

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

```
/*=====*/
SCSFExport scsf_SumAllChartsBar(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_Volume = sc.Subgraph[4];
    SCSubgraphRef Subgraph_OpenInterest = sc.Subgraph[5];
    SCSubgraphRef Subgraph_OHLCAvg = sc.Subgraph[6];
    SCSubgraphRef Subgraph_HLCAvg = sc.Subgraph[7];
    SCSubgraphRef Subgraph_HLAvg = sc.Subgraph[8];
    SCSubgraphRef Subgraph_BidVol = sc.Subgraph[9];
    SCSubgraphRef Subgraph_AskVol = sc.Subgraph[10];

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

        sc.GraphName = "Sum All Charts (Bar)" ;

        sc.StudyDescription = "Sum All Charts (Bar).";

        sc.GraphDrawType = GDT_OHLCBAR;
        sc.StandardChartHeader = 1;

        sc.ValueFormat = 2;

        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(0,255,0);
        Subgraph_Low.DrawZeros = false;

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

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

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

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

```

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

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

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

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

return;
}

// Initialize the data in the destination subgraphs to 0 before looping through the charts.

for (int DataIndex = sc.UpdateStartIndex; DataIndex < sc.ArraySize; DataIndex++)
{
    // Loop through the subgraphs
    for (int SubgraphIndex = 0; SubgraphIndex < 6; SubgraphIndex++)
    {
        sc.Subgraph[SubgraphIndex][DataIndex] = 0;
    }
}

// Loop through the charts
for (int ChartIndex = 1; ChartIndex < 201; ChartIndex++)
{
    SCGraphData ReferenceArrays;
    sc.GetChartData(-ChartIndex, ReferenceArrays);

    if (ReferenceArrays[SC_OPEN].GetArraySize() < 1)
        continue;

    // Loop through the data indexes
    for (int DataIndex = sc.UpdateStartIndex; DataIndex < sc.ArraySize; DataIndex++)
    {
        // Get the matching index
        int RefDataIndex = sc.GetNearestMatchForDateTimeIndex(ChartIndex, DataIndex);

        // Loop through the subgraphs
        for (int SubgraphIndex = 0; SubgraphIndex < 6; SubgraphIndex++)
        {
            // Add in the value of the current chart
            sc.Subgraph[SubgraphIndex][DataIndex]
            += ReferenceArrays[SubgraphIndex][RefDataIndex];
        }
        sc.CalculateOHLCAverages(DataIndex);
    }
}

```

```

return;

}

/*=====*/

SCSFExport scsf_MultiplyAllChartsBar(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_Volume = sc.Subgraph[4];
    SCSubgraphRef Subgraph_OpenInterest = sc.Subgraph[5];
    SCSubgraphRef Subgraph_OHLCAvg = sc.Subgraph[6];
    SCSubgraphRef Subgraph_HLCAvg = sc.Subgraph[7];
    SCSubgraphRef Subgraph_HLAvg = sc.Subgraph[8];
    SCSubgraphRef Subgraph_BidVol = sc.Subgraph[9];
    SCSubgraphRef Subgraph_AskVol = sc.Subgraph[10];

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

        sc.GraphName = "Multiply All Charts" ;

        sc.GraphDrawType = GDT_OHLCBAR;
        sc.StandardChartHeader = 1;

        sc.ValueFormat = 2;

        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(0,255,0);
        Subgraph_Low.DrawZeros = false;

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

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

        Subgraph_OpenInterest.Name = "# of Trades / OI";
        Subgraph_OpenInterest.DrawStyle = DRAWSTYLE_IGNORE;
        Subgraph_OpenInterest.PrimaryColor = RGB(127,0,255);
        Subgraph_OpenInterest.DrawZeros = false;

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

```

```

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

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

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

return;
}

// Initialize the data in the destination subgraphs to 1.0 before looping through the charts.
for (int DataIndex = sc.UpdateStartIndex; DataIndex < sc.ArraySize; DataIndex++)
{
    // Loop through the subgraphs
    for (int SubgraphIndex = 0; SubgraphIndex < 6; SubgraphIndex++)
    {
        sc.Subgraph[SubgraphIndex][DataIndex] = 1.0;
    }
}

// Loop through the charts. Up to 200.
for (int ChartIndex = 1; ChartIndex < 201; ChartIndex++)
{
    SCGraphData ReferenceArrays;
    sc.GetChartData(-ChartIndex, ReferenceArrays);

    if (ReferenceArrays[SC_OPEN].GetArraySize() < 1)
        continue;

    // Loop through the data indexes
    for (int DataIndex = sc.UpdateStartIndex; DataIndex < sc.ArraySize; DataIndex++)
    {
        // Get the matching index
        int RefDataIndex = sc.GetNearestMatchForDateTimeIndex(ChartIndex, DataIndex);

        // Loop through the subgraphs
        for (int SubgraphIndex = 0; SubgraphIndex < 6; SubgraphIndex++)
        {
            // Add in the value of the current chart
            sc.Subgraph[SubgraphIndex][DataIndex]
                *= ReferenceArrays[SubgraphIndex][RefDataIndex];
        }
        sc.CalculateOHLCAverages(DataIndex);
    }
}

```

```

return;
}

/* ===== */
SCSFExport scsf_SumChartsFromList(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_Volume = sc.Subgraph[4];
    SCSubgraphRef Subgraph_OpenInterest = sc.Subgraph[5];
    SCSubgraphRef Subgraph_OHLCAvg = sc.Subgraph[6];
    SCSubgraphRef Subgraph_HLCAvg = sc.Subgraph[7];
    SCSubgraphRef Subgraph_HLAv = sc.Subgraph[8];
    SCSubgraphRef Subgraph_BidVol = sc.Subgraph[9];
    SCSubgraphRef Subgraph_AskVol = sc.Subgraph[10];

    SCInputRef Input_Divisor = sc.Input[0];
    SCInputRef Input_SyncCharts = sc.Input[1];

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

        sc.GraphName = "Sum Charts From List" ;

        sc.StudyDescription = "Sum Charts From List.";

        sc.GraphDrawType = GDT_OHLCBAR;
        sc.StandardChartHeader = 1;

        sc.ValueFormat = 2;

        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(0,255,0);
        Subgraph_Low.DrawZeros = false;

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

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

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

```

```

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

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

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

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

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

Input_Divisor.Name = "Divisor. Use 0 for divisor equal to number of charts.";
Input_Divisor.SetFloat(1);

Input_SyncCharts.Name = "Synchronize charts";
Input_SyncCharts.SetYesNo(false);

sc.TextInputName = "List of Chart Numbers (comma separated)";

return;
}

// Do data processing

int Sync = 1;

if(Input_SyncCharts.GetYesNo())
    Sync = -1;

// Make a copy of the list of chart numbers from the text input
SCString ChartNumberList = sc.TextInput;

// Get each of the chart numbers (from the text input) and put them into a
// vector.
std::vector<int> ChartNumbers;
std::vector<float> ChartMultipliers;

char* TokenContext = NULL;
char* Token = strtok_s((char *)ChartNumberList.GetChars(), ",", &TokenContext);
while (Token != NULL)
{
    ChartNumbers.push_back(atoi(Token));

    float Multiplier = 0.0;
    char* FoundMultiplier = strchr(Token, '*');
    if (FoundMultiplier != NULL)
        Multiplier = static_cast<float>(atof(FoundMultiplier + 1));

    if (Multiplier == 0.0)
        Multiplier = 1.0;
}

```

```

ChartMultipliers.push_back(Multiplier);

// Get the next chart number
Token = strtok_s(NULL, ",", &TokenContext);
}

float DivisorValue = Input_Divisor.GetFloat();
if (DivisorValue == 0)
    DivisorValue = static_cast<float>(ChartNumbers.size());

int Index = 0;
for (std::vector<int>::iterator Iter_ChartNumber = ChartNumbers.begin(); Iter_ChartNumber != ChartNumbers.end();
++Iter_ChartNumber, ++Index)
{
    int ChartNumber = *Iter_ChartNumber;
    float ChartMultiplier = ChartMultipliers[Index];

    SCGraphData ReferenceArrays;
    sc.GetChartData(ChartNumber*Sync, ReferenceArrays);

    if (ReferenceArrays[SC_OPEN].GetArraySize() < 1)
        continue;

    // Loop through the data indexes
    for (int DataIndex = sc.UpdateStartIndex; DataIndex < sc.ArraySize; DataIndex++)
    {
        // Get the matching index
        int RefDataIndex = sc.GetNearestMatchForDateTimeIndex(ChartNumber, DataIndex);

        // Loop through the subgraphs
        for (int SubgraphIndex = 0; SubgraphIndex <= SC_LAST; SubgraphIndex++)
        {
            // Add in the value of the current chart
            if (Index == 0)
            {
                sc.Subgraph[SubgraphIndex][DataIndex] = ReferenceArrays[SubgraphIndex][RefDataIndex] *
ChartMultiplier / DivisorValue;
            }
            else
            {
                sc.Subgraph[SubgraphIndex][DataIndex] += ReferenceArrays[SubgraphIndex][RefDataIndex] *
ChartMultiplier / DivisorValue;
            }
        }
    }

    sc.Subgraph[SC_HIGH][DataIndex] = max(sc.Subgraph[SC_OPEN][DataIndex],
        max(sc.Subgraph[SC_HIGH][DataIndex],
        max(sc.Subgraph[SC_LOW][DataIndex], sc.Subgraph[SC_LAST][DataIndex])
        );

    sc.Subgraph[SC_LOW][DataIndex] = min(sc.Subgraph[SC_OPEN][DataIndex],
        min(sc.Subgraph[SC_HIGH][DataIndex],
        min(sc.Subgraph[SC_LOW][DataIndex], sc.Subgraph[SC_LAST][DataIndex])
        );

    // Loop through the subgraphs
    for (int SubgraphIndex = SC_VOLUME; SubgraphIndex <= SC_NUM_TRADES; SubgraphIndex++)
    {
        // Add in the value of the current chart

```

```

        if (Index == 0)
            sc.Subgraph[SubgraphIndex][DataIndex] = ReferenceArrays[SubgraphIndex][RefDataIndex] * (1.0f /
ChartMultiplier) / DivisorValue;
        else
            sc.Subgraph[SubgraphIndex][DataIndex] += ReferenceArrays[SubgraphIndex][RefDataIndex] * (1.0f /
ChartMultiplier) / DivisorValue;
    }

    sc.CalculateOHLCAverages(DataIndex);
}
}
}

/* ===== */
-----*/
SCSFExport scsf_BidAskVolumeRatio(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_VolRatioAvg = sc.Subgraph[0];
    SCSubgraphRef Subgraph_ZeroLine = sc.Subgraph[1];

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

    SCInputRef Input_MALength = sc.Input[0];
    SCInputRef Input_MAType = sc.Input[1];
    SCInputRef Input_DifferenceCalculation = sc.Input[2];

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

        sc.GraphName = "Bid Ask Volume Ratio";

        sc.StudyDescription = "Bid Ask Volume Ratio";

        sc.AutoLoop = 0;

        Subgraph_VolRatioAvg.Name = "Vol Ratio Avg";
        Subgraph_VolRatioAvg.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_VolRatioAvg.PrimaryColor = RGB(0,255,0);
        Subgraph_VolRatioAvg.DrawZeros = false;

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

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

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

        Input_DifferenceCalculation.Name = "Difference Calculation Method";
        Input_DifferenceCalculation.SetCustomInputStrings("Ask Volume - Bid Volume;Bid Volume - Ask Volume");
        Input_DifferenceCalculation.SetCustomInputIndex(0);

        return;
    }

    // Do data processing
    for (int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize; BarIndex++)
    {
        const float AskVolume = sc.AskVolume[BarIndex];

```



```

const float BidVolume = sc.BidVolume[BarIndex];

float TotalVol = AskVolume + BidVolume;

if (Input_DifferenceCalculation.GetIndex() == 0 && TotalVol > 0)
    Array_ExtraArrayRatio[BarIndex] = 100.0f * (AskVolume - BidVolume) / TotalVol;
else if (Input_DifferenceCalculation.GetIndex() == 1 && TotalVol > 0)
    Array_ExtraArrayRatio[BarIndex] = 100.0f * (BidVolume - AskVolume) / TotalVol;
else
    Array_ExtraArrayRatio[BarIndex] = 0.0f;

    sc.MovingAverage(Array_ExtraArrayRatio, Subgraph_VolRatioAvg, Input_MAType.GetMovAvgType(), BarIndex,
Input_MALength.GetInt());
}

}

/*=====
-----*/
SCSFExport scsf_StudyAngle(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Angle = sc.Subgraph[0];
    SCSubgraphRef Subgraph_ZeroLine = sc.Subgraph[1];

    SCInputRef Input_Data = sc.Input[0];
    SCInputRef Input_Length = sc.Input[1];
    SCInputRef Input_ValuePerPoint = sc.Input[2];
    SCInputRef Input_SkipCalculationAtStartOfTradingDay = sc.Input[3];

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

        sc.GraphName = "Study Angle";
        sc.AutoLoop = 1;

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

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

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

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

        Input_ValuePerPoint.Name = "Value Per Point";
        Input_ValuePerPoint.SetFloat(1.0f);

        Input_SkipCalculationAtStartOfTradingDay.Name = "Skip Calculation at Start of Trading Day";
        Input_SkipCalculationAtStartOfTradingDay.SetYesNo(false);

        return;
    }

    int Length = Input_Length.GetInt();

    sc.DataStartIndex = Length;

```

```

if (Input_ValuePerPoint.GetFloat() == 0.0f)
    Input_ValuePerPoint.SetFloat(0.01f);

int &r_IndexOfNewDay = sc.GetPersistentInt(1);
if (sc.IsFullRecalculation && sc.Index == 0)
    r_IndexOfNewDay = -1;

// Do data processing
bool SkipCalculation = false;

if (Input_SkipCalculationAtStartOfTradingDay.GetYesNo())
{
    if (sc.IsNewTradingDay(sc.Index))
    {
        r_IndexOfNewDay = sc.Index;
    }

    SkipCalculation = r_IndexOfNewDay == sc.Index || (sc.Index - Length) < r_IndexOfNewDay ;
}

if(!SkipCalculation)
{
    SCFloatArrayRef InData = sc.BaseData[Input_Data.GetInputDataIndex()];

    float BackValue = InData[sc.Index - Length];
    float CurrentValue = InData[sc.Index];

    float PointChange = (CurrentValue - BackValue) / Input_ValuePerPoint.GetFloat();

    Subgraph_Angle[sc.Index] = static_cast<float>(atan2(static_cast<double>(PointChange), static_cast<double>
(Length)) * 180.0 / M_PI);
}
else
{
    Subgraph_Angle[sc.Index] = 0;
}
}

/*=====*/
SCSFExport scsf_DetrendedOscillator(SCStudyInterfaceRef sc)
{
    // Section 1 - Set the configuration variables

    SCSubgraphRef Subgraph_DPO = sc.Subgraph[0];

    SCSubgraphRef Subgraph_Overbought = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Oversold = sc.Subgraph[2];

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

    SCInputRef Input_Length = sc.Input[0];
    SCInputRef Input_Overbought = sc.Input[1];
    SCInputRef Input_Oversold = sc.Input[2];
    SCInputRef Input_MovingAverageType = sc.Input[3];

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

        sc.GraphName = "Detrended Oscillator";
    }
}

```

```

sc.StudyDescription = "Detrended Oscillator.";

sc.AutoLoop = 1; // true

Subgraph_DPO.Name = "DO";
Subgraph_DPO.DrawStyle = DRAWSTYLE_LINE;
Subgraph_DPO.PrimaryColor = RGB(0,255,0);

Subgraph_Overbought.Name = "Level 1";
Subgraph_Overbought.DrawStyle = DRAWSTYLE_LINE;
Subgraph_Overbought.PrimaryColor = RGB(255,255,0);

Subgraph_Oversold.Name = "Level 2";
Subgraph_Oversold.DrawStyle = DRAWSTYLE_LINE;
Subgraph_Oversold.PrimaryColor = RGB(255,255,0);

Input_Length.Name = "Length";
Input_Length.SetInt(20);

Input_Overbought.Name = "Overbought Level";
Input_Overbought.SetFloat(0.5);

Input_Oversold.Name = "Oversold Level";
Input_Oversold.SetFloat(-0.5);

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

return;
}

// Section 2 - Do data processing here

sc.DataStartIndex = Input_Length.GetInt();

int N = Input_Length.GetInt()/2;

Subgraph_Overbought[sc.Index] = Input_Overbought.GetFloat();
Subgraph_Oversold[sc.Index] = Input_Oversold.GetFloat();

sc.MovingAverage(sc.Close, Array_MA, Input_MovingAverageType.GetMovAvgType() , sc.Index, N);

int N2 = (N/2) + 1;

Subgraph_DPO[sc.Index] = (sc.Close[sc.Index] - Array_MA[sc.Index - N2]);
}

/*=====*/
SCSFExport scsf_RangeBarPredictor(SCStudyInterfaceRef sc)
{
    // Section 1 - Set the configuration variables

    SCSubgraphRef Subgraph_High = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Low = sc.Subgraph[1];

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

        sc.GraphName = "Range Bar Predictor";

        sc.StudyDescription = "Range Bar Predictor.";

```

```

sc.GraphRegion = 0;

sc.AutoLoop = 0; // manual looping

Subgraph_High.Name = "Predicted High";
Subgraph_High.DrawStyle = DRAWSTYLE_DASH;
Subgraph_High.PrimaryColor = RGB(0, 255, 0);
Subgraph_High.LineWidth = 2;
Subgraph_High.DrawZeros = false;

Subgraph_Low.Name = "Predicted Low";
Subgraph_Low.DrawStyle = DRAWSTYLE_DASH;
Subgraph_Low.PrimaryColor = RGB(255, 0, 0);
Subgraph_Low.LineWidth = 2;
Subgraph_Low.DrawZeros = false;

return;
}

// Section 2 - Do data processing here
for (int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize; BarIndex++)
{
    if (BarIndex == sc.ArraySize - 1)
    {
        n_ACSIL::s_BarPeriod BarPeriodParameters;
        sc.GetBarPeriodParameters(BarPeriodParameters);

        if (!(BarPeriodParameters.IntradayChartBarPeriodType == IBPT_RANGE_IN_TICKS_FILL_GAPS
            || BarPeriodParameters.IntradayChartBarPeriodType ==
IBPT_RANGE_IN_TICKS_NEW_BAR_ON_RANGE_MET_OPEN_EQUALS_PRIOR_CLOSE
            || BarPeriodParameters.IntradayChartBarPeriodType ==
IBPT_RANGE_IN_TICKS_NEWBAR_ON_RANGEMET
            || BarPeriodParameters.IntradayChartBarPeriodType == IBPT_RANGE_IN_TICKS_OPEN_EQUAL_CLOSE
            || BarPeriodParameters.IntradayChartBarPeriodType == IBPT_RANGE_IN_TICKS_STANDARD
            || BarPeriodParameters.IntradayChartBarPeriodType == IBPT_RANGE_IN_TICKS_TRUE))
        {
            return;
        }

        float RangePerBar = BarPeriodParameters.IntradayChartBarPeriodParameter1 * sc.TickSize;

        Subgraph_High[BarIndex] = sc.Low[BarIndex] + RangePerBar;
        Subgraph_Low[BarIndex] = sc.High[BarIndex] - RangePerBar;
    }
    else
    {
        Subgraph_High[BarIndex] = 0;
        Subgraph_Low[BarIndex] = 0;
    }
}
}

/*=====*/
SCSFExport scsf_RenkoBarPredictor(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_High = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Low = sc.Subgraph[1];

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

        sc.GraphName = "Renko Bar Predictor";

        sc.StudyDescription = "Renko Bar Predictor.";
    }
}

```

```

sc.GraphRegion = 0;

sc.AutoLoop = 0; // manual looping

Subgraph_High.Name = "Predicted High";
Subgraph_High.DrawStyle = DRAWSTYLE_DASH;
Subgraph_High.PrimaryColor = RGB(0,255,0);
Subgraph_High.LineWidth = 2;
Subgraph_High.DrawZeros = false;

Subgraph_Low.Name = "Predicted Low";
Subgraph_Low.DrawStyle = DRAWSTYLE_DASH;
Subgraph_Low.PrimaryColor = RGB(255,0,0);
Subgraph_Low.LineWidth = 2;
Subgraph_Low.DrawZeros = false;

return;
}

for (int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize; BarIndex++)
{
    if (BarIndex > 0 && BarIndex == sc.ArraySize - 1)
    {
        float PriorRenkoOpen = sc.BaseData[SC_RENKO_OPEN][BarIndex - 1];
        float PriorRenkoClose = sc.BaseData[SC_RENKO_CLOSE][BarIndex - 1];
        bool PriorBarUp = PriorRenkoOpen < PriorRenkoClose;

        n_ACSIL::s_BarPeriod BarPeriod;
        sc.GetBarPeriodParameters(BarPeriod);
        int RenkoTicksPerBar = BarPeriod.IntradayChartBarPeriodParameter1;
        int RenkoTrendOpenOffsetInTicks = BarPeriod.IntradayChartBarPeriodParameter2;
        int RenkoReversalOpenOffsetInTicks = BarPeriod.IntradayChartBarPeriodParameter3;

        if (BarPeriod.IntradayChartBarPeriodParameter4 <= 2)//exceeded mode
            RenkoTrendOpenOffsetInTicks -= 1;

        if (PriorBarUp)
        {
            if (BarPeriod.IntradayChartBarPeriodType != IBPT_FLEX_RENKO_IN_TICKS_INVERSE_SETTINGS)
            {
                Subgraph_High[BarIndex] = PriorRenkoClose - RenkoTrendOpenOffsetInTicks * sc.TickSize +
RenkoTicksPerBar * sc.TickSize;
                Subgraph_Low[BarIndex] = PriorRenkoClose + RenkoReversalOpenOffsetInTicks * sc.TickSize - 2.0f *
RenkoTicksPerBar * sc.TickSize;
            }
            else
            {
                Subgraph_High[BarIndex] = PriorRenkoClose + RenkoTrendOpenOffsetInTicks * sc.TickSize;
                Subgraph_Low[BarIndex] = PriorRenkoClose - RenkoReversalOpenOffsetInTicks * sc.TickSize;
            }
        }
        else
        {
            if (BarPeriod.IntradayChartBarPeriodType != IBPT_FLEX_RENKO_IN_TICKS_INVERSE_SETTINGS)
            {
                Subgraph_High[BarIndex] = PriorRenkoClose - RenkoReversalOpenOffsetInTicks * sc.TickSize + 2.0f *
RenkoTicksPerBar * sc.TickSize;
                Subgraph_Low[BarIndex] = PriorRenkoClose + RenkoTrendOpenOffsetInTicks * sc.TickSize -
RenkoTicksPerBar * sc.TickSize;
            }
            else
            {
                Subgraph_High[BarIndex] = PriorRenkoClose + RenkoReversalOpenOffsetInTicks * sc.TickSize;

```

```

        Subgraph_Low[BarIndex] = PriorRenkoClose - RenkoTrendOpenOffsetInTicks * sc.TickSize;
    }
}
}
else
{
    Subgraph_High[BarIndex] = 0;
    Subgraph_Low[BarIndex] = 0;
}
}
}

/*=====*/
SCSFExport scsf_FastStochastic(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_PercentK = sc.Subgraph[0];
    SCSubgraphRef Subgraph_PercentD = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Line1 = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Line2 = sc.Subgraph[3];

    SCInputRef Input_Unused2 = sc.Input[2];
    SCInputRef Input_KLength = sc.Input[3];
    SCInputRef Input_DLength = sc.Input[4];
    SCInputRef Input_Line1Value = sc.Input[5];
    SCInputRef Input_Line2Value = sc.Input[6];
    SCInputRef Input_MovAvgType = sc.Input[7];
    SCInputRef Input_DataHigh = sc.Input[8];
    SCInputRef Input_DataLow = sc.Input[9];
    SCInputRef Input_DataLast = sc.Input[10];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Stochastic - Fast";

        sc.ValueFormat = 2;

        Subgraph_PercentK.Name = "%K";
        Subgraph_PercentK.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_PercentK.PrimaryColor = RGB(0,255,0);
        Subgraph_PercentK.DrawZeros = true;

        Subgraph_PercentD.Name = "%D";
        Subgraph_PercentD.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_PercentD.PrimaryColor = RGB(255,0,255);
        Subgraph_PercentD.DrawZeros = true;

        Subgraph_Line1.Name = "Line1";
        Subgraph_Line1.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Line1.PrimaryColor = RGB(255,255,0);
        Subgraph_Line1.DrawZeros = true;

        Subgraph_Line2.Name = "Line2";
        Subgraph_Line2.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Line2.PrimaryColor = RGB(255,127,0);
        Subgraph_Line2.DrawZeros = true;

        Input_Unused2.Name = "";
        Input_Unused2.SetInt(10);
        Input_Unused2.SetIntLimits(1,MAX_STUDY_LENGTH);

        Input_KLength.Name = "%K Length";
        Input_KLength.SetInt(10);
        Input_KLength.SetIntLimits(1,MAX_STUDY_LENGTH);

        Input_DLength.Name = "%D Length";

```

```

    Input_DLength.SetInt(3);
    Input_DLength.SetIntLimits(1,MAX_STUDY_LENGTH);

    Input_Line1Value.Name = "Line1 Value";
    Input_Line1Value.SetFloat(70);

    Input_Line2Value.Name = "Line2 Value";
    Input_Line2Value.SetFloat(30);

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

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

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

    Input_DataLast.Name = "Input Data for Last";
    Input_DataLast.SetInputDataIndex(SC_LAST);

    sc.AutoLoop = true;
    return;
}

sc.DataStartIndex = Input_KLength.GetInt() + Input_DLength.GetInt();

sc.Stochastic(
    sc.BaseData[Input_DataHigh.GetInputDataIndex()],
    sc.BaseData[Input_DataLow.GetInputDataIndex()],
    sc.BaseData[Input_DataLast.GetInputDataIndex()],
    Subgraph_PercentK, // Data member is Fast %K
    Input_KLength.GetInt(),
    Input_DLength.GetInt(),
    0,
    Input_MovAvgType.GetMovAvgType()
);

Subgraph_PercentD[sc.Index] = Subgraph_PercentK.Arrays[0][sc.Index]; // Fast %D (Slow %K)

Subgraph_Line1[sc.Index] = Input_Line1Value.GetFloat();
Subgraph_Line2[sc.Index] = Input_Line2Value.GetFloat();
}

/*=====*/
SCSFExport scsf_KDFastStochastic(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_PercentK = sc.Subgraph[0];
    SCSubgraphRef Subgraph_PercentD = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Line1 = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Line2 = sc.Subgraph[3];

    SCInputRef Input_Unused2 = sc.Input[2];
    SCInputRef Input_KLength = sc.Input[3];
    SCInputRef Input_DLength = sc.Input[4];
    SCInputRef Input_Line1Value = sc.Input[5];
    SCInputRef Input_Line2Value = sc.Input[6];
    SCInputRef Input_MovAvgType = sc.Input[7];
    SCInputRef Input_DataHigh = sc.Input[8];
    SCInputRef Input_DataLow = sc.Input[9];
    SCInputRef Input_DataLast = sc.Input[10];

    if (sc.SetDefaults)

```

```

{
    sc.GraphName = "KD - Fast";

    sc.ValueFormat = 2;

    Subgraph_PercentK.Name = "%K";
    Subgraph_PercentK.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_PercentK.PrimaryColor = RGB(0,255,0);
    Subgraph_PercentK.DrawZeros = true;

    Subgraph_PercentD.Name = "%D";
    Subgraph_PercentD.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_PercentD.PrimaryColor = RGB(255,0,255);
    Subgraph_PercentD.DrawZeros = true;

    Subgraph_Line1.Name = "Line1";
    Subgraph_Line1.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_Line1.PrimaryColor = RGB(255,255,0);
    Subgraph_Line1.DrawZeros = true;

    Subgraph_Line2.Name = "Line2";
    Subgraph_Line2.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_Line2.PrimaryColor = RGB(255,127,0);
    Subgraph_Line2.DrawZeros = true;

    Input_Unused2.Name = "";
    Input_Unused2.SetInt(10);
    Input_Unused2.SetIntLimits(1,MAX_STUDY_LENGTH);

    Input_KLength.Name = "%K Length";
    Input_KLength.SetInt(10);
    Input_KLength.SetIntLimits(1,MAX_STUDY_LENGTH);

    Input_DLength.Name = "%D Length";
    Input_DLength.SetInt(3);
    Input_DLength.SetIntLimits(1,MAX_STUDY_LENGTH);

    Input_Line1Value.Name = "Line1 Value";
    Input_Line1Value.SetFloat(70);

    Input_Line2Value.Name = "Line2 Value";
    Input_Line2Value.SetFloat(30);

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

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

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

    Input_DataLast.Name = "Input Data for Last";
    Input_DataLast.SetInputDataIndex(SC_LAST);

    sc.AutoLoop = true;
    return;
}

sc.DataStartIndex = Input_KLength.GetInt() + Input_DLength.GetInt();

sc.Stochastic(
    sc.BaseData[Input_DataHigh.GetInputDataIndex()],

```



```

    sc.BaseData[Input_DataLow.GetInputDataIndex()],
    sc.BaseData[Input_DataLast.GetInputDataIndex()],
    Subgraph_PercentK, // Data member is Fast %K
    Input_KLength.GetInt(),
    Input_DLength.GetInt(),
    0,
    Input_MovAvgType.GetMovAvgType()
);

Subgraph_PercentD[sc.Index] = Subgraph_PercentK.Arrays[0][sc.Index]; // Fast %D (Slow %K)

Subgraph_Line1[sc.Index] = Input_Line1Value.GetFloat();
Subgraph_Line2[sc.Index] = Input_Line2Value.GetFloat();
}

/*=====*/
SCSFExport scsf_SlowStochastic(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_SlowK = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SlowD = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Line1 = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Line2 = sc.Subgraph[3];
    SCSubgraphRef Subgraph_Temp4 = sc.Subgraph[4];

    SCInputRef Input_FastKLength = sc.Input[2];
    SCInputRef Input_FastDLength = sc.Input[3];
    SCInputRef Input_SlowDLength = sc.Input[4];
    SCInputRef Input_Line1Value = sc.Input[5];
    SCInputRef Input_Line2Value = sc.Input[6];
    SCInputRef Input_MovAvgType = sc.Input[7];
    SCInputRef Input_DataHigh = sc.Input[8];
    SCInputRef Input_DataLow = sc.Input[9];
    SCInputRef Input_DataLast = sc.Input[10];
    SCInputRef Input_UpdateFlag = sc.Input[11];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Stochastic - Slow";

        sc.ValueFormat = 2;

        Subgraph_SlowK.Name = "Slow %K";
        Subgraph_SlowK.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_SlowK.PrimaryColor = RGB(0,255,0);
        Subgraph_SlowK.DrawZeros = true;

        Subgraph_SlowD.Name = "Slow %D";
        Subgraph_SlowD.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_SlowD.PrimaryColor = RGB(255,0,255);
        Subgraph_SlowD.DrawZeros = true;

        Subgraph_Line1.Name = "Line1";
        Subgraph_Line1.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Line1.PrimaryColor = RGB(255,255,0);
        Subgraph_Line1.DrawZeros = true;

        Subgraph_Line2.Name = "Line2";
        Subgraph_Line2.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Line2.PrimaryColor = RGB(255,127,0);
        Subgraph_Line2.DrawZeros = true;

        Input_FastKLength.Name = "Fast %K Length";
        Input_FastKLength.SetInt(10);
        Input_FastKLength.SetIntLimits(1,MAX_STUDY_LENGTH);

```

```

Input_FastDLength.Name = "Fast %D Length (Slow %K)";
Input_FastDLength.SetInt(3);
Input_FastDLength.SetIntLimits(1,MAX_STUDY_LENGTH);

Input_SlowDLength.Name = "Slow %D Length";
Input_SlowDLength.SetInt(3);
Input_SlowDLength.SetIntLimits(1,MAX_STUDY_LENGTH);

Input_Line1Value.Name = "Line1 Value";
Input_Line1Value.SetFloat(70);

Input_Line2Value.Name = "Line2 Value";
Input_Line2Value.SetFloat(30);

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

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

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

Input_DataLast.Name = "Input Data for Last";
Input_DataLast.SetInputDataIndex(SC_LAST);

Input_UpdateFlag.SetInt(1); //update flag

sc.AutoLoop = true;
return;
}

sc.DataStartIndex = Input_FastKLength.GetInt() + Input_FastDLength.GetInt() + Input_SlowDLength.GetInt();

sc.Stochastic(
    sc.BaseData[Input_DataHigh.GetInputDataIndex()],
    sc.BaseData[Input_DataLow.GetInputDataIndex()],
    sc.BaseData[Input_DataLast.GetInputDataIndex()],
    Subgraph_Temp4, // Data member is Fast %K
    Input_FastKLength.GetInt(),
    Input_FastDLength.GetInt(),
    Input_SlowDLength.GetInt(),
    Input_MovAvgType.GetMovAvgType()
);

Subgraph_SlowK[sc.Index] = Subgraph_Temp4.Arrays[0][sc.Index];
Subgraph_SlowD[sc.Index] = Subgraph_Temp4.Arrays[1][sc.Index];

Subgraph_Line1[sc.Index] = Input_Line1Value.GetFloat();
Subgraph_Line2[sc.Index] = Input_Line2Value.GetFloat();

return;

}

/*=====*/
SCSFExport scsf_KDSlowStochastic(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_SlowK = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SlowD = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Line1 = sc.Subgraph[2];

```

```

SCSubgraphRef Subgraph_Line2 = sc.Subgraph[3];
SCSubgraphRef Subgraph_Temp4 = sc.Subgraph[4];

SCInputRef Input_FastKLength = sc.Input[2];
SCInputRef Input_FastDLength = sc.Input[3];
SCInputRef Input_SlowDLength = sc.Input[4];
SCInputRef Input_Line1Value = sc.Input[5];
SCInputRef Input_Line2Value = sc.Input[6];
SCInputRef Input_MovAvgType = sc.Input[7];
SCInputRef Input_DataHigh = sc.Input[8];
SCInputRef Input_DataLow = sc.Input[9];
SCInputRef Input_DataLast = sc.Input[10];

if (sc.SetDefaults)
{
    sc.GraphName = "KD - Slow";

    sc.ValueFormat = 2;

    Subgraph_SlowK.Name = "Slow %K";
    Subgraph_SlowK.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_SlowK.PrimaryColor = RGB(0,255,0);
    Subgraph_SlowK.DrawZeros = true;

    Subgraph_SlowD.Name = "Slow %D";
    Subgraph_SlowD.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_SlowD.PrimaryColor = RGB(255,0,255);
    Subgraph_SlowD.DrawZeros = true;

    Subgraph_Line1.Name = "Line1";
    Subgraph_Line1.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_Line1.PrimaryColor = RGB(255,255,0);
    Subgraph_Line1.DrawZeros = true;

    Subgraph_Line2.Name = "Line2";
    Subgraph_Line2.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_Line2.PrimaryColor = RGB(255,127,0);
    Subgraph_Line2.DrawZeros = true;

    Input_FastKLength.Name = "Fast %K Length";
    Input_FastKLength.SetInt(10);
    Input_FastKLength.SetIntLimits(1,MAX_STUDY_LENGTH);

    Input_FastDLength.Name = "Fast %D Length (Slow %K)";
    Input_FastDLength.SetInt(3);
    Input_FastDLength.SetIntLimits(1,MAX_STUDY_LENGTH);

    Input_SlowDLength.Name = "Slow %D Length";
    Input_SlowDLength.SetInt(3);
    Input_SlowDLength.SetIntLimits(1,MAX_STUDY_LENGTH);

    Input_Line1Value.Name = "Line1 Value";
    Input_Line1Value.SetFloat(70);

    Input_Line2Value.Name = "Line2 Value";
    Input_Line2Value.SetFloat(30);

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

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

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

```

```

Input_DataLast.Name = "Input Data for Last";
Input_DataLast.SetInputDataIndex(SC_LAST);

sc.AutoLoop = true;
return;
}

sc.DataStartIndex = Input_FastKLength.GetInt() + Input_FastDLength.GetInt() + Input_SlowDLength.GetInt();

sc.Stochastic(
    sc.BaseData[Input_DataHigh.GetInputDataIndex()],
    sc.BaseData[Input_DataLow.GetInputDataIndex()],
    sc.BaseData[Input_DataLast.GetInputDataIndex()],
    Subgraph_Temp4, // Data member is Fast %K
    Input_FastKLength.GetInt(),
    Input_FastDLength.GetInt(),
    Input_SlowDLength.GetInt(),
    Input_MovAvgType.GetMovAvgType()
);

Subgraph_SlowK[sc.Index] = Subgraph_Temp4.Arrays[0][sc.Index];
Subgraph_SlowD[sc.Index] = Subgraph_Temp4.Arrays[1][sc.Index];

Subgraph_Line1[sc.Index] = Input_Line1Value.GetFloat();
Subgraph_Line2[sc.Index] = Input_Line2Value.GetFloat();

return;

}
/*=====*/
SCSFExport scsf_DoubleTrix(SCStudyInterfaceRef sc)
{
    // Section 1 - Set the configuration variables

    SCInputRef Input_TRIXLength = sc.Input[0];
    SCInputRef Input_SigLineXMALen = sc.Input[1];

    SCFloatArrayRef Array_EMA1 = sc.Subgraph[0].Arrays[0];
    SCFloatArrayRef Array_EMA2 = sc.Subgraph[0].Arrays[1];
    SCFloatArrayRef Array_EMA3 = sc.Subgraph[0].Arrays[2];

    SCSubgraphRef Subgraph_TRIXLine = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SignalLine = sc.Subgraph[1];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Double Trix";

        sc.StudyDescription = "DoubleTrix.";

        sc.AutoLoop = 1; // true

        Subgraph_TRIXLine.Name = "TRIXLine";
        Subgraph_TRIXLine.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_TRIXLine.PrimaryColor = RGB(0,255,0);
        Subgraph_TRIXLine.LineWidth = 1;

        Subgraph_SignalLine.Name = "SignalLine";
        Subgraph_SignalLine.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_SignalLine.PrimaryColor = RGB(255,0,0);
        Subgraph_SignalLine.LineWidth = 1;
    }
}

```

```

Input_TRIXLength.Name = "TRIXLength";
Input_TRIXLength.SetInt(5);

Input_SigLineXMALen.Name = "SigLineXMALen";
Input_SigLineXMALen.SetInt(3);

return;
}

// Section 2 - Do data processing here

int TRIXLength = Input_TRIXLength.GetInt();
int SigLineXMALen = Input_SigLineXMALen.GetInt();

sc.ExponentialMovAvg(sc.Close,Array_EMA1,TRIXLength);
sc.ExponentialMovAvg(Array_EMA1,Array_EMA2,TRIXLength);
sc.ExponentialMovAvg(Array_EMA2,Array_EMA3,TRIXLength);

if(Array_EMA3[sc.Index-1] != 0)
    Subgraph_TRIXLine[sc.Index] = 10 * ( Array_EMA3[sc.Index] - Array_EMA3[sc.Index-1] ) / Array_EMA3[sc.Index-1]
;
else
    Subgraph_TRIXLine[sc.Index] = 0;

sc.ExponentialMovAvg(Subgraph_TRIXLine,Subgraph_SignalLine,SigLineXMALen);
}

/*=====*/
SCSFExport scsf_BollingerSqueeze2(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_BollingerSqueeze = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SqueezeZeros = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Keltner = sc.Subgraph[2];
    SCSubgraphRef Subgraph_BB = sc.Subgraph[3];
    SCSubgraphRef Subgraph_AverageTrueRange = sc.Subgraph[4];
    SCSubgraphRef Subgraph_Momentum = sc.Subgraph[5];
    SCSubgraphRef Subgraph_TopBollingerBand = sc.Subgraph[6];
    SCSubgraphRef Subgraph_BottomBollingerBand = sc.Subgraph[7];
    SCSubgraphRef Subgraph_TopKeltnerBand = sc.Subgraph[8];
    SCSubgraphRef Subgraph_BottomKeltnerBand = sc.Subgraph[9];

    SCInputRef Input_Data = sc.Input[0];
    SCInputRef Input_MAType = sc.Input[1];
    SCInputRef Input_KeltnerLength = sc.Input[2];
    SCInputRef Input_KeltnerTRAvgLength = sc.Input[3];
    SCInputRef Input_KeltnerMultiplier = sc.Input[4];
    SCInputRef Input_BollingerLength = sc.Input[5];
    SCInputRef Input_BollingerBandsMult = sc.Input[6];
    SCInputRef Input_MomentumLength = sc.Input[7];

    if (sc.SetDefaults)
    {
        sc.GraphName="Bollinger Squeeze 2";
        sc.StudyDescription="Bollinger Squeeze 2";

        // Subgraphs
        Subgraph_BollingerSqueeze.Name = "Momentum Avg";
        Subgraph_BollingerSqueeze.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_BollingerSqueeze.PrimaryColor = RGB(0,255,0);
        Subgraph_BollingerSqueeze.SecondaryColor = RGB(255,0,0);
        Subgraph_BollingerSqueeze.SecondaryColorUsed = true;
    }
}

```

```

Subgraph_SqueezeZeros.Name = "Squeeze Indicator";
Subgraph_SqueezeZeros.DrawStyle = DRAWSTYLE_POINT;
Subgraph_SqueezeZeros.LineWidth = 3;
Subgraph_SqueezeZeros.PrimaryColor = RGB(0,255,0);
Subgraph_SqueezeZeros.SecondaryColor = RGB(255,0,0);
Subgraph_SqueezeZeros.SecondaryColorUsed = true;

Subgraph_Momentum.Name = "Momentum";
Subgraph_Momentum.DrawStyle = DRAWSTYLE_IGNORE;

Subgraph_Keltner.Name = "Keltner Average";
Subgraph_Keltner.DrawStyle = DRAWSTYLE_IGNORE;

Subgraph_AverageTrueRange.Name = "ATR";
Subgraph_AverageTrueRange.DrawStyle = DRAWSTYLE_IGNORE;

Subgraph_TopBollingerBand.Name = "Top Bollinger Band";
Subgraph_TopBollingerBand.DrawStyle = DRAWSTYLE_IGNORE;

Subgraph_BottomBollingerBand.Name = "Bottom Bollinger Band";
Subgraph_BottomBollingerBand.DrawStyle = DRAWSTYLE_IGNORE;

Subgraph_TopKeltnerBand.Name = "Top Keltner Band";
Subgraph_TopKeltnerBand.DrawStyle = DRAWSTYLE_IGNORE;

Subgraph_BottomKeltnerBand.Name = "Bottom Keltner Band";
Subgraph_BottomKeltnerBand.DrawStyle = DRAWSTYLE_IGNORE;

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

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

Input_KeltnerLength.Name="Keltner Bands Length";
Input_KeltnerLength.SetInt(20);
Input_KeltnerLength.SetIntLimits(1,MAX_STUDY_LENGTH);

Input_KeltnerTRAvgLength.Name = "Keltner True Range MovAvg Length";
Input_KeltnerTRAvgLength.SetInt(20);
Input_KeltnerTRAvgLength.SetIntLimits(1,MAX_STUDY_LENGTH);

Input_KeltnerMultiplier.Name="Keltner Bands Multiplier";
Input_KeltnerMultiplier.SetFloat(2.0f);
Input_KeltnerMultiplier.SetFloatLimits(.001f, static_cast<float>(MAX_STUDY_LENGTH));

Input_BollingerLength.Name="Bollinger Bands Length";
Input_BollingerLength.SetInt(20);
Input_BollingerLength.SetIntLimits(1,MAX_STUDY_LENGTH);

Input_BollingerBandsMult.Name="Bollinger Bands Multiplier";
Input_BollingerBandsMult.SetFloat(10.0f);
Input_BollingerBandsMult.SetFloatLimits(.001f, static_cast<float>(MAX_STUDY_LENGTH));

Input_MomentumLength.Name = "Momentum Length";
Input_MomentumLength.SetInt(20);
Input_MomentumLength.SetIntLimits(1, MAX_STUDY_LENGTH);

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

return;
}

```

```

int InputData = Input_Data.GetInputDataIndex();
int MAType = Input_MAType.GetMovAvgType();
int KeltnerLength = Input_KeltnerLength.GetInt();
int KeltnerTRAVGLength = Input_KeltnerTRAVGLength.GetInt();
float KeltnerMult = Input_KeltnerMultiplier.GetFloat();
int BollingerLength = Input_BollingerLength.GetInt();
float BandsMult = Input_BollingerBandsMult.GetFloat();
int MomentumLength = Input_MomentumLength.GetInt();

sc.DataStartIndex = max(KeltnerLength, max(KeltnerTRAVGLength, max(BollingerLength, MomentumLength))) - 1;

// calculate Bollinger Bands
sc.BollingerBands(sc.BaseData[InputData], Subgraph_BB, BollingerLength, BandsMult, MAType);

// BollingerSqueezeSubgraph.Arrays[0][sc.Index] =
//   sc.Close[sc.Index] - sc.Close[sc.Index - MomentumLength];

Subgraph_Momentum[sc.Index] = sc.Close[sc.Index] - sc.Close[sc.Index - MomentumLength];

sc.ExponentialMovAvg(Subgraph_Momentum, Subgraph_BollingerSqueeze, MomentumLength);

// calculate Keltner
sc.Keltner( sc.BaseData,sc.BaseData[InputData], Subgraph_Keltner, KeltnerLength, MAType,
    KeltnerTRAVGLength, MOVAVGTYPE_WILDERS, KeltnerMult,KeltnerMult);

// calculate Average True Range: SimpleMovingAverage(True Range), use Arrays[0] to store True Range
Subgraph_AverageTrueRange.Arrays[0][sc.Index] = sc.GetTrueRange(sc.BaseData, sc.Index);
sc.SimpleMovAvg(Subgraph_AverageTrueRange.Arrays[0], Subgraph_AverageTrueRange, KeltnerLength);

Subgraph_TopKeltnerBand[sc.Index] = Subgraph_Keltner[sc.Index] + (KeltnerMult *
Subgraph_AverageTrueRange[sc.Index]);
Subgraph_BottomKeltnerBand[sc.Index] = Subgraph_Keltner[sc.Index] - (KeltnerMult *
Subgraph_AverageTrueRange[sc.Index]);
Subgraph_TopBollingerBand[sc.Index] = Subgraph_BB.Arrays[0][sc.Index];
Subgraph_BottomBollingerBand[sc.Index] = Subgraph_BB.Arrays[1][sc.Index];

if ((Subgraph_TopBollingerBand[sc.Index] > Subgraph_TopKeltnerBand[sc.Index]) &&
(Subgraph_BottomBollingerBand[sc.Index] < Subgraph_BottomKeltnerBand[sc.Index]))
{
    Subgraph_SqueezeZeros.DataColor[sc.Index] = Subgraph_SqueezeZeros.PrimaryColor;
    Subgraph_SqueezeZeros[sc.Index] = 0.0001f;
}
else
{
    Subgraph_SqueezeZeros.DataColor[sc.Index] = Subgraph_SqueezeZeros.SecondaryColor;
    Subgraph_SqueezeZeros[sc.Index] = -0.0001f;
}

if ((Subgraph_BollingerSqueeze[sc.Index] > 0) &&
(Subgraph_BollingerSqueeze[sc.Index] >= Subgraph_BollingerSqueeze[sc.Index - 1]))
{
    Subgraph_BollingerSqueeze.DataColor[sc.Index] = Subgraph_BollingerSqueeze.PrimaryColor;
}
else if ((Subgraph_BollingerSqueeze[sc.Index] > 0) &&
(Subgraph_BollingerSqueeze[sc.Index] < Subgraph_BollingerSqueeze[sc.Index-1]))
{
    Subgraph_BollingerSqueeze.DataColor[sc.Index] =
    RGB(GetRValue(Subgraph_BollingerSqueeze.PrimaryColor)/2,
    GetGValue(Subgraph_BollingerSqueeze.PrimaryColor)/2,
    GetBValue(Subgraph_BollingerSqueeze.PrimaryColor)/2);
}

```



```

}
else if ((Subgraph_BollingerSqueeze[sc.Index] < 0) &&
(Subgraph_BollingerSqueeze[sc.Index] <= Subgraph_BollingerSqueeze[sc.Index-1]))
    Subgraph_BollingerSqueeze.DataColor[sc.Index] = Subgraph_BollingerSqueeze.SecondaryColor;
else if ((Subgraph_BollingerSqueeze[sc.Index] < 0) &&
(Subgraph_BollingerSqueeze[sc.Index] > Subgraph_BollingerSqueeze[sc.Index-1]))
{
    Subgraph_BollingerSqueeze.DataColor[sc.Index] =
        RGB(GetRValue(Subgraph_BollingerSqueeze.SecondaryColor)/2,
        GetGValue(Subgraph_BollingerSqueeze.SecondaryColor)/2,
        GetBValue(Subgraph_BollingerSqueeze.SecondaryColor)/2);
}
}
}

/*=====*/

```

```

SCSFExport scsf_BollingerSqueeze3(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Ratio = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SqueezeIndicator = sc.Subgraph[1];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_MovAvgType = sc.Input[1];
    SCInputRef Input_Length = sc.Input[2];
    SCInputRef Input_MovingAverageMultiplier = sc.Input[3];
    SCInputRef Input_StandardDeviationMultiplier = sc.Input[4];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Bollinger Squeeze 3";

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

        Subgraph_Ratio.Name = "Ratio";
        Subgraph_Ratio.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_Ratio.PrimaryColor = RGB(0, 255, 0);
        Subgraph_Ratio.SecondaryColor = RGB(255, 0, 0);
        //Ratio.SecondaryColorUsed = true;
        Subgraph_Ratio.DrawZeros = true;

        Subgraph_SqueezeIndicator.Name = "Squeeze Indicator";
        Subgraph_SqueezeIndicator.DrawStyle = DRAWSTYLE_POINT;
        Subgraph_SqueezeIndicator.LineWidth = 3;
        Subgraph_SqueezeIndicator.PrimaryColor = RGB(0, 255, 0);
        Subgraph_SqueezeIndicator.SecondaryColor = RGB(255, 0, 0);
        Subgraph_SqueezeIndicator.SecondaryColorUsed = true;
        Subgraph_SqueezeIndicator.DrawZeros = true;

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

        Input_MovAvgType.Name = "Moving Average Type for Calculations";
        Input_MovAvgType.SetMovAvgType(MOVAVGTYPE_SIMPLE);

        Input_Length.Name = "Study Length";
        Input_Length.SetInt(20);
        Input_Length.SetIntLimits(1, MAX_STUDY_LENGTH);

        Input_MovingAverageMultiplier.Name = "Moving Average Multiplier";
        Input_MovingAverageMultiplier.SetFloat(1.5f);

        Input_StandardDeviationMultiplier.Name = "Standard Deviation Multiplier";
        Input_StandardDeviationMultiplier.SetFloat(2.0f);
    }
}

```



```

    return;
}

sc.StdDeviation(sc.BaseData[Input_InputData.GetInputDataIndex()], Subgraph_Ratio.Arrays[0], Input_Length.GetInt());

sc.TrueRange(sc.BaseData, Subgraph_Ratio.Arrays[1]);
sc.MovingAverage(Subgraph_Ratio.Arrays[1], Subgraph_Ratio.Arrays[2], Input_MovAvgType.GetMovAvgType(),
Input_Length.GetInt());

float StandardDeviationValue = Subgraph_Ratio.Arrays[0][sc.Index];
Subgraph_Ratio[sc.Index] = (Input_StandardDeviationMultiplier.GetFloat() * StandardDeviationValue) /
(Input_MovingAverageMultiplier.GetFloat() * Subgraph_Ratio.Arrays[2][sc.Index]);

if (Subgraph_Ratio[sc.Index] >= 1.0)
    Subgraph_SqueezeIndicator.DataColor[sc.Index] = Subgraph_SqueezeIndicator.PrimaryColor; //No squeeze
else
    Subgraph_SqueezeIndicator.DataColor[sc.Index] = Subgraph_SqueezeIndicator.SecondaryColor; //Squeeze on
}
/*=====*/
SCSFExport scsf_VolumeColoredBasedOnVolume(SCStudyInterfaceRef sc)
{
    // Section 1 - Set the configuration variables

    SCSubgraphRef Subgraph_Volume = sc.Subgraph[0];

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

        sc.GraphName = "Volume-Colored Based on Volume";

        sc.AutoLoop = 1; // true

        Subgraph_Volume.Name = "Volume";
        Subgraph_Volume.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_Volume.SecondaryColorUsed = 1;
        Subgraph_Volume.PrimaryColor = RGB(0,255,0);
        Subgraph_Volume.SecondaryColor = RGB(255,0,0);

        return;
    }

    // Section 2 - Do data processing here

    Subgraph_Volume[sc.Index] = sc.Volume[sc.Index];

    if(sc.Volume[sc.Index-1] <= sc.Volume[sc.Index])
        Subgraph_Volume.DataColor[sc.Index] = Subgraph_Volume.PrimaryColor;
    else
        Subgraph_Volume.DataColor[sc.Index] = Subgraph_Volume.SecondaryColor;
}
/*=====*/
SCSFExport scsf_ExtendedArrayExample(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_ADX = sc.Subgraph[0];
    SCSubgraphRef Subgraph_ADXForwardShifted = sc.Subgraph[1];

    SCSubgraphRef Subgraph_ADXForwardShiftedPlus = sc.Subgraph[2];

    SCInputRef Input_DXLength = sc.Input[3];

```

```
SCInputRef Input_DXMovAngLength = sc.Input[4];
```

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

```
{
```

```
    sc.GraphName = "Extended Array Example";
```

```
    sc.StudyDescription = "Extended Array Example. Based on ADX.";
```

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

```
    sc.GraphRegion = 1;
```

```
    Subgraph_ADX.Name = "ADX";
```

```
    Subgraph_ADX.DrawStyle = DRAWSTYLE_LINE;
```

```
    Subgraph_ADX.PrimaryColor = RGB(0,255,0);
```

```
    Subgraph_ADX.DrawZeros = false;
```

```
    Subgraph_ADXForwardShifted.Name = "ADX Forward Shifted";
```

```
    Subgraph_ADXForwardShifted.DrawStyle = DRAWSTYLE_LINE;
```

```
    Subgraph_ADXForwardShifted.PrimaryColor = RGB(255,0,255);
```

```
    Subgraph_ADXForwardShifted.DrawZeros = false;
```

```
    // Set the ADXForwardShifted subgraph to support additional array elements for forward projection
```

```
    Subgraph_ADXForwardShifted.ExtendedArrayElementsToGraph = 50;
```

```
    Subgraph_ADXForwardShiftedPlus.Name = "ADX Plus Value";
```

```
    Subgraph_ADXForwardShiftedPlus.DrawStyle = DRAWSTYLE_LINE;
```

```
    Subgraph_ADXForwardShiftedPlus.ExtendedArrayElementsToGraph = 50;
```

```
    Subgraph_ADXForwardShiftedPlus.PrimaryColor = RGB(255,255,0);
```

```
    Subgraph_ADXForwardShiftedPlus.DrawZeros = false;
```

```
    Input_DXLength.Name = "DX Length";
```

```
    Input_DXLength.SetInt(14);
```

```
    Input_DXLength.SetIntLimits(1, INT_MAX);
```

```
    Input_DXMovAngLength.Name = "DX Mov Avg Length";
```

```
    Input_DXMovAngLength.SetInt(14);
```

```
    Input_DXMovAngLength.SetIntLimits(1, INT_MAX);
```

```
    return;
```

```
}
```

```
// Do data processing
```

```
sc.DataStartIndex = Input_DXLength.GetInt() + Input_DXMovAngLength.GetInt() - 1;
```

```
// Calculate ADX
```

```
sc.ADX(
```

```
    sc.BaseData,
```

```
    Subgraph_ADX,
```

```
    Input_DXLength.GetInt(),
```

```
    Input_DXMovAngLength.GetInt());
```

```
int NumberOfBarsToForwardShift = Subgraph_ADXForwardShifted.ExtendedArrayElementsToGraph;
```

```
Subgraph_ADXForwardShifted[sc.Index] = Subgraph_ADX[sc.Index-NumberOfBarsToForwardShift];
```

```
Subgraph_ADXForwardShiftedPlus[sc.Index] = Subgraph_ADXForwardShifted[sc.Index] + 10;
```

```
// Extended elements are cleared when the number of bars in the chart grows. Therefore, we need to fill them in every time. However, only when we are on the final index when using sc.AutoLoop = 1.
```

```
if(sc.Index+1 >= sc.ArraySize)
```

```
{
```

```
    for (int i = sc.ArraySize; i < sc.ArraySize + NumberOfBarsToForwardShift ;i++)
```

```

    {
        Subgraph_ADXForwardShifted[i] = Subgraph_ADX[i-NumberOfBarsToForwardShift];
        Subgraph_ADXForwardShiftedPlus[i] = Subgraph_ADXForwardShifted[i] + 10;
    }
}

return;
}

/*=====*/
SCSFExport scsf_ForewardProjectionBars(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];

    SCInputRef Input_NumberOfBarsToProjectForward = sc.Input[0];
    SCInputRef Input_DeletePriorBars = sc.Input[1];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Forward Projection Bars";

        sc.AutoLoop = 0;
        sc.GraphRegion = 0;
        sc.GraphDrawType=GDT_OHLCBAR;
        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(0,255,0);
        Subgraph_Low.DrawZeros = false;

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

        Input_NumberOfBarsToProjectForward.Name = "Number of Bars to Project Forward";
        Input_NumberOfBarsToProjectForward.SetInt(10);

        Input_DeletePriorBars.Name = "Delete Prior Bars";
        Input_DeletePriorBars.SetYesNo(0);

        return;
    }

    // Do data processing
    int NumberOfBarsToProjectForward = Input_NumberOfBarsToProjectForward.GetInt();

    Subgraph_Open.ExtendedArrayElementsToGraph = NumberOfBarsToProjectForward;
    Subgraph_High.ExtendedArrayElementsToGraph = NumberOfBarsToProjectForward;
    Subgraph_Low.ExtendedArrayElementsToGraph = NumberOfBarsToProjectForward;

```

```
Subgraph_Last.ExtendedArrayElementsToGraph = NumberOfBarsToProjectForward;
```

```
if (Input_DeletePriorBars.GetYesNo() == 1)
{
    if (sc.GetCalculationStartIndexForStudy() == 0)
    {
        for (int BarIndex = 0; BarIndex < sc.ArraySize; BarIndex++)
        {
            Subgraph_Open[BarIndex] = 0.0f;
            Subgraph_High[BarIndex] = 0.0f;
            Subgraph_Low[BarIndex] = 0.0f;
            Subgraph_Last[BarIndex] = 0.0f;
        }
    }
    else
    {
        Subgraph_Open[sc.ArraySize - 1] = 0.0f;
        Subgraph_High[sc.ArraySize - 1] = 0.0f;
        Subgraph_Low[sc.ArraySize - 1] = 0.0f;
        Subgraph_Last[sc.ArraySize - 1] = 0.0f;
    }
}

for (int BarIndex = sc.ArraySize - NumberOfBarsToProjectForward; BarIndex < sc.ArraySize; BarIndex++)
{
    Subgraph_Open[BarIndex + NumberOfBarsToProjectForward] = sc.Open[BarIndex];
    Subgraph_High[BarIndex + NumberOfBarsToProjectForward] = sc.High[BarIndex];
    Subgraph_Low[BarIndex + NumberOfBarsToProjectForward] = sc.Low[BarIndex];
    Subgraph_Last[BarIndex + NumberOfBarsToProjectForward] = sc.Close[BarIndex];
}

return;
}

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

```
SCSFExport scsf_ArrayValueAtNthOccurrenceSample(SCStudyInterfaceRef sc)
```

```
{
    SCSubgraphRef Subgraph_StochasticData = sc.Subgraph[0];
    SCSubgraphRef Subgraph_ValueAtOccurrence = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Line1 = sc.Subgraph[2];

    SCInputRef Input_KLength = sc.Input[3];
    SCInputRef Input_Line1Value = sc.Input[5];
    SCInputRef Input_MovAvgType = sc.Input[7];
    SCInputRef Input_DataHigh = sc.Input[8];
    SCInputRef Input_DataLow = sc.Input[9];
    SCInputRef Input_DataLast = sc.Input[10];
    SCInputRef Input_NumberOfOccurrences = sc.Input[11];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Array Value At Nth Occurrence Sample";
        sc.StudyDescription = "Array Value At Nth Occurrence. Based on %K array of Fast Stochastic";

        sc.ValueFormat = 2;

        Subgraph_StochasticData.Name = "%K";
        Subgraph_StochasticData.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_StochasticData.PrimaryColor = RGB(0,255,0);
        Subgraph_StochasticData.DrawZeros = true;

        Subgraph_ValueAtOccurrence.Name = "Value At Nth Occurrence";
    }
}
```

```

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

Subgraph_Line1.Name = "Line1";
Subgraph_Line1.DrawStyle = DRAWSTYLE_LINE;
Subgraph_Line1.PrimaryColor = RGB(255,255,0);
Subgraph_Line1.DrawZeros = true;

Input_KLength.Name = "%K Length";
Input_KLength.SetInt(10);
Input_KLength.SetIntLimits(1,MAX_STUDY_LENGTH);

Input_Line1Value.Name = "Line1 Value";
Input_Line1Value.SetFloat(70);

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

Input_DataHigh.Name = "Input Data";
Input_DataHigh.SetInputDataIndex(SC_HIGH);

Input_DataLow.Name = "Input Data";
Input_DataLow.SetInputDataIndex(SC_LOW);

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

Input_NumberOfOccurrences.Name = "Number Of Occurrences";
Input_NumberOfOccurrences.SetInt(10);
Input_NumberOfOccurrences.SetIntLimits(1,MAX_STUDY_LENGTH);


sc.AutoLoop = true;
return;
}

// This sample is based on Fast Stochastic Study.
// The sample study use only %K subgraph
// If %K value greater than Line1 value, then TRUE will be put to TrueFalse array, otherwise FALSE
// The ValueAtOccurrence Subgraph will have the values returned by sc.ArrayValueAtNthOccurrence function

int InDLength = 3;

sc.DataStartIndex = Input_KLength.GetInt() + InDLength;

sc.Stochastic(sc.BaseDataIn, sc.Subgraph[4], Input_KLength.GetInt(), 1, 1, 1);

sc.Stochastic(
    sc.BaseData[Input_DataHigh.GetInputDataIndex()],
    sc.BaseData[Input_DataLow.GetInputDataIndex()],
    sc.BaseData[Input_DataLast.GetInputDataIndex()],
    Subgraph_StochasticData,
    Input_KLength.GetInt(),
    InDLength,
    0,
    Input_MovAvgType.GetMovAvgType()
);

// fill the TrueFalse array
if (Subgraph_StochasticData[sc.Index] > Input_Line1Value.GetFloat())
    Subgraph_StochasticData.Arrays[1][sc.Index] = 1.0f;
else
    Subgraph_StochasticData.Arrays[1][sc.Index] = 0.0f;

```

```

Subgraph_ValueAtOccurrence[sc.Index] = sc.ArrayValueAtNthOccurrence(Subgraph_StochasticData.Arrays[1],
Subgraph_StochasticData, Input_NumberOfOccurrences.GetInt());

```

```

Subgraph_Line1[sc.Index] = Input_Line1Value.GetFloat();
}

```

```

/*=====*/

```

```

SCSFExport scsf_SwingHighAndLow(SCStudyInterfaceRef sc)

```

```

{
    SCSubgraphRef Subgraph_SwingHigh = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SwingLow = sc.Subgraph[1];
    SCSubgraphRef Subgraph_LineProperties = sc.Subgraph[2];

```

```

    SCInputRef Input_ArrowOffsetValue = sc.Input[0];
    SCInputRef Input_Length = sc.Input[1];
    SCInputRef Input_AllowEqualBars = sc.Input[2];
    SCInputRef Input_ExtendSwings = sc.Input[3];

```

```

    if (sc.SetDefaults)
    {

```

```

        sc.GraphName = "Swing High And Low";

```

```

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

```

```

        Subgraph_SwingHigh.Name = "Swing High";
        Subgraph_SwingHigh.DrawStyle = DRAWSTYLE_ARROW_DOWN;
        Subgraph_SwingHigh.PrimaryColor = RGB(0,255,0);
        Subgraph_SwingHigh.DrawZeros = false;
        Subgraph_SwingHigh.PrimaryColor = RGB(255,0,0);
        Subgraph_SwingHigh.LineWidth = 3;

```

```

        Subgraph_SwingLow.Name = "Swing Low";
        Subgraph_SwingLow.DrawStyle = DRAWSTYLE_ARROW_UP;
        Subgraph_SwingLow.PrimaryColor = RGB(255,0,255);
        Subgraph_SwingLow.DrawZeros = false;
        Subgraph_SwingLow.LineWidth = 3;
        Subgraph_SwingLow.PrimaryColor = RGB(0,255,0);

```

```

        Subgraph_LineProperties.Name = "Line Properties";
        Subgraph_LineProperties.DrawStyle = DRAWSTYLE_SUBGRAPH_NAME_AND_VALUE_LABELS_ONLY;
        Subgraph_LineProperties.LineWidth = 1;
        Subgraph_LineProperties.PrimaryColor = RGB(255, 0, 0);
        Subgraph_LineProperties.SecondaryColor = RGB(0, 255, 0);
        Subgraph_LineProperties.SecondaryColorUsed = true;
        Subgraph_LineProperties.DrawZeros = false;

```

```

        Input_ArrowOffsetValue.Name = "Arrow Offset as Percentage";
        Input_ArrowOffsetValue.SetFloat(3);

```

```

        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);

```

```

        Input_ExtendSwings.Name = "Extend Swings Until Future Intersection";
        Input_ExtendSwings.SetYesNo(false);

```

```

    return;
}

```

```

int IndexToEvaluate = sc.Index - Input_Length.GetInt();

```

```

if(IndexToEvaluate - Input_Length.GetInt() < 0)
{
    return;
}

sc.EarliestUpdateSubgraphDataArrayIndex = IndexToEvaluate;

float ArrowOffset=(sc.High[IndexToEvaluate] - sc.Low[IndexToEvaluate] )*(Input_ArrowOffsetValue.GetFloat() * 0.01f);

Subgraph_SwingHigh[IndexToEvaluate] = 0;
Subgraph_SwingLow[IndexToEvaluate] = 0;

// check for Swing High
if (!Input_AllowEqualBars.GetYesNo())
{
    if (sc.IsSwingHigh(sc.High, IndexToEvaluate, Input_Length.GetInt()))
        Subgraph_SwingHigh[IndexToEvaluate] = sc.High[IndexToEvaluate] + ArrowOffset;
}
else if (IsSwingHighAllowEqual_S(sc, true, IndexToEvaluate, Input_Length.GetInt()))
    Subgraph_SwingHigh[IndexToEvaluate] = sc.High[IndexToEvaluate] + ArrowOffset;

// check for Swing Low
if (!Input_AllowEqualBars.GetYesNo())
{
    if (sc.IsSwingLow(sc.Low, IndexToEvaluate, Input_Length.GetInt()))
        Subgraph_SwingLow[IndexToEvaluate] = sc.Low[IndexToEvaluate] - ArrowOffset;
}
else if (IsSwingLowAllowEqual_S(sc, true, IndexToEvaluate, Input_Length.GetInt()))
    Subgraph_SwingLow[IndexToEvaluate] = sc.Low[IndexToEvaluate] - ArrowOffset;

if (Input_ExtendSwings.GetYesNo())
{
    if (Subgraph_SwingHigh[IndexToEvaluate] != 0)
    {
        sc.AddLineUntilFutureIntersection
        (IndexToEvaluate
        , 1 // LineIDForBar
        , Subgraph_SwingHigh[IndexToEvaluate]
        , Subgraph_LineProperties.PrimaryColor
        , Subgraph_LineProperties.LineWidth
        , Subgraph_LineProperties.LineStyle
        , false
        , false
        , ""
        );
    }
    else
    {
        sc.DeleteLineUntilFutureIntersection(IndexToEvaluate, 1);
    }
}

if (Input_ExtendSwings.GetYesNo())
{
    if (Subgraph_SwingLow[IndexToEvaluate] != 0)
    {
        sc.AddLineUntilFutureIntersection
        (IndexToEvaluate
        , 2 // LineIDForBar
        , Subgraph_SwingLow[IndexToEvaluate]
        , Subgraph_LineProperties.SecondaryColor
        , Subgraph_LineProperties.LineWidth
        , Subgraph_LineProperties.LineStyle

```

```

        , false
        , false
        , ""
    );
}
else
{
    sc.DeleteLineUntilFutureIntersection(IndexToEvaluate, 2);
}
}
}

/*****
SCSFExport scsf_ForceIndexAverage(SCStudyInterfaceRef sc)
{
    // Section 1 - Set the configuration variables

    SCSubgraphRef Subgraph_ForceAverage = sc.Subgraph[0];

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

    SCInputRef Input_AverageLength = sc.Input[0];

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

        sc.GraphName = "Force Index Average";

        sc.StudyDescription = "";

        sc.AutoLoop = 1;

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

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

        Input_AverageLength.Name = "Moving Average Length";
        Input_AverageLength.SetInt(20);
        Input_AverageLength.SetIntLimits(1, MAX_STUDY_LENGTH);

        return;
    }

    // Section 2 - Do data processing here

    sc.DataStartIndex = Input_AverageLength.GetInt();

    int i = sc.Index;

    Array_ForceIndex[i] = sc.Volume[i] * (sc.Close[i] - sc.Close[i-1]);

    sc.ExponentialMovAvg(Array_ForceIndex, Subgraph_ForceAverage, Input_AverageLength.GetInt());
}

/*****
SCSFExport scsf_ForceIndex(SCStudyInterfaceRef sc)
{
    // Section 1 - Set the configuration variables

```



```
SCSubgraphRef Subgraph_ForceIndex = sc.Subgraph[0];
```

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

    sc.GraphName = "Force Index";

    sc.StudyDescription = "";

    sc.AutoLoop = 1;

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

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

    return;
}
```

```
// Section 2 - Do data processing here
```

```
int i = sc.Index;
```

```
Subgraph_ForceIndex[i] = sc.Volume[i] * (sc.Close[i] - sc.Close[i-1]);
```

```
}
/*****
```

```
*****/
```

```
*
```

```
* True Strength Index (Also known as Ergodic)
```

```
*
```

```
* Array 3-8 are used for calculations
```

```
*
```

```
* Numerator = EMA( EMA(Price - LastPrice, LongExpMALength), ShortExpMALength)
```

```
* Denominator = EMA( EMA( Abs(Price - LastPrice), LongExpMALength), ShortExpMALength)
```

```
*
```

```
* TSI = Multiplier * Numerator / Denominator
```

```
* SignalMA = EMA(TSI, SignalMALength)
```

```
* Oscillator = TSI - SignalMA
```

```
*
```

```
*****/
```

```
SCSFExport scsf_TrueStrengthIndex(SCStudyInterfaceRef sc)
```

```
{
```

```
SCSubgraphRef Subgraph_TSI = sc.Subgraph[0];
```

```
SCSubgraphRef Subgraph_SignalMA = sc.Subgraph[1];
```

```
SCSubgraphRef Subgraph_Oscillator = 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_LongExpMALength = sc.Input[3];
```

```
SCInputRef Input_ShortExpMALength = sc.Input[4];
```

```
SCInputRef Input_Multiplier = sc.Input[5];
```

```

SCInputRef Input_SignalLineMALength = sc.Input[6];

if (sc.SetDefaults)
{
    sc.GraphName = "True Strength Index";

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

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

    Subgraph_SignalMA.Name = "Signal MA";
    Subgraph_SignalMA.DrawStyle = DRAWSTYLE_HIDDEN;
    Subgraph_SignalMA.PrimaryColor = RGB(255,0,255);
    Subgraph_SignalMA.DrawZeros = false;

    Subgraph_Oscillator.Name = "Oscillator";
    Subgraph_Oscillator.DrawStyle = DRAWSTYLE_HIDDEN;
    Subgraph_Oscillator.PrimaryColor = RGB(255,255,0);
    Subgraph_Oscillator.DrawZeros = false;

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

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

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

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

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

    return;
}

// Subgraphs 3-7 are used for calculation only
sc.DataStartIndex = ((Input_LongExpMALength.GetInt() ) + (Input_ShortExpMALength.GetInt()) +
Input_SignalLineMALength.GetInt()) ;

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

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

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

int SignalStart = sc.Index;

```

```

    if (SignalStart < Input_ShortExpMALength.GetInt() - 1 + Input_SignalLineMALength.GetInt() -
1+Input_LongExpMALength.GetInt() - 1) //4
        SignalStart = Input_ShortExpMALength.GetInt() - 1 + Input_SignalLineMALength.GetInt() - 1
+Input_LongExpMALength.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, MOVAVGTYPE_EXPONENTIAL,
Input_LongExpMALength.GetInt());

    if(sc.Index >= ShortStart)
        sc.MovingAverage(Subgraph_Temp4, Subgraph_Temp5, MOVAVGTYPE_EXPONENTIAL,
Input_ShortExpMALength.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, MOVAVGTYPE_EXPONENTIAL,
Input_LongExpMALength.GetInt());

    if(sc.Index >= ShortStart)
    {
        sc.MovingAverage(Subgraph_Temp7, Subgraph_Temp8, MOVAVGTYPE_EXPONENTIAL,
Input_ShortExpMALength.GetInt());

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

    // Calculate the Signal Line ( EMA[TSI] )
    if(sc.Index >= SignalStart)
        sc.MovingAverage(Subgraph_TSI, Subgraph_SignalMA, MOVAVGTYPE_EXPONENTIAL,
Input_SignalLineMALength.GetInt());

    // Calculate the Oscillator (TSI - EMA[TSI])
    if(sc.Index >= sc.DataStartIndex)
        Subgraph_Oscillator[sc.Index] = Subgraph_TSI[sc.Index] - Subgraph_SignalMA[sc.Index];
}

/*=====*/
SCSFExport scsf_CommodityChannelIndex(SCStudyInterfaceRef sc)
{
    SCInputRef Input_InputData = 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];
    SCInputRef Input_MAType = sc.Input[7];
    SCInputRef Input_Version = sc.Input[8];

    SCSubgraphRef Subgraph_CCI = sc.Subgraph[0];

```

```

SCSubgraphRef Subgraph_Line1 = sc.Subgraph[1];
SCSubgraphRef Subgraph_Line2 = sc.Subgraph[2];
SCSubgraphRef Subgraph_Line3 = sc.Subgraph[3];

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

    sc.GraphName = "Commodity Channel Index";

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

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

    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 = false;

    Subgraph_Line3.Name = "Line 3";
    Subgraph_Line3.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_Line3.PrimaryColor = RGB(255,127,0);
    Subgraph_Line3.DrawZeros = false;

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

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

    Input_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);

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

    Input_Version.SetInt(1);

    return;
}

if(Input_Version.GetInt() < 1)
{
    Input_Version.SetInt(1);
    Input_MAType.SetMovAvgType(MOVAVGTYPE_SIMPLE);
}

```

```

}

sc.DataStartIndex = Input_Length.GetInt();

float Multiplier = Input_Multiplier.GetFloat();

if(Multiplier == 0.0f)
    Multiplier = 0.015f;

sc.CCI(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_CCI, sc.Index, Input_Length.GetInt(),
Multiplier, Input_MAType.GetMovAvgType());

Subgraph_Line2[sc.Index] = Input_Line2Value.GetFloat();
Subgraph_Line3[sc.Index] = Input_Line3Value.GetFloat();
}

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

    SCInputRef Input_Data = sc.Input[0];

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

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

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

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

        return;
    }

    sc.DataStartIndex = 1;

    if(sc.Index < 1)
        return;

    int Index = sc.Index;

    float Diff = sc.BaseDataIn[Input_Data.GetInputDataIndex()][Index] - sc.BaseDataIn[Input_Data.GetInputDataIndex()][Index-1];

    if(sc.FormattedEvaluate(static_cast<float> (Diff),sc.BaseGraphValueFormat,GREATER_OPERATOR,sc.TickSize
,sc.BaseGraphValueFormat) != 0)
        Subgraph_TVI[Index] = Subgraph_TVI[Index-1] + sc.Volume[Index];
    else if(sc.FormattedEvaluate(static_cast<float> (Diff),sc.BaseGraphValueFormat,LESS_OPERATOR,sc.TickSize*-1
,sc.BaseGraphValueFormat) != 0)
        Subgraph_TVI[Index] = Subgraph_TVI[Index-1] - sc.Volume[Index];
    else
        Subgraph_TVI[Index] = Subgraph_TVI[Index-1];
}

```

```
/******
```

```
*
```

```
* Demand Index
```

```
*
```

```
* Array Number Usage
```

```
*
```

```
* Array 0 - Demand Index
```

```
* Array 1 - Demand Index Moving Average
```

```
* Array 2 - Zero Line
```

```
* Array 3 - Moving Average Sell Power
```

```
* Array 4 - High + Low + 2 * Last
```

```
* Array 5 - Max of last 2 High's - Min of last 2 Low's
```

```
* Array 6 - EMA of Array 5
```

```
* Array 7 - EMA of Volume
```

```
* Array 8 - BuyPower
```

```
* Array 9 - SellPower
```

```
* Array 10 - Moving Average Buy Power
```

```
*
```

```
*****/
```

```
SCSFExport scsf_DemandIndex(SCStudyInterfaceRef sc)
```

```
{  
    SCSubgraphRef Subgraph_OutDemandIndex = sc.Subgraph[0]; // Demand Index
```

```
    SCSubgraphRef Subgraph_OutDemandIndexMA = sc.Subgraph[1]; // Demand Index Moving Average
```

```
    SCSubgraphRef Subgraph_OutZeroLine = sc.Subgraph[2]; // Zero Line
```

```
    SCFloatArrayRef Array_HL2C = sc.Subgraph[0].Arrays[0]; // High + Low + 2 * Last
```

```
    SCFloatArrayRef Array_Range = sc.Subgraph[0].Arrays[1]; // Max of last 2 High's - Min of last 2 Low's
```

```
    SCFloatArrayRef Array_RangeMA = sc.Subgraph[1].Arrays[1]; // EMA of Array 5
```

```
    SCFloatArrayRef Array_VolumeMA = sc.Subgraph[1].Arrays[0]; // EMA of Volume
```

```
    SCFloatArrayRef Array_BuyPower = sc.Subgraph[0].Arrays[2]; // BuyPower
```

```
    SCFloatArrayRef Array_SellPower = sc.Subgraph[0].Arrays[3]; // SellPower
```

```
    SCFloatArrayRef Array_MABuyPower = sc.Subgraph[1].Arrays[2]; // Moving Average Buy Power
```

```
    SCFloatArrayRef Array_MASellPower = sc.Subgraph[1].Arrays[3]; // Moving Average Sell Power
```

```
    SCInputRef Input_BSPowerLength = sc.Input[0];
```

```
    SCInputRef Input_BSPowerMovAvgLength = sc.Input[1];
```

```
    SCInputRef Input_DIMovAvgLength = sc.Input[2];
```

```
    SCInputRef Input_DIMovAvgType = sc.Input[3];
```

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

```
    {  
        sc.GraphName = "Demand Index";
```

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

```
        Subgraph_OutDemandIndex.Name = "DI";
```

```
        Subgraph_OutDemandIndex.PrimaryColor = RGB(0,255,0);
```

```
        Subgraph_OutDemandIndex.DrawStyle = DRAWSTYLE_LINE;
```

```
        Subgraph_OutDemandIndex.LineWidth = 1;
```

```
        Subgraph_OutDemandIndex.DrawZeros = true;
```

```
        Subgraph_OutDemandIndexMA.Name = "DI MovAvg";
```

```
        Subgraph_OutDemandIndexMA.PrimaryColor = RGB(255,125,125);
```

```
        Subgraph_OutDemandIndexMA.DrawStyle = DRAWSTYLE_LINE;
```

```
        Subgraph_OutDemandIndexMA.LineWidth = 1;
```

```
        Subgraph_OutDemandIndexMA.DrawZeros = true;
```

```
        Subgraph_OutZeroLine.Name = "Line";
```

```
        Subgraph_OutZeroLine.PrimaryColor = RGB(125,125,125);
```

```
        Subgraph_OutZeroLine.DrawStyle = DRAWSTYLE_LINE;
```

```
        Subgraph_OutZeroLine.LineWidth = 1;
```

```
        Subgraph_OutZeroLine.DrawZeros = true;
```

```
        Input_BSPowerLength.Name = "Buy/Sell Power Length";
```

```

Input_BSPowerLength.SetInt(19);
Input_BSPowerLength.SetIntLimits(1, 100);

Input_BSPowerMovAvgLength.Name = "Buy/Sell Power Moving Average Length";
Input_BSPowerMovAvgLength.SetInt(19);
Input_BSPowerMovAvgLength.SetIntLimits(1, 100);

Input_DIMovAvgLength.Name = "Demand Index Moving Average Length";
Input_DIMovAvgLength.SetInt(30);
Input_DIMovAvgLength.SetIntLimits(1, 100);

Input_DIMovAvgType.Name = "Demand Index Moving Average Type";
Input_DIMovAvgType.SetMovAvgType(MOVAVGTYPE_EXPONENTIAL);

return;
}

Array_HL2C[sc.Index] = sc.High[sc.Index] + sc.Low[sc.Index] + 2 * sc.Close[sc.Index];

if (sc.Index == 0)
{
    Array_Range[0] = sc.High[0] - sc.Low[0];
}
else
{
    Array_Range[sc.Index]
        = max(sc.High[sc.Index], sc.High[sc.Index - 1])
        - min(sc.Low[sc.Index], sc.Low[sc.Index - 1]);
}

const float SmallValue = sc.TickSize / 4.0f;

//
sc.MovingAverage(Array_Range, Array_RangeMA, Input_DIMovAvgType.GetMovAvgType(),
Input_BSPowerLength.GetInt());
sc.MovingAverage(sc.Volume, Array_VolumeMA, Input_DIMovAvgType.GetMovAvgType(),
Input_BSPowerLength.GetInt());

//
if (sc.Index >= 1)
{
    if (Array_VolumeMA[sc.Index] > SmallValue)
    {
        Array_BuyPower[sc.Index] = sc.Volume[sc.Index] / Array_VolumeMA[sc.Index];
        Array_SellPower[sc.Index] = sc.Volume[sc.Index] / Array_VolumeMA[sc.Index];
    }
    else
    {
        Array_BuyPower[sc.Index] = 0.0f;
        Array_SellPower[sc.Index] = 0.0f;
    }

    if (Array_HL2C[sc.Index] > SmallValue)
    {
        if (Array_HL2C[sc.Index] < Array_HL2C[sc.Index - 1])
        {
            Array_BuyPower[sc.Index]
                = Array_BuyPower[sc.Index]
                / exp(
                    (
                        0.375f
                        * (Array_HL2C[sc.Index] + Array_HL2C[sc.Index - 1])
                        / Array_RangeMA[0]
                    )
                    * (Array_HL2C[sc.Index - 1] - Array_HL2C[sc.Index])
                )
        }
    }
}

```

```

        / Array_HL2C[sc.Index]
    );
}
else if (Array_HL2C[sc.Index] > Array_HL2C[sc.Index - 1])
{
    Array_SellPower[sc.Index]
    = Array_SellPower[sc.Index]
    / exp(
        (
            0.375f
            * (Array_HL2C[sc.Index] + Array_HL2C[sc.Index - 1])
            / Array_RangeMA[0]
        )
        * (Array_HL2C[sc.Index] - Array_HL2C[sc.Index - 1])
        / Array_HL2C[sc.Index - 1]
    );
}
}
else
{
    Array_BuyPower[sc.Index] = 0.0f;
    Array_SellPower[sc.Index] = 0.0f;
}
}

sc.MovingAverage(Array_BuyPower, Array_MABuyPower, Input_DIMovAvgType.GetMovAvgType(),
Input_BSPowerMovAvgLength.GetInt());
sc.MovingAverage(Array_SellPower, Array_MASellPower, Input_DIMovAvgType.GetMovAvgType(),
Input_BSPowerMovAvgLength.GetInt());

float Divisor = 0;
float Dividend = 0;
if (Array_MABuyPower[sc.Index] > Array_MASellPower[sc.Index])
{
    Divisor = Array_MABuyPower[sc.Index];
    Dividend = Array_MASellPower[sc.Index];
}
else if (Array_MABuyPower[sc.Index] < Array_MASellPower[sc.Index])
{
    Divisor = Array_MASellPower[sc.Index];
    Dividend = Array_MABuyPower[sc.Index];
}
else
{
    Divisor = Array_MASellPower[sc.Index];
    Dividend = Array_MASellPower[sc.Index];
}

float Calculation1 = 0.0f;
if (Divisor > SmallValue)
    Calculation1 = 1 - (Dividend / Divisor);

if (Array_MASellPower[sc.Index] > Array_MABuyPower[sc.Index])
    Calculation1 *= -1;

Subgraph_OutDemandIndex[sc.Index] = Calculation1 * 100;
Subgraph_OutZeroLine[sc.Index] = 0;

//
sc.MovingAverage(Subgraph_OutDemandIndex, Subgraph_OutDemandIndexMA, MOVAVGTYPE_SIMPLE,
Input_DIMovAvgLength.GetInt());
}

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

```



```

{
    SCSubgraphRef Subgraph_BBBandWidth = sc.Subgraph[0];
    SCSubgraphRef Subgraph_OutBollingerBands = sc.Subgraph[1];

    SCInputRef Input_Data = sc.Input[0];

    SCInputRef Input_MovingAverageType = sc.Input[1];

    SCInputRef Input_Length = sc.Input[3];
    SCInputRef Input_StandardDeviations = sc.Input[4];
    SCInputRef Input_Version = sc.Input[5];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Bollinger Bands: Bandwidth";
        sc.ValueFormat = 3;
        sc.GraphRegion = 1;
        sc.AutoLoop = 1;

        Subgraph_BBBandWidth.Name = "BollingerBands: Bandwidth";
        Subgraph_BBBandWidth.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_BBBandWidth.PrimaryColor = RGB(0,255,0);
        Subgraph_BBBandWidth.DrawZeros = true;

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

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

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

        Input_StandardDeviations.Name = "Standard Deviations";
        Input_StandardDeviations.SetFloat(2.0f);
        Input_StandardDeviations.SetFloatLimits(0.00001f, static_cast<float>(MAX_STUDY_LENGTH));

        Input_Version.SetInt(1);

        return;
    }

    if (Input_Version.GetInt() < 1)
    {
        Input_MovingAverageType.SetMovAvgType(MOVAVGTYPE_SIMPLE);
        Input_Version.SetInt(1);
    }

    int i = sc.Index;

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

    sc.BollingerBands(sc.BaseDataIn[Input_Data.GetInputDataIndex()], Subgraph_OutBollingerBands,
    Input_Length.GetInt(), Input_StandardDeviations.GetFloat(),Input_MovingAverageType.GetMovAvgType());

    SCFloatArrayRef MiddleBand = Subgraph_OutBollingerBands;
    SCFloatArrayRef UpperBand = Subgraph_OutBollingerBands.Arrays[0];
    SCFloatArrayRef LowerBand = Subgraph_OutBollingerBands.Arrays[1];

    Subgraph_BBBandWidth[i] = (UpperBand[i] - LowerBand[i]) / MiddleBand[i];
}

```

```

/*=====*/
SCSFExport scsf_BollingerBandsPercentB(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_PercentB = sc.Subgraph[0];
    SCSubgraphRef Subgraph_OutBollingerBands = sc.Subgraph[1];
    SCSubgraphRef Subgraph_UpperThreshold = sc.Subgraph[2];
    SCSubgraphRef Subgraph_LowerThreshold = sc.Subgraph[3];

    SCInputRef Input_Data = sc.Input[0];
    SCInputRef Input_MovAvgType = sc.Input[1];
    SCInputRef Input_Length = sc.Input[3];
    SCInputRef Input_StandardDeviations = sc.Input[4];
    SCInputRef Input_BaseBand = sc.Input[5];
    SCInputRef Input_UpperThresholdValue = sc.Input[6];
    SCInputRef Input_LowerThresholdValue = sc.Input[7];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Bollinger Bands %B";
        sc.ValueFormat = 3;
        sc.GraphRegion = 1;
        sc.AutoLoop = 1;

        Subgraph_PercentB.Name = "%B";
        Subgraph_PercentB.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_PercentB.PrimaryColor = RGB(0,255,0);
        Subgraph_PercentB.DrawZeros = true;

        Subgraph_UpperThreshold.Name = "Upper Threshold";
        Subgraph_UpperThreshold.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_UpperThreshold.LineStyle = LINESTYLE_DASH;
        Subgraph_UpperThreshold.PrimaryColor = COLOR_GRAY;
        Subgraph_UpperThreshold.DrawZeros = false;

        Subgraph_LowerThreshold.Name = "Lower Threshold";
        Subgraph_LowerThreshold.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_LowerThreshold.LineStyle = LINESTYLE_DASH;
        Subgraph_LowerThreshold.PrimaryColor = COLOR_GRAY;
        Subgraph_LowerThreshold.DrawZeros = false;

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

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

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

        Input_StandardDeviations.Name = "Standard Deviations";
        Input_StandardDeviations.SetFloat(2.0f);

        Input_BaseBand.Name = "Base Band";
        Input_BaseBand.SetCustomInputStrings( "Lower Band;Middle Band");
        Input_BaseBand.SetCustomInputIndex(0);

        Input_UpperThresholdValue.Name = "Upper Line Threshold Value";
        Input_UpperThresholdValue.SetFloat(0);

        Input_LowerThresholdValue.Name = "Lower Line Threshold Value";
        Input_LowerThresholdValue.SetFloat(0);

        return;
    }
}

```

```

int Index = sc.Index;

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

// Calculate Bollinger bands
sc.BollingerBands(sc.BaseDataIn[Input_Data.GetInputDataIndex()],
    Subgraph_OutBollingerBands, Input_Length.GetInt(), Input_StandardDeviations.GetFloat(),
Input_MovAvgType.GetMovAvgType());

SCFloatArrayRef MiddleBand = Subgraph_OutBollingerBands;
SCFloatArrayRef UpperBand = Subgraph_OutBollingerBands.Arrays[0];
SCFloatArrayRef LowerBand = Subgraph_OutBollingerBands.Arrays[1];

//Standard calculation: %B = (Price - Bottom Band)/(Top Band - Bottom Band)

if (Input_BaseBand.GetIndex()== 1)
{
    float BandsDifference= UpperBand[Index] - MiddleBand[Index];
    if(BandsDifference)
        Subgraph_PercentB[Index] = (sc.BaseDataIn[Input_Data.GetInputDataIndex()][Index] - MiddleBand[Index]) /
BandsDifference;
    else
        Subgraph_PercentB[Index] = 0.0;
}
else
{

    float BandsDifference= UpperBand[Index] - LowerBand[Index];
    if(BandsDifference)
        Subgraph_PercentB[Index] = (sc.BaseDataIn[Input_Data.GetInputDataIndex()][Index] - LowerBand[Index]) /
BandsDifference;
    else
        Subgraph_PercentB[Index] = 0.0;
}

Subgraph_UpperThreshold[Index] = Input_UpperThresholdValue.GetFloat();
Subgraph_LowerThreshold[Index] = Input_LowerThresholdValue.GetFloat();
}

/*=====*/
SCSFExport scsf_DifferenceBar(SCStudyInterfaceRef sc)
{
    SCInputRef Input_Chart2Number = sc.Input[3];
    SCInputRef Input_Chart1Multiplier = sc.Input[4];
    SCInputRef Input_Chart2Multiplier = sc.Input[5];
    SCInputRef Input_Chart1Addition = sc.Input[6];
    SCInputRef Input_ZeroOutputWhenNonExactDateTimeMatch = sc.Input[7];
    SCInputRef Input_UseLatestSourceDataForLastBar = sc.Input[8];
    SCInputRef Input_ZeroOutputWhenOneSourceGraphHasZeroValues = sc.Input[9];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Difference (Bar)";
        sc.ValueFormat = VALUEFORMAT_INHERITED;
        sc.GraphRegion = 1;
        sc.GraphUsesChartColors = 1;
        sc.GraphDrawType = GDT_OHLCBAR;
        sc.StandardChartHeader = 1;

        // We are using Manual looping.
        sc.AutoLoop = false;

        for (int SubgraphIndex = 0; SubgraphIndex <= SC_ASKVOL; ++SubgraphIndex)

```

```

{
    sc.Subgraph[SubgraphIndex].Name = sc.GetStudySubgraphName(0, SubgraphIndex);
    sc.Subgraph[SubgraphIndex].DrawZeros = false;
    if (SubgraphIndex < SC_VOLUME)
        sc.Subgraph[SubgraphIndex].DrawStyle = DRAWSTYLE_LINE;
    else
        sc.Subgraph[SubgraphIndex].DrawStyle = DRAWSTYLE_IGNORE;

    if (SubgraphIndex == SC_LAST)
    {
        sc.Subgraph[SubgraphIndex].LineLabel = LL_DISPLAY_VALUE | LL_VALUE_ALIGN_VALUES_SCALE |
LL_VALUE_ALIGN_CENTER;
    }
}

Input_Chart2Number.Name = "Chart 2 Number and Graph";
Input_Chart2Number.SetChartStudyValues(1, 0);

Input_Chart1Multiplier.Name = "Chart 1 Multiplier";
Input_Chart1Multiplier.SetFloat(1.0f);

Input_Chart2Multiplier.Name = "Chart 2 Multiplier";
Input_Chart2Multiplier.SetFloat(1.0f);

Input_Chart1Addition.Name = "Chart 1 Addition";
Input_Chart1Addition.SetFloat(0);

Input_ZeroOutputWhenNonExactDateTimeMatch.Name = "Zero Output When Non Exact Date-Time Match";
Input_ZeroOutputWhenNonExactDateTimeMatch.SetYesNo(false);

Input_UseLatestSourceDataForLastBar.Name = "Use Latest Source Data For Last Bar";
Input_UseLatestSourceDataForLastBar.SetYesNo(0);

Input_ZeroOutputWhenOneSourceGraphHasZeroValues.Name = "Zero Output When one Source Graph has Zero
Values";
Input_ZeroOutputWhenOneSourceGraphHasZeroValues.SetYesNo(0);

return;
}

int LastProcessedSourceBarIndex = sc.GetPersistentInt(1);

if (sc.IsFullRecalculation )
{
    // Create text indicating both charts and set to sc.GraphName
    SCString Chart1Name = sc.GetStudyNameFromChart(sc.ChartNumber, 0);
    SCString Chart2Name = sc.GetStudyNameFromChart(Input_Chart2Number.GetChartNumber(),
Input_Chart2Number.GetStudyID());
    sc.GraphName.Format("Difference %s * %g - %s * %g", Chart1Name.GetChars(),
Input_Chart1Multiplier.GetFloat(), Chart2Name.GetChars(), Input_Chart2Multiplier.GetFloat());

    LastProcessedSourceBarIndex = 0;
}

// Obtain a reference to the Base Data in the specified chart. This call is relatively efficient, but it should be called as
minimally as possible. To reduce the number of calls we have it outside of the primary "for" loop in this study function. And
we also use Manual Looping by using sc.AutoLoop = 0. In this way, sc.GetChartData /
sc.GetStudyArraysFromChartUsingID is called only once per call to this study function and there is a minimal number of
calls to this study function. sc.GetChartData is a function to get all of the Base Data arrays with one efficient call.
SCGraphData Chart2BaseData;
//sc.GetChartData(-Chart2Number.GetChartNumber(), Chart2BaseData);
sc.GetStudyArraysFromChartUsingID(-Input_Chart2Number.GetChartNumber(), Input_Chart2Number.GetStudyID(),
Chart2BaseData);

```

```

int CalculationStartIndex = sc.GetCalculationStartIndexForStudy();

if (!sc.IsFullRecalculation)
{
    SCDateTimeArray Chart2DateTimeArray;
    sc.GetChartDateTimeArray(Input_Chart2Number.GetChartNumber(), Chart2DateTimeArray);

    int DestinationStartIndex = sc.GetContainingIndexForSCDateTime(sc.ChartNumber,
Chart2DateTimeArray[LastProcessedSourceBarIndex].GetAsDouble());

    if (DestinationStartIndex < CalculationStartIndex)
        CalculationStartIndex = DestinationStartIndex;
}

for (int Index = CalculationStartIndex; Index < sc.ArraySize; Index++)
{
    int Chart2Index = 0;

    if(Input_ZeroOutputWhenNonExactDateTimeMatch.GetYesNo() == false)
    {
        Chart2Index = sc.GetNearestMatchForDateTimeIndex(Input_Chart2Number.GetChartNumber(), Index);

        //When 'use latest source data for last bar' is set to Yes, replay is not running and at last bar in the destination
chart, then use the data from the very latest bar in the source chart.
        if(Input_UseLatestSourceDataForLastBar.GetYesNo() && !sc.IsReplayRunning() && Index == sc.ArraySize - 1)
        {
            Chart2Index = Chart2BaseData[0].GetArraySize() - 1;
        }

        LastProcessedSourceBarIndex = Chart2Index;
    }
    else
    {
        Chart2Index = sc.GetExactMatchForSCDateTime(Input_Chart2Number.GetChartNumber(),
sc.BaseDateTimeln[Index]);

        LastProcessedSourceBarIndex = Chart2Index;

        if(Chart2Index == -1)
        {
            for (int SubgraphIndex = 0; SubgraphIndex <= SC_HL_AVG; SubgraphIndex++)
            {
                sc.Subgraph[SubgraphIndex][Index] = 0.0;
            }
            continue;
        }
    }

    if (Input_ZeroOutputWhenOneSourceGraphHasZeroValues.GetYesNo())
    {
        bool AllZeros = true;
        for (int SubgraphIndex = 0; SubgraphIndex <= SC_LAST; SubgraphIndex++)
        {
            if (Chart2BaseData[SubgraphIndex][Chart2Index] != 0)
            {
                AllZeros = false;
                break;
            }
        }
    }

    if (AllZeros)
        continue;
}

```

```

AllZeros = true;

for (int SubgraphIndex = 0; SubgraphIndex <= SC_LAST; SubgraphIndex++)
{
    if (sc.BaseDataIn[SubgraphIndex][Index] != 0)
    {
        AllZeros = false;
        break;
    }
}

if (AllZeros)
    continue;
}

for (int SubgraphIndex = 0; SubgraphIndex <= SC_NUM_TRADES; SubgraphIndex++)
{
    sc.Subgraph[SubgraphIndex][Index]
    = (sc.BaseDataIn[SubgraphIndex][Index] * Input_Chart1Multiplier.GetFloat() + Input_Chart1Addition.GetFloat())
    - (Chart2BaseData[SubgraphIndex][Chart2Index] * Input_Chart2Multiplier.GetFloat());
}

sc.Subgraph[SC_HIGH][Index] = max(sc.Subgraph[SC_OPEN][Index],
    max(sc.Subgraph[SC_HIGH][Index],
    max(sc.Subgraph[SC_LOW][Index], sc.Subgraph[SC_LAST][Index])
    )
);

sc.Subgraph[SC_LOW][Index] = min(sc.Subgraph[SC_OPEN][Index],
    min(sc.Subgraph[SC_HIGH][Index],
    min(sc.Subgraph[SC_LOW][Index], sc.Subgraph[SC_LAST][Index])
    )
);

sc.CalculateOHLCAverages(Index);
}

sc.EarliestUpdateSubgraphDataArrayIndex = CalculationStartIndex;
}

/*=====*/
SCSFExport scsf_DifferenceBidAskPrices(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_BidPrice = sc.Subgraph[0];
    SCSubgraphRef Subgraph_AskPrice = sc.Subgraph[1];

    SCInputRef Input_Chart2Number = sc.Input[3];
    SCInputRef Input_Chart1Multiplier = sc.Input[4];
    SCInputRef Input_Chart2Multiplier = sc.Input[5];
    SCInputRef Input_Chart1Addition = sc.Input[6];
    SCInputRef Input_ZeroOutputWhenNonExactDateTimeMatch = sc.Input[7];
    SCInputRef Input_UseLatestSourceDataForLastBar = sc.Input[8];
    SCInputRef Input_ZeroOutputWhenOneSourceGraphHasZeroValues = sc.Input[9];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Difference (Bid and Ask Prices)";
        sc.ValueFormat = VALUEFORMAT_INHERITED;
        sc.GraphRegion = 1;

        sc.MaintainAdditionalChartDataArrays = true; // needed to maintain bid and ask arrays in the base graph

        // Using Manual looping.
        sc.AutoLoop = false;
    }
}

```

```

Subgraph_BidPrice.Name = "Bid";
Subgraph_BidPrice.DrawZeros = false;
Subgraph_BidPrice.DrawStyle = DRAWSTYLE_LINE;
Subgraph_BidPrice.LineWidth = 2;
Subgraph_BidPrice.LineLabel = LL_DISPLAY_VALUE | LL_VALUE_ALIGN_VALUES_SCALE |
LL_VALUE_ALIGN_CENTER;

Subgraph_AskPrice.Name = "Ask";
Subgraph_AskPrice.DrawZeros = false;
Subgraph_AskPrice.DrawStyle = DRAWSTYLE_LINE;
Subgraph_AskPrice.LineWidth = 2;
Subgraph_AskPrice.LineLabel = LL_DISPLAY_VALUE | LL_VALUE_ALIGN_VALUES_SCALE |
LL_VALUE_ALIGN_CENTER;


Input_Chart2Number.Name = "Chart 2 Number and Graph";
Input_Chart2Number.SetChartStudyValues(1, 0);

Input_Chart1Multiplier.Name = "Chart 1 Multiplier";
Input_Chart1Multiplier.SetFloat(1.0f);

Input_Chart2Multiplier.Name = "Chart 2 Multiplier";
Input_Chart2Multiplier.SetFloat(1.0f);

Input_Chart1Addition.Name = "Chart 1 Addition";
Input_Chart1Addition.SetFloat(0);

Input_ZeroOutputWhenNonExactDateTimeMatch.Name = "Zero Output When Non Exact Date-Time Match";
Input_ZeroOutputWhenNonExactDateTimeMatch.SetYesNo(false);

Input_UseLatestSourceDataForLastBar.Name = "Use Latest Source Data For Last Bar";
Input_UseLatestSourceDataForLastBar.SetYesNo(0);

Input_ZeroOutputWhenOneSourceGraphHasZeroValues.Name = "Zero Output When one Source Graph has Zero
Values";
Input_ZeroOutputWhenOneSourceGraphHasZeroValues.SetYesNo(0);

return;
}

if (sc.IsFullRecalculation)
{
    // Create text indicating both charts and set to sc.GraphName
    SCString Chart1Name = sc.GetStudyNameFromChart(sc.ChartNumber, 0);
    SCString Chart2Name = sc.GetStudyNameFromChart(Input_Chart2Number.GetChartNumber(),
Input_Chart2Number.GetStudyID());
    sc.GraphName.Format("Difference %s * %g - %s * %g", Chart1Name.GetChars(),
Input_Chart1Multiplier.GetFloat(), Chart2Name.GetChars(), Input_Chart2Multiplier.GetFloat());
}

// Obtain a reference to the Base Data in the specified chart.
SCGraphData Chart2BaseData;
sc.GetStudyArraysFromChartUsingID(-Input_Chart2Number.GetChartNumber(), Input_Chart2Number.GetStudyID(),
Chart2BaseData);

int CalculationStartIndex = sc.GetCalculationStartIndexForStudy();

for (int Index = CalculationStartIndex; Index < sc.ArraySize; Index++)
{
    int Chart2Index = 0;

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

```

```

Chart2Index = sc.GetNearestMatchForDateTimeIndex(Input_Chart2Number.GetChartNumber(), Index);

//When 'use latest source data for last bar' is set to Yes, replay is not running and at last bar in the destination
chart, then use the data from the very latest bar in the source chart.
if (Input_UseLatestSourceDataForLastBar.GetYesNo() && !sc.IsReplayRunning() && Index == sc.ArraySize - 1)
    Chart2Index = Chart2BaseData[0].GetArraySize() - 1;

}
else
{
    Chart2Index = sc.GetExactMatchForSCDateTime(Input_Chart2Number.GetChartNumber(),
sc.BaseDateIn[Index]);

    if (Chart2Index == -1)
    {
        Subgraph_BidPrice[Index] = 0;
        Subgraph_AskPrice[Index] = 0;
        continue;
    }
}

if (Input_ZeroOutputWhenOneSourceGraphHasZeroValues.GetYesNo())
{
    bool AllZeros = true;
    for (int SubgraphIndex = SC_BID_PRICE; SubgraphIndex <= SC_ASK_PRICE; SubgraphIndex++)
    {
        if (Chart2BaseData[SubgraphIndex][Chart2Index] != 0)
        {
            AllZeros = false;
            break;
        }
    }

    if (AllZeros)
        continue;

    for (int SubgraphIndex = SC_BID_PRICE; SubgraphIndex <= SC_ASK_PRICE; SubgraphIndex++)
    {
        if (sc.BaseDataIn[SubgraphIndex][Index] != 0)
        {
            AllZeros = false;
            break;
        }
    }

    if (AllZeros)
        continue;
}

for (int SubgraphIndex = 0; SubgraphIndex <= 1; SubgraphIndex++)
{
    sc.Subgraph[SubgraphIndex][Index]
        = (sc.BaseDataIn[SubgraphIndex == 0 ? SC_BID_PRICE : SC_ASK_PRICE][Index] *
Input_Chart1Multiplier.GetFloat() + Input_Chart1Addition.GetFloat())
        - (Chart2BaseData[SubgraphIndex == 0 ? SC_BID_PRICE : SC_ASK_PRICE][Chart2Index] *
Input_Chart2Multiplier.GetFloat());
}

}

sc.EarliestUpdateSubgraphDataArrayIndex = CalculationStartIndex;
}

```



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

```
SCSFExport scsf_HighLowForTimePeriodExtendedLines(SCStudyInterfaceRef sc)
```

```
{
    SCSubgraphRef Subgraph_HighOfDay = sc.Subgraph[0];
    SCSubgraphRef Subgraph_LowOfDay = sc.Subgraph[1];

    SCInputRef Input_StartTime = sc.Input[0];
    SCInputRef Input_EndTime = sc.Input[1];
    SCInputRef Input_NumberDaysToCalculate = sc.Input[2];
    SCInputRef Input_Version = sc.Input[4];
    SCInputRef Input_LineStopTimeInput = sc.Input[5];
    SCInputRef Input_DisplayHighLowIncrementally = sc.Input[6];

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

    if (sc.SetDefaults)
    {
        sc.GraphName = "High/Low for Time Period - Extended";
        sc.GraphRegion = 0;
        sc.AutoLoop = 0;

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

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

        Input_StartTime.Name = "Start Time";
        Input_StartTime.SetTime(0);

        Input_EndTime.Name = "End Time";
        Input_EndTime.SetTime(SECONDS_PER_DAY - 1);

        Input_NumberDaysToCalculate.Name = "Number Of Days To Calculate";
        Input_NumberDaysToCalculate.SetInt(120);

        Input_LineStopTimeInput.Name = "Line Stop Time";
        Input_LineStopTimeInput.SetTime(SECONDS_PER_DAY - 1);

        Input_DisplayHighLowIncrementally.Name = "Display High Low Incrementally";
        Input_DisplayHighLowIncrementally.SetYesNo(true);

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

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

        Input_FridayEveningExtendsIntoSunday.Name = "Friday Evening Extends Into Sunday";
        Input_FridayEveningExtendsIntoSunday.SetYesNo(false);

        Input_Version.SetInt(3);

        return;
    }

    if (Input_Version.GetInt() == 1)
    {
        Input_InputDataHigh.SetInputDataIndex(SC_HIGH);
    }
}
```

```

    Input_InputDataLow.SetInputDataIndex(SC_LOW);

    Input_Version.SetInt(2);
}

if (Input_Version.GetInt() <= 2)
{
    Input_LineStopTimeInput.SetTime(sc.EndTime1);
    Input_Version.SetInt(3);
}

if (Input_NumberDaysToCalculate.GetInt() <= 0)
    Input_NumberDaysToCalculate.SetInt(10000);

float & HighOfPeriod = sc.GetPersistentFloatFast(1);
float & LowOfPeriod = sc.GetPersistentFloatFast(2);

const bool IsInputTimesReversed = Input_StartTime.GetTime() > Input_EndTime.GetTime();

const bool EndTimeHasMilliseconds = Input_EndTime.GetDateTime().GetMillisecond() > 0;

//Make sure the Line Stop Time is outside of the time range or at the end of the time range.
if (IsInputTimesReversed)
{
    if (Input_LineStopTimeInput.GetTime() >= Input_StartTime.GetTime()
        || Input_LineStopTimeInput.GetTime() < Input_EndTime.GetTime())
    {
        Input_LineStopTimeInput.SetTime(Input_EndTime.GetTime());
    }
}
else
{
    if (Input_LineStopTimeInput.GetTime() >= Input_StartTime.GetTime()
        && Input_LineStopTimeInput.GetTime() < Input_EndTime.GetTime())
    {
        Input_LineStopTimeInput.SetTime(Input_EndTime.GetTime());
    }
}

SCDateTimeMS DaysToCalculateStartDateTime;
DaysToCalculateStartDateTime.SetDate(sc.BaseDateTimeIn[sc.ArraySize - 1].GetDate());
DaysToCalculateStartDateTime.SubtractDays(Input_NumberDaysToCalculate.GetInt() - 1);
DaysToCalculateStartDateTime.SetTime(Input_StartTime.GetTime());

if (IsInputTimesReversed)
{
    if(sc.BaseDateTimeIn[sc.ArraySize - 1].GetTime() < Input_StartTime.GetTime())
        DaysToCalculateStartDateTime.SubtractDays(1);
}

int InitialCalculationIndex = 0;

// Loop through chart bars starting at the Update Start Index
for (int Index = sc.UpdateStartIndex; Index < sc.ArraySize; Index++)
{
    const SCDateTimeMS CurrentBarDateTime = sc.BaseDateTimeIn[Index];

    const SCDateTimeMS PreviousBarDateTime = sc.BaseDateTimeIn[max(0, Index - 1)];
    const SCDateTimeMS NextBarDateTime = sc.BaseDateTimeIn[Index + 1];

    SCDateTimeMS ResetTime;
    ResetTime.SetTime(Input_LineStopTimeInput.GetTime());

    bool NeedReset =

```

```

(PreviousBarDateTime.GetDate() == CurrentBarDateTime.GetDate()
 && PreviousBarDateTime.GetTime() < ResetTime.GetTime()
 && CurrentBarDateTime.GetTime() >= ResetTime.GetTime())
|| (PreviousBarDateTime.GetDate() != CurrentBarDateTime.GetDate()
 && PreviousBarDateTime.GetTime() < ResetTime.GetTime());

```

```

NeedReset |= Index == InitialCalculationIndex;

```

```

if (sc.BaseDateTimeln[Index] < DaysToCalculateStartDateTime)
{
    InitialCalculationIndex = Index + 1;
    continue;
}

```

```

SCDateTimeMS StartDateTime;
SCDateTimeMS EndDateTime;

```

```

StartDateTime = CurrentBarDateTime.GetDate();
EndDateTime = CurrentBarDateTime.GetDate();

```

```

StartDateTime.SetTime(Input_StartTime.GetTime());
EndDateTime.SetTime(Input_EndTime.GetTime());

```

```

if (!EndTimeHasMilliseconds)
{
    //To make EndDateTime at the end of the specified second.
    EndDateTime.AddSeconds(1);
    EndDateTime.SubtractMicroseconds(1);
}

```

```

if (IsInputTimesReversed)
{
    if (CurrentBarDateTime.GetTimelnSeconds() < Input_StartTime.GetTime())
    {
        StartDateTime.SubtractDays(1);
    }
    else
    {
        EndDateTime.AddDays(1);
    }
}

```

```

if(IsInputTimesReversed && Input_FridayEveningExtendsIntoSunday.GetYesNo())
{
    SCDateTime TradingDayDate(sc.GetTradingDayDate(CurrentBarDateTime));
    int DayOfWeek = TradingDayDate.GetDayOfWeek();
    if (DayOfWeek == MONDAY)
    {
        StartDateTime.SubtractDays(2);
        NeedReset = false;
    }
}

```

```

//reset
if (NeedReset)
{
    HighOfPeriod = -FLT_MAX;
    LowOfPeriod = FLT_MAX;
}

```

```

bool OutsideTimeRange = true;

```

```

bool IsCurrentBarContainingOrGreaterThanStartDateTime =

```

```

(CurrentBarDateTime < StartDateTime
  && NextBarDateTime > StartDateTime)
|| CurrentBarDateTime >= StartDateTime;

if (IsCurrentBarContainingOrGreaterThanStartDateTime && CurrentBarDateTime <= EndDateTime)
{
  OutsideTimeRange = false;

  if (HighOfPeriod < sc.BaseData[Input_InputDataHigh.GetInputDataIndex()][Index])
    HighOfPeriod = sc.BaseData[Input_InputDataHigh.GetInputDataIndex()][Index];

  if (LowOfPeriod > sc.BaseData[Input_InputDataLow.GetInputDataIndex()][Index])
    LowOfPeriod = sc.BaseData[Input_InputDataLow.GetInputDataIndex()][Index];
}

if (HighOfPeriod == -FLT_MAX)
  continue;

// Set/update all values for current day
int BackIndex = Index;

while (true)
{
  if(BackIndex < 0)
    break;

  const SCDateTimeMS BackIndexDateTime = sc.BaseDateTimeIn[BackIndex];
  const SCDateTimeMS NextBackIndexDateTime = sc.BaseDateTimeIn[BackIndex+1];

  bool IsCurrentBarContainingOrGreaterThanStartDateTime =
    (BackIndexDateTime < StartDateTime
     && NextBackIndexDateTime > StartDateTime)
    || BackIndexDateTime >= StartDateTime;

  if (!OutsideTimeRange && !IsCurrentBarContainingOrGreaterThanStartDateTime)
    break;

  Subgraph_HighOfDay[BackIndex] = HighOfPeriod;
  Subgraph_LowOfDay[BackIndex] = LowOfPeriod;

  if (OutsideTimeRange || Input_DisplayHighLowIncrementally.GetYesNo())
    break;

  BackIndex--;

  if(sc.UpdateStartIndex != 0 && BackIndex >= 0)
    sc.EarliestUpdateSubgraphDataArrayIndex = BackIndex;
}
}

/*=====*/
SCSFExport scsf_HighLowForTimePeriod(SCStudyInterfaceRef sc)
{
  SCSubgraphRef Subgraph_HighOfDay = sc.Subgraph[0];
  SCSubgraphRef Subgraph_LowOfDay = sc.Subgraph[1];

  SCFloatArrayRef Array_TimeRangeHigh = sc.Subgraph[0].Arrays[0];
  SCFloatArrayRef Array_TimeRangeLow = sc.Subgraph[0].Arrays[1];

  SCInputRef Input_StartTime = sc.Input[0];
  SCInputRef Input_EndTime = sc.Input[1];
  SCInputRef Input_Version = sc.Input[2];
  SCInputRef Input_DataHigh = sc.Input[3];

```

```

SCInputRef Input_DataLow = sc.Input[4];
SCInputRef Input_DisplayHighLowIncrementally = sc.Input[5];
SCInputRef Input_NumberDaysToCalculate = sc.Input[6];

if (sc.SetDefaults)
{
    sc.GraphName          = "High/Low for Time Period";
    sc.GraphRegion        = 0;
    sc.AutoLoop           = 0;

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

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

    Input_StartTime.Name = "Start Time";
    Input_StartTime.SetTime(0);

    Input_EndTime.Name = "End Time";
    Input_EndTime.SetTime(SECONDS_PER_DAY - 1);

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

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

    Input_DisplayHighLowIncrementally.Name = "Display High Low Incrementally";
    Input_DisplayHighLowIncrementally.SetYesNo(false);

    Input_NumberDaysToCalculate.Name = "Number Of Days To Calculate";
    Input_NumberDaysToCalculate.SetInt(10000);

    Input_Version.SetInt(2);

    return;
}

if(Input_Version.GetInt() < 2 )
{
    Input_DataHigh.SetInputDataIndex(SC_HIGH);
    Input_DataLow.SetInputDataIndex(SC_LOW);
    Input_Version.SetInt(2);
}

if (Input_NumberDaysToCalculate.GetInt() <= 0)
    Input_NumberDaysToCalculate.SetInt(10000);

int InputDataHighIndex = Input_DataHigh.GetInputDataIndex();
int InputDataLowIndex = Input_DataLow.GetInputDataIndex();

SCDateTimeMS InStartTime = Input_StartTime.GetDateTime();
SCDateTimeMS InEndTime = Input_EndTime.GetDateTime();

const bool IsInputTimesReversed = InStartTime.GetTimeInSeconds() > InEndTime.GetTimeInSeconds();

const bool EndTimeHasMilliseconds = InEndTime.GetMillisecond() > 0;

SCDateTimeMS DaysToCalculateStartTime;

```

```

DaysToCalculateStartDateTime.SetDate(sc.BaseDateTimeIn[sc.ArraySize - 1].GetDate());
DaysToCalculateStartDateTime.SubtractDays(Input_NumberDaysToCalculate.GetInt() - 1);
DaysToCalculateStartDateTime.SetTime(Input_StartTime.GetTime());

if (IsInputTimesReversed)
{
    if (sc.BaseDateTimeIn[sc.ArraySize - 1].GetTime() < InStartTime.GetTime())
        DaysToCalculateStartDateTime.SubtractDays(1);
}

for (int Index = sc.UpdateStartIndex; Index < sc.ArraySize; Index++)
{
    if (sc.BaseDateTimeIn[Index] < DaysToCalculateStartDateTime)
        continue;

    SCDatetimeMS StartDateTime;
    SCDatetimeMS EndDateTime;

    SCDatetimeMS BarDate = sc.BaseDateTimeIn[Index].GetDate();

    if (!IsInputTimesReversed)
    {
        StartDateTime = BarDate + InStartTime;
        EndDateTime = BarDate + InEndTime;
    }
    //Times are reversed and the current time is greater than or equal to the start time
    else if (sc.BaseDateTimeIn[Index].GetTimeInSeconds() >= InStartTime.GetTimeInSeconds() )
    {
        StartDateTime = BarDate + InStartTime;
        EndDateTime = BarDate + SCDatetime::DAYS(1) + InEndTime;
    }
    //Times are reversed and the current time is less than or equal to the end time
    else if (sc.BaseDateTimeIn[Index].GetTimeInSeconds() <= InEndTime.GetTimeInSeconds() )
    {
        StartDateTime = BarDate + InStartTime - SCDatetime::DAYS(1);
        EndDateTime = BarDate + InEndTime;
    }

    if (!EndTimeHasMilliseconds)
    {
        //To make EndDateTime at the end of the specified second.
        EndDateTime.AddSeconds(1);
        EndDateTime.SubtractMicroseconds(1);
    }

    //Initial calculations or start of new time range
    if (Index == 0 || sc.BaseDateTimeIn[Index - 1] < StartDateTime)
    {
        Array_TimeRangeHigh[Index] = -FLT_MAX;
        Array_TimeRangeLow[Index] = FLT_MAX;
    }
    else
    {
        Array_TimeRangeHigh[Index] = Array_TimeRangeHigh[Index - 1];
        Array_TimeRangeLow[Index] = Array_TimeRangeLow[Index - 1];
    }

    //Outside of range
    if (sc.BaseDateTimeIn[Index] > EndDateTime || sc.BaseDateTimeIn[Index] < StartDateTime)
        continue;
}

```

```

if(sc.BaseData[InputDataHighIndex].GetArraySize() > 0)
{
    if(Array_TimeRangeHigh[Index] < sc.BaseData[InputDataHighIndex][Index])
        Array_TimeRangeHigh[Index] = sc.BaseData[InputDataHighIndex][Index];
}
else
{
    if(Array_TimeRangeHigh[Index] < sc.BaseData[0][Index])
        Array_TimeRangeHigh[Index] = sc.BaseData[0][Index];
}

if(sc.BaseData[InputDataLowIndex].GetArraySize() > 0)
{
    if(Array_TimeRangeLow[Index] > sc.BaseData[InputDataLowIndex][Index])
        Array_TimeRangeLow[Index] = sc.BaseData[InputDataLowIndex][Index];
}
else
{
    if(Array_TimeRangeLow[Index] < sc.BaseData[0][Index])
        Array_TimeRangeLow[Index] = sc.BaseData[0][Index];
}

if (Input_DisplayHighLowIncrementally.GetYesNo())
{
    Subgraph_HighOfDay[Index] = Array_TimeRangeHigh[Index];
    Subgraph_LowOfDay[Index] = Array_TimeRangeLow[Index];
}
else
{
    int BackIndex = Index;

    while(true)
    {
        if(BackIndex < 0)
            break;

        SCDateTimeMS IndexDateTime = sc.BaseDateTimeln[BackIndex];

        if(IndexDateTime < StartDateTime)
            break;

        Subgraph_HighOfDay[BackIndex] = Array_TimeRangeHigh[Index];
        Subgraph_LowOfDay[BackIndex] = Array_TimeRangeLow[Index];

        BackIndex--;

        if(sc.UpdateStartIndex != 0)
            sc.EarliestUpdateSubgraphDataArrayIndex = BackIndex;
    }
}
}

```

```

/*=====*/
SCSFExport scsf_WriteBarDataToFile(SCStudyInterfaceRef sc)
{
    SCInputRef Input_Separator = sc.Input[0];
    SCInputRef Input_UseUTCTime = sc.Input[1];

    if (sc.SetDefaults)

```

```

{
    sc.GraphName = "Write Bar Data To File";
    sc.StudyDescription = "Write Bar Data To File";

    sc.GraphRegion = 0;

    Input_Separator.Name = "Separator";
    Input_Separator.SetCustomInputStrings("Comma;Tab");
    Input_Separator.SetCustomInputIndex(0);

    Input_UseUTCTime.Name = "Use UTC Time";
    Input_UseUTCTime.SetYesNo(0);

    sc.TextInputName = "File Path";

    sc.AutoLoop = 0;//manual looping for efficiency
    return;
}

if (sc.LastCallToFunction)
    return;

SCString OutputPathAndFileName;
if (!sc.TextInput.IsEmpty())
{
    OutputPathAndFileName = sc.TextInput;
}
else
{
    OutputPathAndFileName = sc.DataFilesFolder();
    OutputPathAndFileName += "\\";
    OutputPathAndFileName += sc.Symbol.GetChars();
    OutputPathAndFileName += "-BarData.txt";
}

n_ACSIL::s_WriteBarAndStudyDataToFile WriteBarAndStudyDataToFileParams;
WriteBarAndStudyDataToFileParams.StartingIndex = sc.UpdateStartIndex;
WriteBarAndStudyDataToFileParams.OutputPathAndFileName = OutputPathAndFileName;
WriteBarAndStudyDataToFileParams.IncludeHiddenStudies = 0;
WriteBarAndStudyDataToFileParams.IncludeHiddenSubgraphs = 0;
WriteBarAndStudyDataToFileParams.AppendOnlyToFile = 0;
WriteBarAndStudyDataToFileParams.IncludeLastBar = 0;
WriteBarAndStudyDataToFileParams.UseUTCTime = Input_UseUTCTime.GetYesNo();
WriteBarAndStudyDataToFileParams.WriteStudyData = 0;
WriteBarAndStudyDataToFileParams.UseTabDelimiter = Input_Separator.GetInt() == 1;
sc.WriteBarAndStudyDataToFileEx(WriteBarAndStudyDataToFileParams);
}

/*=====*/
SCSFExport scsf_WriteBarAndStudyDataToFile(SCStudyInterfaceRef sc)
{
    SCInputRef Input_IncludeHiddenStudies = sc.Input[0];
    SCInputRef Input_UseGMTTime = sc.Input[1];
    SCInputRef Input_IncludeHiddenSubgraphs = sc.Input[2];
    SCInputRef Input_OutputMilliseconds = sc.Input[3];

    if(sc.SetDefaults)
    {
        sc.GraphName="Write Bar and Study Data To File";

        sc.GraphRegion = 0;

        Input_IncludeHiddenStudies.Name = "Include Hidden Studies";
        Input_IncludeHiddenStudies.SetYesNo(true);
    }

```



```

//Input_UseGMTTime.Name = "Use GMT Time";
//Input_UseGMTTime.SetYesNo(0);

Input_IncludeHiddenSubgraphs.Name = "Include Hidden Subgraphs";
Input_IncludeHiddenSubgraphs.SetYesNo(false);

//Input_OutputMilliseconds.Name = "Output Milliseconds";
//Input_OutputMilliseconds.SetYesNo(false);

sc.TextInputName = "Path and File Name";

sc.CalculationPrecedence = VERY_LOW_PREC_LEVEL;

sc.AutoLoop = 0; //manual looping required
return;
}

if (sc.LastCallToFunction)
    return;

SCString OutputPathAndFileName;
if(!sc.TextInput.IsEmpty())
{
    OutputPathAndFileName = sc.TextInput;
}
else
{
    OutputPathAndFileName.Format("%s-BarStudyData.csv", sc.Symbol.GetChars());
}

//sc.WriteBarAndStudyDataToFile(sc.UpdateStartIndex, OutputPathAndFileName, IncludeHiddenStudies.GetYesNo(),
IncludeHiddenSubgraphs.GetYesNo());

n_ACSIL::s_WriteBarAndStudyDataToFile WriteBarAndStudyDataToFileParams;
WriteBarAndStudyDataToFileParams.StartingIndex = sc.UpdateStartIndex;
WriteBarAndStudyDataToFileParams.OutputPathAndFileName = OutputPathAndFileName;
WriteBarAndStudyDataToFileParams.IncludeHiddenStudies = Input_IncludeHiddenStudies.GetYesNo();
WriteBarAndStudyDataToFileParams.IncludeHiddenSubgraphs = Input_IncludeHiddenSubgraphs.GetYesNo();
WriteBarAndStudyDataToFileParams.AppendOnlyToFile = 0;
WriteBarAndStudyDataToFileParams.IncludeLastBar = 0;
sc.WriteBarAndStudyDataToFileEx(WriteBarAndStudyDataToFileParams);

return;
}

/*=====*/
bool IsFractalBuySignal(SCFloatArrayRef InDataLow, SCFloatArrayRef InDataHigh, int Index); //Long Fractal

bool IsFractalSellSignal(SCFloatArrayRef InDataLow, int Index); //Short Fractal

SCSFExport scsf_FractalSignals(SCStudyInterfaceRef sc)
{
    enum SubgraphIndexes
    {
        INDEX_SIGNAL_BUY = 0,
        INDEX_SIGNAL_SELL
    };

    SCSubgraphRef Subgraph_Buy = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Sell = sc.Subgraph[1];

    SCInputRef Input_SubgraphOffset = sc.Input[0];
    SCInputRef Input_Version = sc.Input[6];

```

```

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

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

    Subgraph_Buy.Name = "Buy";
    Subgraph_Buy.DrawStyle = DRAWSTYLE_TRIANGLE_UP;
    Subgraph_Buy.PrimaryColor = RGB(0, 255, 0);
    Subgraph_Buy.LineWidth = 5;
    Subgraph_Buy.DrawZeros = false;

    Subgraph_Sell.Name = "Sell";
    Subgraph_Sell.DrawStyle = DRAWSTYLE_TRIANGLE_DOWN;
    Subgraph_Sell.PrimaryColor = RGB(255, 0, 0);
    Subgraph_Sell.LineWidth = 5;
    Subgraph_Sell.DrawZeros = false;

    Input_Version.SetInt(2);

    Input_SubgraphOffset.Name = "Arrow Offset Percentage";
    Input_SubgraphOffset.SetFloat(2.0f);

    sc.AutoLoop = 0;

    return;
}

int CalculationStartIndex = max(0, sc.UpdateStartIndex - 2);
sc.EarliestUpdateSubgraphDataArrayIndex = CalculationStartIndex;

for (int Index = CalculationStartIndex; Index < sc.ArraySize; Index++)
{
    float Range = sc.High[Index] - sc.Low[Index];
    float Offset = Range * (Input_SubgraphOffset.GetFloat()*0.01f);

    // detect buy fractal signal
    if(IsFractalBuySignal(sc.Low, sc.High, Index))
        Subgraph_Buy[Index] = sc.High[Index] + Offset;
    else
        Subgraph_Buy[Index] = 0;

    // detect sell fractal signal
    if(IsFractalSellSignal(sc.Low, Index))
        Subgraph_Sell[Index] = sc.Low[Index] - Offset;
    else
        Subgraph_Sell[Index] = 0;
}
}

/*=====*/
bool IsFractalBuySignal(SCFloatArrayRef InDataLow, SCFloatArrayRef InDataHigh, int Index)
{
    int ArraySize = InDataLow.GetArraySize();

    if (
        Index + 2 < ArraySize
        && Index - 2 >= 0
        && InDataHigh[Index] > InDataHigh[Index + 1]
        && InDataHigh[Index] > InDataHigh[Index + 2]
        && InDataHigh[Index] > InDataHigh[Index - 1]

```

```

        && InDataHigh[Index] > InDataHigh[Index - 2]
    )
    return true;

return false;
}

/*=====*/
bool IsFractalSellSignal(SCFloatArrayRef InDataLow, int Index)
{
    int ArraySize = InDataLow.GetArraySize();

    if (
        Index + 2 < ArraySize
        && Index - 2 >= 0
        && InDataLow[Index] < InDataLow[Index + 1]
        && InDataLow[Index] < InDataLow[Index + 2]
        && InDataLow[Index] < InDataLow[Index - 1]
        && InDataLow[Index] < InDataLow[Index - 2]
    )
        return true;

    return false;
}

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

    SCInputRef Input_Data = sc.Input[0];

    if(sc.SetDefaults)
    {
        sc.GraphName="Round Study Subgraph to Tick Size";
        sc.StudyDescription="Rounds the elements in the array from the Base Data selected by the Input Data option, to the
nearest value based upon the Tick Size setting in the chart";

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

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

        sc.AutoLoop = 0;
        return;
    }

    int CalculationStartIndex = sc.GetCalculationStartIndexForStudy();

    for (int Index = CalculationStartIndex; Index < sc.ArraySize; Index++)
    {
        Subgraph_Value[Index] = static_cast<float>(sc.RoundToTickSize(sc.BaseDataIn[Input_Data.GetInputDataIndex()]
[Index], sc.TickSize));
    }

    sc.EarliestUpdateSubgraphDataArrayIndex = CalculationStartIndex;
}

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

```

```

SCSubgraphRef Subgraph_Macd    = sc.Subgraph[0];
SCSubgraphRef Subgraph_MaMacd  = sc.Subgraph[1];
SCSubgraphRef Subgraph_MacdDiff = sc.Subgraph[2];
SCSubgraphRef Subgraph_RefLine = sc.Subgraph[3];

SCInputRef Input_FastLen      = sc.Input[0];
SCInputRef Input_SlowLen      = sc.Input[1];
SCInputRef Input_MacdLen      = sc.Input[2];
SCInputRef Input_Data         = sc.Input[3];

if(sc.SetDefaults)
{
    sc.GraphName="MACD-Volume Weighted";

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

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

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

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

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

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

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

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

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

    return;
}

sc.DataStartIndex = Input_MacdLen.GetInt() + max(Input_FastLen.GetInt(), Input_SlowLen.GetInt());
int i = sc.Index;

SCFloatArrayRef FastOut = Subgraph_Macd.Arrays[0];
sc.VolumeWeightedMovingAverage(sc.BaseDataIn[Input_Data.GetInputDataIndex()], sc.Volume, FastOut,
Input_FastLen.GetInt());
SCFloatArrayRef SlowOut = Subgraph_Macd.Arrays[1];
sc.VolumeWeightedMovingAverage(sc.BaseDataIn[Input_Data.GetInputDataIndex()], sc.Volume, SlowOut,

```

```

Input_SlowLen.GetInt());

Subgraph_Macd[i] = FastOut[i] - SlowOut[i];

if(i < max(Input_SlowLen.GetInt(), Input_FastLen.GetInt()) + Input_MacdLen.GetInt())
    return;

sc.MovingAverage(Subgraph_Macd, Subgraph_MaMacd, MOVAVGTYPE_EXPONENTIAL, Input_MacdLen.GetInt());

Subgraph_MacdDiff[i] = Subgraph_Macd[i] - Subgraph_MaMacd[i];
Subgraph_RefLine[i] = 0;
}

/*=====*/
SCSFExport scsf_StandardDeviation(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Deviation = sc.Subgraph[0];
    SCInputRef Input_Data = sc.Input[0];
    SCInputRef Input_Length = sc.Input[1];

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

        sc.AutoLoop = 1;

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

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

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

        return;
    }

    sc.DataStartIndex = Input_Length.GetInt();

    sc.StdDeviation(sc.BaseDataIn[Input_Data.GetInputDataIndex()], Subgraph_Deviation, Input_Length.GetInt());
}

/*=====*/
SCSFExport scsf_AutoRetracementProjection(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_PeriodHigh = sc.Subgraph[0];
    SCSubgraphRef Subgraph_PeriodLow = sc.Subgraph[1];
    const int SubgraphLevelsStartIndex = 2;
    const int NumberOfLevels = 14;

    SCInputRef Input_TimePeriodType = sc.Input[0];
    SCInputRef Input_TimePeriodLength = sc.Input[1];
    SCInputRef Input_PlotType = sc.Input[2];
    SCInputRef Input_Version = sc.Input[17];
    SCInputRef Input_NewPeriodDaySessionStart = sc.Input[18];
    SCInputRef Input_CalculationReferenceValue = sc.Input[19];
    SCInputRef Input_RoundToTickSize = sc.Input[20];
    SCInputRef Input_ReferenceSingleSubgraph = sc.Input[21];
    SCInputRef Input_NumberOfDaysToCalculate = sc.Input[22];

```

```

if(sc.SetDefaults)
{
    sc.GraphName = "Auto Retracement/Projection";
    sc.GraphRegion = 0;
    sc.AutoLoop = 0;//Manual looping
    Subgraph_PeriodHigh.Name = "Period High";
    Subgraph_PeriodHigh.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_PeriodHigh.PrimaryColor = RGB(255, 165, 0);
    Subgraph_PeriodHigh.LineWidth = 2;
    Subgraph_PeriodHigh.DrawZeros = false;

    Subgraph_PeriodLow.Name = "Period Low";
    Subgraph_PeriodLow.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_PeriodLow.PrimaryColor = RGB(255, 165, 0);
    Subgraph_PeriodLow.LineWidth = 2;
    Subgraph_PeriodLow.DrawZeros = false;

    for (int LevelIndex = 0; LevelIndex < NumberOfLevels; LevelIndex++)
    {
        SCString TmpStr;
        TmpStr.Format("Level%d", LevelIndex + 1);
        sc.Subgraph[LevelIndex + SubgraphLevelsStartIndex].Name = TmpStr;
        sc.Subgraph[LevelIndex + SubgraphLevelsStartIndex].DrawStyle = DRAWSTYLE_DASH;
        sc.Subgraph[LevelIndex + SubgraphLevelsStartIndex].LineStyle = LINESTYLE_SOLID;
        sc.Subgraph[LevelIndex + SubgraphLevelsStartIndex].LineWidth = 2;
        sc.Subgraph[LevelIndex + SubgraphLevelsStartIndex].PrimaryColor = RGB(0, 255, 255);
        sc.Subgraph[LevelIndex + SubgraphLevelsStartIndex].DrawZeros = false;

        TmpStr.Format("Level %d Percentage .01 = 1%%", LevelIndex + 1);
        sc.Input[3 + LevelIndex].Name = TmpStr;
    }

    Input_PlotType.Name = "Calculation Method";
    Input_PlotType.SetCustomInputStrings("Projection;Retracement");
    Input_PlotType.SetCustomInputIndex( 1);

    sc.Input[3].SetFloat(0.236f);
    sc.Input[4].SetFloat(0.382f);
    sc.Input[5].SetFloat(0.50f);
    sc.Input[6].SetFloat(0.618f);

    for (int LevelIndex = 7; LevelIndex < NumberOfLevels + 3; LevelIndex++)
        sc.Input[LevelIndex].SetFloat(0.0f);

    Input_TimePeriodType.Name = "Time Period Unit";
    Input_TimePeriodType.SetTimePeriodType(TIME_PERIOD_LENGTH_UNIT_DAYS);

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

    Input_Version.SetInt(2);

    Input_NewPeriodDaySessionStart .Name = "New Period at Day Session Start when Using Evening Session";
    Input_NewPeriodDaySessionStart.SetYesNo(false);

    Input_CalculationReferenceValue.Name = "Calculation Reference Value";
    Input_CalculationReferenceValue.SetCustomInputStrings("Automatic;Use High;Use Low");
    Input_CalculationReferenceValue.SetCustomInputIndex(1);

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

    Input_ReferenceSingleSubgraph.Name = "Reference Single Subgraph";
    Input_ReferenceSingleSubgraph.SetYesNo(false);

```

```

Input_NumberOfDaysToCalculate.Name = "Number of Days to Calculate";
Input_NumberOfDaysToCalculate.SetInt(120);
Input_NumberOfDaysToCalculate.SetIntLimits(1, MAX_STUDY_LENGTH);

return;
}

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

if (Input_Version.GetInt() < 2)
{
    Input_NumberOfDaysToCalculate.SetInt(1000);
    Input_Version.SetInt(2);
}

float& OpenOfDay = sc.GetPersistentFloat(1);

int TimePeriodTypeInput = Input_TimePeriodType.GetTimePeriodType();
int TimePeriodLengthInput = Input_TimePeriodLength.GetInt();

float LevelsPercentages[NumberOfLevels];
for (int Index = 0; Index < NumberOfLevels; Index++)
{
    LevelsPercentages[Index] = sc.Input[Index + 3].GetFloat();
}

int OpenSubgraph = SC_OPEN;
int HighSubgraph = SC_HIGH;
int LowSubgraph = SC_LOW;
int LastSubgraph = SC_LAST;

if (Input_ReferenceSingleSubgraph.GetYesNo())
{
    OpenSubgraph = SC_OPEN;
    HighSubgraph = SC_OPEN;
    LowSubgraph = SC_OPEN;
    LastSubgraph = SC_OPEN;
}

// Main loop
for (int Index = sc.UpdateStartIndex; Index < sc.ArraySize; Index++)
{
    SCDatetime TradingDayDateForBar = sc.GetTradingDayDate(sc.BaseDateTimeIn[Index]);
    SCDatetime TradingDayDateForLastBar = sc.GetTradingDayDate(sc.BaseDateTimeIn[sc.ArraySize - 1]);

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

    SCDatetime PriorCurrentPeriodStartDateTime = sc.GetStartOfPeriodForDateTime(sc.BaseDateTimeIn[Index - 1],
    TimePeriodTypeInput, TimePeriodLengthInput, 0, Input_NewPeriodDaySessionStart.GetYesNo());

    SCDatetime CurrentPeriodStartDateTime = sc.GetStartOfPeriodForDateTime(sc.BaseDateTimeIn[Index],
    TimePeriodTypeInput, TimePeriodLengthInput, 0, Input_NewPeriodDaySessionStart.GetYesNo());

    bool IsInNewPeriod = PriorCurrentPeriodStartDateTime != CurrentPeriodStartDateTime ;

```

```

if (IsInNewPeriod) //reset values
{
    OpenOfDay = sc.BaseData[OpenSubgraph][Index];
    Subgraph_PeriodHigh[Index] = sc.BaseData[HighSubgraph][Index];
    Subgraph_PeriodLow[Index] = sc.BaseData[LowSubgraph][Index];
}
else
{
    if (sc.BaseData[HighSubgraph][Index] > Subgraph_PeriodHigh[Index - 1] || Index == 0)
        Subgraph_PeriodHigh[Index] = sc.BaseData[HighSubgraph][Index];
    else
        Subgraph_PeriodHigh[Index] = Subgraph_PeriodHigh[Index - 1];

    if (sc.BaseData[LowSubgraph][Index] < Subgraph_PeriodLow[Index - 1] || Index == 0)
        Subgraph_PeriodLow[Index] = sc.BaseData[LowSubgraph][Index];
    else
        Subgraph_PeriodLow[Index] = Subgraph_PeriodLow[Index - 1];
}

float Range = Subgraph_PeriodHigh[Index] - Subgraph_PeriodLow[Index];

for (int LevelIndex = 0; LevelIndex < NumberOfLevels; LevelIndex++)
{
    if (LevelsPercentages[LevelIndex] == 0)
        continue;

    float Value = 0;

    if (Input_PlotType.GetInt() == 1) //Retracement
    {
        //rising prices
        if ((Input_CalculationReferenceValue.GetIndex() == 0 && sc.BaseData[LastSubgraph][Index] >= OpenOfDay)
||Input_CalculationReferenceValue.GetIndex() == 1)
            Value = Subgraph_PeriodHigh[Index] - LevelsPercentages[LevelIndex] * Range;
        else
            Value = LevelsPercentages[LevelIndex] * Range + Subgraph_PeriodLow[Index];
    }
    else //Projection
    {
        if ((Input_CalculationReferenceValue.GetIndex() == 0 && sc.BaseData[LastSubgraph][Index] >= OpenOfDay)
||Input_CalculationReferenceValue.GetIndex() == 1)
            Value = LevelsPercentages[LevelIndex] * Range + Subgraph_PeriodHigh[Index];
        else
            Value = Subgraph_PeriodLow[Index] - LevelsPercentages[LevelIndex] * Range;
    }

    if (Input_RoundToTickSize.GetYesNo())
        Value = static_cast<float>(sc.RoundToTickSize(Value, sc.TickSize));

    sc.Subgraph[SubgraphLevelsStartIndex + LevelIndex][Index] = Value;
}
}
}

```

/\*=====\*/

SCSFExport scsf\_NewHighLowAlert(SCStudyInterfaceRef sc)

```

{
    SCInputRef Input_HighAlert = sc.Input[0];
    SCInputRef Input_LowAlert = sc.Input[1];

    if(sc.SetDefaults)
    {
        sc.GraphName = "New Daily High/Low Alert";
    }
}

```



```

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

Input_HighAlert.Name = "High Alert Sound";
Input_HighAlert.SetAlertSoundNumber(2);

Input_LowAlert.Name = "Low Alert Sound";
Input_LowAlert.SetAlertSoundNumber(2);

return;
}

float & PreviousHigh = sc.GetPersistentFloatFast(1);
float & PreviousLow = sc.GetPersistentFloatFast(2);

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

float DailyOpen;
float DailyHigh;
float DailyLow;
float DailyClose;

if(!sc.GetOHLCForDate(TradingDayDate, DailyOpen, DailyHigh, DailyLow, DailyClose))
    return;

if(sc.UpdateStartIndex == 0)
{
    PreviousHigh = DailyHigh;
    PreviousLow = DailyLow;

    return;
}

SCString Message;
if(Input_HighAlert.GetAlertSoundNumber() > 0 && PreviousHigh != 0.0 && DailyHigh > PreviousHigh)
{
    Message.Format("New Daily High: %.f. Previous high: %.f", DailyHigh, PreviousHigh);
    sc.PlaySound(Input_HighAlert.GetAlertSoundNumber() - 1, Message);
}

if(Input_LowAlert.GetAlertSoundNumber() > 0 && PreviousLow != 0.0 && DailyLow < PreviousLow)
{
    Message.Format("New Daily Low: %.f. Previous low: %.f", DailyLow, PreviousLow);
    sc.PlaySound(Input_LowAlert.GetAlertSoundNumber() - 1, Message);
}

PreviousHigh = DailyHigh;
PreviousLow = DailyLow;
}

/*=====*/
SCSFExport scsf_Spread4Chart(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_Volume = sc.Subgraph[4];
    SCSubgraphRef Subgraph_OpenInterest = sc.Subgraph[5];
    SCSubgraphRef Subgraph_OHLCAvg = sc.Subgraph[6];
    SCSubgraphRef Subgraph_HLCAvg = sc.Subgraph[7];
    SCSubgraphRef Subgraph_HLAvG = sc.Subgraph[8];

```

```
SCInputRef Input_Chart2Number = sc.Input[0];
SCInputRef Input_Chart3Number = sc.Input[1];
SCInputRef Input_Chart4Number = sc.Input[2];
```

```
SCInputRef Input_Chart1Multiplier = sc.Input[3];
SCInputRef Input_Chart2Multiplier = sc.Input[4];
SCInputRef Input_Chart3Multiplier = sc.Input[5];
SCInputRef Input_Chart4Multiplier = sc.Input[6];
```

```
if (sc.SetDefaults)
{
    sc.GraphName          = "Spread - 4 Chart";
    sc.GraphRegion        = 1;
    sc.UseGlobalChartColors = 1;
    sc.GraphDrawType      = GDT_OHLCBAR;
    sc.StandardChartHeader = 1;

    // We are using Manual looping.
    sc.AutoLoop = false;

    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(0,255,0);
    Subgraph_Low.DrawZeros = false;

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

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

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

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

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

    Subgraph_HLAvg.Name = "HL Avg";
    Subgraph_HLAvg.DrawStyle = DRAWSTYLE_IGNORE;
```

```

Subgraph_HLAvg.PrimaryColor = RGB(0,127,255);
Subgraph_HLAvg.DrawZeros = false;

Input_Chart2Number.Name = "Chart 2 Number";
Input_Chart2Number.SetChartNumber(1);

Input_Chart3Number.Name = "Chart 3 Number";
Input_Chart3Number.SetChartNumber(1);

Input_Chart4Number.Name = "Chart 4 Number";
Input_Chart4Number.SetChartNumber(1);

Input_Chart1Multiplier.Name = "Chart 1 Multiplier";
Input_Chart1Multiplier.SetFloat(1.0);

Input_Chart2Multiplier.Name = "Chart 2 Multiplier";
Input_Chart2Multiplier.SetFloat(1.0);

Input_Chart3Multiplier.Name = "Chart 3 Multiplier";
Input_Chart3Multiplier.SetFloat(1.0);

Input_Chart4Multiplier.Name = "Chart 4 Multiplier";
Input_Chart4Multiplier.SetFloat(1.0);

return;
}

```

// Obtain a reference to the Base Data in the specified chart. This call is relatively efficient, but it should be called as minimally as possible. To reduce the number of calls we have it outside of the primary "for" loop in this study function. And we also use Manual Looping by using `sc.AutoLoop = 0`. In this way, `sc.GetChartBaseData` is called only once per call to this study function and there are minimal number of calls to the function. `sc.GetChartBaseData` is a new function to get all of the Base Data arrays with one efficient call.

```

SCGraphData Chart2BaseData;
sc.GetChartBaseData(-Input_Chart2Number.GetChartNumber(), Chart2BaseData);

SCGraphData Chart3BaseData;
sc.GetChartBaseData(-Input_Chart3Number.GetChartNumber(), Chart3BaseData);

SCGraphData Chart4BaseData;
sc.GetChartBaseData(-Input_Chart4Number.GetChartNumber(), Chart4BaseData);

for (int Index = sc.UpdateStartIndex; Index < sc.ArraySize; Index++)
{
    int Chart2Index = sc.GetNearestMatchForDateTimeIndex(Input_Chart2Number.GetChartNumber(), Index);
    int Chart3Index = sc.GetNearestMatchForDateTimeIndex(Input_Chart3Number.GetChartNumber(), Index);
    int Chart4Index = sc.GetNearestMatchForDateTimeIndex(Input_Chart4Number.GetChartNumber(), Index);

    for (int SubgraphIndex = 0; SubgraphIndex < 6; SubgraphIndex++)
    {
        sc.Subgraph[SubgraphIndex][Index]
        = (sc.BaseDataIn[SubgraphIndex][Index] * Input_Chart1Multiplier.GetFloat()
        - Chart2BaseData[SubgraphIndex][Chart2Index] * Input_Chart2Multiplier.GetFloat()
        - (Chart3BaseData[SubgraphIndex][Chart3Index] * Input_Chart3Multiplier.GetFloat()
        - Chart4BaseData[SubgraphIndex][Chart4Index] * Input_Chart4Multiplier.GetFloat()));
    }

    sc.Subgraph[ SC_HIGH][Index] = max(sc.Subgraph[SC_OPEN][Index],
    max(sc.Subgraph[SC_HIGH][Index],
    max(sc.Subgraph[SC_LOW][Index],sc.Subgraph[SC_LAST][Index])
    )
    );

    sc.Subgraph[SC_LOW][Index] = min(sc.Subgraph[SC_OPEN][Index],
    min(sc.Subgraph[SC_HIGH][Index],

```

```

        min(sc.Subgraph[SC_LOW][Index],sc.Subgraph[SC_LAST][Index])
    )
);

sc.CalculateOHLCAverages(Index);
}

//SCString Chart1Name = sc.GetStudyNameFromChart(sc.ChartNumber, 0);
//SCString Chart2Name = sc.GetStudyNameFromChart(Chart2Number.GetChartNumber(), 0);
//sc.GraphName.Format("Difference %s - %s", Chart1Name.GetChars(), Chart2Name.GetChars());
}

/*=====*/

```

```

SCSFExport scsf_QStick(SCStudyInterfaceRef sc)

```

```

{
    SCSubgraphRef Subgraph_QStick = sc.Subgraph[0];
    SCInputRef Input_Length = sc.Input[0];
    if (sc.SetDefaults)
    {
        // Set the configuration and defaults

        sc.GraphName = "Q Stick";
        sc.AutoLoop = 1;

        Subgraph_QStick.Name = "QS";
        Subgraph_QStick.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_QStick.LineWidth = 2;
        Subgraph_QStick.PrimaryColor = RGB(0,255,0);
        Input_Length.Name = "Length";
        Input_Length.SetInt(4);
        sc.GraphRegion = 1;

        return;
    }
}

```

```

// Do data processing

```

```

Subgraph_QStick.Arrays[0][sc.Index] = sc.BaseData[SC_LAST][sc.Index] - sc.BaseData[SC_OPEN][sc.Index] ;
sc.Summation( Subgraph_QStick.Arrays[0], Subgraph_QStick.Arrays[1], Input_Length.GetInt() );

```

```

Subgraph_QStick.Data[sc.Index] = Subgraph_QStick.Arrays[1][sc.Index] / Input_Length.GetInt() ;

```

```

}

```

```

/*=====*/

```

```

SCSFExport scsf_HorizontalLineAtTime(SCStudyInterfaceRef sc)

```

```

{
    SCSubgraphRef Subgraph_HorizontalLine = sc.Subgraph[0];
    SCSubgraphRef Subgraph_ExtensionLines = sc.Subgraph[1];

    SCInputRef Input_Data = sc.Input[0];

    SCInputRef Input_StartTime = sc.Input[1];
    SCInputRef Input_UseStopTime = sc.Input[2];
    SCInputRef Input_StopTime = sc.Input[3];
    SCInputRef Input_LimitHorizontalLineFromTimeToOneDay = sc.Input[4];
    SCInputRef Input_IgnoreWeekends = sc.Input[5];
    SCInputRef Input_AddExtensionLinesUntilFutureIntersection = sc.Input[6];
    SCInputRef Input_UseAllowedRangeForMatchInMinutes = sc.Input[7];
}

```

```
SCInputRef Input_AllowedRangeForMatchInMinutes = sc.Input[8];
```

```
if (sc.SetDefaults)
{
    sc.GraphName = "Horizontal Line at Time";
    sc.GraphRegion = 0;
    sc.AutoLoop = 1;

    Subgraph_HorizontalLine.Name = "Line";
    Subgraph_HorizontalLine.DrawStyle = DRAWSTYLE_STAIR_STEP;
    Subgraph_HorizontalLine.LineWidth = 2;
    Subgraph_HorizontalLine.PrimaryColor = RGB(0, 255, 0);
    Subgraph_HorizontalLine.DrawZeros = false;

    Subgraph_ExtensionLines.Name = "Extension Lines";
    Subgraph_ExtensionLines.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_ExtensionLines.LineWidth = 2;
    Subgraph_ExtensionLines.PrimaryColor = RGB(255, 128, 0);
    Subgraph_ExtensionLines.DrawZeros = false;

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

    Input_StartTime.Name = "Start Time";
    Input_StartTime.SetTime(HMS_TIME(12, 00, 0));

    Input_UseStopTime.Name = "Use Stop Time";
    Input_UseStopTime.SetYesNo(false);

    Input_StopTime.Name = "Stop Time";
    Input_StopTime.SetTime(HMS_TIME(23, 59, 59));

    Input_LimitHorizontalLineFromTimeToOneDay.Name = "Limit Horizontal Line From Time To 1 Day";
    Input_LimitHorizontalLineFromTimeToOneDay.SetYesNo(true);

    Input_IgnoreWeekends.Name = "Ignore Weekends";
    Input_IgnoreWeekends.SetYesNo(false);

    Input_AddExtensionLinesUntilFutureIntersection.Name = "Add Extension Lines Until Future Intersection";
    Input_AddExtensionLinesUntilFutureIntersection.SetYesNo(false);

    Input_UseAllowedRangeForMatchInMinutes.Name = "Use Allowed Range for Match in Minutes";
    Input_UseAllowedRangeForMatchInMinutes.SetYesNo(false);

    Input_AllowedRangeForMatchInMinutes.Name = "Allowed Range for Match in Minutes";
    Input_AllowedRangeForMatchInMinutes.SetInt(60);

    return;
}
```

```
float& r_PriorValue = sc.GetPersistentFloat(1);
SCDateTime& r_StopDateTime = sc.GetPersistentSCDateTime(2);
int & r_LastUsedBarIndexForExtensionLine = sc.GetPersistentInt(1);
```

```
if (sc.Index == 0)
{
    r_StopDateTime = 0;
    r_PriorValue = FLT_MIN;
    r_LastUsedBarIndexForExtensionLine = -1;
}
```

```
SCDateTime AllowedTimeRange = SCDateTime::MINUTES(Input_AllowedRangeForMatchInMinutes.GetInt());
```

```
SCDateTimeMS StartDateTime = sc.BaseDateTimeIn[sc.Index].GetDateAsSCDateTime() +
Input_StartTime.GetDateTime().GetTimeAsSCDateTimeMS();
```

```

SCDateTime TradingDayDate = sc.GetTradingDayDate(sc.BaseDateTimeln[sc.Index]);
bool IgnoreDate = Input_IgnoreWeekends.GetYesNo() && TradingDayDate.IsWeekend() ;

bool IsTimeRangeMatch = !Input_UseAllowedRangeForMatchInMinutes.GetYesNo()
    || r_StopDateTime.IsGivenTimeWithinGivenTimeRange(sc.BaseDateTimeln[sc.Index], AllowedTimeRange);

if (r_StopDateTime != 0
    && sc.BaseDateTimeln[sc.Index] > r_StopDateTime
    && !IgnoreDate
    && IsTimeRangeMatch)
    r_PriorValue = FLT_MIN;

if (r_LastUsedBarIndexForExtensionLine != -1
    && r_LastUsedBarIndexForExtensionLine == sc.Index)
{
    sc.DeleteLineUntilFutureIntersection(r_LastUsedBarIndexForExtensionLine, 0);
}

IsTimeRangeMatch = !Input_UseAllowedRangeForMatchInMinutes.GetYesNo()
    || StartDateTime.IsGivenTimeWithinGivenTimeRange(sc.BaseDateTimeln[sc.Index], AllowedTimeRange);

//Start time encountered
if (!IgnoreDate
    && ((sc.Index == 0 && sc.BaseDateTimeln[sc.Index] == StartDateTime)
    || (sc.BaseDateTimeln[sc.Index - 1] < StartDateTime
        && sc.BaseDateTimeln[sc.Index] >= StartDateTime) )
    && IsTimeRangeMatch
)
{
    Subgraph_HorizontalLine[sc.Index] = sc.BaseData[Input_Data.GetInputDataIndex()][sc.Index];
    r_PriorValue = Subgraph_HorizontalLine[sc.Index];

    if(Input_UseStopTime.GetYesNo())
    {
        if(Input_StartTime.GetTime() <= Input_StopTime.GetTime())
        {
            r_StopDateTime.SetDateTime(sc.BaseDateTimeln[sc.Index].GetDate(), Input_StopTime.GetTime());
        }
        else
        {
            r_StopDateTime.SetDateTime(sc.BaseDateTimeln[sc.Index].GetDate(), Input_StopTime.GetTime());
            r_StopDateTime.AddDays(1);
        }
    }
    else if (Input_LimitHorizontalLineFromTimeToOneDay.GetYesNo())
    {
        SCDateTime StartDateTimeWithoutMilliseconds = StartDateTime;
        StartDateTimeWithoutMilliseconds.RoundToNearestSecond();
        r_StopDateTime = StartDateTimeWithoutMilliseconds + SCDateTime(0, HMS_TIME(23, 59, 59));
    }

    if (Input_AddExtensionLinesUntilFutureIntersection.GetYesNo())
    {
        sc.AddLineUntilFutureIntersection
        (sc.Index
        , 0 // LineIDForBar
        , r_PriorValue
        , Subgraph_ExtensionLines.PrimaryColor
        , Subgraph_ExtensionLines.LineWidth
        , Subgraph_ExtensionLines.LineStyle
        , false

```

```

        , false
        , ""
    );

    r_LastUsedBarIndexForExtensionLine = sc.Index;
}

}

if (r_PriorValue != FLT_MIN)
{
    Subgraph_HorizontalLine[sc.Index] = r_PriorValue;
}

}

/*=====*/
SCSFExport scsf_BarsWithZeros(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_Volume = sc.Subgraph[4];
    SCSubgraphRef Subgraph_OpenInterest = sc.Subgraph[5];
    SCSubgraphRef Subgraph_OHLCAvg = sc.Subgraph[6];
    SCSubgraphRef Subgraph_HLCAvg = sc.Subgraph[7];
    SCSubgraphRef Subgraph_HLAvG = sc.Subgraph[8];
    SCSubgraphRef Subgraph_BidVol = sc.Subgraph[9];
    SCSubgraphRef Subgraph_AskVol = sc.Subgraph[10];

    if (sc.SetDefaults)
    {
        sc.GraphName          = "Chart Bars with Zero Values";
        sc.ValueFormat         = VALUEFORMAT_INHERITED;
        sc.GraphRegion = 0;
        sc.DisplayAsMainPriceGraph = true;
        sc.GraphUsesChartColors = true;
        sc.GraphDrawType      = GDT_OHLCBAR;
        sc.StandardChartHeader = 1;

        sc.AutoLoop = true;

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

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

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

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

        Subgraph_Volume.Name = "Volume";

```

```

Subgraph_Volume.DrawStyle = DRAWSTYLE_IGNORE;
Subgraph_Volume.PrimaryColor = RGB(255,0,0);
Subgraph_Volume.DrawZeros = true;

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

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

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

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

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

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

return;
}

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

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

    if (sc.SetDefaults)
    {
        sc.GraphName = "Numbers Bars Avg Volume/Price Graph";
        sc.GraphRegion = 1;
        sc.ValueFormat = VALUEFORMAT_WHOLE_NUMBER;
        sc.AutoLoop = true;

        sc.MaintainVolumeAtPriceData = 1;

        Subgraph_AvgVolumePerPrice.Name = "Avg Volume/Price";
        Subgraph_AvgVolumePerPrice.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_AvgVolumePerPrice.PrimaryColor = RGB(0,255,255);
        Subgraph_AvgVolumePerPrice.DrawZeros = true;

        return;
    }
}

```



```

int NumPrices = sc.VolumeAtPriceForBars->GetSizeAtBarIndex(sc.Index);

// Avg Vol/Price (Average Volume At Price)
if (NumPrices== 0)
    Subgraph_AvgVolumePerPrice[sc.Index] = 0.0f;
else
    Subgraph_AvgVolumePerPrice[sc.Index] = sc.Volume[sc.Index] / NumPrices;
}

/* ===== */
SCSFExport scsf_InitialBalance(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_IBHExt6 = sc.Subgraph[0];
    SCSubgraphRef Subgraph_IBHExt5 = sc.Subgraph[1];
    SCSubgraphRef Subgraph_IBHExt4 = sc.Subgraph[2];
    SCSubgraphRef Subgraph_IBHExt3 = sc.Subgraph[3];
    SCSubgraphRef Subgraph_IBHExt2 = sc.Subgraph[4];
    SCSubgraphRef Subgraph_IBHExt1 = sc.Subgraph[5];
    SCSubgraphRef Subgraph_IBHigh = sc.Subgraph[6];
    SCSubgraphRef Subgraph_IBMid = sc.Subgraph[7];
    SCSubgraphRef Subgraph_IBLow = sc.Subgraph[8];
    SCSubgraphRef Subgraph_IBLExt1 = sc.Subgraph[9];
    SCSubgraphRef Subgraph_IBLExt2 = sc.Subgraph[10];
    SCSubgraphRef Subgraph_IBLExt3 = sc.Subgraph[11];
    SCSubgraphRef Subgraph_IBLExt4 = sc.Subgraph[12];
    SCSubgraphRef Subgraph_IBLExt5 = sc.Subgraph[13];
    SCSubgraphRef Subgraph_IBLExt6 = sc.Subgraph[14];

    SCInputRef Input_IBType = sc.Input[0];
    SCInputRef Input_StartTime = sc.Input[1];
    SCInputRef Input_EndTime = sc.Input[2];
    SCInputRef Input_NumDays = sc.Input[3];
    SCInputRef Input_RoundExt = sc.Input[4];
    SCInputRef Input_NumberDaysToCalculate = sc.Input[5];
    SCInputRef Input_NumberOfMinutes = sc.Input[6];
    SCInputRef Input_StartEndTimeMethod = sc.Input[7];
    SCInputRef Input_PeriodEndAsMinutesFromSessionStart = sc.Input[8];

    SCInputRef Input_Multiplier1 = sc.Input[10];
    SCInputRef Input_Multiplier2 = sc.Input[11];
    SCInputRef Input_Multiplier3 = sc.Input[12];
    SCInputRef Input_Multiplier4 = sc.Input[13];
    SCInputRef Input_Multiplier5 = sc.Input[14];
    SCInputRef Input_Multiplier6 = sc.Input[15];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Initial Balance";
        sc.DrawZeros = 0;
        sc.GraphRegion = 0;
        sc.AutoLoop = 1;

        sc.ScaleRangeType = SCALE_SAMEASREGION;

        Subgraph_IBHExt6.Name = "IB High Ext 6";
        Subgraph_IBHExt6.PrimaryColor = RGB(0, 255, 0);
        Subgraph_IBHExt6.DrawStyle = DRAWSTYLE_IGNORE;
        Subgraph_IBHExt6.DrawZeros = false;

        Subgraph_IBHExt5.Name = "IB High Ext 5";
        Subgraph_IBHExt5.PrimaryColor = RGB(0, 255, 0);
        Subgraph_IBHExt5.DrawStyle = DRAWSTYLE_IGNORE;
    }
}

```

Subgraph\_IBHExt5.DrawZeros = false;

Subgraph\_IBHExt4.Name = "IB High Ext 4";  
Subgraph\_IBHExt4.PrimaryColor = RGB(0, 255, 0);  
Subgraph\_IBHExt4.DrawStyle = DRAWSTYLE\_IGNORE;  
Subgraph\_IBHExt4.DrawZeros = false;

Subgraph\_IBHExt3.Name = "IB High Ext 3";  
Subgraph\_IBHExt3.PrimaryColor = RGB(0, 255, 0);  
Subgraph\_IBHExt3.DrawStyle = DRAWSTYLE\_IGNORE;  
Subgraph\_IBHExt3.DrawZeros = false;

Subgraph\_IBHExt2.Name = "IB High Ext 2";  
Subgraph\_IBHExt2.PrimaryColor = RGB(0, 255, 0);  
Subgraph\_IBHExt2.DrawStyle = DRAWSTYLE\_IGNORE;  
Subgraph\_IBHExt2.DrawZeros = false;

Subgraph\_IBHExt1.Name = "IB High Ext 1";  
Subgraph\_IBHExt1.PrimaryColor = RGB(0, 255, 0);  
Subgraph\_IBHExt1.DrawStyle = DRAWSTYLE\_IGNORE;  
Subgraph\_IBHExt1.DrawZeros = false;

Subgraph\_IBHigh.Name = "IB High";  
Subgraph\_IBHigh.PrimaryColor = RGB(128, 255, 128);  
Subgraph\_IBHigh.DrawStyle = DRAWSTYLE\_DASH;  
Subgraph\_IBHigh.DrawZeros = false;

Subgraph\_IBMid.Name = "IB Mid";  
Subgraph\_IBMid.PrimaryColor = RGB(255, 255, 255);  
Subgraph\_IBMid.DrawStyle = DRAWSTYLE\_DASH;  
Subgraph\_IBMid.DrawZeros = false;

Subgraph\_IBLow.Name = "IB Low";  
Subgraph\_IBLow.PrimaryColor = RGB(255, 128, 128);  
Subgraph\_IBLow.DrawStyle = DRAWSTYLE\_DASH;  
Subgraph\_IBLow.DrawZeros = false;

Subgraph\_IBLExt1.Name = "IB Low Ext 1";  
Subgraph\_IBLExt1.PrimaryColor = RGB(255, 0, 0);  
Subgraph\_IBLExt1.DrawStyle = DRAWSTYLE\_IGNORE;  
Subgraph\_IBLExt1.DrawZeros = false;

Subgraph\_IBLExt2.Name = "IB Low Ext 2";  
Subgraph\_IBLExt2.PrimaryColor = RGB(255, 0, 0);  
Subgraph\_IBLExt2.DrawStyle = DRAWSTYLE\_IGNORE;  
Subgraph\_IBLExt2.DrawZeros = false;

Subgraph\_IBLExt3.Name = "IB Low Ext 3";  
Subgraph\_IBLExt3.PrimaryColor = RGB(255, 0, 0);  
Subgraph\_IBLExt3.DrawStyle = DRAWSTYLE\_IGNORE;  
Subgraph\_IBLExt3.DrawZeros = false;

Subgraph\_IBLExt4.Name = "IB Low Ext 4";  
Subgraph\_IBLExt4.PrimaryColor = RGB(255, 0, 0);  
Subgraph\_IBLExt4.DrawStyle = DRAWSTYLE\_IGNORE;  
Subgraph\_IBLExt4.DrawZeros = false;

Subgraph\_IBLExt5.Name = "IB Low Ext 5";  
Subgraph\_IBLExt5.PrimaryColor = RGB(255, 0, 0);  
Subgraph\_IBLExt5.DrawStyle = DRAWSTYLE\_IGNORE;  
Subgraph\_IBLExt5.DrawZeros = false;

Subgraph\_IBLExt6.Name = "IB Low Ext 6";  
Subgraph\_IBLExt6.PrimaryColor = RGB(255, 0, 0);  
Subgraph\_IBLExt6.DrawStyle = DRAWSTYLE\_IGNORE;

```

Subgraph_IBLExt6.DrawZeros = false;

// Inputs
Input_IBType.Name = "Initial Balance Type";
Input_IBType.SetCustomInputStrings("Daily;Weekly;Weekly Include Sunday;Intraday");
Input_IBType.SetCustomInputIndex(0);

Input_StartTime.Name = "Start Time";
Input_StartTime.SetTime(HMS_TIME(9, 30, 0));

Input_EndTime.Name = "End Time";
Input_EndTime.SetTime(HMS_TIME(10, 29, 59));

Input_NumDays.Name = "Weekly: Number of Days";
Input_NumDays.SetInt(2);
Input_NumDays.SetIntLimits(1, 5);

Input_NumberOfMinutes.Name = "Intraday: Number of Minutes";
Input_NumberOfMinutes.SetInt(15);

Input_RoundExt.Name = "Round Extensions to TickSize";
Input_RoundExt.SetYesNo(1);

Input_NumberDaysToCalculate.Name = "Number of Days to Calculate";
Input_NumberDaysToCalculate.SetInt(100000);
Input_NumberDaysToCalculate.SetIntLimits(1, INT_MAX);

Input_StartEndTimeMethod.Name = "Start End Time Method";
Input_StartEndTimeMethod.SetCustomInputStrings("Use Start/End Time;Use Session Start Time and Minutes From
Start");
Input_StartEndTimeMethod.SetCustomInputIndex(0);

Input_PeriodEndAsMinutesFromSessionStart.Name = "Period End As Minutes from Session Start";
Input_PeriodEndAsMinutesFromSessionStart.SetInt(30);

Input_Multiplier1.Name = "Extension Multiplier 1";
Input_Multiplier1.SetFloat(.5f);
Input_Multiplier2.Name = "Extension Multiplier 2";
Input_Multiplier2.SetFloat(1.0f);
Input_Multiplier3.Name = "Extension Multiplier 3";
Input_Multiplier3.SetFloat(1.5f);
Input_Multiplier4.Name = "Extension Multiplier 4";
Input_Multiplier4.SetFloat(2.0f);
Input_Multiplier5.Name = "Extension Multiplier 5";
Input_Multiplier5.SetFloat(2.5f);
Input_Multiplier6.Name = "Extension Multiplier 6";
Input_Multiplier6.SetFloat(3.0f);

return;
}

// Persist vars
int& PeriodFirstIndex = sc.GetPersistentInt(1);

SCDateTime& PeriodStartDateTime = sc.GetPersistentSCDateTime(1);
SCDateTime& PeriodEndDateTime = sc.GetPersistentSCDateTime(2);

float& PeriodHigh = sc.GetPersistentFloat(1);
float& PeriodLow = sc.GetPersistentFloat(2);
float& PeriodMid = sc.GetPersistentFloat(3);
float& PeriodHighExt1 = sc.GetPersistentFloat(4);
float& PeriodHighExt2 = sc.GetPersistentFloat(5);
float& PeriodHighExt3 = sc.GetPersistentFloat(6);
float& PeriodHighExt4 = sc.GetPersistentFloat(7);

```

```

float& PeriodHighExt5 = sc.GetPersistentFloat(8);
float& PeriodHighExt6 = sc.GetPersistentFloat(9);
float& PeriodLowExt1 = sc.GetPersistentFloat(10);
float& PeriodLowExt2 = sc.GetPersistentFloat(11);
float& PeriodLowExt3 = sc.GetPersistentFloat(12);
float& PeriodLowExt4 = sc.GetPersistentFloat(13);
float& PeriodLowExt5 = sc.GetPersistentFloat(14);
float& PeriodLowExt6 = sc.GetPersistentFloat(15);

// Reset persistent variables upon full calculation
if (sc.Index == 0)
{
    PeriodFirstIndex = -1;
    PeriodStartDateTime = 0;
    PeriodEndDateTime = 0;
    PeriodHigh = -FLT_MAX;
    PeriodLow = FLT_MAX;
}

SCDateTimeMS LastBarDateTime = sc.BaseDateTimeIn[sc.ArraySize-1];
SCDateTimeMS FirstCalculationDate = LastBarDateTime.GetDate() -
SCDateTime::DAYS(Input_NumberDaysToCalculate.GetInt() - 1);

SCDateTimeMS CurrentBarDateTime = sc.BaseDateTimeIn[sc.Index];

SCDateTimeMS PrevBarDateTime;

if (sc.Index > 0)
    PrevBarDateTime = sc.BaseDateTimeIn[sc.Index-1];

if (CurrentBarDateTime.GetDate() < FirstCalculationDate) // Limit calculation to specified number of days back
    return;

bool Daily = Input_IBType.GetIndex() == 0;
bool Weekly = Input_IBType.GetIndex() == 1 || Input_IBType.GetIndex() == 2;
bool Intraday = Input_IBType.GetIndex() == 3;
bool IncludeSunday = Input_IBType.GetIndex() == 2;

SCDateTimeMS StartDateTime = CurrentBarDateTime;

if (Input_StartEndTimeMethod.GetIndex() == 0)
    StartDateTime.SetTime(Input_StartTime.GetTime());
else
    StartDateTime.SetTime(sc.StartTimeOfDay);

if (Weekly)
{
    int PeriodStartDayOfWeek = IncludeSunday ? SUNDAY : MONDAY;

    int DayOfWeek = StartDateTime.GetDayOfWeek();

    if (DayOfWeek != PeriodStartDayOfWeek)
        StartDateTime.AddDays(7 - DayOfWeek + PeriodStartDayOfWeek);
}

if (PrevBarDateTime < StartDateTime && CurrentBarDateTime >= StartDateTime)
{
    PeriodFirstIndex = sc.Index;
    PeriodHigh = -FLT_MAX;
    PeriodLow = FLT_MAX;

    PeriodStartDateTime = StartDateTime;

```

```

PeriodEndTime = PeriodStartTime;

if (Input_StartEndTimeMethod.GetIndex() == 0)
    PeriodEndTime.SetTime(Input_EndTime.GetTime());
else
{
    PeriodEndTime.SetTime(static_cast<int>(sc.StartTimeOfDay +
Input_PeriodEndAsMinutesFromSessionStart.GetInt() * SECONDS_PER_MINUTE - 1));
}

if (Daily || Intraday)
{
    if (SCDateTimeMS(PeriodEndTime) <= PeriodStartTime)
        PeriodEndTime.AddDays(1);
}
else if (Weekly)
{
    int PeriodEndDayOfWeek = IncludeSunday ? Input_NumDays.GetInt() - 1 : Input_NumDays.GetInt();

    int DayOfWeek = PeriodEndTime.GetDayOfWeek();

    if (DayOfWeek != PeriodEndDayOfWeek)
        PeriodEndTime.AddDays(PeriodEndDayOfWeek - DayOfWeek);
}
}

// Check end of period
if (PeriodFirstIndex >= 0)
{
    // Check start of new intraday period
    if (Intraday)
    {
        SCDateTimeMS IntradayEndTime = PeriodStartTime +
SCDateTime::MINUTES(Input_NumberOfMinutes.GetInt()) - SCDateTime::MICROSECONDS(1);

        if (PrevBarDateTime < IntradayEndTime && CurrentBarDateTime > IntradayEndTime)
        {
            PeriodFirstIndex = sc.Index;
            PeriodStartTime.AddMinutes(Input_NumberOfMinutes.GetInt());
            PeriodHigh = -FLT_MAX;
            PeriodLow = FLT_MAX;
        }
    }

    if (CurrentBarDateTime > PeriodEndTime)
    {
        PeriodFirstIndex = -1;

        if (Intraday)
        {
            PeriodHigh = -FLT_MAX;
            PeriodLow = FLT_MAX;
        }
    }
}

// Collecting data, back propagate if changed
if (PeriodFirstIndex >= 0)
{
    bool Changed = false;

    if (sc.High[sc.Index] > PeriodHigh)
    {
        PeriodHigh = sc.High[sc.Index];
        Changed = true;
    }
}

```

```

}

if (sc.Low[sc.Index] < PeriodLow)
{
    PeriodLow = sc.Low[sc.Index];
    Changed = true;
}

if (Changed)
{
    PeriodMid = (PeriodHigh + PeriodLow) / 2.0f;

    float Range = PeriodHigh - PeriodLow;

    PeriodHighExt1 = PeriodHigh + Input_Multiplier1.GetFloat() * Range;
    PeriodHighExt2 = PeriodHigh + Input_Multiplier2.GetFloat() * Range;
    PeriodHighExt3 = PeriodHigh + Input_Multiplier3.GetFloat() * Range;
    PeriodHighExt4 = PeriodHigh + Input_Multiplier4.GetFloat() * Range;
    PeriodHighExt5 = PeriodHigh + Input_Multiplier5.GetFloat() * Range;
    PeriodHighExt6 = PeriodHigh + Input_Multiplier6.GetFloat() * Range;

    PeriodLowExt1 = PeriodLow - Input_Multiplier1.GetFloat() * Range;
    PeriodLowExt2 = PeriodLow - Input_Multiplier2.GetFloat() * Range;
    PeriodLowExt3 = PeriodLow - Input_Multiplier3.GetFloat() * Range;
    PeriodLowExt4 = PeriodLow - Input_Multiplier4.GetFloat() * Range;
    PeriodLowExt5 = PeriodLow - Input_Multiplier5.GetFloat() * Range;
    PeriodLowExt6 = PeriodLow - Input_Multiplier6.GetFloat() * Range;

    if (Input_RoundExt.GetYesNo())
    {
        PeriodHighExt1 = sc.RoundToTickSize(PeriodHighExt1, sc.TickSize);
        PeriodHighExt2 = sc.RoundToTickSize(PeriodHighExt2, sc.TickSize);
        PeriodHighExt3 = sc.RoundToTickSize(PeriodHighExt3, sc.TickSize);
        PeriodHighExt4 = sc.RoundToTickSize(PeriodHighExt4, sc.TickSize);
        PeriodHighExt5 = sc.RoundToTickSize(PeriodHighExt5, sc.TickSize);
        PeriodHighExt6 = sc.RoundToTickSize(PeriodHighExt6, sc.TickSize);

        PeriodLowExt1 = sc.RoundToTickSize(PeriodLowExt1, sc.TickSize);
        PeriodLowExt2 = sc.RoundToTickSize(PeriodLowExt2, sc.TickSize);
        PeriodLowExt3 = sc.RoundToTickSize(PeriodLowExt3, sc.TickSize);
        PeriodLowExt4 = sc.RoundToTickSize(PeriodLowExt4, sc.TickSize);
        PeriodLowExt5 = sc.RoundToTickSize(PeriodLowExt5, sc.TickSize);
        PeriodLowExt6 = sc.RoundToTickSize(PeriodLowExt6, sc.TickSize);
    }

    for (int Index = PeriodFirstIndex; Index < sc.Index; Index++)
    {
        Subgraph_IBHigh[Index] = PeriodHigh;
        Subgraph_IBLow[Index] = PeriodLow;
        Subgraph_IBMid[Index] = PeriodMid;
        Subgraph_IBHExt1[Index] = PeriodHighExt1;
        Subgraph_IBHExt2[Index] = PeriodHighExt2;
        Subgraph_IBHExt3[Index] = PeriodHighExt3;
        Subgraph_IBHExt4[Index] = PeriodHighExt4;
        Subgraph_IBHExt5[Index] = PeriodHighExt5;
        Subgraph_IBHExt6[Index] = PeriodHighExt6;
        Subgraph_IBLExt1[Index] = PeriodLowExt1;
        Subgraph_IBLExt2[Index] = PeriodLowExt2;
        Subgraph_IBLExt3[Index] = PeriodLowExt3;
        Subgraph_IBLExt4[Index] = PeriodLowExt4;
        Subgraph_IBLExt5[Index] = PeriodLowExt5;
        Subgraph_IBLExt6[Index] = PeriodLowExt6;
    }

    sc.EarliestUpdateSubgraphDataArrayIndex = PeriodFirstIndex;

```

```

    }
}

// Plot current values
if (PeriodLow != FLT_MAX)
{
    Subgraph_IBHigh[sc.Index] = PeriodHigh;
    Subgraph_IBLow[sc.Index] = PeriodLow;
    Subgraph_IBMid[sc.Index] = PeriodMid;
    Subgraph_IBHExt1[sc.Index] = PeriodHighExt1;
    Subgraph_IBHExt2[sc.Index] = PeriodHighExt2;
    Subgraph_IBHExt3[sc.Index] = PeriodHighExt3;
    Subgraph_IBHExt4[sc.Index] = PeriodHighExt4;
    Subgraph_IBHExt5[sc.Index] = PeriodHighExt5;
    Subgraph_IBHExt6[sc.Index] = PeriodHighExt6;
    Subgraph_IBLExt1[sc.Index] = PeriodLowExt1;
    Subgraph_IBLExt2[sc.Index] = PeriodLowExt2;
    Subgraph_IBLExt3[sc.Index] = PeriodLowExt3;
    Subgraph_IBLExt4[sc.Index] = PeriodLowExt4;
    Subgraph_IBLExt5[sc.Index] = PeriodLowExt5;
    Subgraph_IBLExt6[sc.Index] = PeriodLowExt6;
}
}

/*=====*/
SCSFExport scsf_RenkoVisualOpenCloseValues(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_RenkoOpen = sc.Subgraph[0];
    SCSubgraphRef Subgraph_RenkoClose = sc.Subgraph[1];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Renko Visual Open/Close Values";
        sc.DrawZeros = 0;
        sc.GraphRegion = 0;
        sc.AutoLoop = 0;

        sc.ScaleRangeType = SCALE_SAMEASREGION;

        Subgraph_RenkoOpen.Name = "Renko Open";
        Subgraph_RenkoOpen.PrimaryColor = RGB(255, 255, 255);
        Subgraph_RenkoOpen.LineWidth = 3;
        Subgraph_RenkoOpen.DrawStyle = DRAWSTYLE_LEFT_PRICE_BAR_DASH;
        Subgraph_RenkoOpen.DrawZeros = false;

        Subgraph_RenkoClose.Name = "Renko Close";
        Subgraph_RenkoClose.PrimaryColor = RGB(255, 255, 255);
        Subgraph_RenkoClose.LineWidth = 3;
        Subgraph_RenkoClose.DrawStyle = DRAWSTYLE_RIGHT_PRICE_BAR_DASH;
        Subgraph_RenkoClose.DrawZeros = false;

        return;
    }

    int RenkoTicksPerBar = 0;

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

    RenkoTicksPerBar = BarPeriod.IntradayChartBarPeriodParameter1;

    if (RenkoTicksPerBar == 0)
        return;

    for(int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize; BarIndex++)

```

```

{
    Subgraph_RenkoOpen[BarIndex] = sc.BaseData[SC_RENKO_OPEN][BarIndex];
    Subgraph_RenkoClose[BarIndex] = sc.BaseData[SC_RENKO_CLOSE][BarIndex];
}
}

/*=====*/
SCSFExport scsf_StochasticPercentile(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Stochastic = sc.Subgraph[0];
    SCSubgraphRef Subgraph_StochasticAverage = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Line1 = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Line2 = sc.Subgraph[3];

    SCInputRef Input_Data = sc.Input[0];
    SCInputRef Input_StochasticLength = sc.Input[1];
    SCInputRef Input_MovingAverageLength = sc.Input[2];
    SCInputRef Input_Line1Value = sc.Input[3];
    SCInputRef Input_Line2Value = sc.Input[4];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Stochastic - Percentile";

        sc.ValueFormat = 2;

        Subgraph_Stochastic.Name = "Stochastic%";
        Subgraph_Stochastic.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Stochastic.PrimaryColor = RGB(0,255,0);
        Subgraph_Stochastic.DrawZeros = true;

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

        Subgraph_Line1.Name = "Line1";
        Subgraph_Line1.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Line1.PrimaryColor = RGB(255, 255, 0);
        Subgraph_Line1.DrawZeros = true;

        Subgraph_Line2.Name = "Line2";
        Subgraph_Line2.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Line2.PrimaryColor = RGB(0, 0, 255);
        Subgraph_Line2.DrawZeros = true;

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

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

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

        Input_Line1Value.Name = "Line 1 Value";
        Input_Line1Value.SetFloat(20);

        Input_Line2Value.Name = "Line 2 Value";
        Input_Line2Value.SetFloat(80);
    }
}

```



```

    sc.AutoLoop = false;
    return;
}

std::vector<float> AscendingValues;

SCFloatArrayRef DataArray = sc.BaseData[Input_Data.GetInputDataIndex()];
sc.DataStartIndex = Input_StochasticLength.GetInt() - 1;

for (int CurrentIndex = sc.UpdateStartIndex; CurrentIndex < sc.ArraySize; CurrentIndex++)
{
    AscendingValues.clear();
    int FirstIndex = CurrentIndex - Input_StochasticLength.GetInt() + 1;
    for (int BarIndex = FirstIndex; BarIndex <= CurrentIndex; BarIndex++)
    {
        AscendingValues.push_back(DataArray[BarIndex]);
    }

    std::sort(AscendingValues.begin(), AscendingValues.end());
    float CurrentPrice = DataArray[CurrentIndex];

    int AscendingValuesSize = static_cast<int>(AscendingValues.size());
    int AscendingValuesIndex = 0;

    for(; AscendingValuesIndex < AscendingValuesSize; AscendingValuesIndex++)
    {
        if (CurrentPrice == AscendingValues[AscendingValuesIndex])
            break;

        if (AscendingValuesIndex == AscendingValuesSize - 1)
            break;
    }

    Subgraph_Stochastic[CurrentIndex] = 100.0f*AscendingValuesIndex/(Input_StochasticLength.GetInt() - 1);

    sc.SimpleMovAvg(Subgraph_Stochastic, Subgraph_StochasticAverage, CurrentIndex,
Input_MovingAverageLength.GetInt());

    Subgraph_Line1[CurrentIndex] = Input_Line1Value.GetFloat();
    Subgraph_Line2[CurrentIndex] = Input_Line2Value.GetFloat();
}

return;

}

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

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

    SCInputRef Input_UseCumVol = sc.Input[0];
    SCInputRef Input_Length = sc.Input[1];
    SCInputRef Input_InputDataClose = sc.Input[2];
    SCInputRef Input_InputDataHigh = sc.Input[3];
    SCInputRef Input_InputDataLow = sc.Input[4];

    if (sc.SetDefaults)
    {

```

```

sc.GraphName = "Moving Average - Elastic Volume Weighted";

sc.GraphRegion = 0;

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

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

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

Input_UseCumVol.Name = "Use Cumulative Volume";
Input_UseCumVol.SetYesNo(0);

Input_Length.Name = "Length";
Input_Length.SetInt(20);
Input_Length.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_InputDataClose.Name = "Input Data Close";
Input_InputDataClose.SetInputDataIndex(SC_LAST);

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

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

sc.AutoLoop = 1;

return;
}

int Length = Input_Length.GetInt();
float Close = sc.BaseData[Input_InputDataClose.GetInputDataIndex()][sc.Index];
float High = sc.BaseData[Input_InputDataHigh.GetInputDataIndex()][sc.Index];
float Low = sc.BaseData[Input_InputDataLow.GetInputDataIndex()][sc.Index];

float Volume = sc.Volume[sc.Index];

if (sc.Index == 0)
{
    Arrays_NumberFloatingShares[sc.Index] = Volume;
    Subgraph_EVWMA[sc.Index] = Close;
    Subgraph_TopBand[sc.Index] = High;
    Subgraph_BottomBand[sc.Index] = Low;
}
else
{
    if (Input_UseCumVol.GetYesNo())
    {
        sc.CumulativeSummation(sc.Volume, Arrays_NumberFloatingShares, sc.Index);
    }
    else
    {
        Arrays_NumberFloatingShares[sc.Index] = sc.GetSummation(sc.Volume, sc.Index, Length);
    }
}

```

```

float NumberFloatingShares = Arrays_NumberFloatingShares[sc.Index];

if (NumberFloatingShares == 0)
{
    Subgraph_EVWMA[sc.Index] = 0.0f;
    Subgraph_TopBand[sc.Index] = 0.0f;
    Subgraph_BottomBand[sc.Index] = 0.0f;
}
else
{
    Subgraph_EVWMA[sc.Index] = (1.0f - Volume / NumberFloatingShares) * Subgraph_EVWMA[sc.Index - 1] +
Volume * Close / NumberFloatingShares;
    Subgraph_TopBand[sc.Index] = (1.0f - Volume / NumberFloatingShares) * Subgraph_TopBand[sc.Index - 1] +
Volume * High / NumberFloatingShares;
    Subgraph_BottomBand[sc.Index] = (1.0f - Volume / NumberFloatingShares) * Subgraph_BottomBand[sc.Index -
1] + Volume * Low / NumberFloatingShares;
}
}
}

/*=====*/
SCSFExport scsf_TradingTheNines(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_BuyNumber = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SellNumber = sc.Subgraph[1];

    SCInputRef Input_TextOffset = sc.Input[0];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Trading the Nines";

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

        Subgraph_BuyNumber.Name = "Buy Number";
        Subgraph_BuyNumber.DrawStyle = DRAWSTYLE_CUSTOM_VALUE_AT_Y;
        Subgraph_BuyNumber.PrimaryColor = RGB(0, 255, 0);
        Subgraph_BuyNumber.SecondaryColorUsed = true;
        Subgraph_BuyNumber.SecondaryColor = RGB(255, 255, 0);
        Subgraph_BuyNumber.LineWidth = 8;
        Subgraph_BuyNumber.DrawZeros = false;

        Subgraph_SellNumber.Name = "Sell Number";
        Subgraph_SellNumber.DrawStyle = DRAWSTYLE_CUSTOM_VALUE_AT_Y;
        Subgraph_SellNumber.PrimaryColor = RGB(0, 255, 0);
        Subgraph_SellNumber.SecondaryColorUsed = true;
        Subgraph_SellNumber.SecondaryColor = RGB(255, 255, 0);
        Subgraph_SellNumber.LineWidth = 8;
        Subgraph_SellNumber.DrawZeros = false;

        Input_TextOffset.Name = "Text Labels Tick Offset";
        Input_TextOffset.SetInt(2);
        Input_TextOffset.SetIntLimits(-10000, 10000);

        return;
    }

    int& r_IsValidBuySetup = sc.GetPersistentInt(1);
    int& r_BuyBarCounter = sc.GetPersistentInt(2);
    int& r_IsBuyPerfection = sc.GetPersistentInt(3);
    int& r_IsValidSellSetup = sc.GetPersistentInt(4);
    int& r_SellBarCounter = sc.GetPersistentInt(5);
    int& r_IsSellPerfection = sc.GetPersistentInt(6);

```

```

for (int Index = sc.UpdateStartIndex; Index < sc.ArraySize; Index++)
{
    if (Index < 4)
        continue;

    // Initialize persistent integers.
    if (Index != sc.ArraySize - 1)
    {
        if (Index == 4)
        {
            r_IsValidBuySetup = 0;
            r_BuyBarCounter = 0;
            r_IsBuyPerfection = 0;
            r_IsValidSellSetup = 0;
            r_SellBarCounter = 0;
            r_IsSellPerfection = 0;

            if (sc.Close[Index] >= sc.Close[Index - 4])
                r_IsValidBuySetup = 1;

            if (sc.Close[Index] <= sc.Close[Index - 4])
                r_IsValidSellSetup = 1;
        }

        // Check for valid buy and sell setups for Index > 4.
        if (Index > 4)
        {
            // Buy Setup
            if (r_IsValidBuySetup == 0)
            {
                if (sc.Close[Index] >= sc.Close[Index - 4])
                    r_IsValidBuySetup = 1;
            }
            else
            {
                if (sc.Close[Index] < sc.Close[Index - 4])
                {
                    r_BuyBarCounter++;

                    if (r_BuyBarCounter != 9)
                    {
                        Subgraph_BuyNumber.DataColor[Index] = Subgraph_BuyNumber.PrimaryColor;
                        Subgraph_BuyNumber[Index] = static_cast<float>(r_BuyBarCounter);
                    }
                    else
                    {
                        if ((sc.Low[Index] <= sc.Low[Index - 2] && sc.Low[Index] <= sc.Low[Index - 3]) || (sc.Low[Index - 1] <=
sc.Low[Index - 2] && sc.Low[Index - 1] <= sc.Low[Index - 3]))
                            r_IsBuyPerfection = 1;

                        if (r_IsBuyPerfection)
                        {
                            Subgraph_BuyNumber.DataColor[Index] = Subgraph_BuyNumber.SecondaryColor;
                            Subgraph_BuyNumber[Index] = static_cast<float>(r_BuyBarCounter);
                        }
                        else
                        {
                            Subgraph_BuyNumber.DataColor[Index] = Subgraph_BuyNumber.PrimaryColor;
                            Subgraph_BuyNumber[Index] = static_cast<float>(r_BuyBarCounter);
                        }
                    }

                    r_IsValidBuySetup = 0;
                    r_BuyBarCounter = 0;
                    r_IsBuyPerfection = 0;
                }
            }
        }
    }
}

```

```

    }
}
else
{
    r_IsValidBuySetup = 0;
    r_BuyBarCounter = 0;

    Subgraph_BuyNumber.DataColor[Index] = Subgraph_BuyNumber.PrimaryColor;
    Subgraph_BuyNumber[Index] = static_cast<float>(r_BuyBarCounter);

    if (sc.Close[Index] >= sc.Close[Index - 4])
        r_IsValidBuySetup = 1;
}
}

// Sell Setup
if (r_IsValidSellSetup == 0)
{
    if (sc.Close[Index] <= sc.Close[Index - 4])
        r_IsValidSellSetup = 1;
}
else
{
    if (sc.Close[Index] > sc.Close[Index - 4])
    {
        r_SellBarCounter += 1;

        if (r_SellBarCounter != 9)
        {
            Subgraph_SellNumber.DataColor[Index] = Subgraph_SellNumber.PrimaryColor;
            Subgraph_SellNumber[Index] = static_cast<float>(r_SellBarCounter);
        }
        else
        {
            if ((sc.High[Index] >= sc.High[Index - 2] && sc.High[Index] >= sc.High[Index - 3]) || (sc.High[Index - 1]
>= sc.High[Index - 2] && sc.High[Index - 1] >= sc.High[Index - 3]))
                r_IsSellPerfection = 1;

            if (r_IsSellPerfection)
            {
                Subgraph_SellNumber.DataColor[Index] = Subgraph_SellNumber.SecondaryColor;
                Subgraph_SellNumber[Index] = static_cast<float>(r_SellBarCounter);
            }
            else
            {
                Subgraph_SellNumber.DataColor[Index] = Subgraph_SellNumber.PrimaryColor;
                Subgraph_SellNumber[Index] = static_cast<float>(r_SellBarCounter);
            }
        }

        r_IsValidSellSetup = 0;
        r_SellBarCounter = 0;
        r_IsSellPerfection = 0;
    }
}
else
{
    r_IsValidSellSetup = 0;
    r_SellBarCounter = 0;

    Subgraph_SellNumber.DataColor[Index] = Subgraph_SellNumber.PrimaryColor;
    Subgraph_SellNumber[Index] = static_cast<float>(r_SellBarCounter);

    if (sc.Close[Index] <= sc.Close[Index - 4])
        r_IsValidSellSetup = 1;
}
}

```

```

    }
}
sc.ValueFormat = 0;

float Offset = sc.TickSize * Input_TextOffset.GetInt();

float BuyVerticalPositionValue = sc.Low[Index] - Offset;
float SellVerticalPositionValue = sc.High[Index] + Offset;

Subgraph_BuyNumber.Arrays[0][Index] = BuyVerticalPositionValue;
Subgraph_SellNumber.Arrays[0][Index] = SellVerticalPositionValue;

}
else
{
    Subgraph_BuyNumber[Index] = 0.0f;
    Subgraph_SellNumber[Index] = 0.0f;
}
}
}

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

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_Length = sc.Input[1];
    SCInputRef Input_NumberOfBarsToCalculate = sc.Input[2];

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

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

        Subgraph_MOMA.Name = "Move-Adjusted Moving Average";
        Subgraph_MOMA.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_MOMA.PrimaryColor = RGB(0, 255, 0);

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

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

        Input_NumberOfBarsToCalculate.Name = "Number of Bars To Calculate";
        Input_NumberOfBarsToCalculate.SetIntLimits(1, MAX_STUDY_LENGTH);
        Input_NumberOfBarsToCalculate.SetInt(1000);

        return;
    }

    if (Input_NumberOfBarsToCalculate.GetInt() < Input_Length.GetInt())
        Input_NumberOfBarsToCalculate.SetInt(Input_Length.GetInt() + 1);

    sc.DataStartIndex = sc.ArraySize - Input_NumberOfBarsToCalculate.GetInt();
    if (sc.DataStartIndex < 0)
        sc.DataStartIndex = 0;

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

```

```

float MOMA_Numerator = 0;
float MOMA_Denominator = 0;

for (int MOMAIndex = sc.Index - Input_Length.GetInt() + 1; MOMAIndex <= sc.Index; MOMAIndex++)
{
    MOMA_Numerator += sc.BaseDataIn[Input_InputData.GetInputDataIndex()][MOMAIndex] *
fabs(sc.BaseDataIn[Input_InputData.GetInputDataIndex()][MOMAIndex] -
sc.BaseDataIn[Input_InputData.GetInputDataIndex()][MOMAIndex - 1]);
    MOMA_Denominator += fabs(sc.BaseDataIn[Input_InputData.GetInputDataIndex()][MOMAIndex] -
sc.BaseDataIn[Input_InputData.GetInputDataIndex()][MOMAIndex - 1]);
}

if (MOMA_Denominator == 0)
    Subgraph_MOMA[sc.Index] = sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index];
else
    Subgraph_MOMA[sc.Index] = MOMA_Numerator / MOMA_Denominator;
}

/*=====*/
SCSFExport scsf_VolumeMoveAdjustedMovingAverage(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_MOMA = sc.Subgraph[0];
    SCSubgraphRef Subgraph_VOMA = sc.Subgraph[1];
    SCSubgraphRef Subgraph_VOMOMA = sc.Subgraph[2];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_Length = sc.Input[1];
    SCInputRef Input_NumberOfBarsToCalculate = sc.Input[2];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Moving Average - Volume Move-Adjusted";

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

        Subgraph_MOMA.Name = "Move-Adjusted Moving Average";
        Subgraph_MOMA.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_VOMA.Name = "Volume Moving Average";
        Subgraph_VOMA.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_VOMOMA.Name = "Volume Move-Adjusted Moving Average";
        Subgraph_VOMOMA.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_VOMOMA.PrimaryColor = RGB(0, 255, 0);

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

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

        Input_NumberOfBarsToCalculate.Name = "Number of Bars To Calculate";
        Input_NumberOfBarsToCalculate.SetIntLimits(1, MAX_STUDY_LENGTH);
        Input_NumberOfBarsToCalculate.SetInt(1000);

        return;
    }

    if (Input_NumberOfBarsToCalculate.GetInt() < Input_Length.GetInt())
        Input_NumberOfBarsToCalculate.SetInt(Input_Length.GetInt() + 1);

    sc.DataStartIndex = sc.ArraySize - Input_NumberOfBarsToCalculate.GetInt();
    if (sc.DataStartIndex < 0)

```

```

    sc.DataStartIndex = 0;

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

    float MOMA_Numerator = 0;
    float MOMA_Denominator = 0;
    float VOMA_Numerator = 0;
    float VOMA_Denominator = 0;

    for (int VOMOMAIIndex = sc.Index - Input_Length.GetInt() + 1; VOMOMAIIndex <= sc.Index; VOMOMAIIndex++)
    {
        MOMA_Numerator += sc.BaseDataIn[Input_InputData.GetInputDataIndex()][VOMOMAIIndex] *
fabs(sc.BaseDataIn[Input_InputData.GetInputDataIndex()][VOMOMAIIndex] -
sc.BaseDataIn[Input_InputData.GetInputDataIndex()][VOMOMAIIndex - 1]);
        MOMA_Denominator += fabs(sc.BaseDataIn[Input_InputData.GetInputDataIndex()][VOMOMAIIndex] -
sc.BaseDataIn[Input_InputData.GetInputDataIndex()][VOMOMAIIndex - 1]);
        VOMA_Numerator += sc.BaseDataIn[Input_InputData.GetInputDataIndex()][VOMOMAIIndex] *
sc.Volume[VOMOMAIIndex];
        VOMA_Denominator += sc.Volume[VOMOMAIIndex];
    }

    if (MOMA_Denominator == 0)
        Subgraph_MOMA[sc.Index] = sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index];
    else
        Subgraph_MOMA[sc.Index] = MOMA_Numerator / MOMA_Denominator;

    if (VOMA_Denominator == 0)
        Subgraph_VOMA[sc.Index] = sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index];
    else
        Subgraph_VOMA[sc.Index] = VOMA_Numerator / VOMA_Denominator;

    Subgraph_VOMOMA[sc.Index] = (Subgraph_MOMA[sc.Index] + Subgraph_VOMA[sc.Index]) / 2.0f;
}

/*=====*/
SCSFExport scsf_WeightVolumeMoveAdjustedMovingAverage(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_MOMA = sc.Subgraph[0];
    SCSubgraphRef Subgraph_VOMA = sc.Subgraph[1];
    SCSubgraphRef Subgraph_WMA = sc.Subgraph[2];
    SCSubgraphRef Subgraph_WEVOMO = sc.Subgraph[3];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_Length = sc.Input[1];
    SCInputRef Input_NumberOfBarsToCalculate = sc.Input[2];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Moving Average - Weight Volume Move-Adjusted";

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

        Subgraph_MOMA.Name = "Move-Adjusted Moving Average";
        Subgraph_MOMA.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_VOMA.Name = "Volume Moving Average";
        Subgraph_VOMA.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_WMA.Name = "Weighted Moving Average";
        Subgraph_WMA.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_WEVOMO.Name = "Weight Volume Move-Adjusted Moving Average";
        Subgraph_WEVOMO.DrawStyle = DRAWSTYLE_LINE;
    }
}

```



```

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

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

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

Input_NumberOfBarsToCalculate.Name = "Number of Bars to Calculate";
Input_NumberOfBarsToCalculate.SetIntLimits(1, MAX_STUDY_LENGTH);
Input_NumberOfBarsToCalculate.SetInt(1000);

return;
}

if (Input_NumberOfBarsToCalculate.GetInt() < Input_Length.GetInt())
    Input_NumberOfBarsToCalculate.SetInt(Input_Length.GetInt() + 1);

sc.DataStartIndex = sc.ArraySize - Input_NumberOfBarsToCalculate.GetInt();
if (sc.DataStartIndex < 0)
    sc.DataStartIndex = 0;

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

float MOMA_Numerator = 0;
float MOMA_Denominator = 0;
float VOMA_Numerator = 0;
float VOMA_Denominator = 0;

for (int WEVOMOIndex = sc.Index - Input_Length.GetInt() + 1; WEVOMOIndex <= sc.Index; WEVOMOIndex++)
{
    MOMA_Numerator += sc.BaseDataIn[Input_InputData.GetInputDataIndex()][WEVOMOIndex] *
fabs(sc.BaseDataIn[Input_InputData.GetInputDataIndex()][WEVOMOIndex] -
sc.BaseDataIn[Input_InputData.GetInputDataIndex()][WEVOMOIndex - 1]);
    MOMA_Denominator += fabs(sc.BaseDataIn[Input_InputData.GetInputDataIndex()][WEVOMOIndex] -
sc.BaseDataIn[Input_InputData.GetInputDataIndex()][WEVOMOIndex - 1]);
    VOMA_Numerator += sc.BaseDataIn[Input_InputData.GetInputDataIndex()][WEVOMOIndex] *
sc.Volume[WEVOMOIndex];
    VOMA_Denominator += sc.Volume[WEVOMOIndex];
}

if (MOMA_Denominator == 0)
    Subgraph_MOMA[sc.Index] = sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index];
else
    Subgraph_MOMA[sc.Index] = MOMA_Numerator / MOMA_Denominator;

if (VOMA_Denominator == 0)
    Subgraph_VOMA[sc.Index] = sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index];
else
    Subgraph_VOMA[sc.Index] = VOMA_Numerator / VOMA_Denominator;

sc.WeightedMovingAverage(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_WMA,
Input_Length.GetInt());

Subgraph_WEVOMO[sc.Index] = (Subgraph_MOMA[sc.Index] + Subgraph_VOMA[sc.Index] +
Subgraph_WMA[sc.Index]) / 3.0f;
}

/*=====*/
SCSFExport scsf_KlingerVolumeOscillator(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_FastMA = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SlowMA = sc.Subgraph[1];

```

```

SCSubgraphRef Subgraph_KVO = sc.Subgraph[2];
SCSubgraphRef Subgraph_Trigger = sc.Subgraph[3];

SCFloatArrayRef Array_Trend = Subgraph_KVO.Arrays[0];
SCFloatArrayRef Array_DailyMeasurement = Subgraph_KVO.Arrays[1];
SCFloatArrayRef Array_CumulativeMeasurement = Subgraph_KVO.Arrays[2];
SCFloatArrayRef Array_VolumeForce = Subgraph_KVO.Arrays[3];

SCInputRef Input_FastLength = sc.Input[0];
SCInputRef Input_SlowLength = sc.Input[1];
SCInputRef Input_KVOMAType = sc.Input[2];
SCInputRef Input_TriggerMALength = sc.Input[3];
SCInputRef Input_TriggerMAType = sc.Input[4];

if (sc.SetDefaults)
{
    sc.GraphName = "Klinger Volume Oscillator";
    sc.AutoLoop = 1;
    sc.GraphRegion = 1;

    Subgraph_FastMA.Name = "Fast Moving Average";
    Subgraph_FastMA.DrawStyle = DRAWSTYLE_IGNORE;

    Subgraph_SlowMA.Name = "Slow Moving Average";
    Subgraph_SlowMA.DrawStyle = DRAWSTYLE_IGNORE;

    Subgraph_KVO.Name = "Klinger Volume Oscillator";
    Subgraph_KVO.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_KVO.DrawZeros = true;
    Subgraph_KVO.PrimaryColor = RGB(0, 255, 0);

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

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

    Input_SlowLength.Name = "Slow Moving Average Length";
    Input_SlowLength.SetInt(55);
    Input_SlowLength.SetIntLimits(1, MAX_STUDY_LENGTH);

    Input_TriggerMALength.Name = "Trigger Moving Average Length";
    Input_TriggerMALength.SetInt(13);
    Input_TriggerMALength.SetIntLimits(1, MAX_STUDY_LENGTH);

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

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

    return;
}

float HLC = sc.High[sc.Index] + sc.Low[sc.Index] + sc.Close[sc.Index];
float PrevHLC = 0.0f;

if (sc.Index > 0)
    PrevHLC = sc.High[sc.Index - 1] + sc.Low[sc.Index - 1] + sc.Close[sc.Index - 1];

if (HLC > PrevHLC)

```

```

    Array_Trend[sc.Index] = 1.0f;
else
    Array_Trend[sc.Index] = -1.0f;

Array_DailyMeasurement[sc.Index] = sc.High[sc.Index] - sc.Low[sc.Index];

if (sc.Index > 0)
{
    if (Array_Trend[sc.Index] == Array_Trend[sc.Index - 1])
        Array_CumulativeMeasurement[sc.Index] = Array_CumulativeMeasurement[sc.Index - 1] +
Array_DailyMeasurement[sc.Index];
    else
        Array_CumulativeMeasurement[sc.Index] = Array_DailyMeasurement[sc.Index - 1] +
Array_DailyMeasurement[sc.Index];
}
else
{
    Array_CumulativeMeasurement[sc.Index] = Array_DailyMeasurement[sc.Index];
}

Array_VolumeForce[sc.Index] = sc.Volume[sc.Index] * fabs(2.0f * (Array_DailyMeasurement[sc.Index] /
Array_CumulativeMeasurement[sc.Index] - 1.0f)) * Array_Trend[sc.Index] * 100.0f;

    sc.MovingAverage(Array_VolumeForce, Subgraph_FastMA, Input_KVOMAType.GetMovAvgType(),
Input_FastLength.GetInt());
    sc.MovingAverage(Array_VolumeForce, Subgraph_SlowMA, Input_KVOMAType.GetMovAvgType(),
Input_SlowLength.GetInt());

    Subgraph_KVO[sc.Index] = Subgraph_FastMA[sc.Index] - Subgraph_SlowMA[sc.Index];

    sc.MovingAverage(Subgraph_KVO, Subgraph_Trigger, Input_TriggerMAType.GetMovAvgType(),
Input_TriggerMALength.GetInt());
}

/*=====*/
SCSFExport scsf_CyclePeriod(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_SmoothedData = sc.Subgraph[0];
    SCSubgraphRef Subgraph_CyberCycle = sc.Subgraph[1];
    SCSubgraphRef Subgraph_CyclePeriod = sc.Subgraph[2];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_CCLength = sc.Input[1];
    SCInputRef Input_MedPhaseChangeLength = sc.Input[2];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Cycle Period";

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

        Subgraph_CyclePeriod.Name = "Cycle Period";
        Subgraph_CyclePeriod.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_CyclePeriod.PrimaryColor = RGB(0, 255, 0);
        Subgraph_CyclePeriod.LineWidth = 1;
        Subgraph_CyclePeriod.DrawZeros = true;

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

        Input_CCLength.Name = "Cyber Cycle Length";
        Input_CCLength.SetInt(28);
        Input_CCLength.SetIntLimits(1, MAX_STUDY_LENGTH);
    }
}

```

```

    Input_MedPhaseChangeLength.Name = "Median Phase Change Length";
    Input_MedPhaseChangeLength.SetInt(5);
    Input_MedPhaseChangeLength.SetIntLimits(1, MAX_STUDY_LENGTH);

    return;
}

sc.FourBarSymmetricalFIRFilter(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_SmoothedData,
sc.Index);

sc.CyberCycle(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_SmoothedData,
Subgraph_CyberCycle, sc.Index, Input_CCLength.GetInt());

sc.DominantCyclePeriod(Subgraph_CyberCycle, Subgraph_CyclePeriod, sc.Index,
Input_MedPhaseChangeLength.GetInt());
}

/*=====*/
SCSFExport scsf_LaguerreFilter(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_SmoothedData = sc.Subgraph[0];
    SCSubgraphRef Subgraph_LaguerreFilter = sc.Subgraph[1];

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

    if (sc.SetDefaults)
    {
        sc.GraphName = "Laguerre Filter";

        sc.GraphRegion = 0;
        sc.ValueFormat = 2;
        sc.AutoLoop = 1;
        sc.AutoLoop = 1;

        Subgraph_SmoothedData.Name = "Smoothed Data";
        Subgraph_SmoothedData.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_SmoothedData.PrimaryColor = RGB(0, 0, 255);
        Subgraph_SmoothedData.LineWidth = 1;
        Subgraph_SmoothedData.DrawZeros = true;

        Subgraph_LaguerreFilter.Name = "Laguerre Filter";
        Subgraph_LaguerreFilter.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_LaguerreFilter.PrimaryColor = RGB(0, 255, 0);
        Subgraph_LaguerreFilter.LineWidth = 1;
        Subgraph_LaguerreFilter.DrawZeros = true;

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

        Input_DampingFactor.Name = "Damping Factor";
        Input_DampingFactor.SetFloat(0.8f);
        Input_DampingFactor.SetFloatLimits(0, 1);

        return;
    }

    sc.FourBarSymmetricalFIRFilter(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_SmoothedData,
sc.Index);

    sc.LaguerreFilter(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_LaguerreFilter, sc.Index,
Input_DampingFactor.GetFloat());
}

```

```

/*=====*/
SCSFExport scsf_LaguerreRSI(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_LaguerreFilter = sc.Subgraph[0];
    SCSubgraphRef Subgraph_LaguerreRSI = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Line1 = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Line2 = sc.Subgraph[3];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_DampingFactor = sc.Input[1];
    SCInputRef Input_Line1Value = sc.Input[2];
    SCInputRef Input_Line2Value = sc.Input[3];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Laguerre RSI";

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

        Subgraph_LaguerreRSI.Name = "Laguerre RSI";
        Subgraph_LaguerreRSI.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_LaguerreRSI.PrimaryColor = RGB(0, 255, 0);
        Subgraph_LaguerreRSI.DrawZeros = true;

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

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

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

        Input_DampingFactor.Name = "Damping Factor";
        Input_DampingFactor.SetFloat(0.5);
        Input_DampingFactor.SetFloatLimits(0, 1);

        Input_Line1Value.Name = "Line 1 Value";
        Input_Line1Value.SetFloat(0.8f);
        Input_Line1Value.SetFloatLimits(0, 1);

        Input_Line2Value.Name = "Line 2 Value";
        Input_Line2Value.SetFloat(0.2f);
        Input_Line2Value.SetFloatLimits(0, 1);

        return;
    }

    sc.LaguerreFilter(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_LaguerreFilter, sc.Index,
    Input_DampingFactor.GetFloat());

    float UpSum = 0.0f;
    float DownSum = 0.0f;

    for (int k = 0; k < 3; k++)
    {
        if (Subgraph_LaguerreFilter.Arrays[k][sc.Index] >= Subgraph_LaguerreFilter.Arrays[k + 1][sc.Index])
            UpSum += Subgraph_LaguerreFilter.Arrays[k][sc.Index] - Subgraph_LaguerreFilter.Arrays[k + 1][sc.Index];
        else

```

```

        DownSum += Subgraph_LaguerreFilter.Arrays[k + 1][sc.Index] - Subgraph_LaguerreFilter.Arrays[k][sc.Index];
    }

    if (UpSum + DownSum != 0.0f)
        Subgraph_LaguerreRSI[sc.Index] = UpSum / (UpSum + DownSum);
    else
        Subgraph_LaguerreRSI[sc.Index] = 0.0f;

    Subgraph_Line1[sc.Index] = Input_Line1Value.GetFloat();
    Subgraph_Line2[sc.Index] = Input_Line2Value.GetFloat();
}

/* ===== */
SCSFExport scsf_VolumePositiveNegative(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_ATR = sc.Subgraph[0];
    SCSubgraphRef Subgraph_VPN = sc.Subgraph[1];
    SCSubgraphRef Subgraph_AvgVPN = sc.Subgraph[2];
    SCSubgraphRef Subgraph_CriticalValue = sc.Subgraph[3];
    SCSubgraphRef Subgraph_CenterLine = sc.Subgraph[4];

    SCFloatArrayRef Array_PositiveVolume = sc.Subgraph[1].Arrays[0];
    SCFloatArrayRef Array_SumPositiveVolume = sc.Subgraph[1].Arrays[1];
    SCFloatArrayRef Array_NegativeVolume = sc.Subgraph[1].Arrays[2];
    SCFloatArrayRef Array_SumNegativeVolume = sc.Subgraph[1].Arrays[3];
    SCFloatArrayRef Array_SumVolume = sc.Subgraph[1].Arrays[4];
    SCFloatArrayRef Array_VolumeRatio = sc.Subgraph[1].Arrays[5];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_ATRCoefficient = sc.Input[1];
    SCInputRef Input_VPNLength = sc.Input[2];
    SCInputRef Input_VPNSmoothLength = sc.Input[3];
    SCInputRef Input_AvgVPNLength = sc.Input[4];
    SCInputRef Input_ATRAvgType = sc.Input[5];
    SCInputRef Input_SmoothVPNAvgType = sc.Input[6];
    SCInputRef Input_AvgVPNAvgType = sc.Input[7];
    SCInputRef Input_CriticalValue = sc.Input[8];

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

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

        Subgraph_ATR.Name = "ATR";
        Subgraph_ATR.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_VPN.Name = "VPN";
        Subgraph_VPN.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_VPN.PrimaryColor = RGB(0, 255, 0);
        Subgraph_VPN.SecondaryColor = RGB(255, 0, 0);
        Subgraph_VPN.LineWidth = 1;
        Subgraph_VPN.DrawZeros = true;

        Subgraph_AvgVPN.Name = "Avg VPN";
        Subgraph_AvgVPN.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_AvgVPN.PrimaryColor = RGB(0, 0, 255);
        Subgraph_AvgVPN.LineWidth = 1;
        Subgraph_AvgVPN.DrawZeros = true;

        Subgraph_CriticalValue.Name = "Critical Value";
        Subgraph_CriticalValue.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_CriticalValue.PrimaryColor = RGB(128, 0, 128);
    }
}

```

```

Subgraph_CriticalValue.LineWidth = 1;
Subgraph_CriticalValue.DrawZeros = true;

Subgraph_CenterLine.Name = "Center Line";
Subgraph_CenterLine.DrawStyle = DRAWSTYLE_LINE;
Subgraph_CenterLine.PrimaryColor = RGB(128, 128, 0);
Subgraph_CenterLine.LineWidth = 1;
Subgraph_CenterLine.DrawZeros = true;

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

Input_ATRCoefficient.Name = "ATR Coefficient";
Input_ATRCoefficient.SetFloat(0.1f);
Input_ATRCoefficient.SetFloatLimits(0, FLT_MAX);

Input_VPNLength.Name = "VPN Length";
Input_VPNLength.SetInt(30);
Input_VPNLength.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_VPNSmoothLength.Name = "VPN Smoothing Length";
Input_VPNSmoothLength.SetInt(3);
Input_VPNSmoothLength.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_AvgVPNLength.Name = "Avg VPN Length";
Input_AvgVPNLength.SetInt(30);
Input_AvgVPNLength.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_ATRAvgType.Name = "ATR Avg Type";
Input_ATRAvgType.SetMovAvgType(MOVAVGTYPE_WILDERS);

Input_SmoothVPNAvgType.Name = "VPN Smoothing Avg Type";
Input_SmoothVPNAvgType.SetMovAvgType(MOVAVGTYPE_EXPONENTIAL);

Input_AvgVPNAvgType.Name = "Avg VPN Avg Type";
Input_AvgVPNAvgType.SetMovAvgType(MOVAVGTYPE_SIMPLE);

Input_CriticalValue.Name = "Critical Value";
Input_CriticalValue.SetFloat(10.0f);
Input_CriticalValue.SetFloatLimits(0, FLT_MAX);

return;
}

sc.ATR(sc.BaseDataIn, Subgraph_ATR, Input_VPNLength.GetInt(), Input_ATRAvgType.GetMovAvgType());

Subgraph_CriticalValue[sc.Index] = Input_CriticalValue.GetFloat();

if (sc.Index == 0)
{
    Array_PositiveVolume[sc.Index] = 0.0f;
    Array_NegativeVolume[sc.Index] = 0.0f;
}

else
{
    if (sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index] >=
sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - 1] + Input_ATRCoefficient.GetFloat() *
Subgraph_ATR[sc.Index])
        Array_PositiveVolume[sc.Index] = sc.Volume[sc.Index];
    else
        Array_PositiveVolume[sc.Index] = 0.0f;

    if (sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index] <=
sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - 1] - Input_ATRCoefficient.GetFloat() *

```

```

Subgraph_ATR[sc.Index])
    Array_NegativeVolume[sc.Index] = sc.Volume[sc.Index];
else
    Array_NegativeVolume[sc.Index] = 0.0f;
}

sc.Summation(Array_PositiveVolume, Array_SumPositiveVolume, Input_VPNLength.GetInt());
sc.Summation(Array_NegativeVolume, Array_SumNegativeVolume, Input_VPNLength.GetInt());
sc.Summation(sc.BaseDataIn[SC_VOLUME], Array_SumVolume, Input_VPNLength.GetInt());

if (Array_SumVolume[sc.Index] != 0.0f)
    Array_VolumeRatio[sc.Index] = 100.0f * (Array_SumPositiveVolume[sc.Index] -
Array_SumNegativeVolume[sc.Index]) / Array_SumVolume[sc.Index];
else
    Array_VolumeRatio[sc.Index] = 0.0f;

sc.MovingAverage(Array_VolumeRatio, Subgraph_VPN, Input_SmoothVPNAvgType.GetMovAvgType(),
Input_VPNSmoothLength.GetInt());

sc.MovingAverage(Subgraph_VPN, Subgraph_AvgVPN, Input_AvgVPNAvgType.GetMovAvgType(),
Input_AvgVPNLength.GetInt());

if (Subgraph_VPN[sc.Index] > Subgraph_CriticalValue[sc.Index])
    Subgraph_VPN.DataColor[sc.Index] = Subgraph_VPN.PrimaryColor;
else
    Subgraph_VPN.DataColor[sc.Index] = Subgraph_VPN.SecondaryColor;

Subgraph_CenterLine[sc.Index] = 0.0f;
}

/*=====*/
SCSFExport scsf_ChoppinessIndex(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Choppiness = sc.Subgraph[0];
    SCSubgraphRef Subgraph_ATR = sc.Subgraph[1];
    SCSubgraphRef Subgraph_ATRSum = sc.Subgraph[2];

    SCFloatArrayRef Array_HighestHigh = sc.Subgraph[0].Arrays[0];
    SCFloatArrayRef Array_LowestLow = sc.Subgraph[0].Arrays[1];

    SCInputRef Input_SumPeriod = sc.Input[0];
    SCInputRef Input_ATRPeriod = sc.Input[1];

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

        sc.AutoLoop = true;
        sc.FreeDLL = 0;

        Subgraph_Choppiness.Name = "Choppiness Index";
        Subgraph_Choppiness.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Choppiness.LineWidth = 1;

        Input_SumPeriod.Name = "Summation Period";
        Input_SumPeriod.SetInt(14);
        Input_SumPeriod.SetIntLimits(2, MAX_STUDY_LENGTH);

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

        return;
    }
}

```



```

int MaxPeriod = max(Input_ATRPeriod.GetInt(), Input_SumPeriod.GetInt());

sc.DataStartIndex = MaxPeriod - 1;

sc.Highest(sc.BaseDataIn[SC_HIGH], Array_HighestHigh, Input_SumPeriod.GetInt());
float HighestHigh = Array_HighestHigh[sc.Index];

sc.Lowest(sc.BaseDataIn[SC_LOW], Array_LowestLow, Input_SumPeriod.GetInt());
float LowestLow = Array_LowestLow[sc.Index];

sc.ATR(sc.BaseDataIn, Subgraph_ATR, Input_ATRPeriod.GetInt(), MOVAVGTYPE_SIMPLE);

sc.Summation(Subgraph_ATR, Subgraph_ATRSum, Input_SumPeriod.GetInt());
float Summation = Subgraph_ATRSum[sc.Index];

if (HighestHigh - LowestLow != 0.0f)
{
    Subgraph_Choppiness[sc.Index]
        = 100 * log10(Summation / (HighestHigh - LowestLow))
        / log10(static_cast<float>(Input_SumPeriod.GetInt()));
}
else
    Subgraph_Choppiness[sc.Index] = 0.0f;
}

/*=====*/
SCSFExport scsf_HalfTrend(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_HalfTrend = sc.Subgraph[0];
    SCSubgraphRef Subgraph_ArrowUp = sc.Subgraph[1];
    SCSubgraphRef Subgraph_ArrowDown = sc.Subgraph[2];
    SCSubgraphRef Subgraph_ATRHigh = sc.Subgraph[3];
    SCSubgraphRef Subgraph_ATRLow = sc.Subgraph[4];

    SCFloatArrayRef Array_MAofLowPrice = Subgraph_HalfTrend.Arrays[0];
    SCFloatArrayRef Array_MAofHighPrice = Subgraph_HalfTrend.Arrays[1];
    SCFloatArrayRef Array_Trend = Subgraph_HalfTrend.Arrays[2];
    SCFloatArrayRef Array_NextTrend = Subgraph_HalfTrend.Arrays[3];
    SCFloatArrayRef Array_ATR = Subgraph_HalfTrend.Arrays[4];
    SCFloatArrayRef Array_TR = Subgraph_HalfTrend.Arrays[5];
    SCFloatArrayRef Array_MinHighPrice = Subgraph_HalfTrend.Arrays[6];
    SCFloatArrayRef Array_MaxLowPrice = Subgraph_HalfTrend.Arrays[7];
    SCFloatArrayRef Array_Up = Subgraph_HalfTrend.Arrays[8];
    SCFloatArrayRef Array_Down = Subgraph_HalfTrend.Arrays[9];

    SCInputRef Input_Length = sc.Input[0];
    SCInputRef Input_ATRLength = sc.Input[1];
    SCInputRef Input_ATRMAType = sc.Input[2];
    SCInputRef Input_ChangeDev = sc.Input[3];

    // Section 1 - Set the configuration variables and defaults
    if (sc.SetDefaults)
    {
        sc.GraphName = "HalfTrend";
        sc.StudyDescription = "HalfTrend.";
        sc.AutoLoop = 1; //Automatic looping is enabled.

        // During development set this flag to 1, so the DLL can be rebuilt without restarting Sierra Chart. When development
        // is completed, set it to 0 to improve performance.
        sc.FreeDLL = 1;

        sc.DrawZeros = false;
        sc.GraphRegion = 0;
        sc.ValueFormat = sc.BaseGraphValueFormat;
    }
}

```

```

Subgraph_HalfTrend.Name = "HalfTrend";
Subgraph_HalfTrend.DrawStyle = DRAWSTYLE_LINE;
Subgraph_HalfTrend.PrimaryColor = COLOR_RED;
Subgraph_HalfTrend.SecondaryColor = COLOR_GREEN;
Subgraph_HalfTrend.LineWidth = 2;
Subgraph_HalfTrend.SecondaryColorUsed = 1;

Subgraph_ArrowUp.Name = "Arrow Up";
Subgraph_ArrowUp.DrawStyle = DRAWSTYLE_ARROWUP;
Subgraph_ArrowUp.PrimaryColor = COLOR_GREEN;
Subgraph_ArrowUp.LineWidth = 4;

Subgraph_ArrowDown.Name = "Arrow Down";
Subgraph_ArrowDown.DrawStyle = DRAWSTYLE_ARROWDOWN;
Subgraph_ArrowDown.PrimaryColor = COLOR_RED;
Subgraph_ArrowDown.LineWidth = 4;

Subgraph_ATRHigh.Name = "ATR channel high";
Subgraph_ATRHigh.DrawStyle = DRAWSTYLE_LINE;
Subgraph_ATRHigh.LineStyle = LINESTYLE_DOT;
Subgraph_ATRHigh.PrimaryColor = COLOR_GREEN;
Subgraph_ATRHigh.LineWidth = 1;

Subgraph_ATRLow.Name = "ATR channel low";
Subgraph_ATRLow.DrawStyle = DRAWSTYLE_LINE;
Subgraph_ATRLow.LineStyle = LINESTYLE_DOT;
Subgraph_ATRLow.PrimaryColor = COLOR_RED;
Subgraph_ATRLow.LineWidth = 1;

Input_Length.Name = "Length";
Input_Length.SetInt(2);
Input_Length.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_ATRLength.Name = "ATR Length";
Input_ATRLength.SetInt(100);
Input_ATRLength.SetIntLimits(1, MAX_STUDY_LENGTH);

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

Input_ChanDev.Name = "Channel Deviation";
Input_ChanDev.SetInt(2);
Input_ChanDev.SetIntLimits(1, MAX_STUDY_LENGTH);

return;
}

// Section 2 - Do data processing here
float LowPrice = sc.GetLowest(sc.BaseDataIn[SC_LOW], Input_Length.GetInt());
float HighPrice = sc.GetHighest(sc.BaseDataIn[SC_HIGH], Input_Length.GetInt());
sc.SimpleMovAvg(sc.BaseDataIn[SC_LOW], Array_MAofLowPrice, Input_Length.GetInt());
sc.SimpleMovAvg(sc.BaseDataIn[SC_HIGH], Array_MAofHighPrice, Input_Length.GetInt());
sc.ATR(sc.BaseDataIn, Array_TR, Array_ATR, Input_ATRLength.GetInt(), Input_ATRMAType.GetMovAvgType());
float Deviation = Input_ChanDev.GetInt() * Array_ATR[sc.Index] / 2.0f;

Array_NextTrend[sc.Index] = Array_NextTrend[sc.Index - 1];
Array_Trend[sc.Index] = Array_Trend[sc.Index - 1];
Subgraph_ArrowUp[sc.Index] = 0;
Subgraph_ArrowDown[sc.Index] = 0;
Array_MinHighPrice[sc.Index] = Array_MinHighPrice[sc.Index - 1];
Array_MaxLowPrice[sc.Index] = Array_MaxLowPrice[sc.Index - 1];

if (sc.Index == 0)

```

```

{
    sc.ValueFormat = sc.BaseGraphValueFormat;
    Subgraph_HalfTrend[sc.Index] = sc.Close[sc.Index];
    Array_Trend[sc.Index] = 0;
    Array_NextTrend[sc.Index] = 0;
    Subgraph_ArrowUp[sc.Index] = 0;
    Subgraph_ArrowDown[sc.Index] = 0;
    Array_MinHighPrice[sc.Index] = sc.BaseDataIn[SC_HIGH][sc.Index];
    Array_MaxLowPrice[sc.Index] = sc.BaseDataIn[SC_LOW][sc.Index];
    return;
}

if (Array_NextTrend[sc.Index] == 1)
{
    Array_MaxLowPrice[sc.Index] = max(LowPrice, Array_MaxLowPrice[sc.Index]);
    if (Array_MAofHighPrice[sc.Index] < Array_MaxLowPrice[sc.Index] && sc.BaseData[SC_LAST][sc.Index] <
sc.BaseData[SC_LOW][sc.Index - 1])
    {
        Array_Trend[sc.Index] = 1.0;
        Array_NextTrend[sc.Index] = 0;
        Array_MinHighPrice[sc.Index] = HighPrice;
    }
}

if (Array_NextTrend[sc.Index] == 0)
{
    Array_MinHighPrice[sc.Index] = min(HighPrice, Array_MinHighPrice[sc.Index]);
    if (Array_MAofLowPrice[sc.Index] > Array_MinHighPrice[sc.Index] && sc.BaseData[SC_LAST][sc.Index] >
sc.BaseData[SC_HIGH][sc.Index - 1])
    {
        Array_Trend[sc.Index] = 0.0;
        Array_NextTrend[sc.Index] = 1;
        Array_MaxLowPrice[sc.Index] = LowPrice;
    }
}

if (Array_Trend[sc.Index] == 0.0)
{
    if (Array_Trend[sc.Index - 1] != 0.0)
    {
        Array_Up[sc.Index] = Array_Down[sc.Index - 1];
        Subgraph_ArrowUp[sc.Index] = Array_Up[sc.Index] - Array_ATR[sc.Index] / 2.0f;
    }
    else
    {
        Array_Up[sc.Index] = max(Array_MaxLowPrice[sc.Index], Array_Up[sc.Index - 1]);
    }
    Subgraph_ATRHigh[sc.Index] = Array_Up[sc.Index] + Deviation;
    Subgraph_ATRLow[sc.Index] = Array_Up[sc.Index] - Deviation;
    Array_Down[sc.Index] = 0.0;
}
else
{
    if (Array_Trend[sc.Index - 1] != 1.0)
    {
        Array_Down[sc.Index] = Array_Up[sc.Index - 1];
        Subgraph_ArrowDown[sc.Index] = Array_Down[sc.Index] + Array_ATR[sc.Index] / 2.0f;
    }
    else
    {
        Array_Down[sc.Index] = min(Array_MinHighPrice[sc.Index], Array_Down[sc.Index - 1]);
    }
    Subgraph_ATRHigh[sc.Index] = Array_Down[sc.Index] + Deviation;
    Subgraph_ATRLow[sc.Index] = Array_Down[sc.Index] - Deviation;
    Array_Up[sc.Index] = 0.0;
}

```

```

}

Subgraph_HalfTrend[sc.Index] = Array_Trend[sc.Index] == 0 ? Array_Up[sc.Index] : Array_Down[sc.Index];
Subgraph_HalfTrend.DataColor[sc.Index] = Array_Trend[sc.Index] > 0 ? Subgraph_HalfTrend.PrimaryColor :
Subgraph_HalfTrend.SecondaryColor;
}

/*=====*/
SCSFExport scsf_AdaptiveCyberCycle(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_SmoothedData = sc.Subgraph[0];
    SCSubgraphRef Subgraph_CyberCycle = sc.Subgraph[1];
    SCSubgraphRef Subgraph_CyclePeriod = sc.Subgraph[2];
    SCSubgraphRef Subgraph_AdaptiveCC = sc.Subgraph[3];
    SCSubgraphRef Subgraph_Trigger = sc.Subgraph[4];
    SCSubgraphRef Subgraph_CenterLine = sc.Subgraph[5];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_Length = sc.Input[1];
    SCInputRef Input_MedPhaseChangeLength = sc.Input[2];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Adaptive Cyber Cycle";

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

        Subgraph_AdaptiveCC.Name = "Adaptive Cyber Cycle";
        Subgraph_AdaptiveCC.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_AdaptiveCC.PrimaryColor = RGB(0, 255, 0);
        Subgraph_AdaptiveCC.LineWidth = 1;
        Subgraph_AdaptiveCC.DrawZeros = true;

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

        Subgraph_CenterLine.Name = "Center Line";
        Subgraph_CenterLine.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_CenterLine.PrimaryColor = RGB(128, 128, 0);
        Subgraph_CenterLine.LineWidth = 1;
        Subgraph_CenterLine.DrawZeros = true;

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

        Input_Length.Name = "Length";
        Input_Length.SetInt(28);
        Input_Length.SetIntLimits(1, MAX_STUDY_LENGTH);

        Input_MedPhaseChangeLength.Name = "Median Phase Change Length";
        Input_MedPhaseChangeLength.SetInt(5);
        Input_MedPhaseChangeLength.SetIntLimits(1, MAX_STUDY_LENGTH);

        return;
    }

    sc.FourBarSymmetricalFIRFilter(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_SmoothedData,
sc.Index);

    sc.CyberCycle(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_SmoothedData,

```

```
Subgraph_CyberCycle, sc.Index, Input_Length.GetInt());
```

```
sc.DominantCyclePeriod(Subgraph_CyberCycle, Subgraph_CyclePeriod, sc.Index,  
Input_MedPhaseChangeLength.GetInt());
```

```
float a;
```

```
if (Subgraph_CyclePeriod[sc.Index] != 0.0f)  
    a = 2.0f / (Subgraph_CyclePeriod[sc.Index] + 1);  
else  
    a = 0.0f;
```

```
if (sc.Index < 6)  
    Subgraph_AdaptiveCC[sc.Index] = 0.25f * (sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index] - 2 *  
sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - 1] + sc.BaseDataIn[Input_InputData.GetInputDataIndex()][  
sc.Index - 2]);  
else  
    Subgraph_AdaptiveCC[sc.Index] = (1 - 0.5f * a) * (1 - 0.5f * a) * (Subgraph_SmoothedData[sc.Index] - 2.0f *  
Subgraph_SmoothedData[sc.Index - 1] + Subgraph_SmoothedData[sc.Index - 2]) + 2.0f * (1.0f - a) *  
Subgraph_AdaptiveCC[sc.Index - 1] - (1.0f - a) * (1.0f - a) * Subgraph_AdaptiveCC[sc.Index - 2];
```

```
Subgraph_Trigger[sc.Index] = Subgraph_AdaptiveCC[sc.Index - 1];
```

```
Subgraph_CenterLine[sc.Index] = 0.0f;
```

```
}
```

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

```
SCSFExport scsf_AdaptiveCGOscillator(SCStudyInterfaceRef sc)
```

```
{
```

```
SCSubgraphRef Subgraph_SmoothedData = sc.Subgraph[0];  
SCSubgraphRef Subgraph_CyberCycle = sc.Subgraph[1];  
SCSubgraphRef Subgraph_CyclePeriod = sc.Subgraph[2];  
SCSubgraphRef Subgraph_AdaptiveCGO = sc.Subgraph[3];  
SCSubgraphRef Subgraph_Trigger = sc.Subgraph[4];  
SCSubgraphRef Subgraph_CenterLine = sc.Subgraph[5];
```

```
SCInputRef Input_InputData = sc.Input[0];  
SCInputRef Input_Length = sc.Input[1];  
SCInputRef Input_MedPhaseChangeLength = sc.Input[2];
```

```
if (sc.SetDefaults)  
{  
    sc.GraphName = "Adaptive Center of Gravity Oscillator";
```

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

```
    Subgraph_AdaptiveCGO.Name = "Adaptive Center of Gravity Oscillator";  
    Subgraph_AdaptiveCGO.DrawStyle = DRAWSTYLE_LINE;  
    Subgraph_AdaptiveCGO.PrimaryColor = RGB(0, 255, 0);  
    Subgraph_AdaptiveCGO.LineWidth = 1;  
    Subgraph_AdaptiveCGO.DrawZeros = true;
```

```
    Subgraph_Trigger.Name = "Trigger Line";  
    Subgraph_Trigger.DrawStyle = DRAWSTYLE_LINE;  
    Subgraph_Trigger.PrimaryColor = RGB(0, 0, 255);  
    Subgraph_Trigger.LineWidth = 1;  
    Subgraph_Trigger.DrawZeros = true;
```

```
    Subgraph_CenterLine.Name = "Center Line";  
    Subgraph_CenterLine.DrawStyle = DRAWSTYLE_LINE;  
    Subgraph_CenterLine.PrimaryColor = RGB(128, 128, 0);  
    Subgraph_CenterLine.LineWidth = 1;  
    Subgraph_CenterLine.DrawZeros = true;
```

```

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

Input_Length.Name = "Length";
Input_Length.SetInt(28);
Input_Length.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_MedPhaseChangeLength.Name = "Median Phase Change Length";
Input_MedPhaseChangeLength.SetInt(5);
Input_MedPhaseChangeLength.SetIntLimits(1, MAX_STUDY_LENGTH);

return;
}

sc.FourBarSymmetricalFIRFilter(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_SmoothedData,
sc.Index);

sc.CyberCycle(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_SmoothedData,
Subgraph_CyberCycle, sc.Index, Input_Length.GetInt());

sc.DominantCyclePeriod(Subgraph_CyberCycle, Subgraph_CyclePeriod, sc.Index,
Input_MedPhaseChangeLength.GetInt());

float Period = Subgraph_CyclePeriod[sc.Index];
int IntPeriod = static_cast<int>(Period/2);

float Num = 0.0f;
float Den = 0.0f;

for (int i = 0; i <= IntPeriod - 1; i++)
{
    Num += (1 + i) * sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - i];
    Den += sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - i];
}

if (Den != 0)
    Subgraph_AdaptiveCGO[sc.Index] = -1.0f * Num / Den + (IntPeriod + 1)/2.0f;
else
    Subgraph_AdaptiveCGO[sc.Index] = 0;

Subgraph_Trigger[sc.Index] = Subgraph_AdaptiveCGO[sc.Index - 1];

Subgraph_CenterLine[sc.Index] = 0.0f;
}

/*=====*/
SCSFExport scsf_AdaptiveRVI(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_SmoothedData = sc.Subgraph[0];
    SCSubgraphRef Subgraph_CyberCycle = sc.Subgraph[1];
    SCSubgraphRef Subgraph_CyclePeriod = sc.Subgraph[2];
    SCSubgraphRef Subgraph_AdaptiveRVI = sc.Subgraph[3];
    SCSubgraphRef Subgraph_Trigger = sc.Subgraph[4];
    SCSubgraphRef Subgraph_CenterLine = sc.Subgraph[5];

    SCFloatArrayRef Array_CloseOpen = sc.Subgraph[3].Arrays[0];
    SCFloatArrayRef Array_HighLow = sc.Subgraph[3].Arrays[1];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_Length = sc.Input[1];
    SCInputRef Input_MedPhaseChangeLength = sc.Input[2];

    if (sc.SetDefaults)
    {

```

```

sc.GraphName = "Adaptive Relative Vigor Index";

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

Subgraph_AdaptiveRVI.Name = "Adaptive Relative Vigor Index";
Subgraph_AdaptiveRVI.DrawStyle = DRAWSTYLE_LINE;
Subgraph_AdaptiveRVI.PrimaryColor = RGB(0, 0, 255);
Subgraph_AdaptiveRVI.LineWidth = 1;
Subgraph_AdaptiveRVI.DrawZeros = true;

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

Subgraph_CenterLine.Name = "Center Line";
Subgraph_CenterLine.DrawStyle = DRAWSTYLE_LINE;
Subgraph_CenterLine.PrimaryColor = RGB(128, 128, 0);
Subgraph_CenterLine.LineWidth = 1;
Subgraph_CenterLine.DrawZeros = true;

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

Input_Length.Name = "Length";
Input_Length.SetInt(28);
Input_Length.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_MedPhaseChangeLength.Name = "Median Phase Change Length";
Input_MedPhaseChangeLength.SetInt(5);
Input_MedPhaseChangeLength.SetIntLimits(1, MAX_STUDY_LENGTH);

return;
}

sc.FourBarSymmetricalFIRFilter(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_SmoothedData,
sc.Index);

sc.CyberCycle(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_SmoothedData,
Subgraph_CyberCycle, sc.Index, Input_Length.GetInt());

sc.DominantCyclePeriod(Subgraph_CyberCycle, Subgraph_CyclePeriod, sc.Index,
Input_MedPhaseChangeLength.GetInt());

float Length = (4.0f * Subgraph_CyclePeriod[sc.Index] + 3.0f * Subgraph_CyclePeriod[sc.Index - 1] + 2.0f *
Subgraph_CyclePeriod[sc.Index - 3] + Subgraph_CyclePeriod[sc.Index - 4]) / 20.0f;
int IntLength = static_cast<int>(Length);

Array_CloseOpen[sc.Index] = ((sc.Close[sc.Index] - sc.Open[sc.Index]) + 2.0f * (sc.Close[sc.Index - 1] -
sc.Open[sc.Index - 1]) + 2.0f * (sc.Close[sc.Index - 2] - sc.Open[sc.Index - 2]) + (sc.Close[sc.Index - 3] - sc.Open[sc.Index
- 3])) / 6.0f;
Array_HighLow[sc.Index] = ((sc.High[sc.Index] - sc.Low[sc.Index]) + 2.0f * (sc.High[sc.Index - 1] - sc.Low[sc.Index - 1])
+ 2.0f * (sc.High[sc.Index - 2] - sc.Low[sc.Index - 2]) + (sc.High[sc.Index - 3] - sc.Low[sc.Index - 3])) / 6.0f;

float Num = 0.0f;
float Den = 0.0f;

for (int i = 0; i <= IntLength - 1; i++)
{
    Num += Array_CloseOpen[sc.Index - i];
    Den += Array_HighLow[sc.Index - i];
}

```



```

if (Den != 0.0f)
    Subgraph_AdaptiveRVI[sc.Index] = Num / Den;
else
    Subgraph_AdaptiveRVI[sc.Index] = 0.0f;

Subgraph_Trigger[sc.Index] = Subgraph_AdaptiveRVI[sc.Index - 1];

Subgraph_CenterLine[sc.Index] = 0.0f;
}

/*=====*/
SCSFExport scsf_SinewaveIndicator(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_SmoothedData = sc.Subgraph[0];
    SCSubgraphRef Subgraph_CyberCycle = sc.Subgraph[1];
    SCSubgraphRef Subgraph_CyclePeriod = sc.Subgraph[2];
    SCSubgraphRef Subgraph_DominantCyclePhase = sc.Subgraph[3];
    SCSubgraphRef Subgraph_SinewaveIndicator = sc.Subgraph[4];
    SCSubgraphRef Subgraph_SinewaveIndicatorLead = sc.Subgraph[5];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_Length = sc.Input[1];
    SCInputRef Input_MedPhaseChangeLength = sc.Input[2];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Sinewave Indicator";

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

        Subgraph_SinewaveIndicator.Name = "Sinewave Indicator";
        Subgraph_SinewaveIndicator.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_SinewaveIndicator.PrimaryColor = RGB(0, 0, 255);
        Subgraph_SinewaveIndicator.LineWidth = 1;
        Subgraph_SinewaveIndicator.DrawZeros = true;

        Subgraph_SinewaveIndicatorLead.Name = "Leading Sinewave Indicator";
        Subgraph_SinewaveIndicatorLead.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_SinewaveIndicatorLead.PrimaryColor = RGB(0, 255, 0);
        Subgraph_SinewaveIndicatorLead.LineWidth = 1;
        Subgraph_SinewaveIndicatorLead.DrawZeros = true;

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

        Input_Length.Name = "Length";
        Input_Length.SetInt(28);
        Input_Length.SetIntLimits(1, MAX_STUDY_LENGTH);

        Input_MedPhaseChangeLength.Name = "Median Phase Change Length";
        Input_MedPhaseChangeLength.SetInt(5);
        Input_MedPhaseChangeLength.SetIntLimits(1, MAX_STUDY_LENGTH);

        return;
    }

    sc.FourBarSymmetricalFIRFilter(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_SmoothedData,
    sc.Index);

    sc.CyberCycle(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_SmoothedData,
    Subgraph_CyberCycle, sc.Index, Input_Length.GetInt());

```



```

    sc.DominantCyclePeriod(Subgraph_CyberCycle, Subgraph_CyclePeriod, sc.Index,
Input_MedPhaseChangeLength.GetInt());

    sc.DominantCyclePhase(Subgraph_CyclePeriod, Subgraph_DominantCyclePhase, sc.Index);

    Subgraph_SinewaveIndicator[sc.Index] = static_cast<float>(sin(Subgraph_DominantCyclePhase[sc.Index] * M_PI /
180));

    Subgraph_SinewaveIndicatorLead[sc.Index] = static_cast<float>(sin((Subgraph_DominantCyclePhase[sc.Index] +
45.0f) * M_PI / 180));
}

/*=====*/
SCSFExport scsf_EvenBetterSinewaveIndicator(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_HP = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Filter = sc.Subgraph[1];
    SCSubgraphRef Subgraph_EBSWI = sc.Subgraph[2];

    SCFloatArrayRef Array_Wave = sc.Subgraph[1].Arrays[0];
    SCFloatArrayRef Array_Power = sc.Subgraph[1].Arrays[1];

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

    if (sc.SetDefaults)
    {
        sc.GraphName = "Even Better Sinewave Indicator";

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

        Subgraph_HP.Name = "HP";
        Subgraph_HP.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_Filter.Name = "Filter";
        Subgraph_Filter.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_EBSWI.Name = "Even Better Sinewave Indicator";
        Subgraph_EBSWI.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_EBSWI.PrimaryColor = RGB(0, 255, 0);
        Subgraph_EBSWI.LineWidth = 1;
        Subgraph_EBSWI.DrawZeros = true;

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

        Input_Length.Name = "Length";
        Input_Length.SetInt(40);
        Input_Length.SetIntLimits(5, MAX_STUDY_LENGTH);

        return;
    }

    float angle = static_cast<float>((360.0f / Input_Length.GetInt()) * (M_PI / 180));
    float alpha1;

    if (cos(angle) != 0)
        alpha1 = (1 - sin(angle)) / cos(angle);
    else
        alpha1 = 0;

    Subgraph_HP[sc.Index] = 0.5f * (1 + alpha1) * (sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index] -
sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - 1]) + alpha1 * Subgraph_HP[sc.Index - 1];

```

```

// Smooth using a Super Smoother Filter
float a1 = static_cast<float>(exp(-1.414 * M_PI / 10));
float b1 = static_cast<float>(2 * a1 * cos((1.414 * 180 / 10) * (M_PI / 180)));
float c2 = b1;
float c3 = -1.0f * a1 * a1;
float c1 = 1 - c2 - c3;

Subgraph_Filter[sc.Index] = c1 * (Subgraph_HP[sc.Index] + Subgraph_HP[sc.Index - 1]) / 2.0f + c2 *
Subgraph_Filter[sc.Index - 1] + c3 * Subgraph_Filter[sc.Index - 2];

sc.SimpleMovAvg(Subgraph_Filter, Array_Wave, 3);

Array_Power[sc.Index] = (Subgraph_Filter[sc.Index] * Subgraph_Filter[sc.Index] + Subgraph_Filter[sc.Index - 1] *
Subgraph_Filter[sc.Index - 1] + Subgraph_Filter[sc.Index - 2] * Subgraph_Filter[sc.Index - 2]) / 3.0f;

if (Array_Power[sc.Index] != 0)
    Subgraph_EBSWI[sc.Index] = Array_Wave[sc.Index] / sqrt(Array_Power[sc.Index]);
else
    Subgraph_EBSWI[sc.Index] = 0.0f;
}

/*=====*/
SCSFExport scsf_SmoothedAdaptiveMomentum(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_SmoothedData = sc.Subgraph[0];
    SCSubgraphRef Subgraph_CyberCycle = sc.Subgraph[1];
    SCSubgraphRef Subgraph_CyclePeriod = sc.Subgraph[2];
    SCSubgraphRef Subgraph_SmoothedAdaptiveMomentum = sc.Subgraph[3];
    SCSubgraphRef Subgraph_CenterLine = sc.Subgraph[4];

    SCFloatArrayRef Array_PriceDifference = sc.Subgraph[2].Arrays[0];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_CCLength = sc.Input[1];
    SCInputRef Input_MedPhaseChangeLength = sc.Input[2];
    SCInputRef Input_SSF3Length = sc.Input[3];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Smoothed Adaptive Momentum";

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

        Subgraph_SmoothedAdaptiveMomentum.Name = "Smoothed Adaptive Momentum";
        Subgraph_SmoothedAdaptiveMomentum.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_SmoothedAdaptiveMomentum.PrimaryColor = RGB(0, 0, 255);
        Subgraph_SmoothedAdaptiveMomentum.LineWidth = 1;
        Subgraph_SmoothedAdaptiveMomentum.DrawZeros = true;

        Subgraph_CenterLine.Name = "Center Line";
        Subgraph_CenterLine.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_CenterLine.PrimaryColor = RGB(128, 128, 0);
        Subgraph_CenterLine.LineWidth = 1;
        Subgraph_CenterLine.DrawZeros = true;

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

        Input_CCLength.Name = "Cyber Cycle Length";
        Input_CCLength.SetInt(28);
        Input_CCLength.SetIntLimits(1, MAX_STUDY_LENGTH);
    }
}

```

```

    Input_MedPhaseChangeLength.Name = "Median Phase Change Length";
    Input_MedPhaseChangeLength.SetInt(5);
    Input_MedPhaseChangeLength.SetIntLimits(1, MAX_STUDY_LENGTH);

    Input_SSF3Length.Name = "3-Pole Super Smoother Length";
    Input_SSF3Length.SetInt(8);
    Input_SSF3Length.SetIntLimits(1, MAX_STUDY_LENGTH);

    return;
}

sc.FourBarSymmetricalFIRFilter(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_SmoothedData,
sc.Index);

sc.CyberCycle(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_SmoothedData,
Subgraph_CyberCycle, sc.Index, Input_CCLength.GetInt());

sc.DominantCyclePeriod(Subgraph_CyberCycle, Subgraph_CyclePeriod, sc.Index,
Input_MedPhaseChangeLength.GetInt());

int IntPeriod = static_cast<int>(Subgraph_CyclePeriod[sc.Index]);

Array_PriceDifference[sc.Index] = sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index] -
sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - IntPeriod];

sc.SuperSmoother3Pole(Array_PriceDifference, Subgraph_SmoothedAdaptiveMomentum,
Input_SSF3Length.GetInt());
}

/*=====*/
SCSFExport scsf_LeadingIndicator(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_LeadingIndicator = sc.Subgraph[0];
    SCSubgraphRef Subgraph_MALeadingIndicator = sc.Subgraph[1];

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

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_Length1 = sc.Input[1];
    SCInputRef Input_Length2 = sc.Input[2];
    SCInputRef Input_MALength = sc.Input[3];
    SCInputRef Input_MovAvgType = sc.Input[4];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Leading Indicator";

        sc.GraphRegion = 0;
        sc.ValueFormat = 2;
        sc.AutoLoop = 1;

        Subgraph_LeadingIndicator.Name = "Leading Indicator";
        Subgraph_LeadingIndicator.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_LeadingIndicator.PrimaryColor = RGB(0, 255, 0);
        Subgraph_LeadingIndicator.LineWidth = 1;
        Subgraph_LeadingIndicator.DrawZeros = true;

        Subgraph_MALeadingIndicator.Name = "MA of Leading Indicator";
        Subgraph_MALeadingIndicator.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_MALeadingIndicator.PrimaryColor = RGB(0, 0, 255);
        Subgraph_MALeadingIndicator.LineWidth = 1;
        Subgraph_MALeadingIndicator.DrawZeros = true;

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

```

```

Input_Length1.Name = "Length 1";
Input_Length1.SetInt(7);
Input_Length1.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_Length2.Name = "Length 2";
Input_Length2.SetInt(5);
Input_Length2.SetIntLimits(1, MAX_STUDY_LENGTH);

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

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

return;
}

float alpha1 = 2.0f / (Input_Length1.GetInt() + 1);
float alpha2 = 2.0f / (Input_Length2.GetInt() + 1);

Array_Lead[sc.Index] = 2 * sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index] + (alpha1 - 2.0f) *
sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - 1] + (1.0f - alpha1) * Array_Lead[sc.Index - 1];

Subgraph_LeadingIndicator[sc.Index] = alpha2 * Array_Lead[sc.Index] + (1.0f - alpha2) * Array_Lead[sc.Index - 1];

sc.MovingAverage(Subgraph_LeadingIndicator, Subgraph_MALeadingIndicator,
Input_MovAvgType.GetMovAvgType(), Input_MALength.GetInt());
}

/*=====*/

```