
    sc.GetChartDateTimeArray(InDailyChartNumber, DailyChartDateTimes);
}

```



```

float HighestHigh = -FLT_MAX, LowestLow = FLT_MAX;

int StartIndex = sc.UpdateStartIndex;

if (StartIndex != 0)
{
    SCDateTime TradingDayStartDateTime = sc.GetTradingDayStartDateTimeOfBar(sc.BaseDateTimeln[StartIndex]);

    if (InIncludeFridayEveningSessionWithSundayEveningSession
        && TradingDayStartDateTime.IsSunday())
    {
        TradingDayStartDateTime.SubtractDays(2);
    }

    StartIndex = sc.GetFirstIndexForDate(sc.ChartNumber, TradingDayStartDateTime.GetDate());
}

//This is used by the Study/Price Overlay study and Spreadsheet study.
sc.EarliestUpdateSubgraphDataArrayIndex = StartIndex;

SCDateTimeMS PriorBarTradingDayDate;

for (int Index = StartIndex; Index < sc.ArraySize; Index++)
{
    sc.Subgraph[SC_OPEN][Index] = 0;
    sc.Subgraph[SC_HIGH][Index] = 0;
    sc.Subgraph[SC_LOW][Index] = 0;
    sc.Subgraph[SC_LAST][Index] = 0;
    sc.Subgraph[SC_VOLUME][Index] = 0;

    bool IsNewDay = false;

    SCDateTime BarDateTime = sc.BaseDateTimeln[Index];

    SCDateTimeMS CurrentBarTradingDayDate = sc.GetTradingDayDate(sc.BaseDateTimeln[Index]);

    if (InIncludeFridayEveningSessionWithSundayEveningSession
        && CurrentBarTradingDayDate.IsSaturday())
    {
        CurrentBarTradingDayDate.AddDays(2);
    }

    if (PriorBarTradingDayDate != CurrentBarTradingDayDate)
    {
        IsNewDay = true;

        PriorBarTradingDayDate = CurrentBarTradingDayDate;

        IsValid =
            CalculateDailyOHLC(
                sc,
                CurrentBarTradingDayDate,
                InNumberOfDaysBack,
                InNumberOfDaysToCalculate,
                InUseSaturdayData,
                InUseThisIntradayChart,
                InDailyChartNumber,
                DailyChartData,
                DailyChartDateTimes,
                Input_UseDaySessionOnly.GetYesNo(),
                Open,
                High,
                Low,
                Close,

```

```

        Volume,
        IncludeFridayEveningSessionWithSundayEveningSession,
        UseSundayData
    );

    HighestHigh = -FLT_MAX;
    LowestLow = FLT_MAX;
}

if (!IsValid)
    continue;

if (InGraphHighLowHistorically)
{
    if (!Input_UseDaySessionOnly.GetYesNo() || (Input_UseDaySessionOnly.GetYesNo()
        && sc.IsDateTimeInDaySession(BarDateTime)) )
    {
        if (sc.High[Index] > HighestHigh)
            HighestHigh = sc.High[Index];

        if (sc.Low[Index] < LowestLow)
            LowestLow = sc.Low[Index];

        High = HighestHigh;
        Low = LowestLow;
    }
    else
    {
        High = 0.0f;
        Low = 0.0f;
    }
}

if (Input_GraphOpenCloseHistorically.GetYesNo())
{
    Open = sc.Open[Index];
    Close = sc.Close[Index];
}

float SubgraphOpen = Open;
float SubgraphHigh = High;
float SubgraphLow = Low;
float SubgraphClose = Close;
float SubgraphVolume = Volume;

if (Input_UseDaySessionOnly.GetYesNo()
    && Input_UsePreviousValuesUntilNewDaySession.GetYesNo()
    && !sc.IsDateTimeInDaySession(BarDateTime)
    && Index > 0)
{
    SubgraphOpen = sc.Subgraph[SC_OPEN][Index - 1];
    SubgraphHigh = sc.Subgraph[SC_HIGH][Index - 1];
    SubgraphLow = sc.Subgraph[SC_LOW][Index - 1];
    SubgraphClose = sc.Subgraph[SC_LAST][Index - 1];
}

if (InRoundToTickSize)
{
    SubgraphOpen = static_cast<float>(sc.RoundToTickSize(SubgraphOpen, sc.TickSize));
    SubgraphHigh = static_cast<float>(sc.RoundToTickSize(SubgraphHigh, sc.TickSize));
    SubgraphLow = static_cast<float>(sc.RoundToTickSize(SubgraphLow, sc.TickSize));
    SubgraphClose = static_cast<float>(sc.RoundToTickSize(SubgraphClose, sc.TickSize));
}

```

```

}

if (Input_DisplayOnDaySessionOnly.GetYesNo() && !sc.IsDateTimeInDaySession (BarDateTime) )
{
    SubgraphOpen = 0;
    SubgraphHigh = 0;
    SubgraphLow = 0;
    SubgraphClose = 0;
    SubgraphVolume = 0;
}

sc.Subgraph[SC_OPEN][Index] = SubgraphOpen;
sc.Subgraph[SC_HIGH][Index] = SubgraphHigh;
sc.Subgraph[SC_LOW][Index] = SubgraphLow;
sc.Subgraph[SC_LAST][Index] = SubgraphClose;
sc.Subgraph[SC_VOLUME][Index] = SubgraphVolume;

sc.CalculateOHLCAverages( Index);
}
}

/*=====*/
SCSFExport scsf_DailyOHLCSinglePoint(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_ExtensionLinePropertiesSubgraph = sc.Subgraph[SC_LAST + 1];

    SCInputRef Input_UseThisIntradayChart = sc.Input[0];
    SCInputRef Input_DailyChartNumber = sc.Input[1];
    SCInputRef Input_UseSaturdayData = sc.Input[4];
    SCInputRef Input_NumberOfDaysToCalculate = sc.Input[5];
    SCInputRef Input_UseDaySessionOnly = sc.Input[7];
    SCInputRef Input_DisplayHighLowPriceLabels = sc.Input[8];
    SCInputRef Input_DrawExtensionLines = sc.Input[9];
    SCInputRef Input_UseSundayData = sc.Input[10];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Daily OHLC - Single Point";

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

        sc.AutoLoop = 0;//manual looping

        sc.Subgraph[SC_OPEN].Name = "Open";
        sc.Subgraph[SC_OPEN].DrawStyle = DRAWSTYLE_HIDDEN;
        sc.Subgraph[SC_OPEN].PrimaryColor = RGB(255,128,0);
        sc.Subgraph[SC_OPEN].DrawZeros = false;

        sc.Subgraph[SC_HIGH].Name = "High";
        sc.Subgraph[SC_HIGH].DrawStyle = DRAWSTYLE_TRIANGLE_DOWN;
        sc.Subgraph[SC_HIGH].PrimaryColor = RGB(0,255,0);
        sc.Subgraph[SC_HIGH].LineWidth = 5;
        sc.Subgraph[SC_HIGH].DrawZeros = false;

        sc.Subgraph[SC_LOW].Name = "Low";
        sc.Subgraph[SC_LOW].DrawStyle = DRAWSTYLE_TRIANGLE_UP;
        sc.Subgraph[SC_LOW].PrimaryColor = RGB(255,0,0);
        sc.Subgraph[SC_LOW].LineWidth = 5;
        sc.Subgraph[SC_LOW].DrawZeros = false;
    }
}

```

```

sc.Subgraph[SC_LAST].Name = "Close";
sc.Subgraph[SC_LAST].DrawStyle = DRAWSTYLE_HIDDEN;
sc.Subgraph[SC_LAST].PrimaryColor = RGB(128,0,255);
sc.Subgraph[SC_LAST].DrawZeros = false;

Subgraph_ExtensionLinePropertiesSubgraph.Name = "Extension Line Properties";
Subgraph_ExtensionLinePropertiesSubgraph.DrawStyle =
DRAWSTYLE_SUBGRAPH_NAME_AND_VALUE_LABELS_ONLY;
Subgraph_ExtensionLinePropertiesSubgraph.LineWidth = 1;
Subgraph_ExtensionLinePropertiesSubgraph.PrimaryColor = RGB (255, 0, 255);
Subgraph_ExtensionLinePropertiesSubgraph.DrawZeros = false;

Input_UseThisIntradayChart.Name = "Use this Intraday Chart";
Input_UseThisIntradayChart.SetYesNo(1);

Input_DailyChartNumber.Name = "Daily Chart Number";
Input_DailyChartNumber.SetChartNumber(1);

Input_UseSaturdayData.Name = "Use Saturday Data";
Input_UseSaturdayData.SetYesNo(0);

Input_NumberOfDaysToCalculate.Name = "Number of Days To Calculate";
Input_NumberOfDaysToCalculate.SetInt(50);
Input_NumberOfDaysToCalculate.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_UseDaySessionOnly.Name = "Use Day Session Only";
Input_UseDaySessionOnly.SetYesNo(0);

Input_DisplayHighLowPriceLabels.Name = "Display High/Low Price Labels";
Input_DisplayHighLowPriceLabels.SetYesNo(0);

Input_DrawExtensionLines.Name = "Draw Extension Lines";
Input_DrawExtensionLines.SetYesNo(false);

Input_UseSundayData.Name = "Use Sunday Data";
Input_UseSundayData.SetYesNo(false);

return;
}

int InUseThisIntradayChart = Input_UseThisIntradayChart.GetYesNo();
int InDailyChartNumber = Input_DailyChartNumber.GetInt();
int InUseSaturdayData = Input_UseSaturdayData.GetYesNo();
int InUseSundayData = Input_UseSundayData.GetYesNo();
int InNumberOfDaysToCalculate = Input_NumberOfDaysToCalculate.GetInt();
int InUseDaySessionOnly = Input_UseDaySessionOnly.GetYesNo();
int DisplayPriceLabels = Input_DisplayHighLowPriceLabels.GetYesNo();
int InDrawExtensionLines = Input_DrawExtensionLines.GetYesNo();

float Open = 0.0f, High = 0.0f, Low = 0.0f, Close = 0.0f, Volume = 0.0f;
int IntradayChartDate = 0;
int IsValid = 1;

int& IndexOfMostRecentHigh = sc.GetPersistentInt(1);
int& IndexOfMostRecentLow = sc.GetPersistentInt(2);
int& LineNumberOfMostRecentHighValue = sc.GetPersistentInt(3);
int& LineNumberOfMostRecentLowValue = sc.GetPersistentInt(4);

// we get chart data only once for speed
SCGraphData DailyChartData;
SCDateTimeArray DailyChartDateTime;

if (!InUseThisIntradayChart)
{

```

```

        sc.GetChartData(InDailyChartNumber, DailyChartData);
        sc.GetChartDateTimeArray(InDailyChartNumber, DailyChartDateTime);
    }

    if (sc.IsFullRecalculation)
    {
        IndexOfMostRecentHigh = -1;
        IndexOfMostRecentLow = -1;
        LineNumberOfMostRecentHighValue = 0;
        LineNumberOfMostRecentLowValue = 0;
    }

    int StartIndex = sc.UpdateStartIndex;
    if (StartIndex != 0)
    {
        SCDateTime TradingDayStartDateTime = sc.GetTradingDayStartDateTimeOfBar(sc.BaseDateTimeIn[StartIndex]);
        StartIndex = sc.GetFirstIndexForDate(sc.ChartNumber, TradingDayStartDateTime.GetDate());
    }

    for (int Index = StartIndex; Index < sc.ArraySize; Index++)
    {
        if (IntradayChartDate != sc.GetTradingDayDate(sc.BaseDateTimeIn[Index]))
        {
            // A new day has been entered, existing text drawings and extension lines need to remain. So remove the
            reference to those.
            if (IndexOfMostRecentHigh < Index || IndexOfMostRecentLow < Index)
            {
                IndexOfMostRecentHigh = -1;
                IndexOfMostRecentLow = -1;
                LineNumberOfMostRecentHighValue = 0;
                LineNumberOfMostRecentLowValue = 0;
            }

            IntradayChartDate = sc.GetTradingDayDate(sc.BaseDateTimeIn[Index]);

            IsValid =
                CalculateDailyOHLC(
                    sc,
                    IntradayChartDate,
                    0,
                    InNumberOfDaysToCalculate,
                    InUseSaturdayData,
                    InUseThisIntradayChart,
                    InDailyChartNumber,
                    DailyChartData,
                    DailyChartDateTime,
                    InUseDaySessionOnly,
                    Open,
                    High,
                    Low,
                    Close,
                    Volume,
                    0,
                    InUseSundayData
                );
        }

        if (!IsValid)
            continue;

        if(sc.FormattedEvaluate(Open,sc.BaseGraphValueFormat,EQUAL_OPERATOR,sc.Open[Index],

```

```

sc.BaseGraphValueFormat))
    sc.Subgraph[SC_OPEN][Index] = Open;
else
    sc.Subgraph[SC_OPEN][Index] = 0.0;

if(sc.FormattedEvaluate(High,sc.BaseGraphValueFormat,EQUAL_OPERATOR,sc.High[Index],
sc.BaseGraphValueFormat))
{
    sc.Subgraph[SC_HIGH][Index] = High;

    if (DisplayPriceLabels)
    {
        s_UseTool Tool;
        Tool.ChartNumber = sc.ChartNumber;
        Tool.DrawingType = DRAWING_TEXT;
        Tool.Region = sc.GraphRegion;

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

        Tool.BeginIndex = Index;
        Tool.BeginValue = High;

        Tool.Text = " ";
        Tool.Text += sc.FormatGraphValue(High, sc.BaseGraphValueFormat);
        Tool.Color = sc.Subgraph[SC_HIGH].PrimaryColor;
        Tool.TextAlignment = DT_LEFT | DT_VCENTER;

        Tool.AddMethod = UTAM_ADD_OR_ADJUST;

        sc.UseTool(Tool);

        LineNumberOfMostRecentHighValue = Tool.LineNumber;
    }

    if (InDrawExtensionLines)
    {
        if (IndexOfMostRecentHigh != -1)//First delete the existing line
            sc.DeleteLineUntilFutureIntersection(IndexOfMostRecentHigh, 1);

        sc.AddLineUntilFutureIntersection
            ( Index
            , 1
            , High
            , Subgraph_ExtensionLinePropertiesSubgraph.PrimaryColor
            , Subgraph_ExtensionLinePropertiesSubgraph.LineWidth
            , Subgraph_ExtensionLinePropertiesSubgraph.LineStyle
            , (Subgraph_ExtensionLinePropertiesSubgraph.LineLabel & LL_DISPLAY_VALUE) != 0
            , (Subgraph_ExtensionLinePropertiesSubgraph.LineLabel & LL_DISPLAY_NAME) != 0
            , "High"
            );
    }

    IndexOfMostRecentHigh = Index;
}
else
    sc.Subgraph[SC_HIGH][Index] = 0.0;

if(sc.FormattedEvaluate(Low,sc.BaseGraphValueFormat,EQUAL_OPERATOR,sc.Low[Index],
sc.BaseGraphValueFormat))
{
    sc.Subgraph[SC_LOW][Index] = Low;

```

```

if (DisplayPriceLabels)
{
    s_UseTool Tool;
    Tool.ChartNumber = sc.ChartNumber;
    Tool.DrawingType = DRAWING_TEXT;
    Tool.Region = sc.GraphRegion;

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

    Tool.BeginIndex = Index;
    Tool.BeginValue = Low;

    Tool.Text = " ";
    Tool.Text += sc.FormatGraphValue(Low, sc.BaseGraphValueFormat);
    Tool.Color = sc.Subgraph[SC_LOW].PrimaryColor;
    Tool.TextAlignment = DT_LEFT | DT_VCENTER;

    Tool.AddMethod = UTAM_ADD_OR_ADJUST;

    sc.UseTool(Tool);

    LineNumberOfMostRecentLowValue = Tool.LineNumber;
}

if (InDrawExtensionLines)
{
    if (IndexOfMostRecentLow != -1) //First delete the existing line
        sc.DeleteLineUntilFutureIntersection(IndexOfMostRecentLow, 2);

    sc.AddLineUntilFutureIntersection
        ( Index
        , 2
        , Low
        , Subgraph_ExtensionLinePropertiesSubgraph.PrimaryColor
        , Subgraph_ExtensionLinePropertiesSubgraph.LineWidth
        , Subgraph_ExtensionLinePropertiesSubgraph.LineStyle
        , (Subgraph_ExtensionLinePropertiesSubgraph.LineLabel & LL_DISPLAY_VALUE) != 0
        , (Subgraph_ExtensionLinePropertiesSubgraph.LineLabel & LL_DISPLAY_NAME) != 0
        , "Low"
        );
}

IndexOfMostRecentLow = Index;
}
else
    sc.Subgraph[SC_LOW][Index] = 0.0;

if(sc.FormattedEvaluate(Close,sc.BaseGraphValueFormat,EQUAL_OPERATOR,sc.Close[Index],
sc.BaseGraphValueFormat))
    sc.Subgraph[SC_LAST][Index] = Close;
else
    sc.Subgraph[SC_LAST][Index] = 0.0;
}
}

/*=====*/
SCSFExport scsf_PivotPointsDaily(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_R1 = sc.Subgraph[0];
    SCSubgraphRef Subgraph_R2 = sc.Subgraph[1];
    SCSubgraphRef Subgraph_S1 = sc.Subgraph[2];
    SCSubgraphRef Subgraph_S2 = sc.Subgraph[3];
    SCSubgraphRef Subgraph_R0_5 = sc.Subgraph[4];

```

```

SCSubgraphRef Subgraph_R1_5 = sc.Subgraph[5];
SCSubgraphRef Subgraph_R2_5 = sc.Subgraph[6];
SCSubgraphRef Subgraph_R3 = sc.Subgraph[7];
SCSubgraphRef Subgraph_S0_5 = sc.Subgraph[8];
SCSubgraphRef Subgraph_S1_5 = sc.Subgraph[9];
SCSubgraphRef Subgraph_S2_5 = sc.Subgraph[10];
SCSubgraphRef Subgraph_S3 = sc.Subgraph[11];
SCSubgraphRef Subgraph_PP = sc.Subgraph[12];
SCSubgraphRef Subgraph_PPHigh = sc.Subgraph[13];
SCSubgraphRef Subgraph_PPLow = sc.Subgraph[14];
SCSubgraphRef Subgraph_R4 = sc.Subgraph[15];
SCSubgraphRef Subgraph_S4 = sc.Subgraph[16];
SCSubgraphRef Subgraph_R3_5 = sc.Subgraph[17];
SCSubgraphRef Subgraph_S3_5 = sc.Subgraph[18];
SCSubgraphRef Subgraph_R5 = sc.Subgraph[19];
SCSubgraphRef Subgraph_S5 = sc.Subgraph[20];
SCSubgraphRef Subgraph_R6 = sc.Subgraph[21];
SCSubgraphRef Subgraph_S6 = sc.Subgraph[22];
SCSubgraphRef Subgraph_R7 = sc.Subgraph[23];
SCSubgraphRef Subgraph_S7 = sc.Subgraph[24];
SCSubgraphRef Subgraph_R8 = sc.Subgraph[25];
SCSubgraphRef Subgraph_S8 = sc.Subgraph[26];
SCSubgraphRef Subgraph_R9 = sc.Subgraph[27];
SCSubgraphRef Subgraph_S9 = sc.Subgraph[28];
SCSubgraphRef Subgraph_R10 = sc.Subgraph[29];
SCSubgraphRef Subgraph_S10 = sc.Subgraph[30];
SCSubgraphRef Subgraph_R4_5 = sc.Subgraph[31];
SCSubgraphRef Subgraph_S4_5 = sc.Subgraph[32];
const int NUMBER_OF_STUDY_SUBGRAPHS = 33;

```

```

SCInputRef Input_FormulaType = sc.Input[4];
SCInputRef Input_NumberOfDays = sc.Input[5];
SCInputRef Input_RoundToTickSize = sc.Input[6];
SCInputRef Input_UseSaturdayData = sc.Input[7];
SCInputRef Input_DailyChartNumber = sc.Input[9]; //previously input 3
SCInputRef Input_ReferenceDailyChartForData = sc.Input[10];
SCInputRef Input_ForwardProjectLines = sc.Input[11];
SCInputRef Input_UseManualValues = sc.Input[12];
SCInputRef Input_UserOpen = sc.Input[13];
SCInputRef Input_UserHigh = sc.Input[14];
SCInputRef Input_UserLow = sc.Input[15];
SCInputRef Input_UserClose = sc.Input[16];
SCInputRef Input_UseDaySessionOnly = sc.Input[17];
SCInputRef Input_Version = sc.Input[18];
SCInputRef Input_UseDailyChartForSettlementOnly = sc.Input[19];

```

```

if (sc.SetDefaults)
{

```

```

    sc.GraphName = "Pivot Points-Daily";

```

```

    sc.ScaleRangeType = SCALE_SAMEASREGION;

```

```

    sc.GraphRegion = 0;

```

```

    sc.ValueFormat = VALUEFORMAT_INHERITED;

```

```

    Subgraph_R1.Name = "R1";

```

```

    Subgraph_R2.Name = "R2";

```

```

    Subgraph_S1.Name = "S1";

```

```

    Subgraph_S2.Name = "S2";

```

```

    Subgraph_R0_5.Name = "R.5";

```

```

    Subgraph_R1_5.Name = "R1.5";

```

```

    Subgraph_R2_5.Name = "R2.5";

```

```

    Subgraph_R3.Name = "R3";

```

```

    Subgraph_S0_5.Name = "S.5";

```



```
Subgraph_S1_5.Name = "S1.5";
Subgraph_S2_5.Name = "S2.5";
Subgraph_S3.Name = "S3";
Subgraph_PP.Name = "PP";
Subgraph_PPHigh.Name = "PP High";
Subgraph_PPLow.Name = "PP Low";
Subgraph_R4.Name = "R4";
Subgraph_S4.Name = "S4";
Subgraph_R3_5.Name = "R3.5";
Subgraph_S3_5.Name = "S3.5";
Subgraph_R5.Name = "R5";
Subgraph_S5.Name = "S5";
Subgraph_R6.Name = "R6";
Subgraph_S6.Name = "S6";
Subgraph_R7.Name = "R7";
Subgraph_S7.Name = "S7";
Subgraph_R8.Name = "R8";
Subgraph_S8.Name = "S8";
Subgraph_R9.Name = "R9";
Subgraph_S9.Name = "S9";
Subgraph_R10.Name = "R10";
Subgraph_S10.Name = "S10";
Subgraph_R4_5.Name = "R4_5";
Subgraph_S4_5.Name = "S4_5";
```

```
Subgraph_R1.DrawStyle = DRAWSTYLE_DASH;
Subgraph_R2.DrawStyle = DRAWSTYLE_DASH;
Subgraph_S1.DrawStyle = DRAWSTYLE_DASH;
Subgraph_S2.DrawStyle = DRAWSTYLE_DASH;
Subgraph_R0_5.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_R1_5.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_R2_5.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_R3.DrawStyle = DRAWSTYLE_DASH;
Subgraph_S0_5.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_S1_5.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_S2_5.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_S3.DrawStyle = DRAWSTYLE_DASH;
Subgraph_PP.DrawStyle = DRAWSTYLE_DASH;
Subgraph_PPHigh.DrawStyle = DRAWSTYLE_DASH;
Subgraph_PPLow.DrawStyle = DRAWSTYLE_DASH;
Subgraph_R4.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_S4.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_R3_5.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_S3_5.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_R5.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_S5.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_R6.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_S6.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_R7.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_S7.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_R8.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_S8.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_R9.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_S9.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_R10.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_S10.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_R4_5.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_S4_5.DrawStyle = DRAWSTYLE_HIDDEN;
```

```
Subgraph_R1.DrawZeros = false;
Subgraph_R2.DrawZeros = false;
Subgraph_S1.DrawZeros = false;
Subgraph_S2.DrawZeros = false;
Subgraph_R0_5.DrawZeros = false;
Subgraph_R1_5.DrawZeros = false;
```

Subgraph\_R2\_5.DrawZeros = false;  
Subgraph\_R3.DrawZeros = false;  
Subgraph\_S0\_5.DrawZeros = false;  
Subgraph\_S1\_5.DrawZeros = false;  
Subgraph\_S2\_5.DrawZeros = false;  
Subgraph\_S3.DrawZeros = false;  
Subgraph\_PP.DrawZeros = false;  
Subgraph\_PPHigh.DrawZeros = false;  
Subgraph\_PPLow.DrawZeros = false;  
Subgraph\_R4.DrawZeros = false;  
Subgraph\_S4.DrawZeros = false;  
Subgraph\_R3\_5.DrawZeros = false;  
Subgraph\_S3\_5.DrawZeros = false;  
Subgraph\_R5.DrawZeros = false;  
Subgraph\_S5.DrawZeros = false;  
Subgraph\_R6.DrawZeros = false;  
Subgraph\_S6.DrawZeros = false;  
Subgraph\_R7.DrawZeros = false;  
Subgraph\_S7.DrawZeros = false;  
Subgraph\_R8.DrawZeros = false;  
Subgraph\_S8.DrawZeros = false;  
Subgraph\_R9.DrawZeros = false;  
Subgraph\_S9.DrawZeros = false;  
Subgraph\_R10.DrawZeros = false;  
Subgraph\_S10.DrawZeros = false;  
Subgraph\_R4\_5.DrawZeros = false;  
Subgraph\_S4\_5.DrawZeros = false;

Subgraph\_PP.PrimaryColor = RGB(255, 0, 255);  
Subgraph\_PPHigh.PrimaryColor = RGB(255, 0, 255);  
Subgraph\_PPLow.PrimaryColor = RGB(255, 0, 255);

Subgraph\_R1.PrimaryColor = RGB(255, 0, 0);  
Subgraph\_R2.PrimaryColor = RGB(255, 0, 0);  
Subgraph\_R0\_5.PrimaryColor = RGB(255, 0, 0);  
Subgraph\_R1\_5.PrimaryColor = RGB(255, 0, 0);  
Subgraph\_R2\_5.PrimaryColor = RGB(255, 0, 0);  
Subgraph\_R3.PrimaryColor = RGB(255, 0, 0);  
Subgraph\_R4.PrimaryColor = RGB(255, 0, 0);  
Subgraph\_R4\_5.PrimaryColor = RGB(255, 0, 0);  
Subgraph\_R3\_5.PrimaryColor = RGB(255, 0, 0);  
Subgraph\_R5.PrimaryColor = RGB(255, 0, 0);  
Subgraph\_R6.PrimaryColor = RGB(255, 0, 0);  
Subgraph\_R7.PrimaryColor = RGB(255, 0, 0);  
Subgraph\_R8.PrimaryColor = RGB(255, 0, 0);  
Subgraph\_R9.PrimaryColor = RGB(255, 0, 0);  
Subgraph\_R10.PrimaryColor = RGB(255, 0, 0);

Subgraph\_S1.PrimaryColor = RGB(0, 255, 0);  
Subgraph\_S2.PrimaryColor = RGB(0, 255, 0);  
Subgraph\_S0\_5.PrimaryColor = RGB(0, 255, 0);  
Subgraph\_S1\_5.PrimaryColor = RGB(0, 255, 0);  
Subgraph\_S2\_5.PrimaryColor = RGB(0, 255, 0);  
Subgraph\_S3.PrimaryColor = RGB(0, 255, 0);  
Subgraph\_S4.PrimaryColor = RGB(0, 255, 0);  
Subgraph\_S4\_5.PrimaryColor = RGB(0, 255, 0);  
Subgraph\_S3\_5.PrimaryColor = RGB(0, 255, 0);  
Subgraph\_S5.PrimaryColor = RGB(0, 255, 0);  
Subgraph\_S6.PrimaryColor = RGB(0, 255, 0);  
Subgraph\_S7.PrimaryColor = RGB(0, 255, 0);  
Subgraph\_S8.PrimaryColor = RGB(0, 255, 0);  
Subgraph\_S9.PrimaryColor = RGB(0, 255, 0);  
Subgraph\_S10.PrimaryColor = RGB(0, 255, 0);

```

for (int SubgraphIndex = 0; SubgraphIndex < NUMBER_OF_STUDY_SUBGRAPHS; SubgraphIndex++)
{
    sc.Subgraph[SubgraphIndex].LineLabel = LL_DISPLAY_NAME | LL_DISPLAY_VALUE |
    LL_NAME_ALIGN_ABOVE | LL_NAME_ALIGN_LEFT | LL_VALUE_ALIGN_CENTER |
    LL_VALUE_ALIGN_VALUES_SCALE;
}

sc.Input[3].SetChartNumber(1);

Input_DailyChartNumber.Name = "Daily Chart Number";
Input_DailyChartNumber.SetChartNumber(1);

Input_FormulaType.Name = "Formula Type";
Input_FormulaType.SetInt(0);

Input_NumberOfDays.Name = "Number of Days To Calculate";
Input_NumberOfDays.SetInt(50);
Input_NumberOfDays.SetIntLimits(1,MAX_STUDY_LENGTH);

Input_RoundToTickSize.Name = "Round to Tick Size";
Input_RoundToTickSize.SetYesNo(0);

Input_UseSaturdayData.Name = "Use Saturday Data";
Input_UseSaturdayData.SetYesNo(0);

Input_ReferenceDailyChartForData.Name = "Reference Daily Chart For Data";
Input_ReferenceDailyChartForData.SetYesNo(false);

Input_ForwardProjectLines.Name = "Forward Project Pivot Point Lines";
Input_ForwardProjectLines.SetYesNo(0);

Input_UseManualValues.Name = "Use User Entered OHLC Values";
Input_UseManualValues.SetYesNo(0);

Input_UserOpen.Name = "User Entered Open Value";
Input_UserOpen.SetFloat(0.0f);

Input_UserHigh.Name = "User Entered High Value";
Input_UserHigh.SetFloat(0.0f);

Input_UserLow.Name = "User Entered Low Value";
Input_UserLow.SetFloat(0.0f);

Input_UserClose.Name = "User Entered Close Value";
Input_UserClose.SetFloat(0.0f);

Input_UseDaySessionOnly.Name = "Use Day Session Only";
Input_UseDaySessionOnly.SetYesNo(false);

Input_UseDailyChartForSettlementOnly.Name = "Use Daily Chart For Settlement Only";
Input_UseDailyChartForSettlementOnly.SetYesNo(false);

Input_Version.SetInt(2);

return;
}
//Upgrade code
if (Input_Version.GetInt()<2)
{
    Input_Version.SetInt(2);
    Input_DailyChartNumber.SetInt(sc.Input[3].GetInt() );
    Input_ReferenceDailyChartForData .SetYesNo(!Input_ReferenceDailyChartForData .GetYesNo());
}

float fPivotPoint= 0, fPivotPointHigh = 0, fPivotPointLow = 0;

```

```

float fR0_5 = 0, fR1 = 0, fR1_5 = 0, fR2 = 0, fR2_5 = 0, fR3 = 0, fR3_5 = 0, fR4 = 0, fR4_5 = 0, fR5 = 0, fR6 = 0, fR7 = 0, fR8 = 0, fR9 = 0, fR10 = 0;
float fS0_5 = 0, fS1 = 0, fS1_5 = 0, fS2 = 0, fS2_5 = 0, fS3 = 0, fS3_5 = 0, fS4 = 0, fS4_5 = 0, fS5 = 0, fS6 = 0, fS7 = 0, fS8 = 0, fS9 = 0, fS10 = 0;

```

```

int IntradayChartDate = 0;
int ValidPivotPoint = 1;

```

```

// we get chart data only once for speed

```

```

SCGraphData DailyChartData;
SCDateTimeArray DailyChartDateTime;
if (Input_ReferenceDailyChartForData.GetYesNo() || Input_UseDailyChartForSettlementOnly.GetYesNo())
{
    sc.GetChartData(Input_DailyChartNumber.GetChartNumber(), DailyChartData);
    sc.GetChartDateTimeArray(Input_DailyChartNumber.GetChartNumber(), DailyChartDateTime);

    if (Input_UseDailyChartForSettlementOnly.GetYesNo())
        Input_ReferenceDailyChartForData.SetYesNo(false);
}

```

```

int NumberOfForwardBars = 0;

```

```

if(Input_ForwardProjectLines.GetYesNo())
{
    NumberOfForwardBars = 20;

    if(sc.UpdateStartIndex == 0)
    {
        for (int SubgraphIndex = 0; SubgraphIndex < NUMBER_OF_STUDY_SUBGRAPHS; SubgraphIndex++)
            sc.Subgraph[SubgraphIndex].ExtendedArrayElementsToGraph = NumberOfForwardBars;
    }
}

```

```

for (int Index = sc.UpdateStartIndex; Index < sc.ArraySize + NumberOfForwardBars; Index++)
{
    if (IntradayChartDate != sc.GetTradingDayDate(sc.BaseDateTimeIn[Index]))
    {
        IntradayChartDate = sc.GetTradingDayDate(sc.BaseDateTimeIn[Index]);

```

```

ValidPivotPoint =
    CalculateDailyPivotPoints(
        sc,
        IntradayChartDate,
        Input_FormulaType.GetInt(),
        Input_DailyChartNumber.GetChartNumber(),
        DailyChartData,
        DailyChartDateTime,
        Input_NumberOfDays.GetInt(),
        Input_UseSaturdayData.GetYesNo(),
        Input_ReferenceDailyChartForData.GetYesNo() ? 0 : 1,
        Input_UseManualValues.GetYesNo(),
        Input_UserOpen.GetFloat(),
        Input_UserHigh.GetFloat(),
        Input_UserLow.GetFloat(),
        Input_UserClose.GetFloat(),
        Input_UseDaySessionOnly.GetYesNo(),
        fPivotPoint,
        fPivotPointHigh,
        fPivotPointLow,
        fR0_5,
        fR1, fR1_5,
        fR2, fR2_5,
        fR3,
        fS0_5,

```

```

        fS1, fS1_5,
        fS2, fS2_5,
        fS3,
        fR3_5,
        fS3_5,
        fR4,
        fR4_5,
        fS4,
        fS4_5,
        fR5,
        fS5,
        fR6,
        fS6,
        fR7,
        fS7,
        fR8,
        fS8,
        fR9,
        fS9,
        fR10,
        fS10,
        Input_UseDailyChartForSettlementOnly.GetYesNo()
    );
}

if (!ValidPivotPoint)
    continue;

if (Input_RoundToTickSize.GetYesNo() != 0)
{
    Subgraph_R1[Index] = static_cast<float>(sc.RoundToTickSize(fR1, sc.TickSize));
    Subgraph_R2[Index] = static_cast<float>(sc.RoundToTickSize(fR2, sc.TickSize));
    Subgraph_S1[Index] = static_cast<float>(sc.RoundToTickSize(fS1, sc.TickSize));
    Subgraph_S2[Index] = static_cast<float>(sc.RoundToTickSize(fS2, sc.TickSize));

    Subgraph_R0_5[Index] = static_cast<float>(sc.RoundToTickSize(fR0_5, sc.TickSize));
    Subgraph_R1_5[Index] = static_cast<float>(sc.RoundToTickSize(fR1_5, sc.TickSize));
    Subgraph_R2_5[Index] = static_cast<float>(sc.RoundToTickSize(fR2_5, sc.TickSize));
    Subgraph_R3[Index] = static_cast<float>(sc.RoundToTickSize(fR3, sc.TickSize));
    Subgraph_S0_5[Index] = static_cast<float>(sc.RoundToTickSize(fS0_5, sc.TickSize));
    Subgraph_S1_5[Index] = static_cast<float>(sc.RoundToTickSize(fS1_5, sc.TickSize));
    Subgraph_S2_5[Index] = static_cast<float>(sc.RoundToTickSize(fS2_5, sc.TickSize));
    Subgraph_S3[Index] = static_cast<float>(sc.RoundToTickSize(fS3, sc.TickSize));
    Subgraph_PP[Index] = static_cast<float>(sc.RoundToTickSize(fPivotPoint, sc.TickSize));
    Subgraph_PPHigh[Index] = static_cast<float>(sc.RoundToTickSize(fPivotPointHigh, sc.TickSize));
    Subgraph_PPLow[Index] = static_cast<float>(sc.RoundToTickSize(fPivotPointLow, sc.TickSize));
    Subgraph_R4[Index] = static_cast<float>(sc.RoundToTickSize(fR4, sc.TickSize));
    Subgraph_R4_5[Index] = static_cast<float>(sc.RoundToTickSize(fR4_5, sc.TickSize));
    Subgraph_S4[Index] = static_cast<float>(sc.RoundToTickSize(fS4, sc.TickSize));
    Subgraph_S4_5[Index] = static_cast<float>(sc.RoundToTickSize(fS4_5, sc.TickSize));
    Subgraph_R3_5[Index] = static_cast<float>(sc.RoundToTickSize(fR3_5, sc.TickSize));
    Subgraph_S3_5[Index] = static_cast<float>(sc.RoundToTickSize(fS3_5, sc.TickSize));
    Subgraph_R5[Index] = static_cast<float>(sc.RoundToTickSize(fR5, sc.TickSize));
    Subgraph_S5[Index] = static_cast<float>(sc.RoundToTickSize(fS5, sc.TickSize));
    Subgraph_R6[Index] = static_cast<float>(sc.RoundToTickSize(fR6, sc.TickSize));
    Subgraph_S6[Index] = static_cast<float>(sc.RoundToTickSize(fS6, sc.TickSize));
    Subgraph_R7[Index] = static_cast<float>(sc.RoundToTickSize(fR7, sc.TickSize));
    Subgraph_S7[Index] = static_cast<float>(sc.RoundToTickSize(fS7, sc.TickSize));
    Subgraph_R8[Index] = static_cast<float>(sc.RoundToTickSize(fR8, sc.TickSize));
    Subgraph_S8[Index] = static_cast<float>(sc.RoundToTickSize(fS8, sc.TickSize));
    Subgraph_R9[Index] = static_cast<float>(sc.RoundToTickSize(fR9, sc.TickSize));
    Subgraph_S9[Index] = static_cast<float>(sc.RoundToTickSize(fS9, sc.TickSize));
    Subgraph_R10[Index] = static_cast<float>(sc.RoundToTickSize(fR10, sc.TickSize));
    Subgraph_S10[Index] = static_cast<float>(sc.RoundToTickSize(fS10, sc.TickSize));
}

```

```

else
{
    Subgraph_R1[Index] = fR1;
    Subgraph_R2[Index] = fR2;
    Subgraph_S1[Index] = fS1;
    Subgraph_S2[Index] = fS2;

    Subgraph_R0_5[Index] = fR0_5;
    Subgraph_R1_5[Index] = fR1_5;
    Subgraph_R2_5[Index] = fR2_5;
    Subgraph_R3[Index] = fR3;
    Subgraph_S0_5[Index] = fS0_5;
    Subgraph_S1_5[Index] = fS1_5;
    Subgraph_S2_5[Index] = fS2_5;
    Subgraph_S3[Index] = fS3;
    Subgraph_PP[Index] = fPivotPoint;
    Subgraph_PPHigh[Index] = fPivotPointHigh;
    Subgraph_PPLow[Index] = fPivotPointLow;
    Subgraph_R4[Index] = fR4;
    Subgraph_R4_5[Index] = fR4_5;
    Subgraph_S4[Index] = fS4;
    Subgraph_S4_5[Index] = fS4_5;
    Subgraph_R3_5[Index] = fR3_5;
    Subgraph_S3_5[Index] = fS3_5;
    Subgraph_R5[Index] = fR5;
    Subgraph_S5[Index] = fS5;
    Subgraph_R6[Index] = fR6;
    Subgraph_S6[Index] = fS6;
    Subgraph_R7[Index] = fR7;
    Subgraph_S7[Index] = fS7;
    Subgraph_R8[Index] = fR8;
    Subgraph_S8[Index] = fS8;
    Subgraph_R9[Index] = fR9;
    Subgraph_S9[Index] = fS9;
    Subgraph_R10[Index] = fR10;
    Subgraph_S10[Index] = fS10;
}
}
}

/*=====*/

```

SCSFExport scsf\_PivotPointsVariablePeriod(SCStudyInterfaceRef sc)

```

{
    SCSubgraphRef Subgraph_R1 = sc.Subgraph[0];
    SCSubgraphRef Subgraph_R2 = sc.Subgraph[1];
    SCSubgraphRef Subgraph_S1 = sc.Subgraph[2];
    SCSubgraphRef Subgraph_S2 = sc.Subgraph[3];
    SCSubgraphRef Subgraph_R0_5 = sc.Subgraph[4];
    SCSubgraphRef Subgraph_R1_5 = sc.Subgraph[5];
    SCSubgraphRef Subgraph_R2_5 = sc.Subgraph[6];
    SCSubgraphRef Subgraph_R3 = sc.Subgraph[7];
    SCSubgraphRef Subgraph_S0_5 = sc.Subgraph[8];
    SCSubgraphRef Subgraph_S1_5 = sc.Subgraph[9];
    SCSubgraphRef Subgraph_S2_5 = sc.Subgraph[10];
    SCSubgraphRef Subgraph_S3 = sc.Subgraph[11];
    SCSubgraphRef Subgraph_PP = sc.Subgraph[12];
    SCSubgraphRef Subgraph_PPHigh = sc.Subgraph[13];
    SCSubgraphRef Subgraph_PPLow = sc.Subgraph[14];
    SCSubgraphRef Subgraph_R4 = sc.Subgraph[15];
    SCSubgraphRef Subgraph_S4 = sc.Subgraph[16];
    SCSubgraphRef Subgraph_R3_5 = sc.Subgraph[17];
    SCSubgraphRef Subgraph_S3_5 = sc.Subgraph[18];
    SCSubgraphRef Subgraph_R5 = sc.Subgraph[19];
    SCSubgraphRef Subgraph_S5 = sc.Subgraph[20];
}

```

```

SCSubgraphRef Subgraph_R6 = sc.Subgraph[21];
SCSubgraphRef Subgraph_S6 = sc.Subgraph[22];
SCSubgraphRef Subgraph_R7 = sc.Subgraph[23];
SCSubgraphRef Subgraph_S7 = sc.Subgraph[24];
SCSubgraphRef Subgraph_R8 = sc.Subgraph[25];
SCSubgraphRef Subgraph_S8 = sc.Subgraph[26];
SCSubgraphRef Subgraph_R9 = sc.Subgraph[27];
SCSubgraphRef Subgraph_S9 = sc.Subgraph[28];
SCSubgraphRef Subgraph_R10 = sc.Subgraph[29];
SCSubgraphRef Subgraph_S10 = sc.Subgraph[30];
SCSubgraphRef Subgraph_R4_5 = sc.Subgraph[31];
SCSubgraphRef Subgraph_S4_5 = sc.Subgraph[32];
const int NUMBER_OF_STUDY_SUBGRAPHS = 33;

SCInputRef Input_TimePeriodType = sc.Input[0];
SCInputRef Input_TimePeriodLength = sc.Input[1];
SCInputRef Input_FormulaType = sc.Input[2];
SCInputRef Input_MinimumRequiredTimePeriodAsPercent = sc.Input[3];
SCInputRef Input_RoundToTickSize = sc.Input[4];
SCInputRef Input_DisplayDebuggingOutput = sc.Input[5];
SCInputRef Input_ForwardProjectLines = sc.Input[6];
SCInputRef Input_NumberOfDaysToCalculate = sc.Input[7];
SCInputRef Input_AutoSkipPeriodOfNoTrading = sc.Input[8];
SCInputRef Input_NumberForwardProjectionBars = sc.Input[9];

if (sc.SetDefaults)
{
    sc.GraphName = "Pivot Points-Variable Period";

    sc.ScaleRangeType = SCALE_SAMEASREGION;

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

    Subgraph_R1.Name = "R1";
    Subgraph_R2.Name = "R2";
    Subgraph_S1.Name = "S1";
    Subgraph_S2.Name = "S2";
    Subgraph_R0_5.Name = "R.5";
    Subgraph_R1_5.Name = "R1.5";
    Subgraph_R2_5.Name = "R2.5";
    Subgraph_R3.Name = "R3";
    Subgraph_S0_5.Name = "S.5";
    Subgraph_S1_5.Name = "S1.5";
    Subgraph_S2_5.Name = "S2.5";
    Subgraph_S3.Name = "S3";
    Subgraph_PP.Name = "PP";
    Subgraph_PPHigh.Name = "PP High";
    Subgraph_PPLow.Name = "PP Low";
    Subgraph_R4.Name = "R4";
    Subgraph_S4.Name = "S4";
    Subgraph_R3_5.Name = "R3.5";
    Subgraph_S3_5.Name = "S3.5";
    Subgraph_R5.Name = "R5";
    Subgraph_S5.Name = "S5";
    Subgraph_R6.Name = "R6";
    Subgraph_S6.Name = "S6";
    Subgraph_R7.Name = "R7";
    Subgraph_S7.Name = "S7";
    Subgraph_R8.Name = "R8";
    Subgraph_S8.Name = "S8";
    Subgraph_R9.Name = "R9";
    Subgraph_S9.Name = "S9";
    Subgraph_R10.Name = "R10";

```



```
Subgraph_S10.Name = "S10";
Subgraph_R4_5.Name = "R4.5";
Subgraph_S4_5.Name = "S4.5";
```

```
Subgraph_R1.DrawStyle = DRAWSTYLE_DASH;
Subgraph_R2.DrawStyle = DRAWSTYLE_DASH;
Subgraph_S1.DrawStyle = DRAWSTYLE_DASH;
Subgraph_S2.DrawStyle = DRAWSTYLE_DASH;
Subgraph_R0_5.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_R1_5.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_R2_5.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_R3.DrawStyle = DRAWSTYLE_DASH;
Subgraph_S0_5.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_S1_5.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_S2_5.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_S3.DrawStyle = DRAWSTYLE_DASH;
Subgraph_PP.DrawStyle = DRAWSTYLE_DASH;
Subgraph_PPHigh.DrawStyle = DRAWSTYLE_DASH;
Subgraph_PPLow.DrawStyle = DRAWSTYLE_DASH;
Subgraph_R4.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_S4.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_R3_5.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_S3_5.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_R5.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_S5.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_R6.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_S6.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_R7.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_S7.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_R8.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_S8.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_R9.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_S9.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_R10.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_S10.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_R4_5.DrawStyle = DRAWSTYLE_HIDDEN;
Subgraph_S4_5.DrawStyle = DRAWSTYLE_HIDDEN;
```

```
Subgraph_R1.DrawZeros = false;
Subgraph_R2.DrawZeros = false;
Subgraph_S1.DrawZeros = false;
Subgraph_S2.DrawZeros = false;
Subgraph_R0_5.DrawZeros = false;
Subgraph_R1_5.DrawZeros = false;
Subgraph_R2_5.DrawZeros = false;
Subgraph_R3.DrawZeros = false;
Subgraph_S0_5.DrawZeros = false;
Subgraph_S1_5.DrawZeros = false;
Subgraph_S2_5.DrawZeros = false;
Subgraph_S3.DrawZeros = false;
Subgraph_PP.DrawZeros = false;
Subgraph_PPHigh.DrawZeros = false;
Subgraph_PPLow.DrawZeros = false;
Subgraph_R4.DrawZeros = false;
Subgraph_S4.DrawZeros = false;
Subgraph_R3_5.DrawZeros = false;
Subgraph_S3_5.DrawZeros = false;
Subgraph_R5.DrawZeros = false;
Subgraph_S5.DrawZeros = false;
Subgraph_R6.DrawZeros = false;
Subgraph_S6.DrawZeros = false;
Subgraph_R7.DrawZeros = false;
Subgraph_S7.DrawZeros = false;
Subgraph_R8.DrawZeros = false;
```



```
Subgraph_S8.DrawZeros = false;
Subgraph_R9.DrawZeros = false;
Subgraph_S9.DrawZeros = false;
Subgraph_R10.DrawZeros = false;
Subgraph_S10.DrawZeros = false;
Subgraph_R4_5.DrawZeros = false;
Subgraph_S4_5.DrawZeros = false;
```

```
Subgraph_PP.PrimaryColor = RGB(255,0,255);
Subgraph_PPHigh.PrimaryColor = RGB(255, 0, 255);
Subgraph_PPLow.PrimaryColor = RGB(255, 0, 255);
```

```
Subgraph_R1.PrimaryColor = RGB(255, 0, 0);
Subgraph_R2.PrimaryColor = RGB(255, 0, 0);
Subgraph_R0_5.PrimaryColor = RGB(255, 0, 0);
Subgraph_R1_5.PrimaryColor = RGB(255, 0, 0);
Subgraph_R2_5.PrimaryColor = RGB(255, 0, 0);
Subgraph_R3.PrimaryColor = RGB(255, 0, 0);
Subgraph_R3_5.PrimaryColor = RGB(255, 0, 0);
Subgraph_R4.PrimaryColor = RGB(255, 0, 0);
Subgraph_R4_5.PrimaryColor = RGB(255, 0, 0);
Subgraph_R5.PrimaryColor = RGB(255, 0, 0);
Subgraph_R6.PrimaryColor = RGB(255, 0, 0);
Subgraph_R7.PrimaryColor = RGB(255, 0, 0);
Subgraph_R8.PrimaryColor = RGB(255, 0, 0);
Subgraph_R9.PrimaryColor = RGB(255, 0, 0);
Subgraph_R10.PrimaryColor = RGB(255, 0, 0);
```

```
Subgraph_S1.PrimaryColor = RGB(0, 255, 0);
Subgraph_S2.PrimaryColor = RGB(0, 255, 0);
Subgraph_S0_5.PrimaryColor = RGB(0, 255, 0);
Subgraph_S1_5.PrimaryColor = RGB(0, 255, 0);
Subgraph_S2_5.PrimaryColor = RGB(0, 255, 0);
Subgraph_S3.PrimaryColor = RGB(0, 255, 0);
Subgraph_S3_5.PrimaryColor = RGB(0, 255, 0);
Subgraph_S4.PrimaryColor = RGB(0, 255, 0);
Subgraph_S4_5.PrimaryColor = RGB(0, 255, 0);
Subgraph_S5.PrimaryColor = RGB(0, 255, 0);
Subgraph_S6.PrimaryColor = RGB(0, 255, 0);
Subgraph_S7.PrimaryColor = RGB(0, 255, 0);
Subgraph_S8.PrimaryColor = RGB(0, 255, 0);
Subgraph_S9.PrimaryColor = RGB(0, 255, 0);
Subgraph_S10.PrimaryColor = RGB(0, 255, 0);
```

```
for (int Index = 0; Index < NUMBER_OF_STUDY_SUBGRAPHS; Index++)
{
    sc.Subgraph[Index].LineLabel =
        LL_DISPLAY_NAME | LL_DISPLAY_VALUE | LL_NAME_ALIGN_ABOVE | LL_NAME_ALIGN_LEFT |
        LL_VALUE_ALIGN_CENTER | LL_VALUE_ALIGN_VALUES_SCALE;
}
```

```
Input_TimePeriodType.Name = "Time Period Type";
Input_TimePeriodType.SetTimePeriodType(TIME_PERIOD_LENGTH_UNIT_MINUTES);
```

```
Input_TimePeriodLength.Name = "Time Period Length";
Input_TimePeriodLength.SetInt(60);
Input_TimePeriodLength.SetIntLimits(1, 7*MINUTES_PER_DAY);
```

```
Input_FormulaType.Name = "Formula Type";
Input_FormulaType.SetInt(0);
```

```
Input_MinimumRequiredTimePeriodAsPercent.Name = "Minimum Required Time Period as %";
Input_MinimumRequiredTimePeriodAsPercent.SetFloat(25.0f);
Input_MinimumRequiredTimePeriodAsPercent.SetFloatLimits(1.0f, 100.0f);
```

```

Input_RoundToTickSize.Name = "Round to Tick Size";
Input_RoundToTickSize.SetYesNo(0);

Input_DisplayDebuggingOutput.Name = "Display Debugging Output (slows study calculations)";
Input_DisplayDebuggingOutput.SetYesNo(0);

Input_ForwardProjectLines.Name = "Forward Project Pivot Point Lines";
Input_ForwardProjectLines.SetYesNo(0);

Input_NumberOfDaysToCalculate.Name = "Number Of Days To Calculate";
Input_NumberOfDaysToCalculate.SetInt(1000);

Input_AutoSkipPeriodOfNoTrading.Name = "Auto Skip Period Of No Trading";
Input_AutoSkipPeriodOfNoTrading.SetYesNo(false);

Input_NumberForwardProjectionBars.Name = "Number of Forward Project Bars";
Input_NumberForwardProjectionBars.SetInt(20);
Input_NumberForwardProjectionBars.SetIntLimits(1,200);
return;
}

float f_PivotPoint= 0, f_PivotPointHigh = 0, f_PivotPointLow = 0;

float fR0_5 = 0, fR1 = 0, fR1_5 = 0, fR2 = 0, fR2_5 = 0, fR3 = 0, fR3_5 = 0, fR4 = 0, fR4_5 = 0, fR5 = 0, fR6 = 0, fR7 = 0, fR8 = 0, fR9 = 0, fR10 = 0;

float fS0_5 = 0, fS1 = 0, fS1_5 = 0, fS2 = 0, fS2_5 = 0, fS3 = 0, fS3_5 = 0, fS4 = 0, fS4_5 = 0, fS5 = 0, fS6 = 0, fS7 = 0, fS8 = 0, fS9 = 0, fS10 = 0;

int ValidPivotPoint = 1;

int PeriodLength = Input_TimePeriodLength.GetInt();

int NumberOfForwardBars = 0;

if (Input_ForwardProjectLines.GetYesNo())
{
    NumberOfForwardBars = Input_NumberForwardProjectionBars.GetInt();

    if(sc.UpdateStartIndex == 0)
    {
        Subgraph_PP.ExtendedArrayElementsToGraph = NumberOfForwardBars;
        Subgraph_PPHigh.ExtendedArrayElementsToGraph = NumberOfForwardBars;
        Subgraph_PPLow.ExtendedArrayElementsToGraph = NumberOfForwardBars;
        Subgraph_R1.ExtendedArrayElementsToGraph = NumberOfForwardBars;
        Subgraph_R2.ExtendedArrayElementsToGraph = NumberOfForwardBars;
        Subgraph_S1.ExtendedArrayElementsToGraph = NumberOfForwardBars;
        Subgraph_S2.ExtendedArrayElementsToGraph = NumberOfForwardBars;
        Subgraph_R0_5.ExtendedArrayElementsToGraph = NumberOfForwardBars;
        Subgraph_R1_5.ExtendedArrayElementsToGraph = NumberOfForwardBars;
        Subgraph_R2_5.ExtendedArrayElementsToGraph = NumberOfForwardBars;
        Subgraph_R3.ExtendedArrayElementsToGraph = NumberOfForwardBars;
        Subgraph_S0_5.ExtendedArrayElementsToGraph = NumberOfForwardBars;
        Subgraph_S1_5.ExtendedArrayElementsToGraph = NumberOfForwardBars;
        Subgraph_S2_5.ExtendedArrayElementsToGraph = NumberOfForwardBars;
        Subgraph_S3.ExtendedArrayElementsToGraph = NumberOfForwardBars;
        Subgraph_R4.ExtendedArrayElementsToGraph = NumberOfForwardBars;
        Subgraph_S4.ExtendedArrayElementsToGraph = NumberOfForwardBars;
        Subgraph_R3_5.ExtendedArrayElementsToGraph = NumberOfForwardBars;
        Subgraph_S3_5.ExtendedArrayElementsToGraph = NumberOfForwardBars;
        Subgraph_R5.ExtendedArrayElementsToGraph = NumberOfForwardBars;
        Subgraph_S5.ExtendedArrayElementsToGraph = NumberOfForwardBars;
        Subgraph_R6.ExtendedArrayElementsToGraph = NumberOfForwardBars;
        Subgraph_S6.ExtendedArrayElementsToGraph = NumberOfForwardBars;
        Subgraph_R7.ExtendedArrayElementsToGraph = NumberOfForwardBars;
    }
}

```

```

        Subgraph_S7.ExtendedArrayElementsToGraph = NumberOfForwardBars;
        Subgraph_R8.ExtendedArrayElementsToGraph = NumberOfForwardBars;
        Subgraph_S8.ExtendedArrayElementsToGraph = NumberOfForwardBars;
        Subgraph_R9.ExtendedArrayElementsToGraph = NumberOfForwardBars;
        Subgraph_S9.ExtendedArrayElementsToGraph = NumberOfForwardBars;
        Subgraph_R10.ExtendedArrayElementsToGraph = NumberOfForwardBars;
        Subgraph_S10.ExtendedArrayElementsToGraph = NumberOfForwardBars;
        Subgraph_R4_5.ExtendedArrayElementsToGraph = NumberOfForwardBars;
        Subgraph_S4_5.ExtendedArrayElementsToGraph = NumberOfForwardBars;
    }
}

if (Input_NumberOfDaysToCalculate.GetInt() < 1)
    Input_NumberOfDaysToCalculate.SetInt(1000);

int LastDateInChart = sc.BaseDateTimeIn[sc.ArraySize - 1].GetDate();
int FirstDateToCalculate = LastDateInChart - Input_NumberOfDaysToCalculate.GetInt() + 1;

float Open = 0, High = 0, Low = 0, Close = 0, NextOpen = 0;

SCDateTime BeginOfRefDateTime, EndOfRefDateTime;
SCDateTime CurrentPeriodBeginDateTime;
SCDateTime CurrentPeriodEndDateTime;

SCDateTime PriorCurrentPeriodStartDateTime;

for (int index = sc.UpdateStartIndex; index < sc.ArraySize+NumberOfForwardBars; index++)
{
    SCDateTime CurrentBarDT = sc.BaseDateTimeIn[index];

    if (CurrentBarDT < FirstDateToCalculate)
        continue;

    bool GetReferenceData = true;

    CurrentPeriodBeginDateTime = sc.GetStartOfPeriodForDateTime(CurrentBarDT,
Input_TimePeriodType.GetTimePeriodType(), PeriodLength, 0);

    if (CurrentPeriodBeginDateTime == PriorCurrentPeriodStartDateTime)
        GetReferenceData = false;

    PriorCurrentPeriodStartDateTime = CurrentPeriodBeginDateTime;

    SCDateTime TimeIncrement = sc.TimePeriodSpan(Input_TimePeriodType.GetTimePeriodType(), PeriodLength);

    if (GetReferenceData)
    {
        BeginOfRefDateTime= sc.GetStartOfPeriodForDateTime( CurrentPeriodBeginDateTime,
Input_TimePeriodType.GetTimePeriodType(), PeriodLength, -1);
        EndOfRefDateTime= CurrentPeriodBeginDateTime - SCDateTime::SECONDS(1);

        CurrentPeriodEndDateTime = sc.GetStartOfPeriodForDateTime(CurrentPeriodBeginDateTime,
Input_TimePeriodType.GetTimePeriodType(), PeriodLength, 1) - SCDateTime::SECONDS(1);

        if (Input_DisplayDebuggingOutput.GetYesNo() != 0)
        {
            SCString Message;

            Message.Format("Current Bar: %s, BeginOfRefDateTime: %s, EndOfRefDateTime: %s, CPBeginDateTime:

```

```

%s, CPendTime: %s",
    sc.FormatDateTime(CurrentBarDT).GetChars(), sc.FormatDateTime(BeginOfRefDateTime).GetChars(),
    sc.FormatDateTime(EndOfRefDateTime).GetChars(), sc.FormatDateTime(CurrentPeriodBeginDateTime).GetChars(),
    sc.FormatDateTime(CurrentPeriodEndDateTime).GetChars());

    sc.AddMessageToLog(Message, 0);
}

int MaxPeriodsToGoBack = 1;

if (Input_AutoSkipPeriodOfNoTrading.GetYesNo())
    MaxPeriodsToGoBack = 32;

for (int WalkBack = 0; WalkBack < MaxPeriodsToGoBack; WalkBack++)
{
    if (WalkBack >= 1) //Walk back 1 period.
    {
        SCDatetime PriorBeginOfRefDateTime = BeginOfRefDateTime;

        BeginOfRefDateTime = sc.GetStartOfPeriodForDateTime(BeginOfRefDateTime,
            Input_TimePeriodType.GetTimePeriodType(), PeriodLength, -1);
        EndOfRefDateTime = PriorBeginOfRefDateTime - SCDatetime::SECONDS(1);

        if (Input_DisplayDebuggingOutput.GetYesNo() != 0)
        {
            SCString Message;

            Message.Format("Moving back 1 period. BeginOfRefDateTime: %s, EndOfRefDateTime: %s.",
                sc.FormatDateTime(BeginOfRefDateTime).GetChars(), sc.FormatDateTime(EndOfRefDateTime).GetChars());

            sc.AddMessageToLog(Message, 0);
        }
    }

    int NumberOfBars = 0;
    SCDatetime TotalTimeSpan;
    int Result = sc.GetOHLCOfTimePeriod(BeginOfRefDateTime, EndOfRefDateTime, Open, High, Low, Close,
        NextOpen, NumberOfBars, TotalTimeSpan);

    if (!Result)
        continue;

    if (Input_DisplayDebuggingOutput.GetYesNo() != 0)
    {
        SCString Message;
        Message.Format("Number of Bars: %d, Total Time Span In Minutes: %d", NumberOfBars, static_cast<int>
            (TotalTimeSpan.GetSecondsSinceBaseDate()/SECONDS_PER_MINUTE));
        sc.AddMessageToLog(Message, 0);

        Message.Format("RefOpen %f, RefHigh %f, RefLow %f, RefClose %f, RefNextOpen %f.", Open, High, Low,
            Close, NextOpen);
        sc.AddMessageToLog(Message, 0);
    }

    SCDatetime MinimumTimeSpan = (TimeIncrement.GetAsDouble() *
        Input_MinimumRequiredTimePeriodAsPercent.GetFloat() / 100.0);

    if (TotalTimeSpan >= MinimumTimeSpan)
        break;
}
}

```

```

ValidPivotPoint =
    CalculatePivotPoints(Open,High,Low,Close,NextOpen,
        f_PivotPoint, f_PivotPointHigh, f_PivotPointLow, fR0_5, fR1, fR1_5, fR2, fR2_5, fR3, fS0_5,
        fS1, fS1_5, fS2, fS2_5, fS3, fR3_5, fS3_5, fR4, fR4_5, fS4, fS4_5, fR5, fS5,
        fR6, fS6, fR7, fS7, fR8, fS8, fR9, fS9, fR10, fS10, Input_FormulaType.GetInt());

if (!ValidPivotPoint)
    continue;

if (Input_RoundToTickSize.GetYesNo() != 0)
{
    Subgraph_R1[index] = static_cast<float>(sc.RoundToTickSize(fR1, sc.TickSize));
    Subgraph_R2[index] = static_cast<float>(sc.RoundToTickSize(fR2, sc.TickSize));
    Subgraph_S1[index] = static_cast<float>(sc.RoundToTickSize(fS1, sc.TickSize));
    Subgraph_S2[index] = static_cast<float>(sc.RoundToTickSize(fS2, sc.TickSize));

    Subgraph_R0_5[index] = static_cast<float>(sc.RoundToTickSize(fR0_5, sc.TickSize));
    Subgraph_R1_5[index] = static_cast<float>(sc.RoundToTickSize(fR1_5, sc.TickSize));
    Subgraph_R2_5[index] = static_cast<float>(sc.RoundToTickSize(fR2_5, sc.TickSize));
    Subgraph_R3[index] = static_cast<float>(sc.RoundToTickSize(fR3, sc.TickSize));
    Subgraph_S0_5[index] = static_cast<float>(sc.RoundToTickSize(fS0_5, sc.TickSize));
    Subgraph_S1_5[index] = static_cast<float>(sc.RoundToTickSize(fS1_5, sc.TickSize));
    Subgraph_S2_5[index] = static_cast<float>(sc.RoundToTickSize(fS2_5, sc.TickSize));
    Subgraph_S3[index] = static_cast<float>(sc.RoundToTickSize(fS3, sc.TickSize));
    Subgraph_PP[index] = static_cast<float>(sc.RoundToTickSize(f_PivotPoint, sc.TickSize));
    Subgraph_PPHigh[index] = static_cast<float>(sc.RoundToTickSize(f_PivotPointHigh, sc.TickSize));
    Subgraph_PPLow[index] = static_cast<float>(sc.RoundToTickSize(f_PivotPointLow, sc.TickSize));
    Subgraph_R4[index] = static_cast<float>(sc.RoundToTickSize(fR4, sc.TickSize));
    Subgraph_R4_5[index] = static_cast<float>(sc.RoundToTickSize(fR4_5, sc.TickSize));
    Subgraph_S4[index] = static_cast<float>(sc.RoundToTickSize(fS4, sc.TickSize));
    Subgraph_S4_5[index] = static_cast<float>(sc.RoundToTickSize(fS4_5, sc.TickSize));
    Subgraph_R3_5[index] = static_cast<float>(sc.RoundToTickSize(fR3_5, sc.TickSize));
    Subgraph_S3_5[index] = static_cast<float>(sc.RoundToTickSize(fS3_5, sc.TickSize));
    Subgraph_R5[index] = static_cast<float>(sc.RoundToTickSize(fR5, sc.TickSize));
    Subgraph_S5[index] = static_cast<float>(sc.RoundToTickSize(fS5, sc.TickSize));
    Subgraph_R6[index] = static_cast<float>(sc.RoundToTickSize(fR6, sc.TickSize));
    Subgraph_S6[index] = static_cast<float>(sc.RoundToTickSize(fS6, sc.TickSize));
    Subgraph_R7[index] = static_cast<float>(sc.RoundToTickSize(fR7, sc.TickSize));
    Subgraph_S7[index] = static_cast<float>(sc.RoundToTickSize(fS7, sc.TickSize));
    Subgraph_R8[index] = static_cast<float>(sc.RoundToTickSize(fR8, sc.TickSize));
    Subgraph_S8[index] = static_cast<float>(sc.RoundToTickSize(fS8, sc.TickSize));
    Subgraph_R9[index] = static_cast<float>(sc.RoundToTickSize(fR9, sc.TickSize));
    Subgraph_S9[index] = static_cast<float>(sc.RoundToTickSize(fS9, sc.TickSize));
    Subgraph_R10[index] = static_cast<float>(sc.RoundToTickSize(fR10, sc.TickSize));
    Subgraph_S10[index] = static_cast<float>(sc.RoundToTickSize(fS10, sc.TickSize));
}
else
{
    Subgraph_R1[index] = fR1;
    Subgraph_R2[index] = fR2;
    Subgraph_S1[index] = fS1;
    Subgraph_S2[index] = fS2;

    Subgraph_R0_5[index] = fR0_5;
    Subgraph_R1_5[index] = fR1_5;
    Subgraph_R2_5[index] = fR2_5;
    Subgraph_R3[index] = fR3;
    Subgraph_S0_5[index] = fS0_5;
    Subgraph_S1_5[index] = fS1_5;
    Subgraph_S2_5[index] = fS2_5;
    Subgraph_S3[index] = fS3;
    Subgraph_PP[index] = f_PivotPoint;
}

```

```

    Subgraph_PPHigh[index] = f_PivotPointHigh;
    Subgraph_PPLow[index] = f_PivotPointLow;
    Subgraph_R4[index] = fR4;
    Subgraph_R4_5[index] = fR4_5;
    Subgraph_S4[index] = fS4;
    Subgraph_S4_5[index] = fS4_5;
    Subgraph_R3_5[index] = fR3_5;
    Subgraph_S3_5[index] = fS3_5;
    Subgraph_R5[index] = fR5;
    Subgraph_S5[index] = fS5;
    Subgraph_R6[index] = fR6;
    Subgraph_S6[index] = fS6;
    Subgraph_R7[index] = fR7;
    Subgraph_S7[index] = fS7;
    Subgraph_R8[index] = fR8;
    Subgraph_S8[index] = fS8;
    Subgraph_R9[index] = fR9;
    Subgraph_S9[index] = fS9;
    Subgraph_R10[index] = fR10;
    Subgraph_S10[index] = fS10;
}
} //for
}

/*=====*/

SCSFExport scsf_PivotRangeVariablePeriod(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_HLCAvgPlusHLCAvgMinusHLAvg = sc.Subgraph[0];
    SCSubgraphRef Subgraph_PreviousHLCAvg = sc.Subgraph[1];
    SCSubgraphRef Subgraph_PreviousHLCAvg_2 = sc.Subgraph[2];
    SCSubgraphRef Subgraph_HLCAvgMinusHLCAvgMinusHLAvg = sc.Subgraph[3];
    SCSubgraphRef Subgraph_HLAvg = sc.Subgraph[4];

    SCInputRef Input_TimePeriodType = sc.Input[0];
    SCInputRef Input_TimePeriodLength = sc.Input[1];

    SCInputRef Input_MinimumRequiredTP = sc.Input[3];
    SCInputRef Input_RoundToTickSize = sc.Input[4];
    SCInputRef Input_DisplayDebuggingOutput = sc.Input[5];
    SCInputRef Input_ForwardProjectLines = sc.Input[6];
    SCInputRef Input_NumberOfDaysToCalculate = sc.Input[7];
    SCInputRef Input_AutoSkipPeriodOfNoTrading = sc.Input[8];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Pivot Range-Variable Period";

        sc.ScaleRangeType = SCALE_SAMEASREGION;

        sc.GraphRegion = 0;
        sc.ValueFormat = VALUEFORMAT_INHERITED;
        sc.DrawStudyUnderneathMainPriceGraph= true;

        sc.AutoLoop = 0;

        Subgraph_HLCAvgPlusHLCAvgMinusHLAvg.Name = "Top Range";
        Subgraph_PreviousHLCAvg.Name = "HLC Avg";

        Subgraph_PreviousHLCAvg_2.Name = "HLC Avg";
        Subgraph_HLCAvgMinusHLCAvgMinusHLAvg.Name = "Bottom Range";

        Subgraph_HLCAvgPlusHLCAvgMinusHLAvg.DrawStyle = DRAWSTYLE_FILL_RECTANGLE_TOP;
        Subgraph_PreviousHLCAvg.DrawStyle = DRAWSTYLE_FILL_RECTANGLE_BOTTOM;
    }
}

```



```

Subgraph_PreviousHLCAvg_2.DrawStyle = DRAWSTYLE_FILL_RECTANGLE_TOP;
Subgraph_HLCAvgMinusHLCAvgMinusHLAvg.DrawStyle = DRAWSTYLE_FILL_RECTANGLE_BOTTOM;

Subgraph_HLCAvgPlusHLCAvgMinusHLAvg.DrawZeros = false;
Subgraph_PreviousHLCAvg.DrawZeros = false;
Subgraph_PreviousHLCAvg_2.DrawZeros = false;
Subgraph_HLCAvgMinusHLCAvgMinusHLAvg.DrawZeros = false;

Subgraph_HLCAvgPlusHLCAvgMinusHLAvg.PrimaryColor = RGB(128, 255, 255);
Subgraph_PreviousHLCAvg.PrimaryColor = RGB(128, 255, 255);
Subgraph_PreviousHLCAvg_2.PrimaryColor = RGB(255, 255, 128);
Subgraph_HLCAvgMinusHLCAvgMinusHLAvg.PrimaryColor = RGB(255, 255, 128);

Subgraph_HLAvg.Name= "HL Avg";
Subgraph_HLAvg.PrimaryColor = RGB(255, 128, 0);
Subgraph_HLAvg.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_HLAvg.DrawZeros = false;

for (int i = 0; i <= 4; i++)
{
    sc.Subgraph[i].LineLabel =
        LL_DISPLAY_NAME | LL_DISPLAY_VALUE | LL_NAME_ALIGN_ABOVE | LL_NAME_ALIGN_RIGHT|
        LL_VALUE_ALIGN_CENTER | LL_VALUE_ALIGN_VALUES_SCALE;
}

Input_TimePeriodType.Name = "Time Period Type";
Input_TimePeriodType.SetTimePeriodType(TIME_PERIOD_LENGTH_UNIT_MINUTES);

Input_TimePeriodLength.Name = "Time Period Length";
Input_TimePeriodLength.SetInt(60);
Input_TimePeriodLength.SetIntLimits(1, 7*MINUTES_PER_DAY);

Input_MinimumRequiredTP.Name = "Minimum Required Time Period as %";
Input_MinimumRequiredTP.SetFloat(25.0f);
Input_MinimumRequiredTP.SetFloatLimits(1.0f, 100.0f);

Input_RoundToTickSize.Name = "Round to Tick Size";
Input_RoundToTickSize.SetYesNo(0);

Input_DisplayDebuggingOutput.Name = "Display Debugging Output (slows study calculations)";
Input_DisplayDebuggingOutput.SetYesNo(0);

Input_ForwardProjectLines.Name = "Forward Project Lines";
Input_ForwardProjectLines.SetYesNo(0);

Input_NumberOfDaysToCalculate.Name = "Number Of Days To Calculate";
Input_NumberOfDaysToCalculate.SetInt(1000);

Input_AutoSkipPeriodOfNoTrading.Name = "Auto Skip Period Of No Trading";
Input_AutoSkipPeriodOfNoTrading.SetYesNo(false);

return;
}

int PeriodLength = Input_TimePeriodLength.GetInt();

int NumberOfForwardBars = 0;

if (Input_ForwardProjectLines.GetYesNo())
{

```

```

NumberOfForwardBars = 20;

if(sc.UpdateStartIndex == 0)
{
    Subgraph_HLCAvgPlusHLCAvgMinusHLAvg.ExtendedArrayElementsToGraph = NumberOfForwardBars;
    Subgraph_PreviousHLCAvg.ExtendedArrayElementsToGraph = NumberOfForwardBars;
    Subgraph_PreviousHLCAvg_2.ExtendedArrayElementsToGraph = NumberOfForwardBars;
    Subgraph_HLCAvgMinusHLCAvgMinusHLAvg.ExtendedArrayElementsToGraph = NumberOfForwardBars;
}
}

if (Input_NumberOfDaysToCalculate.GetInt() < 1)
    Input_NumberOfDaysToCalculate.SetInt(1000);

int LastDateInChart = sc.BaseDateTimeIn[sc.ArraySize - 1].GetDate();
int FirstDateToCalculate = LastDateInChart - Input_NumberOfDaysToCalculate.GetInt() + 1;

float PreviousOpen = 0, PreviousHigh = 0, PreviousLow = 0, PreviousClose = 0, NextOpen = 0;

SCDateTime BeginOfRefDateTime,EndOfRefDateTime;
SCDateTime CurrentPeriodBeginDateTime;
SCDateTime CurrentPeriodEndDateTime;

SCDateTime PriorCurrentPeriodStartDateTime;

for (int index = sc.UpdateStartIndex; index < sc.ArraySize+NumberOfForwardBars; index++)
{
    SCDateTime CurrentBarDT = sc.BaseDateTimeIn[index];

    if (CurrentBarDT < FirstDateToCalculate)
        continue;

    bool GetReferenceData = true;

    CurrentPeriodBeginDateTime = sc.GetStartOfPeriodForDateTime(CurrentBarDT,
Input_TimePeriodType.GetTimePeriodType(), PeriodLength, 0);

    if( CurrentPeriodBeginDateTime == PriorCurrentPeriodStartDateTime)
        GetReferenceData = false;

    PriorCurrentPeriodStartDateTime = CurrentPeriodBeginDateTime;

    SCDateTime TimeIncrement = sc.TimePeriodSpan(Input_TimePeriodType.GetTimePeriodType(), PeriodLength);

    if(GetReferenceData)
    {
        BeginOfRefDateTime= sc.GetStartOfPeriodForDateTime( CurrentPeriodBeginDateTime,
Input_TimePeriodType.GetTimePeriodType(), PeriodLength, -1);
        EndOfRefDateTime= CurrentPeriodBeginDateTime - SCDateTime::SECONDS(1);

        CurrentPeriodEndDateTime=sc.GetStartOfPeriodForDateTime( CurrentPeriodBeginDateTime,
Input_TimePeriodType.GetTimePeriodType(), PeriodLength, 1) - SCDateTime::SECONDS(1);

        if (Input_DisplayDebuggingOutput.GetYesNo() != 0)
        {
            SCString Message;

            Message.Format("Current Bar: %s, BeginOfRefDateTime: %s, EndOfRefDateTime: %s, CPBeginDateTime:
%s, CPEndDateTime: %s",

```



```

        sc.FormatDateTime(CurrentBarDT).GetChars(),sc.FormatDateTime(BeginOfRefDateTime).GetChars(),
sc.FormatDateTime(EndOfRefDateTime).GetChars(),sc.FormatDateTime(CurrentPeriodBeginDateTime).GetChars(),
        sc.FormatDateTime(CurrentPeriodEndDateTime).GetChars());

        sc.AddMessageToLog(Message,0);

    }

    int MaxPeriodsToGoBack = 1;

    if(Input_AutoSkipPeriodOfNoTrading.GetYesNo())
        MaxPeriodsToGoBack = 32;

    for (int WalkBack = 0; WalkBack<MaxPeriodsToGoBack;WalkBack++)
    {
        if (WalkBack >= 1) //Walk back 1 period.
        {
            SCDateTime PriorBeginOfRefDateTime=BeginOfRefDateTime;

            BeginOfRefDateTime= sc.GetStartOfPeriodForDateTime(BeginOfRefDateTime,
Input_TimePeriodType.GetTimePeriodType(), PeriodLength, -1);
            EndOfRefDateTime= PriorBeginOfRefDateTime- SCDateTime::SECONDS(1);

            if (Input_DisplayDebuggingOutput.GetYesNo() != 0)
            {
                SCString Message;

                Message.Format("Moving back 1 period. BeginOfRefDateTime: %s, EndOfRefDateTime: %s.",
sc.FormatDateTime(BeginOfRefDateTime).GetChars(),sc.FormatDateTime(EndOfRefDateTime).GetChars());

                sc.AddMessageToLog(Message,0);
            }
        }

        int NumberOfBars = 0;
        SCDateTime TotalTimeSpan;
        int Result = sc.GetOHLCOfTimePeriod(BeginOfRefDateTime, EndOfRefDateTime, PreviousOpen,
PreviousHigh, PreviousLow, PreviousClose, NextOpen, NumberOfBars, TotalTimeSpan);
        if (!Result)
            continue;

        if (Input_DisplayDebuggingOutput.GetYesNo() != 0)
        {
            SCString Message;
            Message.Format("Number of Bars: %d, Total Time Span In Minutes: %d", NumberOfBars, static_cast<int>
(TotalTimeSpan.GetSecondsSinceBaseDate() / SECONDS_PER_MINUTE));
            sc.AddMessageToLog(Message,0);

            Message.Format("RefOpen %f,RefHigh %f,RefLow %f,RefClose %f,RefNextOpen %f.",PreviousOpen,
PreviousHigh, PreviousLow, PreviousClose, NextOpen);
            sc.AddMessageToLog(Message,0);
        }

        SCDateTime MinimumTimeSpan = (TimeIncrement.GetAsDouble() * Input_MinimumRequiredTP.GetFloat() /
100.0);
        if (TotalTimeSpan >= MinimumTimeSpan)
            break;
    }
}

Subgraph_PreviousHLCAvg[index] = (PreviousHigh + PreviousLow + PreviousClose)/3.0f;

```

```

Subgraph_PreviousHLCAvg_2[index] = Subgraph_PreviousHLCAvg[index] ;

Subgraph_HLAvg[index] = (PreviousHigh + PreviousLow)/2.0f ;

float HLCAvgMinusHLAvg = Subgraph_PreviousHLCAvg[index] - Subgraph_HLAvg[index];

Subgraph_HLCAvgPlusHLCAvgMinusHLAvg [index] = Subgraph_PreviousHLCAvg[index] + HLCAvgMinusHLAvg;
Subgraph_HLCAvgMinusHLCAvgMinusHLAvg [index] = Subgraph_PreviousHLCAvg[index] - HLCAvgMinusHLAvg;

if (Input_RoundToTickSize.GetYesNo() != 0)
{
    Subgraph_PreviousHLCAvg[index] = sc.RoundToTickSize(Subgraph_PreviousHLCAvg[index], sc.TickSize);
    Subgraph_PreviousHLCAvg_2[index] = sc.RoundToTickSize(Subgraph_PreviousHLCAvg_2[index], sc.TickSize);

    Subgraph_HLCAvgPlusHLCAvgMinusHLAvg
[index] = sc.RoundToTickSize(Subgraph_HLCAvgPlusHLCAvgMinusHLAvg[index], sc.TickSize);
    Subgraph_HLCAvgMinusHLCAvgMinusHLAvg
[index] = sc.RoundToTickSize(Subgraph_HLCAvgMinusHLCAvgMinusHLAvg[index], sc.TickSize);
}

} //for
}
/*=====*/
// This function is used internally by scsf_SwingHighAndLowCustom
int IsSwingLowAllowEqualBackwardsOnly(SCStudyInterfaceRef sc, int AllowEqual, int Index, int Length)
{
    for(int i = 1; i <= Length; i++)
    {
        if (AllowEqual)
        {
            if (sc.FormattedEvaluate(sc.BaseData [SC_LOW][Index] , sc.BaseGraphValueFormat, GREATER_OPERATOR,
sc.BaseData [SC_LOW][Index-i], sc.BaseGraphValueFormat))
                return 0;
        }
        else
        {
            if (sc.FormattedEvaluate(sc.BaseData [SC_LOW][Index] , sc.BaseGraphValueFormat,
GREATER_EQUAL_OPERATOR, sc.BaseData [SC_LOW][Index-i], sc.BaseGraphValueFormat))
                return 0;
        }
    }

    return 1;
}

/*=====*/
SCSFExport scsf_SwingHighAndLowCustom(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_SwingHigh = sc.Subgraph[0];
    SCSubgraphRef SwingLow = sc.Subgraph[1];

    SCSubgraphRef Subgraph_Former = sc.Subgraph[2];
    SCSubgraphRef Subgraph_ColorForInsideBar = sc.Subgraph[3];
    SCSubgraphRef Subgraph_ColorForPotentialSwingHigh = sc.Subgraph[4];
    SCSubgraphRef Subgraph_ColorForPotentialSwingLow = sc.Subgraph[5];

    SCInputRef Input_ArrowOffsetValue = sc.Input[0];
    SCInputRef Input_Length = sc.Input[1];
    SCInputRef Input_AllowEqualBars = sc.Input[2];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Swing High And Low Hollow Bar";
        sc.StudyDescription = "Swing High And Low Hollow Bar";
    }

```

```

Subgraph_SwingHigh.LineWidth = 3;
SwingLow.LineWidth = 3;

sc.GraphRegion = 0;

Subgraph_SwingHigh.Name = "Swing High";
Subgraph_SwingHigh.DrawStyle = DRAWSTYLE_ARROW_DOWN;
Subgraph_SwingHigh.PrimaryColor = RGB(255,0,0);
Subgraph_SwingHigh.DrawZeros = false;

SwingLow.Name = "Swing Low";
SwingLow.DrawStyle = DRAWSTYLE_ARROW_UP;
SwingLow.PrimaryColor = RGB(0,255,0);
SwingLow.DrawZeros = false;

Subgraph_Former.Name = "Potential Former";
Subgraph_Former.DrawStyle = DRAWSTYLE_COLOR_BAR_HOLLOW;
Subgraph_Former.PrimaryColor = RGB(255,255,0);
Subgraph_Former.DrawZeros = false;

Subgraph_ColorForInsideBar.Name = "Color for Potential Inside Bar";
Subgraph_ColorForInsideBar.DrawStyle = DRAWSTYLE_LINE;
Subgraph_ColorForInsideBar.PrimaryColor = RGB(255,128,0);
Subgraph_ColorForInsideBar.DrawZeros = false;

Subgraph_ColorForPotentialSwingHigh.Name = "Color for Potential Swing High";
Subgraph_ColorForPotentialSwingHigh.DrawStyle = DRAWSTYLE_LINE;
Subgraph_ColorForPotentialSwingHigh.PrimaryColor = RGB(0,0,255);
Subgraph_ColorForPotentialSwingHigh.DrawZeros = false;

Subgraph_ColorForPotentialSwingLow.Name = "Color for Potential Swing Low";
Subgraph_ColorForPotentialSwingLow.DrawStyle = DRAWSTYLE_LINE;
Subgraph_ColorForPotentialSwingLow.PrimaryColor = RGB(255,255,255);
Subgraph_ColorForPotentialSwingLow.DrawZeros = false;

Input_ArrowOffsetValue.Name = "Arrow Offset as Value";
Input_ArrowOffsetValue.SetFloat(0);

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

Input_AllowEqualBars.Name = "Allow Equal High/Low Bars";
Input_AllowEqualBars.SetYesNo(false);

sc.AutoLoop = true;
return;
}

SCFloatArrayRef High = sc.High;
SCFloatArrayRef Low = sc.Low;

int MiddleIndex = sc.Index - Input_Length.GetInt();

Subgraph_Former[sc.Index] = High[sc.Index];

if (IsSwingHighAllowEqual_S(sc, Input_AllowEqualBars.GetYesNo(), sc.Index, Input_Length.GetInt()))
{
    Subgraph_Former.Arrays[0][sc.Index] = 1;
    Subgraph_Former.DataColor[sc.Index] = Subgraph_ColorForPotentialSwingHigh.PrimaryColor;

    // erase all previous potential swing highs
    for (int BarIndex = sc.Index - 1; BarIndex >= MiddleIndex && BarIndex >= 0; BarIndex--)
    {

```

```

        if (Subgraph_Former.Arrays[0][BarIndex] == 1)
        {
            Subgraph_Former[BarIndex] = 0;
            Subgraph_Former.Arrays[0][BarIndex] = 0;
        }
    }
}
else if (IsSwingLowAllowEqualBackwardsOnly(sc, Input_AllowEqualBars.GetYesNo(), sc.Index, Input_Length.GetInt()))
{
    Subgraph_Former.Arrays[0][sc.Index] = -1;
    Subgraph_Former.DataColor[sc.Index] = Subgraph_ColorForPotentialSwingLow.PrimaryColor;

    // erase all previous potential swing lows
    for (int BarIndex = sc.Index - 1; BarIndex >= MiddleIndex && BarIndex >= 0; BarIndex--)
    {
        if (Subgraph_Former.Arrays[0][BarIndex] == -1)
        {
            Subgraph_Former[BarIndex] = 0;
            Subgraph_Former.Arrays[0][BarIndex] = 0;
        }
    }
}
else
{
    Subgraph_Former.Arrays[0][sc.Index] = 0;
    Subgraph_Former.DataColor[sc.Index] = Subgraph_ColorForInsideBar.PrimaryColor;
}

// erase all previous inside-bars in any case
for (int BarIndex = sc.Index - 1; BarIndex >= MiddleIndex && BarIndex >= 0; BarIndex--)
{
    if (Subgraph_Former.Arrays[0][BarIndex] == 0)
    {
        Subgraph_Former[BarIndex] = 0;
    }
}

if (sc.GetBarHasClosedStatus(sc.Index) != BHCS_BAR_HAS_CLOSED)
    return;

if (sc.Index < 2 * Input_Length.GetInt())
{
    Subgraph_SwingHigh[sc.Index] = 0;
    SwingLow[sc.Index] = 0;

    return;
}

if (IsSwingHighAllowEqual_S(sc, Input_AllowEqualBars.GetYesNo(), MiddleIndex, Input_Length.GetInt()))
    Subgraph_SwingHigh[MiddleIndex] = sc.High[MiddleIndex] + Input_ArrowOffsetValue.GetFloat();

if (IsSwingLowAllowEqual_S(sc, Input_AllowEqualBars.GetYesNo(), MiddleIndex, Input_Length.GetInt()))
    SwingLow[MiddleIndex] = sc.Low[MiddleIndex] - Input_ArrowOffsetValue.GetFloat();

// Former needs to be set to 0, as we have calculated real Swing High/Low
Subgraph_Former[MiddleIndex] = 0;

sc.EarliestUpdateSubgraphDataArrayIndex = MiddleIndex;
}

/*=====*/
SCSFExport scsf_StochasticMomentumIndicator(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_SMI = sc.Subgraph[0];
    SCSubgraphRef Subgraph_AvgSMI = sc.Subgraph[1];

```

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

```
SCInputRef Input_OverboughtInput = sc.Input[0];
SCInputRef Input_OversoldInput = sc.Input[1];
SCInputRef Input_PercentDLength = sc.Input[2];
SCInputRef Input_PercentKLength = sc.Input[3];
SCInputRef Input_ExpMovAvgLength = sc.Input[4];
```

```
if (sc.SetDefaults)
{
    sc.GraphName = "Stochastic Momentum Indicator";
    sc.StudyDescription = "Stochastic Momentum Indicator";

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

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

    Subgraph_AvgSMI.Name = "SMI Average";
    Subgraph_AvgSMI.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_AvgSMI.PrimaryColor = RGB(255,0,255);
    Subgraph_AvgSMI.DrawZeros = false;

    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_OverboughtInput.Name = "Overbought Value";
    Input_OverboughtInput.SetFloat(40.0f);

    Input_OversoldInput.Name = "Oversold Value";
    Input_OversoldInput.SetFloat(-40.0f);

    Input_PercentDLength.Name = "%D Length";
    Input_PercentDLength.SetInt(3);

    Input_PercentKLength.Name = "%K Length";
    Input_PercentKLength.SetInt(5);

    Input_ExpMovAvgLength.Name = "EMA Length";
    Input_ExpMovAvgLength.SetInt(3);

    return;
}
```

```
sc.DataStartIndex = max(Input_PercentKLength.GetInt(), Input_PercentDLength.GetInt());
```

```
if (sc.Index < sc.DataStartIndex)
    return;
```

```
SCFloatArrayRef TempLowest = Subgraph_SMI.Arrays[0];
SCFloatArrayRef TempHighest = Subgraph_SMI.Arrays[1];
SCFloatArrayRef TempRelDiff = Subgraph_SMI.Arrays[2];
```

```

SCFloatArrayRef TempDiff = Subgraph_SMI.Arrays[3];
SCFloatArrayRef TempAvgRelMA = Subgraph_SMI.Arrays[4];
SCFloatArrayRef TempAvgRel = Subgraph_SMI.Arrays[5];
SCFloatArrayRef TempAvgDiffMA = Subgraph_SMI.Arrays[6];
SCFloatArrayRef TempAvgDiff = Subgraph_SMI.Arrays[7];

sc.Lowest(sc.Low, TempLowest, Input_PercentKLength.GetInt());
sc.Highest(sc.High, TempHighest, Input_PercentKLength.GetInt());

TempRelDiff[sc.Index] = sc.Close[sc.Index] - (TempHighest[sc.Index] + TempLowest[sc.Index]) / 2.0f;
TempDiff[sc.Index] = TempHighest[sc.Index] - TempLowest[sc.Index];

sc.ExponentialMovAvg(TempRelDiff, TempAvgRelMA, Input_PercentDLength.GetInt());
sc.ExponentialMovAvg(TempAvgRelMA, TempAvgRel, Input_PercentDLength.GetInt());
sc.ExponentialMovAvg(TempDiff, TempAvgDiffMA, Input_PercentDLength.GetInt());
sc.ExponentialMovAvg(TempAvgDiffMA, TempAvgDiff, Input_PercentDLength.GetInt());

if (TempAvgDiff[sc.Index] != 0.0f)
    Subgraph_SMI[sc.Index] = TempAvgRel[sc.Index] / (TempAvgDiff[sc.Index] / 2.0f) * 100.0f;
else
    Subgraph_SMI[sc.Index] = 0.0f;

sc.ExponentialMovAvg(Subgraph_SMI, Subgraph_AvgSMI, Input_ExpMovAvgLength.GetInt());

Subgraph_Overbought[sc.Index] = Input_OverboughtInput.GetFloat();
Subgraph_Oversold[sc.Index] = Input_OversoldInput.GetFloat();
}

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

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

    if (sc.SetDefaults)
    {
        sc.GraphName = "Moving Average - Double Exponential";

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

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

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

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

        return;
    }

    sc.DataStartIndex = Input_Length.GetInt();
    if (sc.Index < sc.DataStartIndex)
        return;

    SCFloatArrayRef EmaPrice = Subgraph_DEMA.Arrays[0];
    SCFloatArrayRef EmaEmaPrice = Subgraph_DEMA.Arrays[1];

```

```

sc.ExponentialMovAvg(sc.BaseDataIn[Input_Data.GetInputDataIndex()], EmaPrice, Input_Length.GetInt());
sc.ExponentialMovAvg(EmaPrice, EmaEmaPrice, Input_Length.GetInt());

Subgraph_DEMA[sc.Index] = 2 * EmaPrice[sc.Index] - EmaEmaPrice[sc.Index];
}

/* ===== */
SCSFExport scsf_BidVolumeVersusAskVolume(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_AskVolume = sc.Subgraph[0];
    SCSubgraphRef Subgraph_BidVolume = sc.Subgraph[1];
    SCSubgraphRef Subgraph_ZeroLine = sc.Subgraph[3];
    SCSubgraphRef Subgraph_Difference = sc.Subgraph[4];

    SCInputRef Input_InvertStudy = sc.Input[3];
    SCInputRef Input_ShowBidVolumePositive = sc.Input[4];
    SCInputRef Input_TryUseNonzeroVolumes = sc.Input[5];

    if (sc.SetDefaults)
    {
        sc.GraphName = "BidVolume vs. AskVolume";
        sc.ValueFormat= 0;

        Subgraph_AskVolume.Name = "Ask Volume";
        Subgraph_AskVolume.PrimaryColor = RGB(0, 255, 0);
        Subgraph_AskVolume.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_AskVolume.DrawZeros = true;

        Subgraph_BidVolume.Name = "Bid Volume";
        Subgraph_BidVolume.PrimaryColor = RGB(255, 0, 0);
        Subgraph_BidVolume.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_BidVolume.DrawZeros = true;

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

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

        Input_InvertStudy.Name = "Invert Study";
        Input_InvertStudy.SetYesNo(0);

        Input_ShowBidVolumePositive.Name = "Show Bid Volume As Positive";
        Input_ShowBidVolumePositive.SetYesNo(0);

        Input_TryUseNonzeroVolumes.Name = "Try Use Non-Zero Bid/Ask Volumes for Difference";
        Input_TryUseNonzeroVolumes.SetYesNo(0);

        return;
    }

    int Invert = 1;
    if(Input_InvertStudy.GetYesNo() == 1)
        Invert = -1;

    // Check for the appropriate chart type
    if (sc.ChartDataType != INTRADAY_DATA)
    {
        if (sc.Index == 0)

```

```

    {
        sc.AddMessageToLog("This study can only be applied to an Intraday Chart", 1);
    }
    return;
}

int StartIndex = sc.UpdateStartIndex;

// Loop through the new indexes to fill in
for (int DestIndex = StartIndex; DestIndex < sc.ArraySize; ++DestIndex)
{
    float BidVolumeVal = sc.BidVolume[DestIndex];
    float AskVolumeVal = sc.AskVolume[DestIndex];

    Subgraph_AskVolume[DestIndex] = AskVolumeVal * Invert;

    if (Input_ShowBidVolumePositive.GetYesNo() == 1)
        Subgraph_BidVolume[DestIndex] = BidVolumeVal * Invert;
    else
        Subgraph_BidVolume[DestIndex] = - BidVolumeVal * Invert;

    Subgraph_ZeroLine[DestIndex] = 0;

    if(Input_TryUseNonzeroVolumes.GetYesNo() == 1)
    {
        // if both volume are zero skip this iteration
        if(BidVolumeVal == 0 && AskVolumeVal == 0)
            continue;

        // if Bid Volume is zero try to find nonzero volume in previous elements
        if(BidVolumeVal == 0)
        {
            int Index = DestIndex - 1;
            while(Index >= 0 && BidVolumeVal == 0)
            {
                BidVolumeVal = sc.BidVolume[Index];
                Index--;
            }
        }

        // if Ask Volume is zero try to find nonzero volume in previous elements
        if(AskVolumeVal == 0)
        {
            int Index = DestIndex - 1;
            while(Index >= 0 && AskVolumeVal == 0)
            {
                AskVolumeVal = sc.AskVolume[Index];
                Index--;
            }
        }
    }
    Subgraph_Difference[DestIndex] = AskVolumeVal - BidVolumeVal;
}

}

/*=====
Test function for HTTP request.
-----*/
SCSFExport scsf_HTTPTest(SCStudyInterfaceRef sc)
{
    if (sc.SetDefaults)
    {

```



```

// Set the configuration and defaults

sc.GraphName = "HTTP Request Test";

sc.StudyDescription = "This is a study function that demonstrates how to request data from a web server.";

sc.AutoLoop = 1;

return;
}

int& RequestState = sc.GetPersistentInt(1);

// Do data processing

if (sc.Index == 0)
{
    if (RequestState == HTTP_REQUEST_ERROR || RequestState == HTTP_REQUEST_RECEIVED)
    {
        // Make a request for a text file on the server. When the request is complete and all of the data has been
        // downloaded, this study function will be called with the file placed into the sc.HTTPResponse character string array.
        if (!sc.MakeHTTPRequest("http://www.sierrachart.com/ACSLHTTPTest.txt"))
            sc.AddMessageToLog("Error making HTTP request.", 1);

        RequestState = HTTP_REQUEST_MADE;
    }
}

if (RequestState == HTTP_REQUEST_MADE && sc.HTTPRequestID != 0)
{
    RequestState = HTTP_REQUEST_RECEIVED;

    // Display the response from the Web server in the Message Log
    sc.AddMessageToLog(sc.HTTPResponse, 1);
}
}

/*=====
Test function for HTTP request with POST.
-----*/
SCSFExport scsf_HTTPTestWithPOST(SCStudyInterfaceRef sc)
{
    if (sc.SetDefaults)
    {
        // Set the configuration and defaults

        sc.GraphName = "HTTP Request Test with POST";

        sc.StudyDescription = "This is a study function that demonstrates how to request data from a web server.";

        sc.AutoLoop = 0;
        return;
    }

    int& RequestState = sc.GetPersistentInt(1);

    // Do data processing

    if (sc.UpdateStartIndex == 0 && sc.IsFullRecalculation)
    {
        if (RequestState == HTTP_REQUEST_ERROR || RequestState == HTTP_REQUEST_RECEIVED)

```

```

{
    n_ACSIL::s_HTTPHeader HTTPHeader;
    HTTPHeader.Name = "Custom";
    HTTPHeader.Value = "Value";

    // Make a request to the server.
    // When the request is complete and all of the data has been downloaded,
    // this study function will be called with the file placed into the
    // sc.HTTPResponse character string array.
    if (!sc.MakeHTTPPOSTRequest("https://www.sierrachart.com/Test/ACSILPOSTTest.php", "Message=PostData",
&HTTPHeader, 1))
    {
        sc.AddMessageToLog("Error making HTTP request.", 1);
    }

    RequestState = HTTP_REQUEST_MADE;
}
}

if (RequestState == HTTP_REQUEST_MADE && sc.HTTPRequestID != 0)
{
    RequestState = HTTP_REQUEST_RECEIVED;

    // Display the response from the Web server in the Message Log
    sc.AddMessageToLog(sc.HTTPResponse, 1);
}
}

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

    SCInputRef Input_StartTime = sc.Input[0];
    SCInputRef Input_EndTime = sc.Input[1];
    SCInputRef Input_Version = sc.Input[2];

    SCInputRef ForwardProject = sc.Input[4];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Time Range Highlight";
        sc.GraphRegion = 0;
        sc.AutoLoop = 0;
        sc.ValueFormat = 0;
        sc.ScaleRangeType = SCALE_INDEPENDENT;

        Subgraph_ColorBackground.Name = "TR";
        Subgraph_ColorBackground.DrawStyle = DRAWSTYLE_BACKGROUND;
        Subgraph_ColorBackground.PrimaryColor = RGB(64, 64, 64);
        Subgraph_ColorBackground.DrawZeros = false;

        Input_StartTime.Name = "Start Time";
        Input_StartTime.SetTime(0);

        Input_EndTime.Name = "End Time";
        Input_EndTime.SetTime(SECONDS_PER_DAY - 1);

        Input_Version.SetInt(1);

        ForwardProject.Name = "Display in Forward Projection Area";
        ForwardProject.SetYesNo(false);
    }
}

```

```

    return;
}

if(Input_Version.GetInt() < 1)
{
    Input_Version.SetInt(1);
    Subgraph_ColorBackground.DrawStyle = DRAWSTYLE_BACKGROUND;
}

int NumberForwardBars = 0;
if(ForwardProject.GetYesNo())
{
    NumberForwardBars = 200;
    Subgraph_ColorBackground.ExtendedArrayElementsToGraph = NumberForwardBars;
}

for (int Index = sc.UpdateStartIndex; Index < (sc.ArraySize + NumberForwardBars); Index++)
{
    bool InputsReversed = (Input_StartTime.GetTime() > Input_EndTime.GetTime());

    SCDateTime& IndexDateTime = sc.BaseDateTimeIn[Index];
    bool ShouldHighLight = false;

    if (InputsReversed)
        ShouldHighLight = (IndexDateTime.GetTimeInSeconds() >= Input_StartTime.GetTime() ||
IndexDateTime.GetTimeInSeconds() <= Input_EndTime.GetTime());
    else
        ShouldHighLight = (IndexDateTime.GetTimeInSeconds() >= Input_StartTime.GetTime() &&
IndexDateTime.GetTimeInSeconds() <= Input_EndTime.GetTime());

    int CloseIndex = Index;
    if(CloseIndex >= sc.ArraySize)
        CloseIndex = sc.ArraySize - 1;

    if (ShouldHighLight)
        Subgraph_ColorBackground[Index] = sc.Close[CloseIndex];
    else
        Subgraph_ColorBackground[Index] = 0;
}

}
/*=====*/
SCSFExport scsf_TimeRangeHighlightTransparent(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_TimeRange = sc.Subgraph[0];

    SCInputRef Input_StartTime = sc.Input[0];
    SCInputRef Input_EndTime = sc.Input[1];

    SCInputRef ForwardProject = sc.Input[4];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Time Range Highlight - Transparent";
        sc.GraphRegion = 0;
        sc.AutoLoop = 0;//Manual looping
        sc.ValueFormat = 0;
        sc.ScaleRangeType = SCALE_SAMEASREGION;
        sc.DrawStudyUnderneathMainPriceGraph = true;

        Subgraph_TimeRange.Name = "TR";
    }
}

```

```

Subgraph_TimeRange.DrawStyle = DRAWSTYLE_BACKGROUND_TRANSPARENT;
Subgraph_TimeRange.PrimaryColor = RGB(64, 64, 64);
Subgraph_TimeRange.DrawZeros = false;

Input_StartTime.Name = "Start Time";
Input_StartTime.SetTime(0);

Input_EndTime.Name = "End Time";
Input_EndTime.SetTime(SECONDS_PER_DAY - 1);

ForwardProject.Name = "Display in Forward Projection Area";
ForwardProject.SetYesNo(false);

return;
}

bool InputsReversed = (Input_StartTime.GetTime() > Input_EndTime.GetTime());

int NumberForwardBars = 0;
if(ForwardProject.GetYesNo())
{
    NumberForwardBars = 200;
    Subgraph_TimeRange.ExtendedArrayElementsToGraph = NumberForwardBars;
}

for (int Index = sc.UpdateStartIndex; Index < (sc.ArraySize + NumberForwardBars); Index++)
{
    SCDateTime& IndexDateTime = sc.BaseDateTimeln[Index];
    bool ShouldHighLight = false;

    if (InputsReversed)
        ShouldHighLight = (IndexDateTime.GetTimeInSeconds() >= Input_StartTime.GetTime() ||
IndexDateTime.GetTimeInSeconds() <= Input_EndTime.GetTime());
    else
        ShouldHighLight = (IndexDateTime.GetTimeInSeconds() >= Input_StartTime.GetTime() &&
IndexDateTime.GetTimeInSeconds() <= Input_EndTime.GetTime());

    if (ShouldHighLight)
    {
        Subgraph_TimeRange[Index] = 1;
    }
    else
    {
        Subgraph_TimeRange[Index] = 0;
    }
}
}

/*****
SCSFExport scsf_ColoredCCI(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_CCI = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Line1 = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Line2 = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Line3 = sc.Subgraph[3];

    SCInputRef Input_Data = sc.Input[0];
    SCInputRef Input_Length = sc.Input[3];
    SCInputRef Input_Multiplier = sc.Input[4];
    SCInputRef Input_Line2Value = sc.Input[5];

```

```

SCInputRef Input_Line3Value = sc.Input[6];

if (sc.SetDefaults)
{
    // Set the configuration and defaults
    sc.GraphName = "Colored CCI";

    sc.AutoLoop = 1; // true
    sc.GraphRegion = 1;
    sc.ValueFormat = 2;

    Subgraph_CCI.Name = "CCI";
    Subgraph_CCI.DrawStyle = DRAWSTYLE_BAR;
    Subgraph_CCI.DrawZeros= true;
    Subgraph_CCI.PrimaryColor = RGB(0, 255, 255);
    Subgraph_CCI.SecondaryColorUsed = 1;
    Subgraph_CCI.SecondaryColor= RGB(0, 255, 0);

    Subgraph_Line1.Name = "Line 1";
    Subgraph_Line1.DrawStyle = DRAWSTYLE_HIDDEN;
    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;

    Subgraph_Line3.Name = "Line 3";
    Subgraph_Line3.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_Line3.PrimaryColor = RGB(255,127,0);
    Subgraph_Line3.DrawZeros = true;

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

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

    Input_Multiplier.Name = "Multiplier";
    Input_Multiplier.SetFloat(0.015f);

    Input_Line2Value.Name = "Line2 Value";
    Input_Line2Value.SetFloat(100);

    Input_Line3Value.Name = "Line3 Value";
    Input_Line3Value.SetFloat(-100);

    return;
}

float CCILow = -50.0f, CCHigh = 50.0f;

sc.DataStartIndex = Input_Length.GetInt();

float Multiplier = Input_Multiplier.GetFloat();

if(Multiplier == 0.0f)
    Multiplier = 0.015f;

```

```

sc.CCI(sc.BaseDataIn[Input_Data.GetInputDataIndex()], Subgraph_CCI, sc.Index, Input_Length.GetInt(), Multiplier);

Subgraph_Line2[sc.Index] = Input_Line2Value.GetFloat();
Subgraph_Line3[sc.Index] = Input_Line3Value.GetFloat();

if (Subgraph_CCI[sc.Index - 1] < CCLow && Subgraph_CCI[sc.Index] > CCHigh)
    Subgraph_CCI.DataColor[sc.Index] = Subgraph_CCI.SecondaryColor;
else
    Subgraph_CCI.DataColor[sc.Index] = Subgraph_CCI.PrimaryColor;
}

/*****
SCSFExport scsf_ColorBarsBasedOnMA(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Avg = sc.Subgraph[0];
    SCSubgraphRef Subgraph_ColorSMA = sc.Subgraph[1];
    SCSubgraphRef Subgraph_ArrowSMA = sc.Subgraph[2];

    SCFloatArrayRef Array_Slope = Subgraph_Avg.Arrays[0];

    SCInputRef Input_Data = sc.Input[0];
    SCInputRef Input_Length = sc.Input[3];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Color Bars Based on Moving Average";

        sc.GraphRegion = 0;
        sc.ValueFormat = 2;

        sc.AutoLoop = 1;

        Subgraph_Avg.Name = "Avg";

        Subgraph_ColorSMA.Name = "Color SMA";
        Subgraph_ColorSMA.DrawStyle = DRAWSTYLE_COLOR_BAR;
        Subgraph_ColorSMA.PrimaryColor = RGB(0, 255, 0);
        Subgraph_ColorSMA.DrawZeros = false;

        Subgraph_ArrowSMA.Name = "Arrow SMA";
        Subgraph_ArrowSMA.DrawStyle = DRAWSTYLE_ARROW_UP;
        Subgraph_ArrowSMA.PrimaryColor = RGB(0, 255, 0);
        Subgraph_ArrowSMA.LineWidth = 3;
        Subgraph_ArrowSMA.DrawZeros = false;

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

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

        return;
    }

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

    sc.SimpleMovAvg(sc.BaseDataIn[Input_Data.GetInputDataIndex()], Subgraph_Avg, Input_Length.GetInt());
    sc.Slope(Subgraph_Avg, Array_Slope);

    if (sc.BaseDataIn[Input_Data.GetInputDataIndex()][sc.Index] > Subgraph_Avg[sc.Index] && Array_Slope[sc.Index] > 0)
    {
        Subgraph_ColorSMA[sc.Index] = 1;
        Subgraph_ArrowSMA[sc.Index] = sc.Low[sc.Index];
    }
}

```

```

else
{
    Subgraph_ColorSMA[sc.Index] = 0;
    Subgraph_ArrowSMA[sc.Index] = 0;
}
}

/*=====*/
//The best way to examine the operation of this study is to perform a chart replay.
SCSFExport scsf_SetAlertExample(SCStudyInterfaceRef sc)
{
    if (sc.SetDefaults)
    {
        // Set the configuration and defaults

        sc.GraphName = "Set Alert Example";

        sc.StudyDescription = "";

        sc.AutoLoop = true;

        return;
    }

    // Do data processing
    if (sc.GetBarHasClosedStatus(sc.Index) == BHCS_BAR_HAS_NOT_CLOSED)
        return;

    int Index = sc.Index;

    SCString AlertMessageText;
    AlertMessageText.Format("Alert at index %d", Index);

    //To configure the sound file for Alert 5 and other settings, select 'Global Settings >> General Settings >> Alerts' within
    the user interface.
    sc.SetAlert(5, AlertMessageText);
    sc.SetAlert(6, AlertMessageText);
}

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

    SCFloatArrayRef Array_HH = sc.Subgraph[0].Arrays[0];
    SCFloatArrayRef Array_LL = sc.Subgraph[0].Arrays[1];
    SCFloatArrayRef Array_LH = sc.Subgraph[0].Arrays[2];
    SCFloatArrayRef Array_HL = sc.Subgraph[0].Arrays[3];
    SCFloatArrayRef Array_UpSwing = sc.Subgraph[0].Arrays[4];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Clear Method Swing Line";

        sc.AutoLoop = true;

        sc.GraphRegion = 0;

        Subgraph_SwingLine.Name = "SwingLine";
        Subgraph_SwingLine.PrimaryColor = COLOR_CYAN;
        Subgraph_SwingLine.SecondaryColor = COLOR_MAGENTA;
    }
}

```

```

Subgraph_SwingLine.SecondaryColorUsed = 1;

return;
}

// Do data processing

//Carry forward prior values in the background data arrays
Array_HH[sc.Index] = Array_HH[sc.Index-1];
Array_HL[sc.Index] = Array_HL[sc.Index-1];
Array_LL[sc.Index] = Array_LL[sc.Index-1];
Array_LH[sc.Index] = Array_LH[sc.Index-1];
Array_UpSwing[sc.Index] = Array_UpSwing[sc.Index-1];

if (Array_UpSwing[sc.Index] == 1)
{
    if (sc.High[sc.Index] > Array_HH[sc.Index])
        Array_HH[sc.Index] = sc.High[sc.Index];

    if (sc.Low[sc.Index] > Array_HL[sc.Index])
        Array_HL[sc.Index] = sc.Low[sc.Index];

    if (sc.High[sc.Index] < Array_HL[sc.Index])
    {
        Array_UpSwing[sc.Index] = 0;
        Array_LL[sc.Index] = sc.Low[sc.Index];
        Array_LH[sc.Index] = sc.High[sc.Index];
    }
}

if (Array_UpSwing[sc.Index] == 0)
{
    if (sc.Low[sc.Index] < Array_LL[sc.Index])
        Array_LL[sc.Index] = sc.Low[sc.Index];

    if (sc.High[sc.Index] < Array_LH[sc.Index])
        Array_LH[sc.Index] = sc.High[sc.Index];

    if (sc.Low[sc.Index] > Array_LH[sc.Index])
    {
        Array_UpSwing[sc.Index] = 1;
        Array_HH[sc.Index] = sc.High[sc.Index];
        Array_HL[sc.Index] = sc.Low[sc.Index];
    }
}

if (Array_UpSwing[sc.Index] == 1)
{
    Subgraph_SwingLine[sc.Index] = Array_HL[sc.Index];
    Subgraph_SwingLine.DataColor[sc.Index] = Subgraph_SwingLine.PrimaryColor;
}
else
{
    Subgraph_SwingLine[sc.Index] = Array_LH[sc.Index];
    Subgraph_SwingLine.DataColor[sc.Index] = Subgraph_SwingLine.SecondaryColor;
}
}

/*=====*/

```



```

SCSFExport scsf_StudySubgraphsAdd(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Sybgraph_Add = sc.Subgraph[0];

    SCInputRef Input_StudySubgraph1 = sc.Input[0];
    SCInputRef Input_StudySubgraph2 = sc.Input[1];
    SCInputRef Input_Study2SubgraphOffset = sc.Input[2];
    SCInputRef Input_DrawZeros = sc.Input[3];
    SCInputRef Input_PerformAddWithZeroValue = sc.Input[4];
    SCInputRef Input_Version = sc.Input[5];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Study Subgraphs Add";

        sc.AutoLoop = 0;
        sc.CalculationPrecedence = LOW_PREC_LEVEL;

        Sybgraph_Add.Name = "Add";
        Sybgraph_Add.DrawStyle = DRAWSTYLE_LINE;
        Sybgraph_Add.LineWidth = 1;
        Sybgraph_Add.PrimaryColor = RGB(0,255,0);

        Input_StudySubgraph1.Name = "Input Study 1";
        Input_StudySubgraph1.SetStudySubgraphValues(0, 0);

        Input_StudySubgraph2.Name = "Input Study 2";
        Input_StudySubgraph2.SetStudySubgraphValues(0, 0);

        Input_Study2SubgraphOffset.Name = "Study 2 Subgraph Offset";
        Input_Study2SubgraphOffset.SetInt(0);

        Input_DrawZeros.Name = "Draw Zeros";
        Input_DrawZeros.SetYesNo(false);

        Input_PerformAddWithZeroValue.Name = "Perform Add With Zero Value";
        Input_PerformAddWithZeroValue.SetYesNo(true);

        Input_Version.SetInt(2);

        return;
    }

    if (Input_Version.GetInt() < 2)
    {
        Input_PerformAddWithZeroValue.SetYesNo(true);
        Input_Version.SetInt(2);
    }

    Sybgraph_Add.DrawZeros = Input_DrawZeros.GetYesNo();

    SCFloatArray Study1Array;

    sc.GetStudyArrayUsingID(Input_StudySubgraph1.GetStudyID(), Input_StudySubgraph1.GetSubgraphIndex(), Study1Array);

    SCFloatArray Study2Array;

    sc.GetStudyArrayUsingID(Input_StudySubgraph2.GetStudyID(), Input_StudySubgraph2.GetSubgraphIndex(), Study2Array);

    int CalculationStartIndex = sc.GetCalculationStartIndexForStudy();
    sc.EarliestUpdateSubgraphDataArrayIndex = CalculationStartIndex;

    for (int Index = CalculationStartIndex; Index < sc.ArraySize; Index++)

```

```

{

    float Value1 = Study1Array[Index];
    float Value2 = Study2Array[Index - Input_Study2SubgraphOffset.GetInt()];

    if (Input_PerformAddWithZeroValue.GetYesNo()
        || (Value1 != 0.0 && Value2 != 0.0))
    {
        Sybgraph_Add[Index] = Value1 + Value2;
    }
    else
    {
        Sybgraph_Add[Index] = 0;
    }
}

}

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

    SCInputRef Input_StudySubgraph1 = sc.Input[0];
    SCInputRef Input_StudySubgraph2 = sc.Input[1];
    SCInputRef Input_Study2SubgraphOffset = sc.Input[2];

    SCInputRef Input_DrawZeros = sc.Input[3];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Study Subgraphs Multiply";
        sc.AutoLoop = 0;
        sc.CalculationPrecedence = LOW_PREC_LEVEL;

        Subgraph_Multiply.Name = "Multiply";
        Subgraph_Multiply.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Multiply.PrimaryColor = RGB(0,255,0);
        Subgraph_Multiply.LineWidth = 1;

        Input_StudySubgraph1.Name = "Input Study 1";
        Input_StudySubgraph1.SetStudySubgraphValues(0, 0);

        Input_StudySubgraph2.Name = "Input Study 2";
        Input_StudySubgraph2.SetStudySubgraphValues(0, 0);

        Input_Study2SubgraphOffset.Name = "Study 2 Subgraph Offset";
        Input_Study2SubgraphOffset.SetInt(0);

        Input_DrawZeros.Name = "Draw Zeros";
        Input_DrawZeros.SetYesNo(false);

        return;
    }

    Subgraph_Multiply.DrawZeros= Input_DrawZeros.GetYesNo();

    SCFloatArray Study1Array;

    sc.GetStudyArrayUsingID(Input_StudySubgraph1.GetStudyID(),Input_StudySubgraph1.GetSubgraphIndex(),Study1Array);

```

```

SCFloatArray Study2Array;

sc.GetStudyArrayUsingID(Input_StudySubgraph2.GetStudyID(),Input_StudySubgraph2.GetSubgraphIndex(),Study2Array);

int CalculationStartIndex = sc.GetCalculationStartIndexForStudy();

for (int Index = CalculationStartIndex; Index < sc.ArraySize; Index++)
{
    Subgraph_Multiply[Index] = Study1Array[Index] * Study2Array[Index - Input_Study2SubgraphOffset.GetInt()];
}

sc.EarliestUpdateSubgraphDataArrayIndex = CalculationStartIndex;
}

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

    SCInputRef Input_StudySubgraph1 = sc.Input[0];
    SCInputRef Input_StudySubgraph2 = sc.Input[1];
    SCInputRef Input_Study2SubgraphOffset = sc.Input[2];
    SCInputRef Input_DrawZeros = sc.Input[3];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Study Subgraphs Divide/Ratio";

        sc.AutoLoop = 0;
        sc.CalculationPrecedence = LOW_PREC_LEVEL;

        Subgraph_Ratio.Name = "Ratio";
        Subgraph_Ratio.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Ratio.PrimaryColor = RGB(0,255,0);
        Subgraph_Ratio.LineWidth = 1;

        Input_StudySubgraph1.Name = "Input Study 1";
        Input_StudySubgraph1.SetStudySubgraphValues(0, 0);

        Input_StudySubgraph2.Name = "Input Study 2";
        Input_StudySubgraph2.SetStudySubgraphValues(0, 0);

        Input_Study2SubgraphOffset.Name = "Study 2 Subgraph Offset";
        Input_Study2SubgraphOffset.SetInt(0);

        Input_DrawZeros.Name = "Draw Zeros";
        Input_DrawZeros.SetYesNo(false);

        return;
    }

    Subgraph_Ratio.DrawZeros= Input_DrawZeros.GetYesNo();

    SCFloatArray Study1Array;

    sc.GetStudyArrayUsingID(Input_StudySubgraph1.GetStudyID(),Input_StudySubgraph1.GetSubgraphIndex(),Study1Array);

```

```

SCFloatArray Study2Array;

sc.GetStudyArrayUsingID(Input_StudySubgraph2.GetStudyID(),Input_StudySubgraph2.GetSubgraphIndex(),Study2Array);

int CalculationStartIndex = sc.GetCalculationStartIndexForStudy();

for (int Index = CalculationStartIndex; Index < sc.ArraySize; Index++)
{
    float Denominator = Study2Array[Index - Input_Study2SubgraphOffset.GetInt()];
    if (Denominator != 0.0f)
        Subgraph_Ratio[Index] = Study1Array[Index] / Denominator;
    else
        Subgraph_Ratio[Index] = 0.0f;
}

sc.EarliestUpdateSubgraphDataArrayIndex = CalculationStartIndex;
}

/*=====*/
SCSFExport scsf_Inertia(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Inertia = sc.Subgraph[0];
    SCSubgraphRef Subgraph_RVI = sc.Subgraph[1];

    SCInputRef Input_RVILength = sc.Input[0];
    SCInputRef Input_LinearRegressionLength = sc.Input[1];

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

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

        Subgraph_Inertia.Name = "Inertia";
        Subgraph_Inertia.PrimaryColor = COLOR_GREEN;
        Subgraph_Inertia.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Inertia.PrimaryColor = RGB(0,255,0);
        Subgraph_Inertia.DrawZeros = true;
        Subgraph_Inertia.LineWidth = 1;

        Input_RVILength.Name = "Relative Vigor Index Length";
        Input_RVILength.SetInt(10);

        Input_LinearRegressionLength.Name = "Linear Regression Length";
        Input_LinearRegressionLength.SetInt(10);

        return;
    }

    sc.DataStartIndex = max(Input_RVILength.GetInt(), Input_LinearRegressionLength.GetInt());

    // Calculate RVI
    SCFloatArrayRef Value1 = Subgraph_RVI.Arrays[0];
    SCFloatArrayRef Value2 = Subgraph_RVI.Arrays[1];
    SCFloatArrayRef RVINumerator = Subgraph_RVI.Arrays[2];
    SCFloatArrayRef RVIDenominator = Subgraph_RVI.Arrays[3];

    Value1[sc.Index] = ((sc.Close[sc.Index] - sc.Open[sc.Index])
        + 2 * (sc.Close[sc.Index - 1] - sc.Open[sc.Index - 1])

```

```

+ 2 * (sc.Close[sc.Index - 2] - sc.Open[sc.Index - 2])
+ (sc.Close[sc.Index - 3] - sc.Open[sc.Index - 3])) / 6;

Value2[sc.Index] = ((sc.High[sc.Index] - sc.Low[sc.Index])
+ 2 * (sc.High[sc.Index - 1] - sc.Low[sc.Index - 1])
+ 2 * (sc.High[sc.Index - 2] - sc.Low[sc.Index - 2])
+ (sc.High[sc.Index - 3] - sc.Low[sc.Index - 3])) / 6;

if (sc.Index < sc.DataStartIndex)
    return;

sc.Summation(Value1, RVINumerator , Input_RVILength.GetInt());
sc.Summation(Value2, RVIDenominator, Input_RVILength.GetInt());

if (RVIDenominator[sc.Index] != 0.0f)
    Subgraph_RVI[sc.Index] = RVINumerator [sc.Index] / RVIDenominator[sc.Index];
else
    Subgraph_RVI[sc.Index] = 0.0f;

// Calculate Inertia
sc.LinearRegressionIndicator(Subgraph_RVI, Subgraph_Inertia, Input_LinearRegressionLength.GetInt());
}

/*=====*/
SCSFExport scsf_Inertia2(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Inertia = sc.Subgraph[0];
    SCSubgraphRef Subgraph_RVI = sc.Subgraph[1];

    SCInputRef Input_StdDeviationLength = sc.Input[0];
    SCInputRef Input_RviLength = sc.Input[1];
    SCInputRef Input_LinearRegressionLength = sc.Input[2];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Inertia 2";

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

        Subgraph_Inertia.Name = "Inertia";
        Subgraph_Inertia.PrimaryColor = COLOR_GREEN;
        Subgraph_Inertia.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Inertia.DrawZeros = true;
        Subgraph_Inertia.LineWidth = 1;

        Input_StdDeviationLength.Name = "Standard Deviation Length";
        Input_StdDeviationLength.SetInt(9);

        Input_RviLength.Name = "Relative Volatility Index Length";
        Input_RviLength.SetInt(14);

        Input_LinearRegressionLength.Name = "Linear Regression Length";
        Input_LinearRegressionLength.SetInt(10);

        return;
    }

    sc.DataStartIndex = max(Input_StdDeviationLength.GetInt(), max(Input_RviLength.GetInt(),
Input_LinearRegressionLength.GetInt()));
    if (sc.Index < sc.DataStartIndex)
        return;

```

```

// Calculate RVI
SCFloatArrayRef RviUp = Subgraph_RVI.Arrays[0];
SCFloatArrayRef RviDown = Subgraph_RVI.Arrays[1];
SCFloatArrayRef RviUpAvg = Subgraph_RVI.Arrays[2];
SCFloatArrayRef RviDownAvg = Subgraph_RVI.Arrays[3];
SCFloatArrayRef StdDeviation = Subgraph_RVI.Arrays[4];

sc.StdDeviation(sc.Close, StdDeviation, Input_StdDeviationLength.GetInt());

if (sc.Close[sc.Index] > sc.Close[sc.Index - 1])
{
    RviUp[sc.Index] = StdDeviation[sc.Index];
    RviDown[sc.Index] = 0.0f;
}
else
{
    RviUp[sc.Index] = 0.0f;
    RviDown[sc.Index] = StdDeviation[sc.Index];
}

//At the first index calculated the prior RVI up average and down average is going to be zero.
if (Input_RviLength.GetInt() != 0)
{
    RviUpAvg[sc.Index] = (RviUpAvg[sc.Index - 1] * (Input_RviLength.GetInt() - 1) + RviUp[sc.Index]) /
Input_RviLength.GetInt();
    RviDownAvg[sc.Index] = (RviDownAvg[sc.Index - 1] * (Input_RviLength.GetInt() - 1) + RviDown[sc.Index]) /
Input_RviLength.GetInt();
}
else
{
    RviUpAvg[sc.Index] = 0.0f;
    RviDownAvg[sc.Index] = 0.0f;
}

if (RviUpAvg[sc.Index] + RviDownAvg[sc.Index] != 0.0f)
    Subgraph_RVI[sc.Index] = (RviUpAvg[sc.Index] / (RviUpAvg[sc.Index] + RviDownAvg[sc.Index])) * 100;
else
    Subgraph_RVI[sc.Index] = 0.0f;

// Calculate Inertia
sc.LinearRegressionIndicator(Subgraph_RVI, Subgraph_Inertia, Input_LinearRegressionLength.GetInt());
}

/*=====*/
SCSFExport scsf_AskBidVolumeDifferenceBars(SCStudyInterfaceRef sc)
{
    // reference the base data
    SCFloatArrayRef InArray_NumTrades = sc.BaseDataIn[SC_NUM_TRADES];
    SCFloatArrayRef InArray_BidVolume = sc.BaseDataIn[SC_BIDVOL];
    SCFloatArrayRef InArray_AskVolume = sc.BaseDataIn[SC_ASKVOL];
    SCFloatArrayRef InArray_AskBidDiffHigh = sc.BaseDataIn[SC_ASKBID_DIFF_HIGH];
    SCFloatArrayRef InArray_AskBidDiffLow = sc.BaseDataIn[SC_ASKBID_DIFF_LOW];

    SCSubgraphRef Subgraph_Open = sc.Subgraph[0];
    SCSubgraphRef Subgraph_High = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Low = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Close = sc.Subgraph[3];

    SCInputRef Input_IncludePullbackColumn = sc.Input[0];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Ask/Bid Volume Difference Bars";

        sc.MaintainAdditionalChartDataArrays = 1;
    }
}

```

```

sc.AutoLoop = 0;
sc.ValueFormat = 0;
sc.GraphDrawType = GDT_CANDLESTICK;

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

Subgraph_High.Name = "High";
Subgraph_High.PrimaryColor = RGB(0,128,0);
Subgraph_High.DrawStyle = DRAWSTYLE_LINE;
Subgraph_High.DrawZeros = true;

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

Subgraph_Close.Name = "Close";
Subgraph_Close.PrimaryColor = RGB(128,0,0);
Subgraph_Close.DrawStyle = DRAWSTYLE_LINE;
Subgraph_Close.DrawZeros = true;

Input_IncludePullbackColumn.Name = "Include Pullback Column";
Input_IncludePullbackColumn.SetYesNo(false);

return;
}

for (int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize; BarIndex++)
{
    float BidAskDifference = InArray_AskVolume[BarIndex] - InArray_BidVolume[BarIndex];
    if (InArray_NumTrades[BarIndex] == 1)
        Subgraph_Open[BarIndex] = BidAskDifference;
    else
        Subgraph_Open[BarIndex] = 0;

    Subgraph_High[BarIndex] = InArray_AskBidDiffHigh[BarIndex];
    Subgraph_Low[BarIndex] = InArray_AskBidDiffLow[BarIndex];
    Subgraph_Close[BarIndex] = BidAskDifference;

    //Move the Open to within the high to low range
    if (Subgraph_Open[BarIndex] > Subgraph_High[BarIndex])
        Subgraph_Open[BarIndex] = Subgraph_High[BarIndex];

    if (Subgraph_Open[BarIndex] < Subgraph_Low[BarIndex])
        Subgraph_Open[BarIndex] = Subgraph_Low[BarIndex];
}

// Pullback column
if (Input_IncludePullbackColumn.GetYesNo() != 0
    && static_cast<int>(sc.PullbackVolumeAtPrice->GetNumberOfBars()) > 0)
{
    if (sc.IsFullRecalculation)
    {
        Subgraph_Open.ExtendedArrayElementsToGraph = 1;
        Subgraph_High.ExtendedArrayElementsToGraph = 1;
        Subgraph_Low.ExtendedArrayElementsToGraph = 1;
        Subgraph_Close.ExtendedArrayElementsToGraph = 1;
    }

    double PullBackAskVolumeBidVolumeDifference = 0;

    const s_VolumeAtPriceV2* p_VolumeAtPrice = NULL;

```

```

const int VAPSizeAtBarIndex = sc.PullbackVolumeAtPrice->GetSizeAtBarIndex(0);
for (int VAPIndex = 0; VAPIndex < VAPSizeAtBarIndex; VAPIndex++)
{
    if (!sc.PullbackVolumeAtPrice->GetVAPElementAtIndex(0, VAPIndex, &p_VolumeAtPrice))
        break;

    PullBackAskVolumeBidVolumeDifference += static_cast<double>(p_VolumeAtPrice->AskVolume) -
p_VolumeAtPrice->BidVolume;

}

Subgraph_Open[sc.ArraySize] = 0;
Subgraph_High[sc.ArraySize] = 0;
Subgraph_Low[sc.ArraySize] = 0;
Subgraph_Close[sc.ArraySize] = static_cast<float>(PullBackAskVolumeBidVolumeDifference);
}
else
{
    if (sc.IsFullRecalculation)
    {
        Subgraph_Open.ExtendedArrayElementsToGraph = 0;
        Subgraph_High.ExtendedArrayElementsToGraph = 0;
        Subgraph_Low.ExtendedArrayElementsToGraph = 0;
        Subgraph_Close.ExtendedArrayElementsToGraph = 0;
    }
}
}

/*=====*/
SCSFExport scsf_UpDownVolumeDifferenceBars(SCStudyInterfaceRef sc)
{
    // reference the base data
    SCFloatArrayRef Array_NumTrades = sc.BaseDataIn[SC_NUM_TRADES];
    SCFloatArrayRef Array_DownVolume = sc.BaseDataIn[SC_DOWNVOL];
    SCFloatArrayRef Array_UpVolume = sc.BaseDataIn[SC_UPVOL];
    SCFloatArrayRef Array_UpDownVolumeDifferenceHigh = sc.BaseDataIn[SC_UPDOWN_VOL_DIFF_HIGH];
    SCFloatArrayRef Array_UpDownVolumeDifferenceLow = sc.BaseDataIn[SC_UPDOWN_VOL_DIFF_LOW];

    SCSubgraphRef Subgraph_Open = sc.Subgraph[0];
    SCSubgraphRef Subgraph_High = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Low = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Close = sc.Subgraph[3];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Up/Down Volume Difference Bars";

        sc.MaintainAdditionalChartDataArrays = 1;

        sc.ValueFormat = 0;
        sc.GraphDrawType = GDT_CANDLESTICK;

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

        Subgraph_High.Name = "High";
        Subgraph_High.PrimaryColor = RGB(0, 128, 0);
        Subgraph_High.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_High.DrawZeros = true;

        Subgraph_Low.Name = "Low";
        Subgraph_Low.PrimaryColor = RGB(255, 0, 0);

```



```

Subgraph_Low.DrawStyle = DRAWSTYLE_LINE;
Subgraph_Low.DrawZeros = true;

Subgraph_Close.Name = "Close";
Subgraph_Close.PrimaryColor = RGB(128, 0, 0);
Subgraph_Close.DrawStyle = DRAWSTYLE_LINE;
Subgraph_Close.DrawZeros = true;

return;
}

for (int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize; BarIndex++)
{
    float UpVolumeDownVolumeDifference = Array_UpVolume[BarIndex] - Array_DownVolume[BarIndex];
    if (Array_NumTrades[BarIndex] == 1)
        Subgraph_Open[BarIndex] = UpVolumeDownVolumeDifference;
    else
        Subgraph_Open[BarIndex] = 0;

    Subgraph_High[BarIndex] = Array_UpDownVolumeDifferenceHigh[BarIndex];
    Subgraph_Low[BarIndex] = Array_UpDownVolumeDifferenceLow[BarIndex];
    Subgraph_Close[BarIndex] = UpVolumeDownVolumeDifference;

    if (Subgraph_Open[BarIndex] > Subgraph_High[BarIndex])
        Subgraph_Open[BarIndex] = Subgraph_High[BarIndex];

    if (Subgraph_Open[BarIndex] < Subgraph_Low[BarIndex])
        Subgraph_Open[BarIndex] = Subgraph_Low[BarIndex];
}
}

/*=====*/
SCSFExport scsf_CumulativeDeltaBarsVolume(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_BarOpens = sc.Subgraph[SC_OPEN];
    SCSubgraphRef Subgraph_BarHighs = sc.Subgraph[SC_HIGH];
    SCSubgraphRef Subgraph_BarLows = sc.Subgraph[SC_LOW];
    SCSubgraphRef Subgraph_BarCloses = sc.Subgraph[SC_LAST];
    SCSubgraphRef Subgraph_BarVolumes = sc.Subgraph[SC_VOLUME];

    SCSubgraphRef Subgraph_OHLCAvg = sc.Subgraph[SC_OHLC_AVG];
    SCSubgraphRef Subgraph_HLCAvg = sc.Subgraph[SC_HLC_AVG];
    SCSubgraphRef Subgraph_HLAvg = sc.Subgraph[SC_HL_AVG];

    SCInputRef Input_PerformRollingCalculation = sc.Input[0];
    SCInputRef Input_RollingCalculationLength = sc.Input[1];
    SCInputRef Input_ResetAtSessionStart = sc.Input[2];
    SCInputRef Input_ResetAtBothSessionStarts = sc.Input[3];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Cumulative Delta Bars - Volume";

        sc.MaintainAdditionalChartDataArrays = 1;
        sc.AutoLoop = 0;

        sc.ValueFormat = 0;
        sc.GraphDrawType = GDT_CANDLESTICK;

        Subgraph_BarOpens.Name = "Open";
        Subgraph_BarOpens.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_BarOpens.PrimaryColor = RGB(0,255,0);
        Subgraph_BarOpens.SecondaryColor = RGB(0,255,0);
        Subgraph_BarOpens.SecondaryColorUsed = true;
    }
}

```

```

Subgraph_BarOpens.DrawZeros = true;

Subgraph_BarHighs.Name = "High";
Subgraph_BarHighs.DrawStyle = DRAWSTYLE_LINE;
Subgraph_BarHighs.PrimaryColor = RGB(0,128,0);
Subgraph_BarHighs.DrawZeros = true;

Subgraph_BarLows.Name = "Low";
Subgraph_BarLows.DrawStyle = DRAWSTYLE_LINE;
Subgraph_BarLows.PrimaryColor = RGB(255,0,0);
Subgraph_BarLows.SecondaryColor = RGB(255,0,0);
Subgraph_BarLows.SecondaryColorUsed = true;
Subgraph_BarLows.DrawZeros = true;

Subgraph_BarCloses.Name = "Close";
Subgraph_BarCloses.DrawStyle = DRAWSTYLE_LINE;
Subgraph_BarCloses.PrimaryColor = RGB(128,0,0);
Subgraph_BarCloses.DrawZeros = true;

Subgraph_BarVolumes.Name = "Volume";
Subgraph_BarVolumes.DrawStyle = DRAWSTYLE_IGNORE;

Subgraph_OHLCAvg.Name = "OHLC Avg";
Subgraph_OHLCAvg.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_OHLCAvg.PrimaryColor = COLOR_GREEN;
Subgraph_OHLCAvg.DrawZeros = true;

Subgraph_HLCAvg.Name = "HLC Avg";
Subgraph_HLCAvg.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_HLCAvg.PrimaryColor = COLOR_GREEN;
Subgraph_HLCAvg.DrawZeros = true;

Subgraph_HLAvg.Name = "HL Avg";
Subgraph_HLAvg.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_HLAvg.PrimaryColor = COLOR_GREEN;
Subgraph_HLAvg.DrawZeros = true;

Input_ResetAtSessionStart.Name = "Reset at Start of Trading Day";
Input_ResetAtSessionStart.SetYesNo(1);

Input_ResetAtBothSessionStarts.Name = "Reset at Both Session Start Times";
Input_ResetAtBothSessionStarts.SetYesNo(false);

Input_PerformRollingCalculation.Name = "Perform Rolling Calculation";
Input_PerformRollingCalculation.SetYesNo(0);

Input_RollingCalculationLength.Name = "Rolling Calculation Length";
Input_RollingCalculationLength.SetInt(10);

return;
}

SCFloatArrayRef BidVolume = sc.BaseDataIn[SC_BIDVOL];
SCFloatArrayRef AskVolume = sc.BaseDataIn[SC_ASKVOL];
SCFloatArrayRef DifferenceHigh = sc.BaseDataIn[SC_ASKBID_DIFF_HIGH];
SCFloatArrayRef DifferenceLow = sc.BaseDataIn[SC_ASKBID_DIFF_LOW];

for (int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize; BarIndex++)
{
    if (!Input_PerformRollingCalculation.GetYesNo())
    {
        bool Reset = false;

        if (BarIndex == 0)
        {

```

```

    Reset = true;
}
else if (Input_ResetAtBothSessionStarts.GetYesNo() != 0)
{
    SCDatetime PriorStartOfPeriod = sc.GetStartOfPeriodForDateTime(sc.BaseDateTimeIn[BarIndex - 1],
TIME_PERIOD_LENGTH_UNIT_DAYS, 1, 0, 1);

    SCDatetime StartOfPeriod = sc.GetStartOfPeriodForDateTime(sc.BaseDateTimeIn[BarIndex],
TIME_PERIOD_LENGTH_UNIT_DAYS, 1, 0, 1);

    if (StartOfPeriod != PriorStartOfPeriod)
    {
        Reset = true;
    }
}
else if (Input_ResetAtSessionStart.GetYesNo() != 0)
{
    SCDatetime PriorStartOfPeriod = sc.GetStartOfPeriodForDateTime(sc.BaseDateTimeIn[BarIndex - 1],
TIME_PERIOD_LENGTH_UNIT_DAYS, 1, 0, 0);

    SCDatetime StartOfPeriod = sc.GetStartOfPeriodForDateTime(sc.BaseDateTimeIn[BarIndex],
TIME_PERIOD_LENGTH_UNIT_DAYS, 1, 0, 0);

    if (StartOfPeriod != PriorStartOfPeriod)
    {
        Reset = true;
    }
}

sc.CumulativeDeltaVolume(sc.BaseDataIn, Subgraph_BarCloses, BarIndex, Reset);

Subgraph_BarOpens[BarIndex] = Subgraph_BarCloses.Arrays[0][BarIndex];
Subgraph_BarHighs[BarIndex] = Subgraph_BarCloses.Arrays[1][BarIndex];
Subgraph_BarLows[BarIndex] = Subgraph_BarCloses.Arrays[2][BarIndex];

Subgraph_BarVolumes[BarIndex] = sc.Volume[BarIndex];
}
else
{
    int FirstIndex = BarIndex - Input_RollingCalculationLength.GetInt() + 1;

    if (FirstIndex < 0)
        FirstIndex = 0;

    for (int ReferenceIndex = FirstIndex; ReferenceIndex <= BarIndex; ReferenceIndex++)
    {
        if (ReferenceIndex == FirstIndex)
        {
            Subgraph_BarOpens[ReferenceIndex] = 0;
            Subgraph_BarHighs[ReferenceIndex] = DifferenceHigh[ReferenceIndex];
            Subgraph_BarLows[ReferenceIndex] = DifferenceLow[ReferenceIndex];
            Subgraph_BarCloses[ReferenceIndex] = AskVolume[ReferenceIndex] - BidVolume[ReferenceIndex];
        }
        else
        {
            Subgraph_BarOpens[ReferenceIndex] = Subgraph_BarCloses[ReferenceIndex - 1];
            Subgraph_BarHighs[ReferenceIndex] = Subgraph_BarCloses[ReferenceIndex - 1] +
DifferenceHigh[ReferenceIndex];

            Subgraph_BarLows[ReferenceIndex] = Subgraph_BarCloses[ReferenceIndex - 1] +
DifferenceLow[ReferenceIndex];

```

```
        Subgraph_BarCloses[ReferenceIndex - 1] = Subgraph_BarCloses[ReferenceIndex - 1] +  
(AskVolume[ReferenceIndex] - BidVolume[ReferenceIndex]);
```

```
    }
```

```
    if (Subgraph_BarOpens[ReferenceIndex] > Subgraph_BarHighs[ReferenceIndex])  
        Subgraph_BarOpens[ReferenceIndex] = Subgraph_BarHighs[ReferenceIndex];
```

```
    if (Subgraph_BarOpens[ReferenceIndex] < Subgraph_BarLows[ReferenceIndex])  
        Subgraph_BarOpens[ReferenceIndex] = Subgraph_BarLows[ReferenceIndex];
```

```
    }
```

```
}
```

```
    sc.CalculateOHLCAverages(BarIndex);
```

```
}
```

```
}
```

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

```
SCSFExport scsf_CumulativeDeltaBarsTicks(SCStudyInterfaceRef sc)
```

```
{
```

```
    SCSubgraphRef Subgraph_Open   = sc.Subgraph[SC_OPEN];
```

```
    SCSubgraphRef Subgraph_High   = sc.Subgraph[SC_HIGH];
```

```
    SCSubgraphRef Subgraph_Low    = sc.Subgraph[SC_LOW];
```

```
    SCSubgraphRef Subgraph_Close  = sc.Subgraph[SC_LAST];
```

```
    SCSubgraphRef Subgraph_OHLCAvg = sc.Subgraph[SC_OHLC_AVG];
```

```
    SCSubgraphRef Subgraph_HLCAvg  = sc.Subgraph[SC_HLC_AVG];
```

```
    SCSubgraphRef Subgraph_HLAvg   = sc.Subgraph[SC_HL_AVG];
```

```
    SCInputRef Input_ResetAtSessionStart = sc.Input[2];
```

```
    SCInputRef Input_ResetAtBothSessionStarts = sc.Input[3];
```

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

```
    {
```

```
        sc.GraphName = "Cumulative Delta Bars - Trades";
```

```
        sc.MaintainAdditionalChartDataArrays = 1;
```

```
        sc.AutoLoop = 0; //Manual looping
```

```
        sc.ValueFormat = 0;
```

```
        sc.GraphDrawType = GDT_CANDLESTICK;
```

```
        Subgraph_Open.Name = "Open";
```

```
        Subgraph_Open.DrawStyle = DRAWSTYLE_LINE;
```

```
        Subgraph_Open.PrimaryColor = RGB(0,255,0);
```

```
        Subgraph_Open.SecondaryColor = RGB(0,255,0);
```

```
        Subgraph_Open.SecondaryColorUsed = true;
```

```
        Subgraph_Open.DrawZeros = true;
```

```
        Subgraph_High.Name = "High";
```

```
        Subgraph_High.DrawStyle = DRAWSTYLE_LINE;
```

```
        Subgraph_High.PrimaryColor = RGB(0,128,0);
```

```
        Subgraph_High.DrawZeros = true;
```

```
        Subgraph_Low.Name = "Low";
```

```
        Subgraph_Low.DrawStyle = DRAWSTYLE_LINE;
```

```
        Subgraph_Low.PrimaryColor = RGB(255,0,0);
```

```
        Subgraph_Low.SecondaryColor = RGB(255,0,0);
```

```
        Subgraph_Low.SecondaryColorUsed = true;
```

```
        Subgraph_Low.DrawZeros = true;
```

```
        Subgraph_Close.Name = "Close";
```

```
        Subgraph_Close.DrawStyle = DRAWSTYLE_LINE;
```

```
        Subgraph_Close.PrimaryColor = RGB(128,0,0);
```

```

Subgraph_Close.DrawZeros = true;

Subgraph_OHLCAvg.Name = "OHLC Avg";
Subgraph_OHLCAvg.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_OHLCAvg.PrimaryColor = COLOR_GREEN;
Subgraph_OHLCAvg.DrawZeros = true;

Subgraph_HLCAvg.Name = "HLC Avg";
Subgraph_HLCAvg.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_HLCAvg.PrimaryColor = COLOR_GREEN;
Subgraph_HLCAvg.DrawZeros = true;

Subgraph_HLAvg.Name = "HL Avg";
Subgraph_HLAvg.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_HLAvg.PrimaryColor = COLOR_GREEN;
Subgraph_HLAvg.DrawZeros = true;

Input_ResetAtSessionStart.Name = "Reset at Start of Trading Day";
Input_ResetAtSessionStart.SetYesNo(1);

Input_ResetAtBothSessionStarts.Name = "Reset at Both Session Start Times";
Input_ResetAtBothSessionStarts.SetYesNo(false);

return;
}

for (int Index = sc.UpdateStartIndex; Index < sc.ArraySize; Index++)
{
    bool Reset = false;

    if (Index == 0)
        Reset = true;
    else if (Input_ResetAtBothSessionStarts.GetYesNo() != 0 )
    {
        SCDateTime PriorStartOfPeriod = sc.GetStartOfPeriodForDateTime(sc.BaseDateIn[Index - 1],
TIME_PERIOD_LENGTH_UNIT_DAYS ,1,0,1);

        SCDateTime StartOfPeriod = sc.GetStartOfPeriodForDateTime(sc.BaseDateIn[Index],
TIME_PERIOD_LENGTH_UNIT_DAYS ,1,0,1);

        if (StartOfPeriod != PriorStartOfPeriod)
            Reset = true;
    }
    else if (Input_ResetAtSessionStart.GetYesNo() != 0 )
    {
        SCDateTime PriorStartOfPeriod = sc.GetStartOfPeriodForDateTime(sc.BaseDateIn[Index - 1],
TIME_PERIOD_LENGTH_UNIT_DAYS , 1, 0, 0);

        SCDateTime StartOfPeriod = sc.GetStartOfPeriodForDateTime(sc.BaseDateIn[Index],
TIME_PERIOD_LENGTH_UNIT_DAYS , 1, 0, 0);

        if (StartOfPeriod != PriorStartOfPeriod)
            Reset = true;
    }

    sc.CumulativeDeltaTicks(sc.BaseDataIn, Subgraph_Close, Index, Reset);

    Subgraph_Open[Index] = Subgraph_Close.Arrays[0][Index];
    Subgraph_High[Index] = Subgraph_Close.Arrays[1][Index];
    Subgraph_Low[Index] = Subgraph_Close.Arrays[2][Index];

    sc.CalculateOHLCAverages(Index);
}
}

```

```

/*=====*/
SCSFExport scsf_CumulativeDeltaBarsTickVolume(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Open   = sc.Subgraph[SC_OPEN];
    SCSubgraphRef Subgraph_High   = sc.Subgraph[SC_HIGH];
    SCSubgraphRef Subgraph_Low    = sc.Subgraph[SC_LOW];
    SCSubgraphRef Subgraph_Close  = sc.Subgraph[SC_LAST];
    SCSubgraphRef Subgraph_OHLCAvg = sc.Subgraph[SC_OHLC_AVG];
    SCSubgraphRef Subgraph_HLCAvg = sc.Subgraph[SC_HLC_AVG];
    SCSubgraphRef Subgraph_HLAvG  = sc.Subgraph[SC_HL_AVG];

    SCInputRef Input_ResetAtSessionStart = sc.Input[2];
    SCInputRef Input_ResetAtBothSessionStarts = sc.Input[3];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Cumulative Delta Bars - Up/Down Tick Volume";

        sc.MaintainAdditionalChartDataArrays = 1;
        sc.AutoLoop = 0; //Manual looping

        sc.ValueFormat = 0;
        sc.GraphDrawType = GDT_CANDLESTICK;

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

        Subgraph_High.Name = "High";
        Subgraph_High.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_High.PrimaryColor = RGB(0,128,0);
        Subgraph_High.DrawZeros = true;

        Subgraph_Low.Name = "Low";
        Subgraph_Low.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Low.PrimaryColor = RGB(255,0,0);
        Subgraph_Low.SecondaryColor = RGB(255,0,0);
        Subgraph_Low.SecondaryColorUsed = true;
        Subgraph_Low.DrawZeros = true;

        Subgraph_Close.Name = "Close";
        Subgraph_Close.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Close.PrimaryColor = RGB(128,0,0);
        Subgraph_Close.DrawZeros = true;

        Subgraph_OHLCAvg.Name = "OHLC Avg";
        Subgraph_OHLCAvg.DrawStyle = DRAWSTYLE_IGNORE;
        Subgraph_OHLCAvg.PrimaryColor = COLOR_GREEN;
        Subgraph_OHLCAvg.DrawZeros = true;

        Subgraph_HLCAvg.Name = "HLC Avg";
        Subgraph_HLCAvg.DrawStyle = DRAWSTYLE_IGNORE;
        Subgraph_HLCAvg.PrimaryColor = COLOR_GREEN;
        Subgraph_HLCAvg.DrawZeros = true;

        Subgraph_HLAvG.Name = "HL Avg";
        Subgraph_HLAvG.DrawStyle = DRAWSTYLE_IGNORE;
        Subgraph_HLAvG.PrimaryColor = COLOR_GREEN;
        Subgraph_HLAvG.DrawZeros = true;

        Input_ResetAtSessionStart.Name = "Reset at Start of Trading Day";
    }
}

```

```

Input_ResetAtSessionStart.SetYesNo(1);

Input_ResetAtBothSessionStarts.Name = "Reset at Both Session Start Times";
Input_ResetAtBothSessionStarts.SetYesNo(false);

return;
}

for (int Index = sc.UpdateStartIndex; Index < sc.ArraySize; Index++)
{
    bool Reset = false;

    if (Index == 0)
        Reset = true;
    else if (Input_ResetAtBothSessionStarts.GetYesNo() != 0 )
    {
        SCDateTime PriorStartOfPeriod = sc.GetStartOfPeriodForDateTime(sc.BaseDateTimeIn[Index - 1],
TIME_PERIOD_LENGTH_UNIT_DAYS, 1, 0, 1);

        SCDateTime StartOfPeriod = sc.GetStartOfPeriodForDateTime(sc.BaseDateTimeIn[Index],
TIME_PERIOD_LENGTH_UNIT_DAYS, 1, 0, 1);

        if (StartOfPeriod != PriorStartOfPeriod)
            Reset = true;
    }
    else if (Input_ResetAtSessionStart.GetYesNo() != 0 )
    {
        SCDateTime PriorStartOfPeriod = sc.GetStartOfPeriodForDateTime(sc.BaseDateTimeIn[Index - 1],
TIME_PERIOD_LENGTH_UNIT_DAYS, 1, 0, 0);

        SCDateTime StartOfPeriod = sc.GetStartOfPeriodForDateTime(sc.BaseDateTimeIn[Index],
TIME_PERIOD_LENGTH_UNIT_DAYS, 1, 0, 0);

        if (StartOfPeriod != PriorStartOfPeriod)
            Reset = true;
    }

    sc.CumulativeDeltaTickVolume(sc.BaseDataIn, Subgraph_Close, Index, Reset);

    Subgraph_Open[Index] = Subgraph_Close.Arrays[0][Index];
    Subgraph_High[Index] = Subgraph_Close.Arrays[1][Index];
    Subgraph_Low[Index] = Subgraph_Close.Arrays[2][Index];

    sc.CalculateOHLCAverages(Index);
}
}

/*=====*/
SCSFExport scsf_BackgroundDrawStyleExample(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Background = sc.Subgraph[0];
    SCSubgraphRef Subgraph_BackgroundDC = sc.Subgraph[1];

    if (sc.SetDefaults)
    {
        // Set the configuration and defaults
        sc.GraphName = "Background DrawStyle Example";
        sc.StudyDescription = "Background DrawStyle Example";

        sc.AutoLoop = true;

        sc.GraphRegion = 0;

```

```

sc.DrawStudyUnderneathMainPriceGraph = 1; // not required in studies, but want color behind price for this study

Subgraph_Background.Name = "Background";
Subgraph_Background.DrawStyle = DRAWSTYLE_BACKGROUND;
Subgraph_Background.PrimaryColor = COLOR_LIGHTGREEN;
Subgraph_Background.SecondaryColor = COLOR_LIGHTPINK;
Subgraph_Background.SecondaryColorUsed = true; // turn on if both colors are to be used
Subgraph_Background.AutoColoring = AUTOCOLOR_POSNEG; // use positive/negative values to signify colors

Subgraph_BackgroundDC.Name = "Background DataColor";
Subgraph_BackgroundDC.DrawStyle = DRAWSTYLE_BACKGROUND;
Subgraph_BackgroundDC.PrimaryColor = RGB(255,0,255);

return;
}

// Do data processing
int min = sc.BaseDateTimeln[sc.Index].GetMinute();

if (min > 0 && min < 15)
    Subgraph_Background[sc.Index] = 0; // do not color background
else if (min >= 15 && min < 30)
    Subgraph_Background[sc.Index] = 1; // use primary color
else if (min >= 30 && min < 45)
    Subgraph_Background[sc.Index] = -1; // use secondary color
else if (min >= 45 && min < 60)
{
    Subgraph_BackgroundDC[sc.Index] = 1;
    Subgraph_BackgroundDC.DataColor[sc.Index] = RGB(0, 0, 17*(60-min));
}
}

/*=====*/
SCSFExport scsf_NumericInformationGraphDrawTypeExample(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Grid = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Open = sc.Subgraph[1];
    SCSubgraphRef Subgraph_High = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Low = sc.Subgraph[3];
    SCSubgraphRef Subgraph_Close = sc.Subgraph[4];
    SCSubgraphRef Subgraph_Volume = sc.Subgraph[5];
    SCSubgraphRef Subgraph_AskBidVolumeDiff = sc.Subgraph[6];
    SCSubgraphRef Subgraph_PriceSMA = sc.Subgraph[7];

    SCInputRef Input_FontSize = sc.Input[0];

    if (sc.SetDefaults)
    {
        // Set the configuration and defaults
        sc.GraphName = "Numeric Information Graph Draw Type Example";
        sc.StudyDescription = "Numeric Information Graph Draw Type Example";

        sc.AutoLoop = true;

        sc.GraphDrawType = GDT_NUMERIC_INFORMATION;

        Input_FontSize.Name = "Font Size";
        Input_FontSize.SetInt(8);
        Input_FontSize.SetIntLimits(0,100);

        Subgraph_Grid.Name = "Grid Style";
        Subgraph_Grid.DrawStyle = DRAWSTYLE_LINE;
    }
}

```



```

Subgraph_Grid.PrimaryColor = COLOR_GRAY;

Subgraph_Open.Name = "Open";
Subgraph_Open.DrawStyle = DRAWSTYLE_LINE;
Subgraph_Open.PrimaryColor = COLOR_CYAN;

Subgraph_High.Name = "High";
Subgraph_High.DrawStyle = DRAWSTYLE_LINE;
Subgraph_High.PrimaryColor = COLOR_GREEN;

Subgraph_Low.Name = "Low";
Subgraph_Low.DrawStyle = DRAWSTYLE_LINE;
Subgraph_Low.PrimaryColor = COLOR_RED;

Subgraph_Close.Name = "Close";
Subgraph_Close.DrawStyle = DRAWSTYLE_LINE;
Subgraph_Close.PrimaryColor = COLOR_PURPLE;

Subgraph_Volume.Name = "Volume";
Subgraph_Volume.DrawStyle = DRAWSTYLE_LINE;
Subgraph_Volume.PrimaryColor = COLOR_WHITE;

Subgraph_AskBidVolumeDiff.Name = "AB Vol Diff";
Subgraph_AskBidVolumeDiff.DrawStyle = DRAWSTYLE_LINE;
Subgraph_AskBidVolumeDiff.PrimaryColor = COLOR_WHITE;

Subgraph_PriceSMA.Name = "SMA";
Subgraph_PriceSMA.DrawStyle = DRAWSTYLE_LINE;
Subgraph_PriceSMA.PrimaryColor = COLOR_WHITE;

return;
}

if (sc.Index == 0)
{
    sc.ValueFormat = sc.BaseGraphValueFormat;

    // set up the information graph draw type
    s_NumericInformationGraphDrawTypeConfig NumericInformationGraphDrawTypeConfig;

    // only need to override defaults (see s_NumericInformationGraphDrawTypeConfig for defaults and other options to
    override)

    NumericInformationGraphDrawTypeConfig.TransparentTextBackground = false;
    NumericInformationGraphDrawTypeConfig.GridLineStyleSubgraphIndex = 0; // Subgraph 1 to specify grid line style
    NumericInformationGraphDrawTypeConfig.FontSize = Input_FontSize.GetInt();
    NumericInformationGraphDrawTypeConfig.ValueFormat[5] = 0; // set value format for volume, others are inherited
    NumericInformationGraphDrawTypeConfig.ShowPullback = true;

    sc.SetNumericInformationGraphDrawTypeConfig(NumericInformationGraphDrawTypeConfig);

    // display volume first
    sc.SetNumericInformationDisplayOrderFromString("6,2,3,4,5,7,8");
}

// Do data processing
Subgraph_Open[sc.Index] = sc.Open[sc.Index];
Subgraph_High[sc.Index] = sc.High[sc.Index];
Subgraph_Low[sc.Index] = sc.Low[sc.Index];
Subgraph_Close[sc.Index] = sc.Close[sc.Index];
Subgraph_Volume[sc.Index] = sc.Volume[sc.Index];

SCFloatArrayRef SMA = Subgraph_Close.Arrays[0];

sc.MovingAverage(sc.Close, SMA, MOVAVGTYPE_SIMPLE, 10);

```

```

Subgraph_PriceSMA[sc.Index] = SMA[sc.Index];

Subgraph_AskBidVolumeDiff[sc.Index] = sc.AskVolume[sc.Index] - sc.BidVolume[sc.Index];

//Set value in column after last chart bar column (pullback column)
if (sc.Index == sc.ArraySize - 1)
    Subgraph_AskBidVolumeDiff[sc.Index + 1] = Subgraph_AskBidVolumeDiff[sc.Index];

Subgraph_Low.DataColor[sc.Index] = sc.CombinedForegroundColorRef(COLOR_BLACK,
COLOR_YELLOW);

// set cell foreground/background color
if (Subgraph_Close[sc.Index] > Subgraph_Close[sc.Index-1])
    Subgraph_Close.DataColor[sc.Index] = sc.CombinedForegroundColorRef(COLOR_BLACK,
COLOR_GREEN);
else
    Subgraph_Close.DataColor[sc.Index] = sc.CombinedForegroundColorRef(COLOR_WHITE,
COLOR_RED);
}

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

    SCInputRef Input_Data = sc.Input[0];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Study Subgraph Absolute Value";
        sc.StudyDescription = "This study calculates and displays the absolute value of the selected study Subgraph. To
select the study to use, set the Based On study setting to that study and set the Input Data input to the specific study
Subgraph in that study selected.";

        sc.AutoLoop = 0;

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

        Input_Data.Name = "Input Data";
        Input_Data.SetInputDataIndex(0);

        return;
    }

    int CalculationStartIndex = sc.GetCalculationStartIndexForStudy();

    for (int Index = CalculationStartIndex; Index < sc.ArraySize; Index++)
    {
        Subgraph_Result[Index] = fabs(sc.BaseData[Input_Data.GetInputDataIndex()][Index]);
    }

    sc.EarliestUpdateSubgraphDataArrayIndex = CalculationStartIndex;

```

```
}
```

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

```
SCSFExport scsf_VolumeZoneOscillator(SCStudyInterfaceRef sc)
```

```
{
```

```
    SCInputRef Input_Period      = sc.Input[1];
    SCInputRef Input_OverBoughtLevel3 = sc.Input[2];
    SCInputRef Input_OverBoughtLevel2 = sc.Input[3];
    SCInputRef Input_OverBoughtLevel1 = sc.Input[4];
    SCInputRef Input_OverSoldLevel1  = sc.Input[5];
    SCInputRef Input_OverSoldLevel2  = sc.Input[6];
    SCInputRef Input_OverSoldLevel3  = sc.Input[7];
```

```
    SCSubgraphRef Subgraph_OverBought3 = sc.Subgraph[0];
    SCSubgraphRef Subgraph_OverBought2 = sc.Subgraph[1];
    SCSubgraphRef Subgraph_OverBought1 = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Zero      = sc.Subgraph[3];
    SCSubgraphRef Subgraph_OverSold1  = sc.Subgraph[4];
    SCSubgraphRef Subgraph_OverSold2  = sc.Subgraph[5];
    SCSubgraphRef Subgraph_OverSold3  = sc.Subgraph[6];
    SCSubgraphRef Subgraph_VZO       = sc.Subgraph[7];
```

```
    SCFloatArrayRef Array_R = Subgraph_VZO.Arrays[0];
    SCFloatArrayRef Array_VP = Subgraph_VZO.Arrays[1];
    SCFloatArrayRef Array_TV = Subgraph_VZO.Arrays[2];
```

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

```
    {
```

```
        sc.GraphName = "Volume Zone Oscillator";
```

```
        sc.StudyDescription = "TASC Article, May, 2011, In The Volume Zone";
```

```
        sc.GraphRegion = 1;
        sc.ValueFormat = 2;
        sc.AutoLoop = 1;
        sc.DrawZeros = 1;
```

```
        // inputs
```

```
        Input_Period.Name = "Period";
        Input_Period.SetInt(14);
        Input_Period.SetIntLimits(1, MAX_STUDY_LENGTH);
```

```
        Input_OverBoughtLevel3.Name = "Overbought Line3 Value";
        Input_OverBoughtLevel3.SetFloat(60.0f);
```

```
        Input_OverBoughtLevel2.Name = "Overbought Line2 Value";
        Input_OverBoughtLevel2.SetFloat(40.0f);
```

```
        Input_OverBoughtLevel1.Name = "Overbought Line1 Value";
        Input_OverBoughtLevel1.SetFloat(15.0f);
```

```
        Input_OverSoldLevel1.Name = "Oversold Line1 Value";
        Input_OverSoldLevel1.SetFloat(-5.0f);
```

```
        Input_OverSoldLevel2.Name = "Oversold Line2 Value";
        Input_OverSoldLevel2.SetFloat(-40.0f);
```

```
        Input_OverSoldLevel3.Name = "Oversold Line3 Value";
        Input_OverSoldLevel3.SetFloat(-60.0f);
```

```
        // subgraphs
```

```
        Subgraph_VZO.Name = "VZO";
        Subgraph_VZO.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_VZO.PrimaryColor = RGB(0,255,0);
```

```

Subgraph_VZO.DrawZeros = true;

Subgraph_OverBought3.Name = "Over Bought Level 3";
Subgraph_OverBought3.DrawStyle = DRAWSTYLE_LINE;
Subgraph_OverBought3.PrimaryColor = RGB(255,0,255);
Subgraph_OverBought3.DrawZeros = true;
Subgraph_OverBought3.DisplayNameValueInWindowsFlags = 0;

Subgraph_OverBought2.Name = "Over Bought Level 2";
Subgraph_OverBought2.DrawStyle = DRAWSTYLE_LINE;
Subgraph_OverBought2.PrimaryColor = RGB(255,0,255);
Subgraph_OverBought2.DrawZeros = true;
Subgraph_OverBought2.DisplayNameValueInWindowsFlags = 0;

Subgraph_OverBought1.Name = "Over Bought Level 1";
Subgraph_OverBought1.DrawStyle = DRAWSTYLE_LINE;
Subgraph_OverBought1.PrimaryColor = RGB(255,0,255);
Subgraph_OverBought1.DrawZeros = true;
Subgraph_OverBought1.DisplayNameValueInWindowsFlags = 0;

Subgraph_Zero.Name = "Zero";
Subgraph_Zero.DrawStyle = DRAWSTYLE_LINE;
Subgraph_Zero.PrimaryColor = RGB(255,255,255);
Subgraph_Zero.DrawZeros = true;
Subgraph_Zero.DisplayNameValueInWindowsFlags = 0;

Subgraph_OverSold1.Name = "Over Sold Level 1";
Subgraph_OverSold1.DrawStyle = DRAWSTYLE_LINE;
Subgraph_OverSold1.PrimaryColor = RGB(255,255,0);
Subgraph_OverSold1.DrawZeros = true;
Subgraph_OverSold1.DisplayNameValueInWindowsFlags = 0;

Subgraph_OverSold2.Name = "Over Sold Level 2";
Subgraph_OverSold2.DrawStyle = DRAWSTYLE_LINE;
Subgraph_OverSold2.PrimaryColor = RGB(255,255,0);
Subgraph_OverSold2.DrawZeros = true;
Subgraph_OverSold2.DisplayNameValueInWindowsFlags = 0;

Subgraph_OverSold3.Name = "Over Sold Level 3";
Subgraph_OverSold3.DrawStyle = DRAWSTYLE_LINE;
Subgraph_OverSold3.PrimaryColor = RGB(255,255,0);
Subgraph_OverSold3.DrawZeros = true;
Subgraph_OverSold3.DisplayNameValueInWindowsFlags = 0;

return;
}

Subgraph_OverBought3[sc.Index] = Input_OverBoughtLevel3.GetFloat();
Subgraph_OverBought2[sc.Index] = Input_OverBoughtLevel2.GetFloat();
Subgraph_OverBought1[sc.Index] = Input_OverBoughtLevel1.GetFloat();
Subgraph_Zero[sc.Index] = 0;
Subgraph_OverSold1[sc.Index] = Input_OverSoldLevel1.GetFloat();
Subgraph_OverSold2[sc.Index] = Input_OverSoldLevel2.GetFloat();
Subgraph_OverSold3[sc.Index] = Input_OverSoldLevel3.GetFloat();

if (sc.Index == 0)
{
    Array_R[sc.Index] = sc.Volume[sc.Index];
    return;
}
else
{
    if (sc.Close[sc.Index] > sc.Close[sc.Index-1])
        Array_R[sc.Index] = sc.Volume[sc.Index];
    else

```

```

    Array_R[sc.Index] = -sc.Volume[sc.Index];
}

sc.MovingAverage(Array_R, Array_VP, MOVAVGTYPE_EXPONENTIAL, Input_Period.GetInt());
sc.MovingAverage(sc.Volume, Array_TV, MOVAVGTYPE_EXPONENTIAL, Input_Period.GetInt());

if (Array_TV[sc.Index] != 0)
{
    Subgraph_VZO[sc.Index] = Array_VP[sc.Index] / Array_TV[sc.Index] * 100.0f;
}
}

/*=====*/
SCSFExport scsf_FisherTransform(SCStudyInterfaceRef sc)
{
    SCInputRef Input_Price = sc.Input[0];
    SCInputRef Input_Period = sc.Input[1];

    SCSubgraphRef Subgraph_Zero = sc.Subgraph[0];
    SCSubgraphRef Subgraph_FisherTransform = sc.Subgraph[1];
    SCSubgraphRef Subgraph_FisherTransformM1 = sc.Subgraph[2];

    SCFloatArrayRef Array_Value = Subgraph_Zero.Arrays[0];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Fisher Transform";

        sc.StudyDescription = "Fisher Transform";

        sc.GraphRegion = 1;
        sc.ValueFormat = 2;
        sc.AutoLoop = 1;
        sc.DrawZeros = 1;

        // inputs
        Input_Price.Name = "Price";
        Input_Price.SetInputDataIndex(SC_HL_AVG);

        Input_Period.Name = "Period";
        Input_Period.SetInt(10);
        Input_Period.SetIntLimits(1, MAX_STUDY_LENGTH);

        // subgraphs
        Subgraph_FisherTransform.Name = "FisherTransform";
        Subgraph_FisherTransform.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_FisherTransform.PrimaryColor = RGB(255,0,0);
        Subgraph_FisherTransform.DrawZeros = true;

        Subgraph_FisherTransformM1.Name = "FisherTransformOffset";
        Subgraph_FisherTransformM1.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_FisherTransformM1.PrimaryColor = RGB(0,0,255);
        Subgraph_FisherTransformM1.DrawZeros = true;

        Subgraph_Zero.Name = "Zero";
        Subgraph_Zero.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Zero.PrimaryColor = RGB(255,255,255);
        Subgraph_Zero.DrawZeros = true;
        Subgraph_Zero.DisplayNameValueInWindowsFlags = 0;

        return;
    }

    Subgraph_Zero[sc.Index] = 0;

```

```

if (sc.Index != 0)
{
    SCFloatArrayRef Price = sc.BaseData[Input_Price.GetInputDataIndex()];

    float Highest = sc.GetHighest(Price, Input_Period.GetInt());
    float Lowest = sc.GetLowest(Price, Input_Period.GetInt());
    float Range = Highest - Lowest;

    if (Range == 0)
        Array_Value[sc.Index] = 0;
    else
        Array_Value[sc.Index] = .66f * ((Price[sc.Index] - Lowest) / Range - 0.5f) + 0.67f * Array_Value[sc.Index-1];

    float TruncValue = Array_Value[sc.Index];

    if (TruncValue > .99f)
        TruncValue = .999f;
    else if (TruncValue < -.99f)
        TruncValue = -.999f;

    float Fisher = .5f * (log((1 + TruncValue) / (1 - TruncValue)) + Subgraph_FisherTransform[sc.Index-1]);

    Subgraph_FisherTransformM1[sc.Index] = Subgraph_FisherTransform[sc.Index-1];
    Subgraph_FisherTransform[sc.Index] = Fisher;
}
}

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

    SCFloatArrayRef Array_Avg = Subgraph_ZScore.Arrays[0];
    SCFloatArrayRef Array_StdDev = Subgraph_ZScore.Arrays[1];

    SCInputRef Input_Data = sc.Input[0];
    SCInputRef Input_MeanLength = sc.Input[1];
    SCInputRef Input_StdDevLength = sc.Input[2];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Z-Score";

        sc.ValueFormat = 2;
        sc.AutoLoop = 1;

        Subgraph_ZScore.Name = "Z-Score";
        Subgraph_ZScore.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_ZScore.PrimaryColor = RGB(192, 192, 192);
        Subgraph_ZScore.SecondaryColor = RGB(192, 192, 192);
        Subgraph_ZScore.SecondaryColorUsed = 1;
        Subgraph_ZScore.DrawZeros = true;

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

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

        Input_StdDevLength.Name = "Standard Deviation Length";
        Input_StdDevLength.SetInt(10);
        Input_StdDevLength.SetIntLimits(1, MAX_STUDY_LENGTH);

        return;
    }
}

```

```

}

if (sc.Index == 0)
{
    sc.DataStartIndex = max(Input_MeanLength.GetInt() - 1, Input_StdDevLength.GetInt() - 1);
}

sc.SimpleMovAvg(sc.BaseData[Input_Data.GetInputDataIndex()], Array_Avg, Input_MeanLength.GetInt());

sc.StdDeviation(sc.BaseData[Input_Data.GetInputDataIndex()], Array_StdDev, Input_StdDevLength.GetInt());

if (Array_StdDev[sc.Index] <= 0)
{
    Subgraph_ZScore[sc.Index] = 0;
}
else
{
    Subgraph_ZScore[sc.Index] = (sc.BaseData[Input_Data.GetInputDataIndex()][sc.Index] - Array_Avg[sc.Index]) /
Array_StdDev[sc.Index];
}
}

/*=====*/
SCSFExport scsf_VolumeAtPriceThresholdAlert(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_PriceTriggeredForBid = sc.Subgraph[0];
    SCSubgraphRef Subgraph_PriceTriggeredForAsk = sc.Subgraph[1];
    SCSubgraphRef Subgraph_PriceTriggeredForTotal = sc.Subgraph[2];

    SCInputRef Input_BidVolumeThreshold = sc.Input[1];
    SCInputRef Input_BidAlertNum = sc.Input[2];
    SCInputRef Input_AskVolumeThreshold = sc.Input[3];
    SCInputRef Input_AskAlertNum = sc.Input[4];
    SCInputRef Input_TotalVolumeThreshold = sc.Input[5];
    SCInputRef Input_TotalAlertNum = sc.Input[6];

    if (sc.SetDefaults)
    {
        // Set the configuration and defaults
        sc.GraphName = "Volume At Price Threshold Alert";
        sc.StudyDescription = "This study triggers alerts based on configurable volume thresholds for the bid/ask/total volume at price.";

        sc.GraphRegion = 0;
        sc.AutoLoop = 0; //Manual looping
        sc.ValueFormat = sc.BaseGraphValueFormat;

        sc.MaintainVolumeAtPriceData = 1; // true

        Subgraph_PriceTriggeredForBid.Name = "Price Triggered By Bid Threshold";
        Subgraph_PriceTriggeredForBid.DrawStyle = DRAWSTYLE_IGNORE;
        Subgraph_PriceTriggeredForBid.DrawZeros = 0;

        Subgraph_PriceTriggeredForAsk.Name = "Price Triggered By Ask Threshold";
        Subgraph_PriceTriggeredForAsk.DrawStyle = DRAWSTYLE_IGNORE;
        Subgraph_PriceTriggeredForAsk.DrawZeros = 0;

        Subgraph_PriceTriggeredForTotal.Name = "Price Triggered By Total Threshold";
        Subgraph_PriceTriggeredForTotal.DrawStyle = DRAWSTYLE_IGNORE;
        Subgraph_PriceTriggeredForTotal.DrawZeros = 0;

        Input_BidVolumeThreshold.Name = "Bid Volume Threshold (0 disables)";
        Input_BidVolumeThreshold.SetInt(100);

        Input_BidAlertNum.Name = "Bid Volume Alert";
    }
}

```

```

Input_BidAlertNum.SetAlertSoundNumber(0);

Input_AskVolumeThreshold.Name = "Ask Volume Threshold (0 disables)";
Input_AskVolumeThreshold.SetInt(100);

Input_AskAlertNum.Name = "Ask Volume Alert";
Input_AskAlertNum.SetAlertSoundNumber(0);

Input_TotalVolumeThreshold.Name = "Total Volume Threshold (0 disables)";
Input_TotalVolumeThreshold.SetInt(100);

Input_TotalAlertNum.Name = "Total Volume Alert";
Input_TotalAlertNum.SetAlertSoundNumber(0);

return;
}

// Do data processing
if (sc.UpdateStartIndex == 0)
{
    sc.ValueFormat = sc.BaseGraphValueFormat;
}

if (static_cast<int>(sc.VolumeAtPriceForBars->GetNumberOfBars()) < sc.ArraySize)
    return;//This is an indication that the volume at price data does not exist

SCString AlertMessage;

int Volume = 0;
int VolumeThreshold = 0;
bool EnableAlerts = sc.IsFullRecalculation == 0 && !sc.ChartIsDownloadingHistoricalData(sc.ChartNumber);

for (int Index = sc.UpdateStartIndex; Index < sc.ArraySize; Index++)
{
    const s_VolumeAtPriceV2 *p_VolumeAtPrice = nullptr;

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

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

        // check the bid volume at price
        VolumeThreshold = Input_BidVolumeThreshold.GetInt();

        if (VolumeThreshold > 0 && Subgraph_PriceTriggeredForBid[Index] == 0)
        {
            Volume = p_VolumeAtPrice->BidVolume;

            if (Volume >= VolumeThreshold)
            {
                Subgraph_PriceTriggeredForBid[Index] = Price;

                if (EnableAlerts && Input_BidAlertNum.GetAlertSoundNumber() > 0 && Index == sc.ArraySize - 1)
                {
                    AlertMessage.Format("Bid Volume %i triggered alert at %s",
                        Volume,
                        sc.FormatGraphValue(Price, sc.ValueFormat).GetChars()
                    );

                    sc.SetAlert(Input_BidAlertNum.GetAlertSoundNumber() - 1, AlertMessage);
                }
            }
        }
    }
}

```



```

// check the ask volume at price
VolumeThreshold = Input_AskVolumeThreshold.GetInt();

if (VolumeThreshold > 0 && Subgraph_PriceTriggeredForAsk[Index] == 0)
{
    Volume = p_VolumeAtPrice->AskVolume;

    if (Volume >= VolumeThreshold)
    {
        Subgraph_PriceTriggeredForAsk[Index] = Price;

        if (EnableAlerts && Input_AskAlertNum.GetAlertSoundNumber() > 0 && Index == sc.ArraySize - 1)
        {
            AlertMessage.Format("Ask Volume %i triggered alert at %s",
                Volume,
                sc.FormatGraphValue(Price, sc.ValueFormat).GetChars()
            );

            sc.SetAlert(Input_AskAlertNum.GetAlertSoundNumber() - 1, AlertMessage);
        }
    }
}

// check the total volume at price
VolumeThreshold = Input_TotalVolumeThreshold.GetInt();

if (VolumeThreshold > 0 && Subgraph_PriceTriggeredForTotal[Index] == 0)
{
    Volume = p_VolumeAtPrice->Volume;

    if (Volume >= VolumeThreshold)
    {
        Subgraph_PriceTriggeredForTotal[Index] = Price;

        if (EnableAlerts && Input_TotalAlertNum.GetAlertSoundNumber() > 0 && Index == sc.ArraySize - 1)
        {
            AlertMessage.Format("Total Volume %i triggered alert at %s",
                Volume,
                sc.FormatGraphValue(Price, sc.ValueFormat).GetChars()
            );

            sc.SetAlert(Input_TotalAlertNum.GetAlertSoundNumber() - 1, AlertMessage);
        }
    }
}
}
}
}
}

/*=====*/
SCSFExport scsf_VolumeAtPriceThresholdAlertV2(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_ExtensionLineProperties = sc.Subgraph[SC_SUBGRAPHS_AVAILABLE - 1];
    SCSubgraphRef Subgraph_CountOfAlerts = sc.Subgraph[SC_SUBGRAPHS_AVAILABLE - 2];

    int AdjacentAlertsHighlightSubgraphStartingNumber = SC_SUBGRAPHS_AVAILABLE - 12;

    SCInputRef Input_ComparisonMethod = sc.Input[0];
    SCInputRef Input_VolumeThreshold = sc.Input[1];
    SCInputRef Input_AlertNumber = sc.Input[2];
    SCInputRef Input_DrawExtensionLines = sc.Input[3];
    SCInputRef Input_DrawExtensionLinesWithTransparentRange = sc.Input[4];
    SCInputRef Input_PercentageThreshold = sc.Input[5];
    SCInputRef Input_AdditionalVolumeThreshold = sc.Input[6];

```

```

SCInputRef Input_AllowZeroValueCompares = sc.Input[7];
SCInputRef Input_DivideByZeroAction = sc.Input[8];
SCInputRef Input_HighlightAdjacentAlertsGroupSize = sc.Input[9];
SCInputRef Input_DrawExtensionLinesUntilEndOfChart = sc.Input[10];
SCInputRef Input_NumberOfDaysToCalculate = sc.Input[11];
SCInputRef Input_Version = sc.Input[12];

if (sc.SetDefaults)
{
    // Set the configuration and defaults
    sc.GraphName = "Volume At Price Threshold Alert V2";

    sc.GraphRegion = 0;
    sc.AutoLoop = 0; //Manual looping
    sc.ValueFormat = sc.BaseGraphValueFormat;

    sc.MaintainVolumeAtPriceData = 1; // true

    for (int SubgraphIndex = 0; SubgraphIndex < SC_SUBGRAPHS_AVAILABLE - 13; ++SubgraphIndex)
    {
        SCString SubgraphName;
        SubgraphName.Format("Trigger %d", SubgraphIndex);

        sc.Subgraph[SubgraphIndex].Name = SubgraphName;
        sc.Subgraph[SubgraphIndex].PrimaryColor = RGB(255, 128, 0);
        sc.Subgraph[SubgraphIndex].DrawStyle = DRAWSTYLE_SQUARE_OFFSET_LEFT_FOR_CANDLESTICK;
        sc.Subgraph[SubgraphIndex].LineWidth = 8;
        sc.Subgraph[SubgraphIndex].DrawZeros = 0;
        sc.Subgraph[SubgraphIndex].DisplayNameValueInWindowsFlags = 0;
    }

    Subgraph_CountOfAlerts.Name = "Count of Alerts";
    Subgraph_CountOfAlerts.DrawStyle = DRAWSTYLE_IGNORE;
    Subgraph_CountOfAlerts.PrimaryColor = RGB(0, 255, 0);
    Subgraph_CountOfAlerts.DrawZeros = 1;

    for (int SubgraphIndex = 0; SubgraphIndex < 10; ++SubgraphIndex)
    {
        SCString SubgraphName;
        SubgraphDrawStyles AdjacentAlertsDrawStyle;
        if (SubgraphIndex % 2 == 0)
        {
            SubgraphName.Format("Adjacent Alert Highlight Bottom %d", static_cast<int>(SubgraphIndex / 2) + 1);
            AdjacentAlertsDrawStyle = DRAWSTYLE_LEFT_OFFSET_BOX_TOP_FOR_CANDLESTICK;
        }
        else
        {
            SubgraphName.Format("Adjacent Alert Highlight Top %d", static_cast<int>(SubgraphIndex / 2) + 1);
            AdjacentAlertsDrawStyle = DRAWSTYLE_LEFT_OFFSET_BOX_BOTTOM_FOR_CANDLESTICK;
        }

        sc.Subgraph[AdjacentAlertsHighlightSubgraphStartingNumber + SubgraphIndex].Name = SubgraphName;
        sc.Subgraph[AdjacentAlertsHighlightSubgraphStartingNumber + SubgraphIndex].PrimaryColor = RGB(255, 255,
0);
        sc.Subgraph[AdjacentAlertsHighlightSubgraphStartingNumber + SubgraphIndex].DrawStyle =
AdjacentAlertsDrawStyle;
        sc.Subgraph[AdjacentAlertsHighlightSubgraphStartingNumber + SubgraphIndex].LineWidth = 8;
        sc.Subgraph[AdjacentAlertsHighlightSubgraphStartingNumber +
SubgraphIndex].DisplayNameValueInWindowsFlags = 0;
    }

    Subgraph_ExtensionLineProperties.Name = "Extension Line Properties";
    Subgraph_ExtensionLineProperties.DrawStyle =
DRAWSTYLE_SUBGRAPH_NAME_AND_VALUE_LABELS_ONLY;
    Subgraph_ExtensionLineProperties.LineWidth = 1;

```

```

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

Input_ComparisonMethod.Name = "Comparison Method";
Input_ComparisonMethod.SetCustomInputStrings("Bid Volume;Ask Volume;Total Volume;Number of Trades;Ask
Volume Bid Volume Difference;Ask Volume Bid Volume Diagonal Difference;Ask Volume Bid Volume Ratio;Ask Volume
Bid Volume Diagonal Ratio;Bid Volume and Ask Volume Separately");
Input_ComparisonMethod.SetCustomInputIndex(2);

Input_VolumeThreshold.Name = "Volume Threshold";
Input_VolumeThreshold.SetInt(100);

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

Input_DrawExtensionLines.Name = "Draw Extension Lines";
// Input_DrawExtensionLines.SetYesNo(false);
Input_DrawExtensionLines.SetCustomInputStrings("None;All Alerts;Lowest Price in Adjacent Alerts;Highest Price in
Adjacent Alerts;All Prices in Adjacent Alerts");
Input_DrawExtensionLines.SetCustomInputIndex(0);

Input_DrawExtensionLinesWithTransparentRange.Name = "Draw Extension Lines With Transparent Range";
Input_DrawExtensionLinesWithTransparentRange.SetYesNo(false);

Input_PercentageThreshold.Name = "Percentage Threshold";
Input_PercentageThreshold.SetInt(150);

Input_AdditionalVolumeThreshold.Name = "Additional Volume Threshold";
Input_AdditionalVolumeThreshold.SetInt(100);

Input_AllowZeroValueCompares.Name = "Enable Zero Bid/Ask Compares";
Input_AllowZeroValueCompares.SetYesNo(0);

Input_DivideByZeroAction.Name = "Zero Value Compare Action";
Input_DivideByZeroAction.SetCustomInputStrings("Set 0 to 1;Set Percentage to +/- 1000%");
Input_DivideByZeroAction.SetCustomInputIndex(0);

Input_HighlightAdjacentAlertsGroupSize.Name = "Highlight Adjacent Alerts Minimum Group Size";
Input_HighlightAdjacentAlertsGroupSize.SetInt(0);

Input_DrawExtensionLinesUntilEndOfChart.Name = "Draw Extension Lines until End of Chart";
Input_DrawExtensionLinesUntilEndOfChart.SetYesNo(false);

Input_NumberOfDaysToCalculate.Name = "Number of Days to Calculate";
Input_NumberOfDaysToCalculate.SetInt(30);
Input_NumberOfDaysToCalculate.SetIntLimits(1, 10000);

Input_Version.SetInt(2);

sc.ValueFormat = VALUEFORMAT_INHERITED;

return;
}

if (Input_Version.GetInt() < 1)
{
    Input_Version.SetInt(1);
    Input_NumberOfDaysToCalculate.SetInt(30);
}

if (Input_Version.GetInt() < 2)
{
    Input_Version.SetInt(2);
    if (Input_DrawExtensionLines.GetYesNo() == false)

```

```

    Input_DrawExtensionLines.SetCustomInputIndex(0);
else
    Input_DrawExtensionLines.SetCustomInputIndex(1);
}

//This is an indication that the volume at price data does not exist
if (static_cast<int>(sc.VolumeAtPriceForBars->GetNumberOfBars()) < sc.ArraySize)
    return;

//The Subgraph display properties need to be the same for all Subgraphs. If the properties at Subgraph index 1 are
different than at subgraph index 0, then apply Subgraph 0 properties to the rest.

if (sc.Subgraph[1].PrimaryColor != sc.Subgraph[0].PrimaryColor
    || sc.Subgraph[1].DrawStyle != sc.Subgraph[0].DrawStyle
    || sc.Subgraph[1].LineWidth != sc.Subgraph[0].LineWidth
)
{
    for (int SubgraphIndex = 1; SubgraphIndex < SC_SUBGRAPHS_AVAILABLE - 13; ++SubgraphIndex)
    {
        sc.Subgraph[SubgraphIndex].PrimaryColor = sc.Subgraph[0].PrimaryColor;
        sc.Subgraph[SubgraphIndex].DrawStyle = sc.Subgraph[0].DrawStyle;
        sc.Subgraph[SubgraphIndex].LineWidth = sc.Subgraph[0].LineWidth;
    }
}

SCString AlertMessage;
bool EnableAlerts = sc.IsFullRecalculation == 0 && !sc.ChartIsDownloadingHistoricalData(sc.ChartNumber);

if (Input_HighlightAdjacentAlertsGroupSize.GetInt() == 0)
{
    for (int SubgraphIndex = 0; SubgraphIndex < 10; ++SubgraphIndex)
    {
        sc.Subgraph[AdjacentAlertsHighlightSubgraphStartingNumber + SubgraphIndex].DrawStyle =
DRAWSTYLE_IGNORE;
    }

    if (Input_DrawExtensionLines.GetIndex() > 1)
    {
        Input_DrawExtensionLines.SetCustomInputIndex(1);
    }
}
else if (Input_HighlightAdjacentAlertsGroupSize.GetInt() != 0)
{
    bool AllSetToIgnore = true;
    for (int SubgraphIndex = 0; SubgraphIndex < 10; ++SubgraphIndex)
    {
        if (sc.Subgraph[AdjacentAlertsHighlightSubgraphStartingNumber + SubgraphIndex].DrawStyle !=
DRAWSTYLE_IGNORE)
        {
            AllSetToIgnore = false;
            break;
        }
    }

    if (AllSetToIgnore)
    {
        for (int SubgraphIndex = 0; SubgraphIndex < 10; ++SubgraphIndex)
        {
            if (SubgraphIndex % 2 == 0)
            {
                sc.Subgraph[AdjacentAlertsHighlightSubgraphStartingNumber + SubgraphIndex].DrawStyle =
DRAWSTYLE_LEFT_OFFSET_BOX_TOP_FOR_CANDLESTICK;
            }
            else
            {

```

```

        sc.Subgraph[AdjacentAlertsHighlightSubgraphStartingNumber + SubgraphIndex].DrawStyle =
DRAWSTYLE_LEFT_OFFSET_BOX_BOTTOM_FOR_CANDLESTICK;
    }
}
}
}

```

```

const int HighlightAdjacentAlertsGroupSizeValue = Input_HighlightAdjacentAlertsGroupSize.GetInt();

```

```

SCDateTimeMS StartDateTimeForCalculations = sc.BaseDateTimeIn[sc.ArraySize - 1];
StartDateTimeForCalculations.SubtractDays(Input_NumberOfDaysToCalculate.GetInt());
StartDateTimeForCalculations = sc.GetStartOfPeriodForDateTime(StartDateTimeForCalculations,
TIME_PERIOD_LENGTH_UNIT_DAYS , 1, 0);

```

```

int LineID = 0;

```

```

for (int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize; BarIndex++)
{
    if (sc.BaseDateTimeIn[BarIndex] < StartDateTimeForCalculations)
    {
        continue;
    }
}

```

```

int AvailableSubgraphIndex = 0;

```

```

Subgraph_CountOfAlerts[BarIndex] = 0;

```

```

bool AdjacentAlertsHaveBottomHighlightAlertPrice = false;
int AdjacentAlertsHighlightBottomPriceIndex = 0;
bool AdjacentAlertsGotGroup = false;
int AdjacentAlertsGroupNumber = -1;
int AdjacentAlertsCountInGroup = 0;

```

```

//Reset all subgraph values

```

```

for (int SubgraphIndex = 0; SubgraphIndex < SC_SUBGRAPHS_AVAILABLE - 2; ++SubgraphIndex)
    sc.Subgraph[SubgraphIndex].Data[BarIndex] = 0;

```

```

bool GotExtensionLineForGroup = false;
float AdjacentAlertsHighlightBottomPrice = 0.0f;

```

```

int NumberOfPricesAtBarIndex = sc.VolumeAtPriceForBars->GetSizeAtBarIndex(BarIndex);

```

```

for (int PriceIndex = 0; PriceIndex < NumberOfPricesAtBarIndex; ++PriceIndex)
{
    s_VolumeAtPriceV2 *p_VolumeAtPrice = NULL;

    if (!sc.VolumeAtPriceForBars->GetVAPElementAtIndex(BarIndex, PriceIndex, &p_VolumeAtPrice))
        continue;

```

```

    s_VolumeAtPriceV2 *p_NextVolumeAtPrice = NULL;

```

```

    if (PriceIndex < NumberOfPricesAtBarIndex - 1)
    {
        sc.VolumeAtPriceForBars->GetVAPElementAtIndex(BarIndex, PriceIndex + 1, &p_NextVolumeAtPrice);
    }

```

```

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

```

```

// Check if condition has been met

```

```

int ComparisonMethodIndex = Input_ComparisonMethod.GetIndex();
bool ConditionMet = false;
unsigned int VolumeThresholdValue = Input_VolumeThreshold.GetInt();
unsigned int AdditionalVolumeThresholdValue = Input_AdditionalVolumeThreshold.GetInt();

```

```

if (ComparisonMethodIndex == 0)//Bid Volume
{
    if ((VolumeThresholdValue > 0
        && p_VolumeAtPrice->BidVolume >= VolumeThresholdValue)
        || (VolumeThresholdValue == 0 && p_VolumeAtPrice->BidVolume == 0)
        )
        ConditionMet = true;
}
else if (ComparisonMethodIndex == 1)//Ask Volume
{
    if (( VolumeThresholdValue > 0
        && p_VolumeAtPrice->AskVolume >= VolumeThresholdValue)
        || (VolumeThresholdValue == 0 && p_VolumeAtPrice->AskVolume == 0)
        )
        ConditionMet = true;
}
else if (ComparisonMethodIndex == 2)//Total Volume
{
    if ((VolumeThresholdValue > 0
        && p_VolumeAtPrice->Volume >= VolumeThresholdValue)
        || (VolumeThresholdValue == 0 && p_VolumeAtPrice->Volume == 0)
        )
        ConditionMet = true;
}
else if (ComparisonMethodIndex == 3)//Number of Trades
{
    if ((VolumeThresholdValue > 0
        && p_VolumeAtPrice->NumberOfTrades >= VolumeThresholdValue)
        || (VolumeThresholdValue == 0 && p_VolumeAtPrice->NumberOfTrades == 0)
        )
        ConditionMet = true;
}
else if (ComparisonMethodIndex == 4)//Ask Volume Bid Volume Difference
{
    int AskVolumeBidVolumeDifference = p_VolumeAtPrice->AskVolume - p_VolumeAtPrice->BidVolume;

    int VolumeThresholdSigned = Input_VolumeThreshold.GetInt();

    if (AskVolumeBidVolumeDifference > 0 && VolumeThresholdSigned > 0 && AskVolumeBidVolumeDifference
    >= VolumeThresholdSigned)
        ConditionMet = true;
    else if (AskVolumeBidVolumeDifference < 0 && VolumeThresholdSigned < 0 &&
    AskVolumeBidVolumeDifference <= VolumeThresholdSigned)
        ConditionMet = true;
}
else if (ComparisonMethodIndex == 5)//Ask Volume Bid Volume Diagonal Difference
{
    int AskVolumeBidVolumeDifference = 0;

    if (p_NextVolumeAtPrice != NULL)
    {
        AskVolumeBidVolumeDifference = p_NextVolumeAtPrice->AskVolume - p_VolumeAtPrice->BidVolume;
        if (p_NextVolumeAtPrice->AskVolume > p_VolumeAtPrice->BidVolume)
            PriceForExtensionLine = p_NextVolumeAtPrice->PriceInTicks * sc.TickSize;
    }

    int VolumeThresholdSigned = Input_VolumeThreshold.GetInt();

    if (AskVolumeBidVolumeDifference > 0 && VolumeThresholdSigned > 0 && AskVolumeBidVolumeDifference
    >= VolumeThresholdSigned)
        ConditionMet = true;
    else if (AskVolumeBidVolumeDifference < 0 && VolumeThresholdSigned < 0 &&
    AskVolumeBidVolumeDifference <= VolumeThresholdSigned)

```

```

        ConditionMet = true;
    }
    else if (ComparisonMethodIndex == 6)//Ask Volume Bid Volume Ratio
    {
        bool AllowZeroValueComparesSetting = Input_AllowZeroValueCompares.GetYesNo();
        unsigned int DivideByZeroActionIndex = Input_DivideByZeroAction.GetIndex();
        int AskVolumeBidVolumeRatioPercent = 0;

        if ((p_VolumeAtPrice->AskVolume > 0 && p_VolumeAtPrice->BidVolume > 0) ||
AllowZeroValueComparesSetting)
        {
            if (p_VolumeAtPrice->AskVolume >= p_VolumeAtPrice->BidVolume)
            {
                if (p_VolumeAtPrice->BidVolume == 0 && DivideByZeroActionIndex == 0)
                    AskVolumeBidVolumeRatioPercent = (p_VolumeAtPrice->AskVolume / 1) * 100;
                else if (p_VolumeAtPrice->BidVolume == 0 && DivideByZeroActionIndex == 1)
                    AskVolumeBidVolumeRatioPercent = 1000;
                else
                    AskVolumeBidVolumeRatioPercent = sc.Round((static_cast<float>(p_VolumeAtPrice->AskVolume) /
p_VolumeAtPrice->BidVolume) * 100);
            }
            else
            {
                if (p_VolumeAtPrice->AskVolume == 0 && DivideByZeroActionIndex == 0)
                    AskVolumeBidVolumeRatioPercent = (p_VolumeAtPrice->BidVolume / 1) * -100;
                else if (p_VolumeAtPrice->AskVolume == 0 && DivideByZeroActionIndex == 1)
                    AskVolumeBidVolumeRatioPercent = -1000;
                else
                    AskVolumeBidVolumeRatioPercent = sc.Round((static_cast<float>(p_VolumeAtPrice->BidVolume) /
p_VolumeAtPrice->AskVolume) * -100);
            }
        }

        int PercentThresholdSigned = Input_PercentageThreshold.GetInt();

        if (AskVolumeBidVolumeRatioPercent > 0 && PercentThresholdSigned > 0 &&
AskVolumeBidVolumeRatioPercent >= PercentThresholdSigned)
            ConditionMet = true;
        else if (AskVolumeBidVolumeRatioPercent < 0 && PercentThresholdSigned < 0 &&
AskVolumeBidVolumeRatioPercent <= PercentThresholdSigned)
            ConditionMet = true;
    }
    else if (ComparisonMethodIndex == 7)//Ask Volume Bid Volume Diagonal Ratio
    {
        bool AllowZeroValueComparesSetting = Input_AllowZeroValueCompares.GetYesNo();
        unsigned int DivideByZeroActionIndex = Input_DivideByZeroAction.GetIndex();
        int AskVolumeBidVolumeRatioPercent = 0;

        if (p_NextVolumeAtPrice != NULL)
        {
            if ((p_NextVolumeAtPrice->AskVolume >= p_VolumeAtPrice->BidVolume) && (p_VolumeAtPrice-
>BidVolume > 0 || AllowZeroValueComparesSetting))
            {
                if (p_VolumeAtPrice->BidVolume == 0 && DivideByZeroActionIndex == 0)
                    AskVolumeBidVolumeRatioPercent = (p_NextVolumeAtPrice->AskVolume / 1) * 100;
                else if (p_VolumeAtPrice->BidVolume == 0 && DivideByZeroActionIndex == 1)
                    AskVolumeBidVolumeRatioPercent = 1000;
                else
                    AskVolumeBidVolumeRatioPercent = sc.Round((static_cast<float>(p_NextVolumeAtPrice-
>AskVolume) / p_VolumeAtPrice->BidVolume) * 100);

                Price = p_NextVolumeAtPrice->PriceInTicks * sc.TickSize;
                PriceForExtensionLine = p_NextVolumeAtPrice->PriceInTicks * sc.TickSize;
            }
        }
    }
}

```



```

        else if (p_VolumeAtPrice->BidVolume > p_NextVolumeAtPrice->AskVolume && (p_NextVolumeAtPrice->AskVolume > 0 || AllowZeroValueComparesSetting))
        {
            if (p_NextVolumeAtPrice->AskVolume == 0 && DivideByZeroActionIndex == 0)
                AskVolumeBidVolumeRatioPercent = (p_VolumeAtPrice->BidVolume / 1) * -100;
            else if (p_NextVolumeAtPrice->AskVolume == 0 && DivideByZeroActionIndex == 1)
                AskVolumeBidVolumeRatioPercent = -1000;
            else
                AskVolumeBidVolumeRatioPercent = sc.Round((static_cast<float>(p_VolumeAtPrice->BidVolume) /
p_NextVolumeAtPrice->AskVolume) * -100);
        }
    }

    int PercentThresholdSigned = Input_PercentageThreshold.GetInt();

    if (AskVolumeBidVolumeRatioPercent > 0 && PercentThresholdSigned > 0 &&
AskVolumeBidVolumeRatioPercent >= PercentThresholdSigned)
        ConditionMet = true;
    else if (AskVolumeBidVolumeRatioPercent < 0 && PercentThresholdSigned < 0 &&
AskVolumeBidVolumeRatioPercent <= PercentThresholdSigned)
        ConditionMet = true;
}
else if (ComparisonMethodIndex == 8)//Bid Volume and Ask Volume Separately
{
    if (
        ((VolumeThresholdValue > 0
        && p_VolumeAtPrice->BidVolume >= VolumeThresholdValue)
        || (VolumeThresholdValue == 0 && p_VolumeAtPrice->BidVolume == 0))
        &&
        ((AdditionalVolumeThresholdValue > 0
        && p_VolumeAtPrice->AskVolume >= AdditionalVolumeThresholdValue)
        || (AdditionalVolumeThresholdValue == 0 && p_VolumeAtPrice->AskVolume == 0))
    )
        ConditionMet = true;
}

if (ConditionMet)
{
    Subgraph_CountOfAlerts[BarIndex]++;

    // Adjacent Alerts Highlight
    if (HighlightAdjacentAlertsGroupSizeValue > 0 && AdjacentAlertsGroupNumber < 5)
    {
        if (AdjacentAlertsHaveBottomHighlightAlertPrice && PriceIndex != AdjacentAlertsHighlightBottomPriceIndex
+ AdjacentAlertsCountInGroup)
        {
            AdjacentAlertsHaveBottomHighlightAlertPrice = false;
            AdjacentAlertsCountInGroup = 0;
            if (!AdjacentAlertsGotGroup)
            {
                AdjacentAlertsGroupNumber--;
            }
            AdjacentAlertsGotGroup = false;
            GotExtensionLineForGroup = false;
        }

        if (AdjacentAlertsHaveBottomHighlightAlertPrice == false)
        {
            AdjacentAlertsGroupNumber++;
            sc.Subgraph[AdjacentAlertsHighlightSubgraphStartingNumber + AdjacentAlertsGroupNumber *
2].Data[BarIndex] = Price;
            AdjacentAlertsHaveBottomHighlightAlertPrice = true;
            AdjacentAlertsCountInGroup++;
            AdjacentAlertsHighlightBottomPriceIndex = PriceIndex;
            AdjacentAlertsHighlightBottomPrice = Price; // Needed for extension lines

```



```

        if (AdjacentAlertsCountInGroup >= HighlightAdjacentAlertsGroupSizeValue)
        {
            sc.Subgraph[AdjacentAlertsHighlightSubgraphStartingNumber + AdjacentAlertsGroupNumber * 2 +
1].Data[BarIndex] = Price;
            AdjacentAlertsGotGroup = true;
        }

    }
    else if (AdjacentAlertsHaveBottomHighlightAlertPrice && PriceIndex ==
AdjacentAlertsHighlightBottomPriceIndex + AdjacentAlertsCountInGroup)
    {
        AdjacentAlertsCountInGroup++;
        if (AdjacentAlertsCountInGroup >= HighlightAdjacentAlertsGroupSizeValue)
        {
            sc.Subgraph[AdjacentAlertsHighlightSubgraphStartingNumber + AdjacentAlertsGroupNumber * 2 +
1].Data[BarIndex] = Price;
            AdjacentAlertsGotGroup = true;
        }
    }
    else if (AdjacentAlertsHaveBottomHighlightAlertPrice && !AdjacentAlertsGotGroup)
    {
        AdjacentAlertsCountInGroup = 0;
        AdjacentAlertsHaveBottomHighlightAlertPrice = false;
        sc.Subgraph[AdjacentAlertsHighlightSubgraphStartingNumber + AdjacentAlertsGroupNumber *
2].Data[BarIndex] = 0.0;
        AdjacentAlertsGroupNumber--;
    }
}

sc.Subgraph[AvailableSubgraphIndex][BarIndex] = Price;

// Extension Lines
if (Input_DrawExtensionLines.GetIndex() == 1 // All Alerts
    && sc.GetBarHasClosedStatus(BarIndex) == BHCS_BAR_HAS_CLOSED)
{
    n_ACSIL::s_LineUntilFutureIntersection LineUntilFutureIntersection;
    LineUntilFutureIntersection.StartBarIndex = BarIndex;
    LineUntilFutureIntersection.LineIDForBar = LineID;

    if (Input_DrawExtensionLinesWithTransparentRange.GetYesNo())
    {
        LineUntilFutureIntersection.LineValue = PriceForExtensionLine + sc.TickSize * 0.5f;
        LineUntilFutureIntersection.UseLineValue2 = true;
        LineUntilFutureIntersection.LineValue2ForRange = PriceForExtensionLine - sc.TickSize * 0.5f;
        LineUntilFutureIntersection.TransparencyLevel = sc.TransparencyLevel;
    }
    else
        LineUntilFutureIntersection.LineValue = PriceForExtensionLine;

    LineUntilFutureIntersection.LineColor = Subgraph_ExtensionLineProperties.PrimaryColor;
    LineUntilFutureIntersection.LineWidth = Subgraph_ExtensionLineProperties.LineWidth;
    LineUntilFutureIntersection.LineStyle = Subgraph_ExtensionLineProperties.LineStyle;

    if (Input_DrawExtensionLinesUntilEndOfChart.GetYesNo())
        LineUntilFutureIntersection.AlwaysExtendToEndOfChart = true;

    sc.AddLineUntilFutureIntersectionEx(LineUntilFutureIntersection);

    LineID++;
}
else if (Input_DrawExtensionLines.GetIndex() == 2 // Lowest Price in Adjacent Alerts
    && AdjacentAlertsGotGroup
    && !GotExtensionLineForGroup
    && sc.GetBarHasClosedStatus(BarIndex) == BHCS_BAR_HAS_CLOSED)

```

```

{
    GotExtensionLineForGroup = true;
    PriceForExtensionLine = AdjacentAlertsHighlightBottomPrice;

    n_ACSIL::s_LineUntilFutureIntersection LineUntilFutureIntersection;
    LineUntilFutureIntersection.StartBarIndex = BarIndex;
    LineUntilFutureIntersection.LineIDForBar = LineID;

    if (Input_DrawExtensionLinesWithTransparentRange.GetYesNo())
    {
        LineUntilFutureIntersection.LineValue = PriceForExtensionLine + sc.TickSize * 0.5f;
        LineUntilFutureIntersection.UseLineValue2 = true;
        LineUntilFutureIntersection.LineValue2ForRange = PriceForExtensionLine - sc.TickSize * 0.5f;
        LineUntilFutureIntersection.TransparencyLevel = sc.TransparencyLevel;
    }
    else
        LineUntilFutureIntersection.LineValue = PriceForExtensionLine;

    LineUntilFutureIntersection.LineColor = Subgraph_ExtensionLineProperties.PrimaryColor;
    LineUntilFutureIntersection.LineWidth = Subgraph_ExtensionLineProperties.LineWidth;
    LineUntilFutureIntersection.LineStyle = Subgraph_ExtensionLineProperties.LineStyle;

    if (Input_DrawExtensionLinesUntilEndOfChart.GetYesNo())
        LineUntilFutureIntersection.AlwaysExtendToEndOfChart = true;

    sc.AddLineUntilFutureIntersectionEx(LineUntilFutureIntersection);

    LineID++;
}
else if (Input_DrawExtensionLines.GetIndex() == 3 // Highest Price in Adjacent Alerts
    && AdjacentAlertsGotGroup
    && sc.GetBarHasClosedStatus(BarIndex) == BHCS_BAR_HAS_CLOSED)
{
    if (!GotExtensionLineForGroup) // Have a group, but we don't know if it is the highest point yet
    {
        GotExtensionLineForGroup = true;
        PriceForExtensionLine = Price;

        n_ACSIL::s_LineUntilFutureIntersection LineUntilFutureIntersection;
        LineUntilFutureIntersection.StartBarIndex = BarIndex;
        LineUntilFutureIntersection.LineIDForBar = LineID;

        if (Input_DrawExtensionLinesWithTransparentRange.GetYesNo())
        {
            LineUntilFutureIntersection.LineValue = PriceForExtensionLine + sc.TickSize * 0.5f;
            LineUntilFutureIntersection.UseLineValue2 = true;
            LineUntilFutureIntersection.LineValue2ForRange = PriceForExtensionLine - sc.TickSize * 0.5f;
            LineUntilFutureIntersection.TransparencyLevel = sc.TransparencyLevel;
        }
        else
            LineUntilFutureIntersection.LineValue = PriceForExtensionLine;

        LineUntilFutureIntersection.LineColor = Subgraph_ExtensionLineProperties.PrimaryColor;
        LineUntilFutureIntersection.LineWidth = Subgraph_ExtensionLineProperties.LineWidth;
        LineUntilFutureIntersection.LineStyle = Subgraph_ExtensionLineProperties.LineStyle;

        if (Input_DrawExtensionLinesUntilEndOfChart.GetYesNo())
            LineUntilFutureIntersection.AlwaysExtendToEndOfChart = true;

        sc.AddLineUntilFutureIntersectionEx(LineUntilFutureIntersection);

        LineID++;
    }
    else // Have a new highest point, so remove previous line and add a new one.
    {

```

```

LineID--;
sc.DeleteLineUntilFutureIntersection(BarIndex, LineID);

PriceForExtensionLine = Price;

n_ACSIL::s_LineUntilFutureIntersection LineUntilFutureIntersection;
LineUntilFutureIntersection.StartBarIndex = BarIndex;
LineUntilFutureIntersection.LineIDForBar = LineID;

if (Input_DrawExtensionLinesWithTransparentRange.GetYesNo())
{
    LineUntilFutureIntersection.LineValue = PriceForExtensionLine + sc.TickSize * 0.5f;
    LineUntilFutureIntersection.UseLineValue2 = true;
    LineUntilFutureIntersection.LineValue2ForRange = PriceForExtensionLine - sc.TickSize * 0.5f;
    LineUntilFutureIntersection.TransparencyLevel = sc.TransparencyLevel;
}
else
    LineUntilFutureIntersection.LineValue = PriceForExtensionLine;

LineUntilFutureIntersection.LineColor = Subgraph_ExtensionLineProperties.PrimaryColor;
LineUntilFutureIntersection.LineWidth = Subgraph_ExtensionLineProperties.LineWidth;
LineUntilFutureIntersection.LineStyle = Subgraph_ExtensionLineProperties.LineStyle;

if (Input_DrawExtensionLinesUntilEndOfChart.GetYesNo())
    LineUntilFutureIntersection.AlwaysExtendToEndOfChart = true;

sc.AddLineUntilFutureIntersectionEx(LineUntilFutureIntersection);

LineID++;
}
}
else if (Input_DrawExtensionLines.GetIndex() == 4 // All Prices in Adjacent Alerts
    && sc.GetBarHasClosedStatus(BarIndex) == BHCS_BAR_HAS_CLOSED)
{
    if (AdjacentAlertsHaveBottomHighlightAlertPrice && !GotExtensionLineForGroup &&
        AdjacentAlertsGotGroup) // Create lines from bottom to entered adjacent value
    {
        GotExtensionLineForGroup = true;

        for (int PriceIndex = 0; PriceIndex < AdjacentAlertsCountInGroup; PriceIndex++)
        {
            PriceForExtensionLine = AdjacentAlertsHighlightBottomPrice + PriceIndex * sc.TickSize;

            n_ACSIL::s_LineUntilFutureIntersection LineUntilFutureIntersection;
            LineUntilFutureIntersection.StartBarIndex = BarIndex;
            LineUntilFutureIntersection.LineIDForBar = LineID;

            if (Input_DrawExtensionLinesWithTransparentRange.GetYesNo())
            {
                LineUntilFutureIntersection.LineValue = PriceForExtensionLine + sc.TickSize * 0.5f;
                LineUntilFutureIntersection.UseLineValue2 = true;
                LineUntilFutureIntersection.LineValue2ForRange = PriceForExtensionLine - sc.TickSize * 0.5f;
                LineUntilFutureIntersection.TransparencyLevel = sc.TransparencyLevel;
            }
            else
                LineUntilFutureIntersection.LineValue = PriceForExtensionLine;

            LineUntilFutureIntersection.LineColor = Subgraph_ExtensionLineProperties.PrimaryColor;
            LineUntilFutureIntersection.LineWidth = Subgraph_ExtensionLineProperties.LineWidth;
            LineUntilFutureIntersection.LineStyle = Subgraph_ExtensionLineProperties.LineStyle;

            if (Input_DrawExtensionLinesUntilEndOfChart.GetYesNo())
                LineUntilFutureIntersection.AlwaysExtendToEndOfChart = true;

            sc.AddLineUntilFutureIntersectionEx(LineUntilFutureIntersection);

```



```

    return;
}

// Do data processing

// Make chart a continuous futures contract chart. This must be supported by the symbol for it to have an effect. This
option will not go into effect until returning from the study function, at that time the chart will be reloaded.
if (sc.ContinuousFuturesContractOption != CFCO_DATE_RULE_ROLLOVER)
{
    sc.ContinuousFuturesContractOption = CFCO_DATE_RULE_ROLLOVER;
}
}

/*=====*/
SCSFExport scsf_RelativeVolume(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_RelativeVolume = sc.Subgraph[0];
    SCSubgraphRef Subgraph_CurrentDayCumulativeVolumeToAverageCumulativeVolumePriorDaysRatio =
sc.Subgraph[1];
    SCSubgraphRef Subgraph_HundredLine = sc.Subgraph[2];
    SCSubgraphRef Subgraph_AverageVolume = sc.Subgraph[3];
    SCSubgraphRef Subgraph_AverageCumulativeVolume = sc.Subgraph[4];
    SCSubgraphRef Subgraph_CurrentDayCumulativeVolume = sc.Subgraph[5];
    SCSubgraphRef Subgraph_VolumeAverageVolumeDiff = sc.Subgraph[6];
    SCSubgraphRef Subgraph_CumulativeVolumeAverageCumulativeVolumeDiff = sc.Subgraph[7];

    SCFloatArrayRef Array_TimespanForDay = sc.Subgraph[0].Arrays[0];
    SCFloatArrayRef Array_AverageVolume =
Subgraph_CurrentDayCumulativeVolumeToAverageCumulativeVolumePriorDaysRatio.Arrays[0];
    SCFloatArrayRef Array_AverageCumulativeVolumeForPriorDays =
Subgraph_CurrentDayCumulativeVolumeToAverageCumulativeVolumePriorDaysRatio.Arrays[1];
    SCFloatArrayRef Array_MedianVolumeTemp =
Subgraph_CurrentDayCumulativeVolumeToAverageCumulativeVolumePriorDaysRatio.Arrays[2];
    SCFloatArrayRef Array_MedianCumulativeVolumeTemp =
Subgraph_CurrentDayCumulativeVolumeToAverageCumulativeVolumePriorDaysRatio.Arrays[3];
    SCFloatArrayRef Array_MedianTemp =
Subgraph_CurrentDayCumulativeVolumeToAverageCumulativeVolumePriorDaysRatio.Arrays[4];

    SCInputRef Input_Period = sc.Input[0];
    SCInputRef Input_AccumulateType = sc.Input[1];
    SCInputRef Input_DaySessionOnly = sc.Input[2];
    SCInputRef Input_VolumeAvergeType = sc.Input[3];
    SCInputRef Input_VolumeSubgraphToUse = sc.Input[4];
    SCInputRef Input_TimeComparisonTolerance = sc.Input[5];
    SCInputRef Input_MinimumRequiredDataForDayAsPercentage = sc.Input[6];
    SCInputRef Input_HighPercentThresh = sc.Input[8];
    SCInputRef Input_HighThresholdColor = sc.Input[9];
    SCInputRef Input_LowPercentThresh = sc.Input[10];
    SCInputRef Input_LowThresholdColor = sc.Input[11];
    SCInputRef Input_Version = sc.Input[13];
    // SCInputRef Input_DisplayAsVolumeAverageVolumeDiff = sc.Input[14];
    // SCInputRef Input_DisplayCumulativeVolumeAverageCumulativeVolumeDiff = sc.Input[15];

    const static int CURRENT_VERSION = 4;

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

        sc.GraphRegion = 1;
        sc.ValueFormat = 1;
        sc.DrawZeros = 0;
    }
}

```

```

sc.AutoLoop = 0;//manual looping
sc.MaintainAdditionalChartDataArrays = 1;// needed to calculate the time span of the bar accurately.

Subgraph_RelativeVolume.Name = "Relative Volume";
Subgraph_RelativeVolume.DrawStyle = DRAWSTYLE_BAR;
Subgraph_RelativeVolume.PrimaryColor = RGB(0, 128, 192);
Subgraph_RelativeVolume.LineWidth = 3;

Subgraph_CurrentDayCumulativeVolumeToAverageCumulativeVolumePriorDaysRatio.Name = "Cumulative Volume
Ratio";
Subgraph_CurrentDayCumulativeVolumeToAverageCumulativeVolumePriorDaysRatio.DrawStyle =
DRAWSTYLE_LINE_SKIP_ZEROS;
Subgraph_CurrentDayCumulativeVolumeToAverageCumulativeVolumePriorDaysRatio.PrimaryColor = RGB(255,
128, 0);
Subgraph_CurrentDayCumulativeVolumeToAverageCumulativeVolumePriorDaysRatio.LineWidth = 2;

Subgraph_AverageVolume.Name = "Average Volume";
Subgraph_AverageVolume.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_AverageVolume.PrimaryColor = RGB(0, 128, 255);
Subgraph_AverageVolume.LineWidth = 1;

Subgraph_AverageCumulativeVolume.Name = "Average Cumulative Volume";
Subgraph_AverageCumulativeVolume.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_AverageCumulativeVolume.PrimaryColor = COLOR_GRAY;
Subgraph_AverageCumulativeVolume.LineWidth = 1;

Subgraph_HundredLine.Name = "100%";
Subgraph_HundredLine.DrawStyle = DRAWSTYLE_LINE;
Subgraph_HundredLine.LineStyle = LINESTYLE_DOT;
Subgraph_HundredLine.PrimaryColor = COLOR_GRAY;

Subgraph_CurrentDayCumulativeVolume.Name = "Cumulative Volume";
Subgraph_CurrentDayCumulativeVolume.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_CurrentDayCumulativeVolume.PrimaryColor = COLOR_GRAY;

Subgraph_VolumeAverageVolumeDiff.Name = "Volume Average Volume Difference";
Subgraph_VolumeAverageVolumeDiff.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_VolumeAverageVolumeDiff.LineWidth = 2;
Subgraph_VolumeAverageVolumeDiff.PrimaryColor = RGB(0, 200, 0);

Subgraph_CumulativeVolumeAverageCumulativeVolumeDiff.Name = "Cum Volume Average Cum Volume
Difference";
Subgraph_CumulativeVolumeAverageCumulativeVolumeDiff.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_CumulativeVolumeAverageCumulativeVolumeDiff.LineWidth = 2;
Subgraph_CumulativeVolumeAverageCumulativeVolumeDiff.PrimaryColor = RGB(0, 0, 200);

Input_Period.Name = "Period - Days To Include";
Input_Period.SetInt(5);
Input_Period.SetIntLimits(1, INT_MAX);

Input_AccumulateType.Name = "Accumulation Type";
Input_AccumulateType.SetCustomInputStrings("All Days;Day Of Week");
Input_AccumulateType.SetCustomInputIndex(0);

Input_DaySessionOnly.Name = "Day Session Only";
Input_DaySessionOnly.SetYesNo(1);

Input_VolumeAvergeType.Name = "Volume Average Type";
Input_VolumeAvergeType.SetCustomInputStrings("Average Volume;Median Volume");
Input_VolumeAvergeType.SetCustomInputIndex(0);

Input_VolumeSubgraphToUse.Name = "Volume Subgraph To Use";
Input_VolumeSubgraphToUse.SetInputDataIndex(SC_VOLUME);

Input_TimeComparisonTolerance.Name = "Time Comparison Tolerance in Minutes";

```

```

Input_TimeComparisonTolerance.SetInt(1);
Input_TimeComparisonTolerance.SetIntLimits(0, 240);

Input_MinimumRequiredDataForDayAsPercentage.Name = "Minimum Required Data for Day as Percentage";
Input_MinimumRequiredDataForDayAsPercentage.SetInt(50);
Input_MinimumRequiredDataForDayAsPercentage.SetIntLimits(0, 100);

Input_HighPercentThresh.Name = "High Percent Threshold (0-disabled, 120->120%)";
Input_HighPercentThresh.SetFloat(120);
Input_HighPercentThresh.SetFloatLimits(0, FLT_MAX);

Input_HighThresholdColor.Name = "High Threshold Color";
Input_HighThresholdColor.SetColor(COLOR_GREEN);

Input_LowPercentThresh.Name = "Low Percent Threshold (0-disabled, 80->80%)";
Input_LowPercentThresh.SetFloat(80);
Input_LowPercentThresh.SetFloatLimits(0, FLT_MAX);

Input_LowThresholdColor.Name = "Low Threshold Color";
Input_LowThresholdColor.SetColor(COLOR_RED);

// Input_DisplayAsVolumeAverageVolumeDiff.Name = "Display As Volume Average Volume Difference";
// Input_DisplayAsVolumeAverageVolumeDiff.SetYesNo(false);
//
// Input_DisplayCumulativeVolumeAverageCumulativeVolumeDiff.Name = "Display Cum Volume Average Cum
Volume Difference";
// Input_DisplayCumulativeVolumeAverageCumulativeVolumeDiff.SetYesNo(false);

Input_Version.SetInt(CURRENT_VERSION);

return;
}

if (Input_Version.GetInt() < 2)
{
    Input_VolumeAvergeType.SetCustomInputIndex(0);
    Subgraph_AverageVolume.DrawStyle = DRAWSTYLE_IGNORE;
    Subgraph_AverageCumulativeVolume.DrawStyle = DRAWSTYLE_IGNORE;

    Input_Version.SetInt(CURRENT_VERSION);
}

if (Input_Version.GetInt() < 3)
{
    Input_VolumeSubgraphToUse.SetInputDataIndex(SC_VOLUME);
    Input_Version.SetInt(CURRENT_VERSION);
}

if (Input_Version.GetInt() < 4)
{
    Input_MinimumRequiredDataForDayAsPercentage.SetInt(50);
    Input_Version.SetInt(CURRENT_VERSION);
}

int VolumeSubgraphIndex = Input_VolumeSubgraphToUse.GetInputDataIndex();

// References to persistent variables
int& LastIndex = sc.GetPersistentInt(1);

if (sc.LastCallToFunction)
{
    return;
}

if (sc.IsFullRecalculation && sc.UpdateStartIndex == 0)

```



```

{
    sc.ClearAllPersistentData();

    LastIndex = 0;

    Subgraph_CurrentDayCumulativeVolume[0] = sc.BaseData[VolumeSubgraphIndex][0];

    // if (Input_DisplayAsVolumeAverageVolumeDiff.GetYesNo() ||
    Input_DisplayCumulativeVolumeAverageCumulativeVolumeDiff.GetYesNo())
    // {
    //     if (Input_DisplayAsVolumeAverageVolumeDiff.GetYesNo() && (Subgraph_VolumeAverageVolumeDiff.DrawStyle
    == DRAWSTYLE_IGNORE
    //         || Subgraph_VolumeAverageVolumeDiff.DrawStyle == DRAWSTYLE_HIDDEN))
    //         Subgraph_VolumeAverageVolumeDiff.DrawStyle = DRAWSTYLE_LINE;
    //
    //     if (Input_DisplayCumulativeVolumeAverageCumulativeVolumeDiff.GetYesNo() &&
    (Subgraph_CumulativeVolumeAverageCumulativeVolumeDiff.DrawStyle == DRAWSTYLE_IGNORE
    //         || Subgraph_CumulativeVolumeAverageCumulativeVolumeDiff.DrawStyle == DRAWSTYLE_HIDDEN))
    //         Subgraph_CumulativeVolumeAverageCumulativeVolumeDiff.DrawStyle = DRAWSTYLE_LINE;
    //
    //     RelativeVolume.DrawStyle = DRAWSTYLE_IGNORE;
    //     CurrentDayCumulativeVolumeToAverageCumulativeVolumePriorDaysRatio.DrawStyle = DRAWSTYLE_IGNORE;
    //     HundredLine.DrawStyle = DRAWSTYLE_IGNORE;
    // }
    // else
    // {
    //     if (!Input_DisplayAsVolumeAverageVolumeDiff.GetYesNo())
    //         Subgraph_VolumeAverageVolumeDiff.DrawStyle = DRAWSTYLE_IGNORE;
    //
    //     if (!Input_DisplayCumulativeVolumeAverageCumulativeVolumeDiff.GetYesNo())
    //         Subgraph_CumulativeVolumeAverageCumulativeVolumeDiff.DrawStyle = DRAWSTYLE_IGNORE;
    //
    //     RelativeVolume.DrawStyle = DRAWSTYLE_BAR;
    //     CurrentDayCumulativeVolumeToAverageCumulativeVolumePriorDaysRatio.DrawStyle =
    DRAWSTYLE_LINE_SKIP_ZEROS;
    //     HundredLine.DrawStyle = DRAWSTYLE_LINE;
    // }
}

enum TradingDateSufficientDataEnum
{
    SufficientData_Unset = 0
    , SufficientData_True = 1
    , SufficientData_False = 2
};

for (int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize; BarIndex++)
{
    SCDateTime BarDateTime = sc.BaseDateTimeIn[BarIndex];

    Subgraph_HundredLine[BarIndex] = 100.0f;

    bool DrawValuesAtIndex = !Input_DaySessionOnly.GetYesNo() || sc.IsDateTimeInDaySession(BarDateTime);

    const int CurrentBarTradeDate = sc.GetTradingDayDate(BarDateTime);
    const int PriorBarTradeDate = sc.GetTradingDayDate(sc.BaseDateTimeIn[BarIndex - 1]);
    const int NextBarTradeDate = sc.GetTradingDayDate(sc.BaseDateTimeIn[BarIndex + 1]);

    bool IsNewDay = BarIndex == 0 || CurrentBarTradeDate != PriorBarTradeDate;

    Array_TimespanForDay[BarIndex] = static_cast<float>(sc.TimeSpanOfBar(BarIndex).GetAsDouble());

    if (BarIndex >= 1 && !IsNewDay)
        Array_TimespanForDay[BarIndex] += Array_TimespanForDay[BarIndex - 1];
}

```



```

if (CurrentBarTradeDate != NextBarTradeDate)
{
    int& SufficientData = sc.GetPersistentInt(CurrentBarTradeDate);

    if (SufficientData == SufficientData_Unset)
    {
        if (Array_TimespanForDay[BarIndex] >= (Input_MinimumRequiredDataForDayAsPercentage.GetInt() * 0.01))
        {
            SufficientData = SufficientData_True;
        }
        else
        {
            SufficientData = SufficientData_False;
#ifdef _DEBUG
            SCString MessageText;
            SCString BarDateString = sc.DateTimeToString(CurrentBarTradeDate, FLAG_DT_COMPLETE_DATE);
            MessageText.Format("Date %s has insufficient data.", BarDateString.GetChars());
            sc.AddMessageToLog(MessageText, 0);
#endif
        }
    }
}

// track cumulative volume
if (!DrawValuesAtIndex)
    Subgraph_CurrentDayCumulativeVolume[BarIndex] = 0;
else if (CurrentBarTradeDate != PriorBarTradeDate)
    Subgraph_CurrentDayCumulativeVolume[BarIndex] = sc.BaseData[VolumeSubgraphIndex][BarIndex];
else
    Subgraph_CurrentDayCumulativeVolume[BarIndex] = Subgraph_CurrentDayCumulativeVolume[BarIndex - 1] +
sc.BaseData[VolumeSubgraphIndex][BarIndex];

bool CalculateMedian = Input_VolumeAvergeType.GetIndex() == 1;

if (BarIndex != LastIndex)
{
    float SumOfBarVol = 0;
    float SumOfAccumVol = 0;
    int NumValues = 0;

    SCDateTime PriorDateTime = BarDateTime;
    const int NumDaysToCalc = Input_Period.GetInt();
    int DaysBack = 0;

    for (; DaysBack < NumDaysToCalc; ++DaysBack)
    {
        int PriorIndex = -1;
        int SkipCount = 0;
        while (SkipCount <= 3)
        {
            if (Input_AccumulateType.GetIndex() == 0) // All days
            {
                PriorDateTime.SubtractDays(1);
            }
            else // Day of week
                PriorDateTime.SubtractDays(7);

            SCDateTime PriorTradingDayDate = sc.GetTradingDayDate(PriorDateTime);

            int& SufficientData = sc.GetPersistentInt(PriorTradingDayDate.GetDate());

            if (SufficientData == SufficientData_False || SufficientData == SufficientData_Unset)
            {
                SkipCount++;
            }
        }
    }
}

```

```

        continue;
    }

    break;
}

PriorIndex = sc.GetContainingIndexForSCDateTime(sc.ChartNumber, PriorDateTime.GetAsDouble());

//If the found Date-Time is within the time comparison tolerance of what is expected, then use it
if (SCDateTime(sc.BaseDateTimeln[PriorIndex] - PriorDateTime).GetAbsoluteValue() >
SCDateTime::MINUTES(Input_TimeComparisonTolerance.GetInt()))
{
    continue;
}

SumOfBarVol += sc.BaseData[VolumeSubgraphIndex][PriorIndex];
SumOfAccumVol += Subgraph_CurrentDayCumulativeVolume[PriorIndex];

if (CalculateMedian && BarIndex >= NumValues)
{
    Array_MedianVolumeTemp[BarIndex - NumValues] = sc.BaseData[VolumeSubgraphIndex][PriorIndex];
    Array_MedianCumulativeVolumeTemp[BarIndex - NumValues] =
Subgraph_CurrentDayCumulativeVolume[PriorIndex];
}

NumValues++;

if (PriorIndex == 0)
    break;
}

if (NumValues > 0 && DaysBack >= NumDaysToCalc - 1)
{
    if (CalculateMedian)
    {
        MovingMedian_S(Array_MedianVolumeTemp, Array_AverageVolume, Array_MedianTemp, BarIndex,
NumValues);
        MovingMedian_S(Array_MedianCumulativeVolumeTemp, Array_AverageCumulativeVolumeForPriorDays,
Array_MedianTemp, BarIndex, NumValues);
    }
    else
    {
        Array_AverageVolume[BarIndex] = SumOfBarVol / NumValues;
        Array_AverageCumulativeVolumeForPriorDays[BarIndex] = SumOfAccumVol / NumValues;
    }
}

LastIndex = BarIndex;
}

if (DrawValuesAtIndex)
{
    // express relative and average accumulated volume as percentages
    if (Array_AverageVolume[BarIndex] != 0)
        Subgraph_RelativeVolume[BarIndex] = (sc.BaseData[VolumeSubgraphIndex][BarIndex] /
Array_AverageVolume[BarIndex]) * 100.0f;

    if (Array_AverageCumulativeVolumeForPriorDays[BarIndex] != 0)
    {
        Subgraph_CurrentDayCumulativeVolumeToAverageCumulativeVolumePriorDaysRatio[BarIndex] =
(Subgraph_CurrentDayCumulativeVolume[BarIndex] / Array_AverageCumulativeVolumeForPriorDays[BarIndex]) * 100.0f;
    }
}

```

```

Subgraph_AverageVolume[BarIndex] = Array_AverageVolume[BarIndex];
Subgraph_AverageCumulativeVolume[BarIndex] = Array_AverageCumulativeVolumeForPriorDays[BarIndex];
Subgraph_VolumeAverageVolumeDiff[BarIndex] = sc.Volume[BarIndex] - Subgraph_AverageVolume[BarIndex];
Subgraph_CumulativeVolumeAverageCumulativeVolumeDiff[BarIndex] =
Subgraph_CurrentDayCumulativeVolume[BarIndex] - Subgraph_AverageCumulativeVolume[BarIndex];

// finally color bars
if (Subgraph_RelativeVolume[BarIndex] <= Input_LowPercentThresh.GetFloat() &&
Input_LowPercentThresh.GetFloat() != 0)
    Subgraph_RelativeVolume.DataColor[BarIndex] = Input_LowThresholdColor.GetColor();
else if (Subgraph_RelativeVolume[BarIndex] >= Input_HighPercentThresh.GetFloat() &&
Input_HighPercentThresh.GetFloat() != 0)
    Subgraph_RelativeVolume.DataColor[BarIndex] = Input_HighThresholdColor.GetColor();
else
    Subgraph_RelativeVolume.DataColor[BarIndex] = Subgraph_RelativeVolume.PrimaryColor;
}

}
}
/*=====*/
SCSFExport scsf_ForecastOscillator(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_ForecastOscillator = sc.Subgraph[0];
    SCSubgraphRef Subgraph__ZeroLine = sc.Subgraph[1];

    SCFloatArrayRef Array_Avg = Subgraph_ForecastOscillator.Arrays[0];

    SCInputRef Input_Data = sc.Input[0];
    SCInputRef Input_Period = sc.Input[1];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Chande Forecast Oscillator";

        sc.ValueFormat = 2;
        sc.DrawZeros = 1;
        sc.AutoLoop = 1;

        // subgraphs
        Subgraph_ForecastOscillator.Name = "Chande Forecast Oscillator";
        Subgraph_ForecastOscillator.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_ForecastOscillator.PrimaryColor = COLOR_GREEN;
        Subgraph_ForecastOscillator.LineWidth = 2;

        Subgraph__ZeroLine.Name = "Zero";
        Subgraph__ZeroLine.DrawStyle = DRAWSTYLE_LINE;
        Subgraph__ZeroLine.LineStyle = LINESTYLE_DOT;
        Subgraph__ZeroLine.PrimaryColor = COLOR_DARKGRAY;
        Subgraph__ZeroLine.DisplayNameValueInWindowsFlags = 0;

        // inputs
        Input_Data.Name = "Input Data";
        Input_Data.SetInputDataIndex(SC_LAST);

        Input_Period.Name = "Period";
        Input_Period.SetInt(14);
        Input_Period.SetIntLimits(1, MAX_STUDY_LENGTH);

        return;
    }

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

```

```

SCFloatArrayRef BaseDataArray = sc.BaseDataIn[Input_Data.GetInputDataIndex()];

sc.LinearRegressionIndicator(BaseDataArray, Array_Avg, Input_Period.GetInt());

float InputValue = BaseDataArray[sc.Index];

if (InputValue != 0)
    Subgraph_ForecastOscillator[sc.Index] = 100.0f * ((InputValue - Array_Avg[sc.Index]) / InputValue);

Subgraph__ZeroLine[sc.Index] = 0.0f;
}

/*=====*/
SCSFExport scsf_DetectingNewBarsAdded(SCStudyInterfaceRef sc)
{
    if (sc.SetDefaults)
    {
        // Set the configuration and defaults
        sc.GraphName = "Detecting New Bars Added";
        sc.AutoLoop = 1;

        return;
    }

    // Do data processing
    int& PriorArraySize = sc.GetPersistentInt(1);

    //This is not a full recalculation of the study
    if (!sc.IsFullRecalculation)
    {
        // If there are new bars added
        if (PriorArraySize < sc.ArraySize)
        {
            // put processing here that is required for when new bars are added to the chart
        }
    }

    PriorArraySize = sc.ArraySize;
}

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

    SCFloatArrayRef Array_TextPosition = Subgraph_DeltaValue.Arrays[0];

    SCInputRef Input_OffsetInTicks = sc.Input[1];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Bar Delta Below Bar";

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

        sc.HideDLLAndFunctionNames = 1;

        // subgraphs
        Subgraph_DeltaValue.Name = "Bar Delta";
        Subgraph_DeltaValue.DrawStyle = DRAWSTYLE_CUSTOM_VALUE_AT_Y;
    }
}

```

```

Subgraph_DeltaValue.PrimaryColor = COLOR_GREEN;
Subgraph_DeltaValue.SecondaryColor = COLOR_RED;
Subgraph_DeltaValue.SecondaryColorUsed = 1;
Subgraph_DeltaValue.AutoColoring = AUTOCOLOR_POSNEG;

Input_OffsetInTicks.Name = "Offset In Ticks";
Input_OffsetInTicks.SetInt(1);

return;
}

Subgraph_DeltaValue[sc.Index] = sc.AskVolume[sc.Index] - sc.BidVolume[sc.Index];

Array_TextPosition[sc.Index] = sc.Low[sc.Index] - (Input_OffsetInTicks.GetInt() * sc.TickSize); // top of text
}

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

    SCFloatArrayRef Array_TextPosition = Subgraph_Difference.Arrays[0];

    SCInputRef Input_DisplayAboveOrBelow = sc.Input[0];
    SCInputRef Input_OffsetInTicks = sc.Input[1];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Ask Volume Bid Volume Difference Text";

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

        Subgraph_Difference.Name = "Difference";
        Subgraph_Difference.DrawStyle = DRAWSTYLE_CUSTOM_VALUE_AT_Y;
        Subgraph_Difference.PrimaryColor = RGB(0,128,0);
        Subgraph_Difference.SecondaryColor = RGB(128,0,0);
        Subgraph_Difference.SecondaryColorUsed = 1;
        Subgraph_Difference.AutoColoring = AUTOCOLOR_POSNEG;
        Subgraph_Difference.LineWidth = 8;

        Input_DisplayAboveOrBelow.Name = "Display Location";
        Input_DisplayAboveOrBelow.SetCustomInputStrings("Top of Bar;Bottom of Bar");
        Input_DisplayAboveOrBelow.SetCustomInputIndex(0);

        Input_OffsetInTicks.Name = "Offset In Ticks";
        Input_OffsetInTicks.SetInt(1);

        return;
    }

    Subgraph_Difference[sc.Index] = sc.AskVolume[sc.Index] - sc.BidVolume[sc.Index];

    if (Input_DisplayAboveOrBelow.GetIndex() == 0)
        Array_TextPosition[sc.Index] = sc.High[sc.Index] + (Input_OffsetInTicks.GetInt() * sc.TickSize);
    else
        Array_TextPosition[sc.Index] = sc.Low[sc.Index] - (Input_OffsetInTicks.GetInt() * sc.TickSize);
}

/*=====*/
SCSFExport scsf_LoadDaySessionOnlyAtOpen(SCStudyInterfaceRef sc)
{
    if (sc.SetDefaults)

```

```

{
    // Set the configuration and defaults

    sc.GraphName = "Load Day Session Only at Open";

    sc.StudyDescription = "If the evening session option is enabled for a chart, then when the Start Time is crossed over
using the actual time, then the evening session is disabled and the chart is reloaded.";

    sc.AutoLoop = 0;

    return;
}

// Do data processing
SCDateTime& r_PreviousCurrentTime = sc.GetPersistentSCDateTime(1);

if (sc.IsFullRecalculation)
    r_PreviousCurrentTime = 0.0;

SCDateTime CurrentDateTime;

if (sc.IsReplayRunning())
    CurrentDateTime = sc.CurrentDateTimeForReplay;
else
    CurrentDateTime = sc.CurrentSystemDateTime;

// This code does not consider if the Start Time is 00:00:00. It is assumed to be greater than that.
if (r_PreviousCurrentTime != 0.0
    && sc.UseSecondStartEndTimes
    && CurrentDateTime.GetTimeInSeconds() >= sc.StartTime1
    && r_PreviousCurrentTime.GetTimeInSeconds() < sc.StartTime1)
{
    sc.UseSecondStartEndTimes = 0;
}

// This code does not consider if the Start Time is 00:00:00. It is assumed to be greater than that.
if (!r_PreviousCurrentTime.IsUnset()
    && !sc.UseSecondStartEndTimes
    && CurrentDateTime.GetTimeInSeconds() > sc.EndTime1
    && r_PreviousCurrentTime.GetTimeInSeconds() <= sc.EndTime1)
{
    sc.UseSecondStartEndTimes = 1;
}

r_PreviousCurrentTime = CurrentDateTime;
}
/*=====*/
SCSFExport scsf_DivergenceDetector(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_MainGraphAngle = sc.Subgraph[0];
    SCFloatArrayRef Array_BaseDataInTicks = sc.Subgraph[0].Arrays[0];
    SCSubgraphRef Subgraph_StudyAngle = sc.Subgraph[1];
    SCFloatArrayRef Array_StudyDataInPoints = sc.Subgraph[1].Arrays[0];
    SCSubgraphRef Subgraph_AngleDifference = sc.Subgraph[2];
    SCSubgraphRef Subgraph_PositiveDivergenceIndicator = sc.Subgraph[3];
    SCSubgraphRef Subgraph_NegativeDivergenceIndicator = sc.Subgraph[4];

    SCInputRef Input_Data = sc.Input[0];
    SCInputRef Input_StudySubgraphReference = sc.Input[1];
    SCInputRef Input_Length = sc.Input[2];

```

```

SCInputRef Input_DivergenceThreshold = sc.Input[3];
SCInputRef Input_OppositeSlopeDivergenceThreshold = sc.Input[4];
SCInputRef Input_ValuePerPointForStudyReference = sc.Input[5];

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

    sc.ValueFormat = 2;
    sc.AutoLoop = 1;

    Subgraph_MainGraphAngle.Name = "Main Graph Angle";
    Subgraph_MainGraphAngle.DrawStyle = DRAWSTYLE_IGNORE;
    Subgraph_MainGraphAngle.DrawZeros = false;

    Subgraph_StudyAngle.Name = "Study Angle";
    Subgraph_StudyAngle.DrawStyle = DRAWSTYLE_IGNORE;
    Subgraph_StudyAngle.DrawZeros = false;

    Subgraph_AngleDifference.Name = "Angle Difference";
    Subgraph_AngleDifference.DrawStyle = DRAWSTYLE_IGNORE;
    Subgraph_AngleDifference.PrimaryColor = RGB(0,192,0);
    Subgraph_AngleDifference.DrawZeros = false;

    Subgraph_PositiveDivergenceIndicator.Name = "Positive Divergence";
    Subgraph_PositiveDivergenceIndicator.DrawStyle = DRAWSTYLE_ARROW_UP;
    Subgraph_PositiveDivergenceIndicator.LineWidth = 3;
    Subgraph_PositiveDivergenceIndicator.DrawZeros = false;

    Subgraph_NegativeDivergenceIndicator.Name = "Negative Divergence";
    Subgraph_NegativeDivergenceIndicator.DrawStyle = DRAWSTYLE_ARROW_DOWN;
    Subgraph_NegativeDivergenceIndicator.LineWidth = 3;
    Subgraph_NegativeDivergenceIndicator.DrawZeros = false;

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

    Input_StudySubgraphReference.Name = "Study Subgraph Reference";
    Input_StudySubgraphReference.SetStudySubgraphValues(0, 0);

    Input_Length.Name = "Divergence Length";
    Input_Length.SetInt(10);

    //If the divergence of the slope of the main price graph and the slope of the study subgraph being referenced
    //exceeds this threshold, then it is considered there is a divergence.
    Input_DivergenceThreshold.Name = "Divergence Threshold in Degrees";
    Input_DivergenceThreshold.SetFloat(45.0f);
    Input_DivergenceThreshold.SetFloatLimits(-90,90);

    Input_OppositeSlopeDivergenceThreshold.Name = "Opposite Slope Divergence Threshold in Degrees";
    Input_OppositeSlopeDivergenceThreshold.SetFloat(10.0f);
    Input_OppositeSlopeDivergenceThreshold.SetFloatLimits(-90,90);

    Input_ValuePerPointForStudyReference.Name = "Value Per Point for Study Reference";
    Input_ValuePerPointForStudyReference.SetFloat(1.0f);

    sc.CalculationPrecedence = LOW_PREC_LEVEL;
    sc.GraphRegion = 0;

    return;
}

```

```

SCFloatArray StudySubgraphArray;

```

```

    sc.GetStudyArrayUsingID(Input_StudySubgraphReference.GetStudyID(),
Input_StudySubgraphReference.GetSubgraphIndex(), StudySubgraphArray);

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

    double MainGraphSlope = 0;
    double MainGraphY_Intercept = 0;
    Array_BaseDataInTicks[sc.Index] = sc.BaseData[Input_Data.GetInputDataIndex()][sc.Index] /
sc.ValueIncrementPerBar;

    sc.CalculateRegressionStatistics(Array_BaseDataInTicks,MainGraphSlope, MainGraphY_Intercept
,Input_Length.GetInt());

    double StudySlope = 0;
    double StudyY_Intercept = 0;
    Array_StudyDataInPoints[sc.Index] = StudySubgraphArray[sc.Index] /
Input_ValuePerPointForStudyReference.GetFloat();
    sc.CalculateRegressionStatistics(Array_StudyDataInPoints, StudySlope, StudyY_Intercept ,Input_Length.GetInt());

    float ThresholdDegrees = Input_DivergenceThreshold.GetFloat();
    float OppositeSlopeDivergenceThresholdDegrees = Input_OppositeSlopeDivergenceThreshold.GetFloat();

    double MainGraphAngle = sc.SlopeToAngleInDegrees(MainGraphSlope);

    double StudyAngle = sc.SlopeToAngleInDegrees(StudySlope);

    double AngleDifference = (MainGraphAngle-StudyAngle);

    Subgraph_AngleDifference[sc.Index] = static_cast<float>(AngleDifference);

    Subgraph_MainGraphAngle.Data[sc.Index] = static_cast<float>(MainGraphAngle);
    Subgraph_StudyAngle.Data[sc.Index] = static_cast<float>(StudyAngle);

    if ((fabs(AngleDifference) >= ThresholdDegrees)
        || (MainGraphAngle > 0 && StudyAngle < 0 && fabs(AngleDifference) >=
OppositeSlopeDivergenceThresholdDegrees)
        || (MainGraphAngle < 0 && StudyAngle > 0 && fabs(AngleDifference) >=
OppositeSlopeDivergenceThresholdDegrees))
    {

        //A positive divergence is when the study angle is greater than the main price graph angle
        if (StudyAngle > MainGraphAngle)
        {
            Subgraph_PositiveDivergenceIndicator[sc.Index] = sc.Low[sc.Index] - sc.TickSize;
            Subgraph_NegativeDivergenceIndicator[sc.Index] = 0;
        }
        else
        {
            Subgraph_PositiveDivergenceIndicator[sc.Index] = 0;
            Subgraph_NegativeDivergenceIndicator[sc.Index] = sc.High[sc.Index] + sc.TickSize;
        }
    }
    else
    {
        Subgraph_PositiveDivergenceIndicator[sc.Index] = 0;
        Subgraph_NegativeDivergenceIndicator[sc.Index] = 0;
    }
}

/*=====*/
SCSFExport scsf_CandlestickBodyGapExtend(SCStudyInterfaceRef sc)

```



```

{
    SCSubgraphRef Subgraph_LineExtension = sc.Subgraph[0];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Candlestick Body Gap Extend";

        sc.AutoLoop = 1;

        Subgraph_LineExtension.Name = "Line Extension";
        Subgraph_LineExtension.DrawStyle = DRAWSTYLE_COLOR_BAR;
        Subgraph_LineExtension.PrimaryColor = RGB(0,255,0);
        Subgraph_LineExtension.DrawZeros = false;

        return;
    }

    if(sc.Index == 0)
        return;

    float PriorBarOpen = sc.BaseData[SC_OPEN][sc.Index-1];
    float PriorBarClose = sc.BaseData[SC_LAST][sc.Index-1];
    float BarOpen = sc.BaseData[SC_OPEN][sc.Index];
    float BarClose = sc.BaseData[SC_LAST][sc.Index];
    float PriorBarMin = min(PriorBarOpen, PriorBarClose);
    float PriorBarMax = max(PriorBarOpen, PriorBarClose);
    float BarMin = min(BarOpen, BarClose);
    float BarMax = max(BarOpen, BarClose);

    if(sc.FormattedEvaluate(PriorBarMin, sc.BaseGraphValueFormat, GREATER_OPERATOR, BarMax,
sc.BaseGraphValueFormat))
        Subgraph_LineExtension[sc.Index] = 1;
    else if(sc.FormattedEvaluate(PriorBarMax, sc.BaseGraphValueFormat, LESS_OPERATOR, BarMin,
sc.BaseGraphValueFormat))
        Subgraph_LineExtension[sc.Index] = 1;
    else
        Subgraph_LineExtension[sc.Index] = 0;

}

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

    if (sc.SetDefaults)
    {
        sc.GraphName = "CalculateTimeSpanAcrossChartBars Example";

        Subgraph_TimeDiff.Name = "Duration";
        Subgraph_TimeDiff.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_TimeDiff.PrimaryColor = RGB(0, 255, 0);

        sc.GraphRegion = 1;
        sc.ValueFormat = VALUEFORMAT_TIME;
        sc.AutoLoop = 1;
    }
}

```

```

    Subgraph_TimeDiff[sc.Index] = static_cast<float>(sc.CalculateTimeSpanAcrossChartBars(sc.Index, sc.Index).GetAsDouble());
}

/*=====*/
SCSFExport scsf_StudySubgraphAsTextAboveBelowBarV2(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_StudyReference = sc.Subgraph[0];

    SCFloatArrayRef Array_TextPosition = Subgraph_StudyReference.Arrays[0];

    SCInputRef Input_StudySubgraph1 = sc.Input[0];
    SCInputRef Input_DisplayAboveOrBelow = sc.Input[1];
    SCInputRef Input_OffsetInTicks = sc.Input[2];
    SCInputRef Input_DrawZeros = sc.Input[3];
    SCInputRef Input_AdditionalForwardColumns = sc.Input[4];
    SCInputRef Input_OffsetInTextHeights = sc.Input[5]; //Graph.m_Arrays[SubgraphIndex][1].

    if (sc.SetDefaults)
    {
        sc.GraphName = "Study Subgraph Above/Below Bar as Text";

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

        //sc.CalculationPrecedence = LOW_PREC_LEVEL;

        Subgraph_StudyReference.Name = "Reference";
        Subgraph_StudyReference.DrawStyle = DRAWSTYLE_CUSTOM_VALUE_AT_Y;
        Subgraph_StudyReference.PrimaryColor = RGB(252, 196, 120);
        Subgraph_StudyReference.LineWidth = 12;

        Input_StudySubgraph1.Name = "Input Study Subgraph";
        Input_StudySubgraph1.SetStudySubgraphValues(0, 0);

        Input_DisplayAboveOrBelow.Name = "Display Location";
        Input_DisplayAboveOrBelow.SetCustomInputStrings("Top of Bar;Bottom of Bar;Bottom of Chart Region");
        Input_DisplayAboveOrBelow.SetCustomInputIndex(0);

        Input_OffsetInTicks.Name = "Offset in Ticks";
        Input_OffsetInTicks.SetInt(1);

        Input_DrawZeros.Name = "Draw Zeros";
        Input_DrawZeros.SetYesNo(false);

        Input_AdditionalForwardColumns.Name = "Additional Forward Columns";
        Input_AdditionalForwardColumns.SetInt(0);

        Input_OffsetInTextHeights.Name = "Offset in Text Heights (+/-)";
        Input_OffsetInTextHeights.SetInt(0);

        return;
    }

    if (Input_DisplayAboveOrBelow.GetIndex() != 2)
    {
        sc.ScaleRangeType = SCALE_AUTO;
    }
    else
    {
        sc.ScaleRangeType = SCALE_USERDEFINED;
        sc.ScaleRangeTop = 100;
    }
}

```

```

    sc.ScaleRangeBottom = 1;
}

if (sc.IsFullRecalculation)
    Subgraph_StudyReference.ExtendedArrayElementsToGraph = Input_AdditionalForwardColumns.GetInt();

Subgraph_StudyReference.DrawZeros = Input_DrawZeros.GetYesNo();

SCFloatArray Study1Array;
sc.GetStudyArrayUsingID(Input_StudySubgraph1.GetStudyID(), Input_StudySubgraph1.GetSubgraphIndex(),
Study1Array);

const int DisplayPositionIndex = Input_DisplayAboveOrBelow.GetIndex();

int EndIndex = sc.ArraySize + Input_AdditionalForwardColumns.GetInt();

for (int BarIndex = sc.UpdateStartIndex; BarIndex < EndIndex; ++BarIndex)
{
    int BarValueIndex = min(BarIndex, sc.ArraySize - 1);

    float ValueFromReferencedSubgraph = Study1Array[BarIndex];

    if (Input_OffsetInTextHeights.GetInt() != 0)
    {
        Subgraph_StudyReference.Arrays[1][BarIndex] = static_cast<float>(Input_OffsetInTextHeights.GetInt());
    }

    if (ValueFromReferencedSubgraph != FLT_MAX && ValueFromReferencedSubgraph != -FLT_MAX)
    {
        Subgraph_StudyReference[BarIndex] = ValueFromReferencedSubgraph;

        if (DisplayPositionIndex == 0)
        {
            Array_TextPosition[BarIndex] = sc.High[BarValueIndex] + (Input_OffsetInTicks.GetInt() * sc.TickSize);
        }
        else if (DisplayPositionIndex == 1)
        {
            Array_TextPosition[BarIndex] = sc.Low[BarValueIndex] - (Input_OffsetInTicks.GetInt() * sc.TickSize);
        }
        else
        {
            Array_TextPosition[BarIndex] = 5;
        }
    }
    else
    {
        Subgraph_StudyReference[BarIndex] = 0;
        Array_TextPosition[BarIndex] = 0;
    }
}
}

/*=====*/

SCSFExport scsf_MovingAverageColored(SCStudyInterfaceRef sc)
{
    if (sc.SetDefaults)
    {
        sc.GraphName = "Moving Average - Colored";

        sc.Subgraph[0].Name = "Up Color Top";
        sc.Subgraph[0].DrawStyle = DRAWSTYLE_FILLTOP;
    }
}

```

```

sc.Subgraph[0].DrawZeros = 1;

sc.Subgraph[1].Name = "Up Color Bottom";
sc.Subgraph[1].DrawStyle = DRAWSTYLE_FILLBOTTOM;
sc.Subgraph[1].DrawZeros = 1;

sc.Subgraph[2].Name = "Down Color";
sc.Subgraph[2].DrawStyle = DRAWSTYLE_IGNORE;
sc.Subgraph[2].DrawZeros = 1;

sc.Subgraph[3].Name = "Down Color";
sc.Subgraph[3].DrawStyle = DRAWSTYLE_IGNORE;
sc.Subgraph[3].DrawZeros = 1;

sc.Input[0].Name = "MA Type";
sc.Input[0].SetMovAvgType(MOVAVGTYPE_EXPONENTIAL);

sc.Input[1].Name = "MA Input";
sc.Input[1].SetInputDataIndex(SC_LAST);

sc.Input[3].Name = "Short MA Length";
sc.Input[3].SetInt(13);

sc.Input[4].Name = "Long MA Length";
sc.Input[4].SetInt(21);


sc.GraphRegion = 0; // First region
return;
}

int& SLength = sc.Input[3].IntValue;
int& LLength = sc.Input[4].IntValue;

for (int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize; BarIndex++)
{
    sc.MovingAverage(sc.BaseDataIn[sc.Input[1].GetInputDataIndex()], sc.Subgraph[5], sc.Input[0].GetMovAvgType(),
BarIndex, SLength);

    sc.MovingAverage(sc.BaseDataIn[sc.Input[1].GetInputDataIndex()], sc.Subgraph[6], sc.Input[0].GetMovAvgType(),
BarIndex, LLength);

    if (sc.Subgraph[5][BarIndex] > sc.Subgraph[6][BarIndex])
    {
        sc.Subgraph[0][BarIndex] = sc.Subgraph[5][BarIndex];
        sc.Subgraph[1][BarIndex] = sc.Subgraph[6][BarIndex];
        if (sc.BaseDataIn[3][BarIndex] > sc.Subgraph[5][BarIndex])
            sc.Subgraph[0].DataColor[BarIndex] = sc.Subgraph[0].PrimaryColor;
        else
            sc.Subgraph[0].DataColor[BarIndex] = sc.Subgraph[1].PrimaryColor;
    }
    else
    {
        sc.Subgraph[0][BarIndex] = sc.Subgraph[6][BarIndex];
        sc.Subgraph[1][BarIndex] = sc.Subgraph[5][BarIndex];

        if (sc.BaseDataIn[3][BarIndex] < sc.Subgraph[5][BarIndex])
            sc.Subgraph[0].DataColor[BarIndex] = sc.Subgraph[3].PrimaryColor;
        else
            sc.Subgraph[0].DataColor[BarIndex] = sc.Subgraph[2].PrimaryColor;
    }
}
}
}

```

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

```
SCSFExport scsf_ConnieBrownCompositeIndex(SCStudyInterfaceRef sc)
```

```
{
    SCSubgraphRef Subgraph_RSI1 = sc.Subgraph[0];
    SCSubgraphRef Subgraph_RSI2 = sc.Subgraph[1];
    SCSubgraphRef Subgraph_RSIMO = sc.Subgraph[2];
    SCSubgraphRef Subgraph_RSIMA = sc.Subgraph[3];
    SCSubgraphRef Subgraph_CBI1 = sc.Subgraph[4];
    SCSubgraphRef Subgraph_CBI2 = sc.Subgraph[5];
    SCSubgraphRef Subgraph_CBI3 = sc.Subgraph[6];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_RSILength1 = sc.Input[1];
    SCInputRef Input_RSILength2 = sc.Input[2];
    SCInputRef Input_MomentumLength = sc.Input[3];
    SCInputRef Input_MovAvgLength1 = sc.Input[4];
    SCInputRef Input_MovAvgLength2 = sc.Input[5];
    SCInputRef Input_MovAvgLength3 = sc.Input[6];
    SCInputRef Input_MovAvgType1 = sc.Input[7];
    SCInputRef Input_MovAvgType2 = sc.Input[8];
    SCInputRef Input_MovAvgType3 = sc.Input[9];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Connie Brown's Composite Index";

        sc.AutoLoop = 1;

        Subgraph_RSI1.Name = "RSI 1";
        Subgraph_RSI1.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_RSI2.Name = "RSI 2";
        Subgraph_RSI2.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_RSIMO.Name = "Momentum of RSI";
        Subgraph_RSIMO.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_RSIMA.Name = "Moving Average of RSI";
        Subgraph_RSIMA.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_CBI1.Name = "Connie Brown Index 1";
        Subgraph_CBI1.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_CBI1.PrimaryColor = RGB(255, 0, 0);
        Subgraph_CBI1.DrawZeros = true;

        Subgraph_CBI2.Name = "Connie Brown Index 2";
        Subgraph_CBI2.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_CBI2.PrimaryColor = RGB(0, 255, 0);
        Subgraph_CBI2.DrawZeros = true;

        Subgraph_CBI3.Name = "Connie Brown Index 3";
        Subgraph_CBI3.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_CBI3.PrimaryColor = RGB(0, 0, 255);
        Subgraph_CBI3.DrawZeros = true;

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

        Input_RSILength1.Name = "RSI Length 1";
        Input_RSILength1.SetInt(14);
        Input_RSILength1.SetIntLimits(1, MAX_STUDY_LENGTH);
    }
}
```

```

Input_RSILength2.Name = "RSI Length 2";
Input_RSILength2.SetInt(3);
Input_RSILength2.SetIntLimits(1, MAX_STUDY_LENGTH);

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

Input_MovAvgLength1.Name = "Moving Average Length 1";
Input_MovAvgLength1.SetInt(3);
Input_MovAvgLength1.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_MovAvgLength2.Name = "Moving Average Length 2";
Input_MovAvgLength2.SetInt(13);
Input_MovAvgLength2.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_MovAvgLength3.Name = "Moving Average Length 3";
Input_MovAvgLength3.SetInt(33);
Input_MovAvgLength3.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_MovAvgType1.Name = "Moving Average Type 1";
Input_MovAvgType1.SetMovAvgType(MOVAVGTYPE_SIMPLE);

Input_MovAvgType2.Name = "Moving Average Type 2";
Input_MovAvgType2.SetMovAvgType(MOVAVGTYPE_SIMPLE);

Input_MovAvgType3.Name = "Moving Average Type 3";
Input_MovAvgType3.SetMovAvgType(MOVAVGTYPE_SIMPLE);

return;
}

sc.DataStartIndex = max(Input_RSILength1.GetInt() + Input_MomentumLength.GetInt(), Input_RSILength2.GetInt() +
Input_MovAvgLength1.GetInt()) + max(Input_MovAvgLength2.GetInt(), Input_MovAvgLength3.GetInt()) - 1;

sc.RSI(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_RSI1,
Input_MovAvgType1.GetMovAvgType(), Input_RSILength1.GetInt());

sc.RSI(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_RSI2,
Input_MovAvgType1.GetMovAvgType(), Input_RSILength2.GetInt());
sc.MovingAverage(Subgraph_RSI2, Subgraph_RSIMA, Input_MovAvgType1.GetMovAvgType(),
Input_MovAvgLength1.GetInt());

Subgraph_RSIMO[sc.Index] = Subgraph_RSI1[sc.Index] - Subgraph_RSI1[sc.Index -
Input_MomentumLength.GetInt()];

Subgraph_CBI1[sc.Index] = Subgraph_RSIMO[sc.Index] + Subgraph_RSIMA[sc.Index];

sc.MovingAverage(Subgraph_CBI1, Subgraph_CBI2, Input_MovAvgType2.GetMovAvgType(),
Input_MovAvgLength2.GetInt());

sc.MovingAverage(Subgraph_CBI1, Subgraph_CBI3, Input_MovAvgType3.GetMovAvgType(),
Input_MovAvgLength3.GetInt());
}

/*=====*/

SCSFExport scsf_MovingAverageBlock(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_ATR = sc.Subgraph[0];
    SCSubgraphRef Subgraph_WorkBoxHalf1 = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Direction1 = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Top1 = sc.Subgraph[3];
    SCSubgraphRef Subgraph_Mid1 = sc.Subgraph[4];
    SCSubgraphRef Subgraph_Bot1 = sc.Subgraph[5];

```

```

SCSubgraphRef Subgraph_WorkBoxHalf2 = sc.Subgraph[6];
SCSubgraphRef Subgraph_Direction2 = sc.Subgraph[7];
SCSubgraphRef Subgraph_Top2 = sc.Subgraph[8];
SCSubgraphRef Subgraph_Mid2 = sc.Subgraph[9];
SCSubgraphRef Subgraph_Bot2 = sc.Subgraph[10];

SCInputRef Input_ATRLength = sc.Input[0];
SCInputRef Input_ATRMovAvgType = sc.Input[1];
SCInputRef Input_BoxMultiplier1 = sc.Input[2];
SCInputRef Input_BoxMultiplier2 = sc.Input[3];

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

    sc.GraphRegion = 0;
    sc.ValueFormat = 2;
    sc.AutoLoop = 1;

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

    Subgraph_WorkBoxHalf1.Name = "Work Box Half 1";
    Subgraph_WorkBoxHalf1.DrawStyle = DRAWSTYLE_IGNORE;

    Subgraph_Direction1.Name = "Direction 1";
    Subgraph_Direction1.DrawStyle = DRAWSTYLE_IGNORE;

    Subgraph_Top1.Name = "Top 1";
    Subgraph_Top1.DrawStyle = DRAWSTYLE_IGNORE;

    Subgraph_Mid1.Name = "Middle 1";
    Subgraph_Mid1.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_Mid1.PrimaryColor = RGB(0, 255, 255); // cyan
    Subgraph_Mid1.SecondaryColor = RGB(255, 0, 255); // magenta
    Subgraph_Mid1.SecondaryColorUsed = true;
    Subgraph_Mid1.DrawZeros = true;

    Subgraph_Bot1.Name = "Bottom 1";
    Subgraph_Bot1.DrawStyle = DRAWSTYLE_IGNORE;

    Subgraph_WorkBoxHalf2.Name = "Work Box Half 2";
    Subgraph_WorkBoxHalf2.DrawStyle = DRAWSTYLE_IGNORE;

    Subgraph_Direction2.Name = "Direction 2";
    Subgraph_Direction2.DrawStyle = DRAWSTYLE_IGNORE;

    Subgraph_Top2.Name = "Top 2";
    Subgraph_Top2.DrawStyle = DRAWSTYLE_IGNORE;

    Subgraph_Mid2.Name = "Middle 2";
    Subgraph_Mid2.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_Mid2.PrimaryColor = RGB(0, 255, 255); // cyan
    Subgraph_Mid2.SecondaryColor = RGB(255, 0, 255); // magenta
    Subgraph_Mid2.SecondaryColorUsed = true;
    Subgraph_Mid2.DrawZeros = true;

    Subgraph_Bot2.Name = "Bottom 2";
    Subgraph_Bot2.DrawStyle = DRAWSTYLE_IGNORE;

    Input_ATRLength.Name = "Average True Range Length";
    Input_ATRLength.SetInt(10);
    Input_ATRLength.SetIntLimits(1, MAX_STUDY_LENGTH);

    Input_ATRMovAvgType.Name = "ATR Moving Average Type";

```

```

Input_ATRMovAvgType.SetMovAvgType(MOVAVGTYPE_SIMPLE);

Input_BoxMultiplier1.Name = "Box Multiplier 1";
Input_BoxMultiplier1.SetFloat(1.0f);

Input_BoxMultiplier2.Name = "Box Multiplier 2";
Input_BoxMultiplier2.SetFloat(2.0f);

return;
}

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

sc.ATR(sc.BaseDataIn, Subgraph_ATR, Input_ATRLength.GetInt(), Input_ATRMovAvgType.GetMovAvgType());

Subgraph_WorkBoxHalf1[sc.Index] = 0.5f*Input_BoxMultiplier1.GetFloat()*Subgraph_ATR[sc.Index];

if (sc.High[sc.Index] > Subgraph_Top1[sc.Index - 1])
{
    Subgraph_Direction1[sc.Index] = 1;
    Subgraph_Top1[sc.Index] = sc.High[sc.Index];
    Subgraph_Mid1[sc.Index] = Subgraph_Top1[sc.Index] - Subgraph_WorkBoxHalf1[sc.Index];
    Subgraph_Bot1[sc.Index] = Subgraph_Top1[sc.Index] - 2 * Subgraph_WorkBoxHalf1[sc.Index];
}

else if (sc.Low[sc.Index] < Subgraph_Bot1[sc.Index - 1])
{
    Subgraph_Direction1[sc.Index] = -1;
    Subgraph_Bot1[sc.Index] = sc.Low[sc.Index];
    Subgraph_Mid1[sc.Index] = Subgraph_Bot1[sc.Index] + Subgraph_WorkBoxHalf1[sc.Index];
    Subgraph_Top1[sc.Index] = Subgraph_Bot1[sc.Index] + 2 * Subgraph_WorkBoxHalf1[sc.Index];
}

else
{
    Subgraph_Direction1[sc.Index] = Subgraph_Direction1[sc.Index - 1];
    Subgraph_Top1[sc.Index] = Subgraph_Top1[sc.Index - 1];
    Subgraph_Mid1[sc.Index] = Subgraph_Mid1[sc.Index - 1];
    Subgraph_Bot1[sc.Index] = Subgraph_Bot1[sc.Index - 1];
}

Subgraph_WorkBoxHalf2[sc.Index] = 0.5f*Input_BoxMultiplier2.GetFloat()*Subgraph_ATR[sc.Index];

if (sc.High[sc.Index] > Subgraph_Top2[sc.Index - 1])
{
    Subgraph_Direction2[sc.Index] = 1;
    Subgraph_Top2[sc.Index] = sc.High[sc.Index];
    Subgraph_Mid2[sc.Index] = Subgraph_Top2[sc.Index] - Subgraph_WorkBoxHalf2[sc.Index];
    Subgraph_Bot2[sc.Index] = Subgraph_Top2[sc.Index] - 2 * Subgraph_WorkBoxHalf2[sc.Index];
}

else if (sc.Low[sc.Index] < Subgraph_Bot2[sc.Index - 1])
{
    Subgraph_Direction2[sc.Index] = -1;
    Subgraph_Bot2[sc.Index] = sc.Low[sc.Index];
    Subgraph_Mid2[sc.Index] = Subgraph_Bot2[sc.Index] + Subgraph_WorkBoxHalf2[sc.Index];
    Subgraph_Top2[sc.Index] = Subgraph_Bot2[sc.Index] + 2 * Subgraph_WorkBoxHalf2[sc.Index];
}

else
{
    Subgraph_Direction2[sc.Index] = Subgraph_Direction2[sc.Index - 1];
    Subgraph_Top2[sc.Index] = Subgraph_Top2[sc.Index - 1];
}

```



```

    Subgraph_Mid2[sc.Index] = Subgraph_Mid2[sc.Index - 1];
    Subgraph_Bot2[sc.Index] = Subgraph_Bot2[sc.Index - 1];
}

if (Subgraph_Direction1[sc.Index] == 1)
    Subgraph_Mid1.DataColor[sc.Index] = Subgraph_Mid1.PrimaryColor;
else if (Subgraph_Direction1[sc.Index] == -1)
    Subgraph_Mid1.DataColor[sc.Index] = Subgraph_Mid1.SecondaryColor;

if (Subgraph_Direction2[sc.Index] == 1)
    Subgraph_Mid2.DataColor[sc.Index] = Subgraph_Mid2.PrimaryColor;
else if (Subgraph_Direction2[sc.Index] == -1)
    Subgraph_Mid2.DataColor[sc.Index] = Subgraph_Mid2.SecondaryColor;
}

/*=====*/

SCSFExport scsf_PolarizedFractalEfficiency(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_PFE = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SmoothedPFE = sc.Subgraph[1];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_Period = sc.Input[1];
    SCInputRef Input_SmoothingPeriod = sc.Input[2];
    SCInputRef Input_MovingAverageType = sc.Input[3];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Polarized Fractal Efficiency";

        sc.GraphRegion = 1;
        sc.ValueFormat = 2;
        sc.AutoLoop = 1;

        Subgraph_PFE.Name = "Polarized Fractal Efficiency";
        Subgraph_PFE.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_PFE.PrimaryColor = RGB(0, 255, 0);
        Subgraph_PFE.LineWidth = 1;
        Subgraph_PFE.DrawZeros = true;

        Subgraph_SmoothedPFE.Name = "Smoothed Polarized Fractal Efficiency";
        Subgraph_SmoothedPFE.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_SmoothedPFE.PrimaryColor = RGB(0, 0, 255);
        Subgraph_SmoothedPFE.LineWidth = 1;
        Subgraph_SmoothedPFE.DrawZeros = true;

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

        Input_Period.Name = "Period";
        Input_Period.SetInt(14);
        Input_Period.SetIntLimits(1, MAX_STUDY_LENGTH);

        Input_SmoothingPeriod.Name = "Smoothing Period";
        Input_SmoothingPeriod.SetInt(3);
        Input_SmoothingPeriod.SetIntLimits(1, MAX_STUDY_LENGTH);

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

        return;
    }
}

```

```

}

float DenominatorSum = 0;

if (sc.Index < Input_Period.GetInt())
    Subgraph_PFE[sc.Index] = 0;
else
{
    for (int Index = 0; Index <= Input_Period.GetInt() - 2; Index++)
    {
        float PriceChangeDenominator = sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - Index] -
sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - Index - 1];
        float PriceChangeDenominatorSquared = PriceChangeDenominator * PriceChangeDenominator;

        DenominatorSum += sqrt(PriceChangeDenominatorSquared + 1);
    }

    float PriceChangeNumerator = sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index] -
sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - Input_Period.GetInt()];
    float PriceChangeNumeratorSquared = PriceChangeNumerator * PriceChangeNumerator;

    Subgraph_PFE[sc.Index] = 100 * sqrt(PriceChangeNumeratorSquared +
Input_Period.GetInt()*Input_Period.GetInt()) / DenominatorSum;

    if (sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index] <
sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - 1])
    {
        Subgraph_PFE[sc.Index] = -1 * Subgraph_PFE[sc.Index];
    }
}

sc.MovingAverage(Subgraph_PFE, Subgraph_SmoothedPFE, Input_MovingAverageType.GetMovAvgType(),
Input_SmoothingPeriod.GetInt());
}

/*=====*/

SCSFExport scsf_PriceMomentumOscillator(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_RateOfChange = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SmoothedRateOfChange = sc.Subgraph[1];
    SCSubgraphRef Subgraph_PriceMomentumOscillatorLine = sc.Subgraph[2];
    SCSubgraphRef Subgraph_PriceMomentumOscillatorSignal = sc.Subgraph[3];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_PriceMomentumOscillatorLineLength1 = sc.Input[1];
    SCInputRef Input_PriceMomentumOscillatorLineLength2 = sc.Input[2];
    SCInputRef Input_PriceMomentumOscillatorSignalLength = sc.Input[3];
    SCInputRef Input_MovingAverageType = sc.Input[4];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Price Momentum Oscillator";

        sc.AutoLoop = 1;

        Subgraph_RateOfChange.Name = "Rate of Change";
        Subgraph_RateOfChange.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_SmoothedRateOfChange.Name = "Smoothed Rate of Change";
        Subgraph_SmoothedRateOfChange.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_PriceMomentumOscillatorLine.Name = "Price Momentum Oscillator Line";

```

```

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

Subgraph_PriceMomentumOscillatorSignal.Name = "Price Momentum Oscillator Signal";
Subgraph_PriceMomentumOscillatorSignal.DrawStyle = DRAWSTYLE_LINE;
Subgraph_PriceMomentumOscillatorSignal.PrimaryColor = RGB(0, 0, 255);
Subgraph_PriceMomentumOscillatorSignal.DrawZeros = true;

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

Input_PriceMomentumOscillatorLineLength1.Name = "PMO Line Length 1";
Input_PriceMomentumOscillatorLineLength1.SetInt(35);
Input_PriceMomentumOscillatorLineLength1.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_PriceMomentumOscillatorLineLength2.Name = "PMO Line Length 2";
Input_PriceMomentumOscillatorLineLength2.SetInt(20);
Input_PriceMomentumOscillatorLineLength2.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_PriceMomentumOscillatorSignalLength.Name = "PMO Signal Line Length";
Input_PriceMomentumOscillatorSignalLength.SetInt(10);
Input_PriceMomentumOscillatorSignalLength.SetIntLimits(1, MAX_STUDY_LENGTH);

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

return;
}

sc.DataStartIndex = max(max(Input_PriceMomentumOscillatorLineLength1.GetInt(),
Input_PriceMomentumOscillatorLineLength2.GetInt()), Input_PriceMomentumOscillatorSignalLength.GetInt()) - 1;

if (sc.Index == 0)
return;

else if (sc.Index == 1)
{
Subgraph_RateOfChange[sc.Index] = 100 * (sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index] -
sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - 1]) / sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - 1];

Subgraph_SmoothedRateOfChange[sc.Index] = Subgraph_RateOfChange[sc.Index];
Subgraph_PriceMomentumOscillatorLine[sc.Index] = 10 * Subgraph_SmoothedRateOfChange[sc.Index];
}
else
{
Subgraph_RateOfChange[sc.Index] = 100 * (sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index] -
sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - 1]) / sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - 1];

Subgraph_SmoothedRateOfChange[sc.Index] = (2.0f / Input_PriceMomentumOscillatorLineLength1.GetInt()) *
(Subgraph_RateOfChange[sc.Index] - Subgraph_SmoothedRateOfChange[sc.Index - 1]) +
Subgraph_SmoothedRateOfChange[sc.Index - 1];

Subgraph_PriceMomentumOscillatorLine[sc.Index] = (2.0f / Input_PriceMomentumOscillatorLineLength2.GetInt()) *
(10 * Subgraph_SmoothedRateOfChange[sc.Index] - Subgraph_PriceMomentumOscillatorLine[sc.Index - 1]) +
Subgraph_PriceMomentumOscillatorLine[sc.Index - 1];
}

sc.MovingAverage(Subgraph_PriceMomentumOscillatorLine, Subgraph_PriceMomentumOscillatorSignal,
Input_MovingAverageType.GetMovAvgType(), Input_PriceMomentumOscillatorSignalLength.GetInt());
}

/*=====*/

```

```

SCSFExport scsf_StudySubgraphReverseOrder(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_ReverseOrder = sc.Subgraph[0];

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

    if (sc.SetDefaults)
    {
        sc.GraphName = "Study Subgraph Reverse Order";

        sc.AutoLoop = 0;

        Subgraph_ReverseOrder.Name = "ReverseOrder";
        Subgraph_ReverseOrder.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_ReverseOrder.PrimaryColor = RGB(0, 255, 0);

        Input_InputData.Name = "Input Data";
        Input_InputData.SetInputDataIndex(0);

        Input_DrawZeros.Name = "Draw Zeros";
        Input_DrawZeros.SetYesNo(false);

        return;
    }

    Subgraph_ReverseOrder.DrawZeros = Input_DrawZeros.GetYesNo();

    int CalculationStartIndex = sc.GetCalculationStartIndexForStudy();

    //Is a full recalculation or there are new chart bars
    if (sc.IsFullRecalculation || (sc.UpdateStartIndex < sc.ArraySize - 1))
        CalculationStartIndex=0;

    int OutputIndex = sc.ArraySize - 1 - CalculationStartIndex;

    if (OutputIndex < 0)
        return;

    for (int Index = CalculationStartIndex; Index < sc.ArraySize; Index++)
    {
        float BasedOnStudySubgraphValue = sc.BaseData[Input_InputData.GetInputDataIndex()][Index];

        Subgraph_ReverseOrder[OutputIndex] = BasedOnStudySubgraphValue;
        OutputIndex--;

        if (OutputIndex < 0)
            break;
    }

    sc.EarliestUpdateSubgraphDataArrayIndex = 0;
}

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

    SCInputRef Input_Data = sc.Input[0];
    SCInputRef Input_DrawZeros = sc.Input[1];

    if (sc.SetDefaults)

```

```

{
    sc.GraphName = "Study Subgraph Invert";

    sc.AutoLoop = 0; // Needed when using sc.GetCalculationStartIndexForStudy

    Subgraph_Result.Name = "Result";
    Subgraph_Result.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_Result.PrimaryColor = RGB(0, 255, 0);

    Input_Data.Name = "Input Data";
    Input_Data.SetInputDataIndex(0);

    Input_DrawZeros.Name = "Draw Zeros";
    Input_DrawZeros.SetYesNo(false);

    return;
}

Subgraph_Result.DrawZeros = Input_DrawZeros.GetYesNo();

int CalculationStartIndex = sc.GetCalculationStartIndexForStudy();

int InputDataIndex = Input_Data.GetInputDataIndex();

for (int Index = CalculationStartIndex; Index < sc.ArraySize; Index++)
{
    Subgraph_Result[Index] = 1.0f/sc.BaseData[InputDataIndex][Index];
}

sc.EarliestUpdateSubgraphDataArrayIndex = CalculationStartIndex;
}

/*=====*/
SCSFExport scsf_ElliottWaves(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_UpWave = sc.Subgraph[0];
    SCSubgraphRef Subgraph_DownWave = sc.Subgraph[1];

    SCFloatArrayRef Array_UpWave = sc.Subgraph[0].Arrays[0];
    SCFloatArrayRef Array_DownWave = sc.Subgraph[1].Arrays[0];

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

    if (sc.SetDefaults)
    {
        sc.GraphName = "Elliott Waves";

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

        Subgraph_UpWave.Name = "Up Wave";
        Subgraph_UpWave.DrawStyle = DRAWSTYLE_LINE_SKIP_ZEROS;
        Subgraph_UpWave.DrawZeros = 0;

        Subgraph_DownWave.Name = "Down Wave";
        Subgraph_DownWave.DrawStyle = DRAWSTYLE_LINE_SKIP_ZEROS;
        Subgraph_DownWave.DrawZeros = 0;

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

        Input_RetracementToleranceLevel.Name = "Retracement Tolerance Fraction (1%=.01,100%=1)";
    }
}

```

```

Input_RetracementToleranceLevel.SetFloat(0.0f);
Input_RetracementToleranceLevel.SetFloatLimits(0.0f, 1.0f);

return;
}

// Define Persistent Integers
int& r_UpWaveBeginIndex = sc.GetPersistentIntFast(1);
int& r_UpWaveEndIndex = sc.GetPersistentIntFast(2);
int& r_DownWaveBeginIndex = sc.GetPersistentIntFast(3);
int& r_DownWaveEndIndex = sc.GetPersistentIntFast(4);

if (sc.IsFullRecalculation)
{
    // Initialize Persistent Integers
    r_UpWaveBeginIndex = 0;
    r_UpWaveEndIndex = 0;
    r_DownWaveBeginIndex = 0;
    r_DownWaveEndIndex = 0;
}

const int InputDataIndex = Input_InputData.GetInputDataIndex();
const int MinimumIndex = 1;

for (int BarIndex = max(MinimumIndex, sc.UpdateStartIndex); BarIndex < sc.ArraySize; BarIndex++)
{
    if (sc.GetBarHasClosedStatus(BarIndex) == BHCS_BAR_HAS_NOT_CLOSED)
        continue;

    Subgraph_UpWave[BarIndex] = 0;
    Subgraph_DownWave[BarIndex] = 0;

    double PriorPrice = sc.BaseDataIn[InputDataIndex][BarIndex - 1];
    double Price = sc.BaseDataIn[InputDataIndex][BarIndex];
    const double RetracementToleranceFraction = Input_RetracementToleranceLevel.GetFloat();

    double FractionOfAbsoluteValuePriceDifference = RetracementToleranceFraction * fabs(Price - PriorPrice);

    // Detect Up Wave
    if (Price >= (PriorPrice - FractionOfAbsoluteValuePriceDifference))
    {
        Array_UpWave[BarIndex] = 1.0f;
    }
    else
        Array_UpWave[BarIndex] = 0;

    // Draw Up Wave
    if (Array_UpWave[BarIndex - 1] == 0.0f && Array_UpWave[BarIndex] == 1.0f)
        r_UpWaveBeginIndex = BarIndex;

    if (Array_UpWave[BarIndex - 1] == 1.0f && Array_UpWave[BarIndex] == 0.0f)
        r_UpWaveEndIndex = BarIndex - 1;

    if (r_UpWaveEndIndex != 0)
    {
        sc.FillSubGraphBetweenBeginEndPoints
        (
            0,
            r_UpWaveBeginIndex,
            r_UpWaveEndIndex,
            sc.BaseDataIn[InputDataIndex][r_UpWaveBeginIndex],
            sc.BaseDataIn[InputDataIndex][r_UpWaveEndIndex]
        );

        // Reset Up Wave Persistent Integers to 0 after Up Wave Is Drawn

```

```

    r_UpWaveBeginIndex = 0;
    r_UpWaveEndIndex = 0;
}

// Detect Down Wave
if (Price <= (PriorPrice + (RetracementToleranceFraction * fabs(Price - PriorPrice))))
{
    Array_DownWave[BarIndex] = 1.0f;
}
else
    Array_DownWave[BarIndex] = 0;

if (Array_DownWave[BarIndex - 1] == 0.0f && Array_DownWave[BarIndex] == 1.0f)
    r_DownWaveBeginIndex = BarIndex;

if (Array_DownWave[BarIndex - 1] == 1.0f && Array_DownWave[BarIndex] == 0.0f)
    r_DownWaveEndIndex = BarIndex - 1;

// Draw Down Wave
if (r_DownWaveEndIndex != 0)
{
    sc.FillSubGraphBetweenBeginEndPoints
    ( 1,
      r_DownWaveBeginIndex,
      r_DownWaveEndIndex,
      sc.BaseDataIn[InputDataIndex][r_DownWaveBeginIndex],
      sc.BaseDataIn[InputDataIndex][r_DownWaveEndIndex]
    );

    // Reset Down Wave Persistent Integers to 0 after Down Wave Is Drawn
    r_DownWaveBeginIndex = 0;
    r_DownWaveEndIndex = 0;
}
}
}

```