

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

```
/*=====*/
SCSFExport scsf_Accumulate(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Summation = sc.Subgraph[0];

    SCInputRef Input_Data = sc.Input[0];

    if(sc.SetDefaults)
    {
        sc.GraphName = "Summation";
        sc.GraphRegion = 1;
        sc.ValueFormat = 2;

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

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

        sc.AutoLoop = 1;

        return;
    }

    sc.CumulativeSummation(sc.BaseData[Input_Data.GetInputDataIndex()], Subgraph_Summation, sc.Index);
}
```

```
/*=====*/
SCSFExport scsf_ArmsEaseOfMovement(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_EMV = sc.Subgraph[0];
    SCSubgraphRef Subgraph_EMVAvg = sc.Subgraph[1];

    SCInputRef Input_VolumeDivisor = sc.Input[0];
    SCInputRef Input_Length = sc.Input[1];
    SCInputRef Input_MovingAverageType = sc.Input[2];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Arms Ease of Movement";

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

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

        Subgraph_EMVAvg.Name = "MA of EMV";
        Subgraph_EMVAvg.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_EMVAvg.PrimaryColor = RGB(0, 0, 255);
        Subgraph_EMVAvg.DrawZeros = false;

        Input_VolumeDivisor.Name = "Volume Divisor";
        Input_VolumeDivisor.SetInt(100000000);

        Input_Length.Name = "Fast Moving Average Length";
        Input_Length.SetInt(14);
    }
}
```

```

Input_Length.SetIntLimits(1, MAX_STUDY_LENGTH);

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

sc.AutoLoop = 1;

    return;
}

if (sc.Index < 1)
    return;

sc.ArmsEMV(sc.BaseDataIn, Subgraph_EMV, Input_VolumeDivisor.GetInt(), sc.Index);

    sc.MovingAverage(Subgraph_EMV, Subgraph_EMVAvg, Input_MovingAverageType.GetMovAvgType(),
Input_Length.GetInt());
}

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

    SCInputRef Input_Length = sc.Input[3];

    if(sc.SetDefaults)
    {
        sc.GraphName = "Chaikin Money Flow";

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

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

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

        sc.AutoLoop = 1;

        return;
    }

    sc.ChaikinMoneyFlow(sc.BaseDataIn, Subgraph_CMF, Input_Length.GetInt());
}

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

    SCInputRef Input_Data = sc.Input[0];
    SCInputRef Input_Length = sc.Input[3];

```

```

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

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

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

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

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

    sc.AutoLoop = 1;

    return;
}

sc.Dispersion(sc.BaseDataIn[static_cast<int>(Input_Data.GetInputDataIndex())], Subgraph_Dispersion, sc.Index,
Input_Length.GetInt());
}

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

    SCInputRef Input_Data = sc.Input[0];
    SCInputRef Input_Value = sc.Input[3];
    SCInputRef Input_BandsType = sc.Input[4];

    if(sc.SetDefaults)
    {
        sc.GraphName = "Bands/Envelope";

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

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

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

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

        Input_Value.Name = "Percentage/Fixed Value/Ticks";
        Input_Value.SetFloat(0.05f);
        Input_Value.SetFloatLimits(0.00001f, static_cast<float>(MAX_STUDY_LENGTH));
    }
}

```

```

Input_BandsType.Name = "Bands Type";
Input_BandsType.SetCustomInputStrings("Percentage;Fixed Value;Number of Ticks");
Input_BandsType.SetCustomInputIndex(0);

sc.AutoLoop = 1;

return;
}

SCFloatArrayRef In = sc.BaseDataIn[Input_Data.GetInputDataIndex()];

if(Input_BandsType.GetIndex() == 0)
{
    sc.EnvelopePct(In, Subgraph_TopBand, Input_Value.GetFloat());
    Subgraph_BottomBand[sc.Index] = Subgraph_TopBand.Arrays[0][sc.Index];
}
else if(Input_BandsType.GetIndex() == 1)
{
    sc.EnvelopeFixed(In, Subgraph_TopBand, Input_Value.GetFloat());
    Subgraph_BottomBand[sc.Index] = Subgraph_TopBand.Arrays[0][sc.Index];
}
else
{
    sc.EnvelopeFixed(In, Subgraph_TopBand, Input_Value.GetFloat() * sc.TickSize);
    Subgraph_BottomBand[sc.Index] = Subgraph_TopBand.Arrays[0][sc.Index];
}
}

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

    SCInputRef Input_Data = sc.Input[0];
    SCInputRef Input_Length = sc.Input[3];

    if(sc.SetDefaults)
    {
        sc.GraphName = "Vertical Horizontal Filter";

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

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

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

        Input_Length.Name = "Length";
        Input_Length.SetInt(28);
        Input_Length.SetIntLimits(1,MAX_STUDY_LENGTH);

        sc.AutoLoop = 1;

return;

```

```

    }
    if(sc.Index < Input_Length.GetInt())
        return;

    SCFloatArrayRef In = sc.BaseDataIn[static_cast<int>(Input_Data.GetInputDataIndex())];
    sc.VHF(In, Subgraph_VHF, sc.Index, Input_Length.GetInt());
}

/*=====*/
SCSFExport scsf_RWI(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_RWIHigh = sc.Subgraph[0];
    SCSubgraphRef Subgraph_RWILow = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Temp2 = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Temp3 = sc.Subgraph[3];
    SCSubgraphRef Subgraph_Temp4 = sc.Subgraph[4];

    SCInputRef Input_Length = sc.Input[3];

    if(sc.SetDefaults)
    {
        sc.GraphName = "Random Walk Index";

        sc.GraphRegion = 1;

        Subgraph_RWIHigh.Name = "RWI High";
        Subgraph_RWIHigh.PrimaryColor = RGB(0,255,0);
        Subgraph_RWIHigh.DrawStyle = DRAWSTYLE_LINE;

        Subgraph_RWILow.Name = "RWI Low";
        Subgraph_RWILow.PrimaryColor = RGB(255,0,255);
        Subgraph_RWILow.DrawStyle = DRAWSTYLE_LINE;

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

        sc.AutoLoop = 1;

        return;
    }

    sc.DataStartIndex = Input_Length.GetInt() - 1;
    sc.InternalRWI(sc.BaseDataIn, Subgraph_RWIHigh, Subgraph_RWILow, Subgraph_Temp2, Subgraph_Temp3,
    Subgraph_Temp4, sc.Index, Input_Length.GetInt());
}

/*=====*/
SCSFExport scsf_UltimateOscillator(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_UltimateOscillatorOut1 = sc.Subgraph[0];
    SCSubgraphRef Subgraph_UltimateOscillatorOut2 = sc.Subgraph[1];

    SCInputRef Input_Length1 = sc.Input[3];
    SCInputRef Input_Length2 = sc.Input[4];
    SCInputRef Input_Length3 = sc.Input[5];

    if(sc.SetDefaults)
    {
        sc.GraphName = "Ultimate Oscillator";
    }

```

```

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

Subgraph_UltimateOscillatorOut1.Name = "UO";
Subgraph_UltimateOscillatorOut1.DrawZeros = false;
Subgraph_UltimateOscillatorOut1.DrawStyle = DRAWSTYLE_LINE;
Subgraph_UltimateOscillatorOut1.PrimaryColor = RGB(0,255,0);

Input_Length1.Name = "Length 1";
Input_Length1.SetInt(7);
Input_Length1.SetIntLimits(1,MAX_STUDY_LENGTH);

Input_Length2.Name = "Length 2";
Input_Length2.SetInt(14);
Input_Length2.SetIntLimits(1,MAX_STUDY_LENGTH);

Input_Length3.Name = "Length 3";
Input_Length3.SetInt(28);
Input_Length3.SetIntLimits(1,MAX_STUDY_LENGTH);

sc.AutoLoop = 1;

return;
}

sc.DataStartIndex = max(Input_Length1.GetInt(), max(Input_Length2.GetInt(),Input_Length3.GetInt()));

sc.UltimateOscillator(sc.BaseData, Subgraph_UltimateOscillatorOut1, Subgraph_UltimateOscillatorOut2,
Input_Length1.GetInt(), Input_Length2.GetInt(), Input_Length3.GetInt());
}

/*=====*/
SCSFExport scsf_WilliamsAD(SCStudyInterfaceRef sc)
{
    SCSUBGRAPHREF Subgraph_AccumulationDistribution = sc.Subgraph[0];

    if(sc.SetDefaults)
    {
        sc.GraphName="Accumulation/Distribution (Williams)";

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

        Subgraph_AccumulationDistribution.Name = "AD";
        Subgraph_AccumulationDistribution.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_AccumulationDistribution.PrimaryColor = RGB(0,255,0);

        sc.AutoLoop = 1;

        return;
    }

    sc.DataStartIndex = 1;

    sc.WilliamsAD(sc.BaseDataIn, Subgraph_AccumulationDistribution, sc.Index);
}

```

```

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

    SCInputRef Input_Length = sc.Input[3];
    SCInputRef Input_Invert = sc.Input[4];
    SCInputRef Input_DataHigh = sc.Input[5];
    SCInputRef Input_DataLow = sc.Input[6];
    SCInputRef Input_DataLast = sc.Input[7];
    SCInputRef Input_UpdateFlag = sc.Input[8];

    if(sc.SetDefaults)
    {
        sc.GraphName = "Williams %R";

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

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

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

        Input_Invert.Name = "Invert Output";
        Input_Invert.SetYesNo(1);

        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 = 1;

        return;
    }

    if (Input_UpdateFlag.GetInt() == 0)
    {
        Input_DataHigh.SetInputDataIndex(SC_HIGH);
        Input_DataLow.SetInputDataIndex(SC_LOW);
        Input_DataLast.SetInputDataIndex(SC_LAST);
        Input_UpdateFlag.SetInt(1);
    }

    sc.DataStartIndex = Input_Length.GetInt();

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

```

```

    sc.BaseData[Input_DataLow.GetInputDataIndex()],
    sc.BaseData[Input_DataLast.GetInputDataIndex()],
    Subgraph_PercentR,
    sc.Index,
    Input_Length.GetInt()
);

if (Input_Invert.GetYesNo())
    Subgraph_PercentR[sc.Index] *= -1;
}

/*=====*/
SCSFExport scsf_IslandReversal(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_IslandReversal = sc.Subgraph[0];
    SCSubgraphRef Subgraph_GapUp = sc.Subgraph[1];
    SCSubgraphRef Subgraph_GapDown = sc.Subgraph[2];
    SCSubgraphRef Subgraph_GapUpIndex = sc.Subgraph[3];
    SCSubgraphRef Subgraph_GapDownIndex = sc.Subgraph[4];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Island Reversal";

        sc.GraphRegion = 0;
        sc.ValueFormat = 0;

        Subgraph_IslandReversal.Name = "IR";
        Subgraph_IslandReversal.DrawStyle = DRAWSTYLE_POINT;
        Subgraph_IslandReversal.PrimaryColor = RGB(0, 255, 0);
        Subgraph_IslandReversal.LineWidth = 8;
        Subgraph_IslandReversal.DrawZeros = false;

        Subgraph_GapUp.Name = "Gap Up";
        Subgraph_GapUp.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_GapDown.Name = "Gap Down";
        Subgraph_GapDown.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_GapUpIndex.Name = "Gap Up Index";
        Subgraph_GapUpIndex.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_GapDownIndex.Name = "Gap Down Index";
        Subgraph_GapDownIndex.DrawStyle = DRAWSTYLE_IGNORE;

        sc.AutoLoop = 1;

        return;
    }

    //Detect gaps in consecutive bars for sc.Index > 0
    if (sc.Index > 0)
    {
        //Gap Up
        if (sc.Low[sc.Index] > sc.High[sc.Index - 1])
        {
            Subgraph_GapUp[sc.Index] = 1;
            Subgraph_GapDown[sc.Index] = Subgraph_GapDown[sc.Index - 1];
            Subgraph_GapUpIndex[sc.Index] = static_cast<float>(sc.Index);
            Subgraph_GapDownIndex[sc.Index] = Subgraph_GapDownIndex[sc.Index - 1];
        }
    }
}

```



```

//Gap Down
else if (sc.High[sc.Index] < sc.Low[sc.Index - 1])
{
    Subgraph_GapUp[sc.Index] = Subgraph_GapUp[sc.Index - 1];
    Subgraph_GapDown[sc.Index] = 1;
    Subgraph_GapUpIndex[sc.Index] = Subgraph_GapUpIndex[sc.Index - 1];
    Subgraph_GapDownIndex[sc.Index] = static_cast<float>(sc.Index);
}

//No Gap (Overlapping Bars)
else
{
    Subgraph_GapUp[sc.Index] = Subgraph_GapUp[sc.Index - 1];
    Subgraph_GapDown[sc.Index] = Subgraph_GapDown[sc.Index - 1];
    Subgraph_GapUpIndex[sc.Index] = Subgraph_GapUpIndex[sc.Index - 1];
    Subgraph_GapDownIndex[sc.Index] = Subgraph_GapDownIndex[sc.Index - 1];
}
}

//Detect Island Reversal
if (Subgraph_GapUp[sc.Index] == 1 && Subgraph_GapDown[sc.Index] == 1)
{
    //Gap Up occurs before Gap Down
    if (Subgraph_GapUpIndex[sc.Index] <= Subgraph_GapDownIndex[sc.Index] - 1)
    {
        float GapLevel = max(sc.High[static_cast<int>(Subgraph_GapUpIndex[sc.Index]) - 1], sc.High[static_cast<int>(Subgraph_GapDownIndex[sc.Index])]);
        int AreBarsAboveGapLevel = 1;

        //Check each bar between Up Gap and Down Gap to make sure it does not go below the Gap Up Level
        for (int BarIndex = static_cast<int>(Subgraph_GapUpIndex[sc.Index]); BarIndex < static_cast<int>(Subgraph_GapDownIndex[sc.Index]); BarIndex++)
        {
            if (sc.Low[BarIndex] < GapLevel)
                AreBarsAboveGapLevel = 0;
        }

        if (AreBarsAboveGapLevel == 1)
        {
            Subgraph_IslandReversal[static_cast<int>(Subgraph_GapUpIndex[sc.Index])] = GapLevel;
            Subgraph_IslandReversal[static_cast<int>(Subgraph_GapDownIndex[sc.Index]) - 1] = GapLevel;
        }

        //Disallow single-bar islands
        if (Subgraph_GapUpIndex[sc.Index] == Subgraph_GapDownIndex[sc.Index] - 1)
            Subgraph_IslandReversal[static_cast<int>(Subgraph_GapUpIndex[sc.Index])] = 0;

        //Reset these variables to check for next pair of Gaps
        Subgraph_GapUp[sc.Index] = 0;
        Subgraph_GapUpIndex[sc.Index] = 0;
    }

    //Gap Down occurs before Gap Up
    else if (Subgraph_GapDownIndex[sc.Index] <= Subgraph_GapUpIndex[sc.Index] - 1)
    {
        float GapLevel = min(sc.Low[static_cast<int>(Subgraph_GapDownIndex[sc.Index]) - 1], sc.Low[static_cast<int>(Subgraph_GapUpIndex[sc.Index])]);
        int AreBarsBelowGapLevel = 1;

        //Check each bar between Down Gap and Up Gap to make sure it does not go above the Gap Down Level
        for (int BarIndex = static_cast<int>(Subgraph_GapDownIndex[sc.Index]); BarIndex < static_cast<int>(Subgraph_GapUpIndex[sc.Index]); BarIndex++)
        {
            if (sc.High[BarIndex] > GapLevel)

```

```

        AreBarsBelowGapLevel = 0;
    }

    if (AreBarsBelowGapLevel == 1)
    {
        Subgraph_IslandReversal[static_cast<int>(Subgraph_GapUpIndex[sc.Index]) - 1] = GapLevel;
        Subgraph_IslandReversal[static_cast<int>(Subgraph_GapDownIndex[sc.Index])] = GapLevel;
    }

    //Disallow single-bar islands
    if (Subgraph_GapDownIndex[sc.Index] == Subgraph_GapUpIndex[sc.Index] - 1)
        Subgraph_IslandReversal[static_cast<int>(Subgraph_GapDownIndex[sc.Index])] = 0;

    //Reset these variables to check for next pair of Gaps
    Subgraph_GapDown[sc.Index] = 0;
    Subgraph_GapDownIndex[sc.Index] = 0;
}
}
}
}
/*=====*/
SCSFExport scsf_WeightedAverageOscillator(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Oscillator = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Temp1 = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Temp2 = sc.Subgraph[2];

    SCInputRef Input_Data = sc.Input[0];
    SCInputRef Input_SlowAvgLength = sc.Input[3];
    SCInputRef Input_FastAvgLength = sc.Input[4];

    if(sc.SetDefaults)
    {
        sc.GraphName = "Weighted Average Oscillator";

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

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

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

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

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

        sc.AutoLoop = 1;

        return;
    }
    sc.DataStartIndex = max( Input_SlowAvgLength.GetInt(),Input_FastAvgLength.GetInt());

    sc.WeightedMovingAverage(sc.BaseDataIn[static_cast<int>(Input_Data.GetInputDataIndex())], Subgraph_Temp1,

```

```

Input_SlowAvgLength.GetInt());
    sc.WeightedMovingAverage(sc.BaseDataIn[static_cast<int>(Input_Data.GetInputDataIndex())], Subgraph_Temp2,
Input_FastAvgLength.GetInt());

    sc.Oscillator(Subgraph_Temp2, Subgraph_Temp1, Subgraph_Oscillator);
}

/*=====*/
SCSFExport scsf_PreferredStochasticDiNapoli(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_FastKLength = sc.Input[3];
    SCInputRef Input_FastDMovAvgLength = sc.Input[4];
    SCInputRef Input_ModifiedMovAvgLength = sc.Input[5];
    SCInputRef Input_Line1Value = sc.Input[6];
    SCInputRef Input_Line2Value = sc.Input[7];

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

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

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

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

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

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

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

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

        Input_ModifiedMovAvgLength.Name = "Modified Mov Avg Length (Slow %D)";
        Input_ModifiedMovAvgLength.SetInt(3);
        Input_ModifiedMovAvgLength.SetIntLimits(1,MAX_STUDY_LENGTH);

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

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

```

```

sc.AutoLoop = 1;

return;
}

int Index = sc.Index;
sc.DataStartIndex = Input_FastKLength.GetInt() + Input_FastDMovAvgLength.GetInt() + 1;

sc.Stochastic(sc.BaseDataIn, sc.Subgraph[4], Input_FastKLength.GetInt(), 3, 3, MOVAVGTYPE_SIMPLE);

sc.ExponentialMovAvg(sc.Subgraph[4], Subgraph_PercentK, Index, Input_FastDMovAvgLength.GetInt());

Subgraph_PercentD[Index] = Subgraph_PercentD[Index - 1] +
    ((Subgraph_PercentK[Index] - Subgraph_PercentD[Index - 1]) / Input_ModifiedMovAvgLength.GetInt());

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

/*=====*/
SCSFExport scsf_ElderRay(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_BullPower = sc.Subgraph[0];
    SCSubgraphRef Subgraph_BearPower = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Zero = sc.Subgraph[2];
    SCSubgraphRef Subgraph_MovAvgTemp = sc.Subgraph[3];

    SCInputRef Input_MovAvgLength = sc.Input[3];

    if(sc.SetDefaults)
    {
        sc.GraphName = "Elder Ray";

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

        Subgraph_BullPower.Name = "Bull Power";
        Subgraph_BullPower.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_BullPower.PrimaryColor = RGB(0,255,0);
        Subgraph_BullPower.DrawZeros = true;

        Subgraph_BearPower.Name = "Bear Power";
        Subgraph_BearPower.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_BearPower.PrimaryColor = RGB(255,0,255);
        Subgraph_BearPower.DrawZeros = true;

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

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

        sc.AutoLoop = 1;
    }
}

```

```

    return;
}

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

int i = sc.Index;
sc.ExponentialMovAvg(sc.Close, Subgraph_MovAvgTemp, i, Input_MovAvgLength.GetInt());

Subgraph_BullPower[i] = sc.High[i] - Subgraph_MovAvgTemp[i];
Subgraph_BearPower[i] = sc.Low[i] - Subgraph_MovAvgTemp[i];
}

/*=====*/
SCSFExport scsf_MomentumWithMovingAverage(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Momentum = sc.Subgraph[0];
    SCSubgraphRef Subgraph_MovingAverage = sc.Subgraph[1];
    SCSubgraphRef Subgraph_CenterLine = sc.Subgraph[2];

    SCInputRef Input_Data = sc.Input[0];
    SCInputRef Input_MomentumLength = sc.Input[3];
    SCInputRef Input_MovAvgLength = sc.Input[4];
    SCInputRef Input_MomentumType = sc.Input[5];

    if(sc.SetDefaults)
    {
        sc.GraphName = "Momentum with Moving Average";

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

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

        Subgraph_MovingAverage.Name = "Moving Average";
        Subgraph_MovingAverage.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_MovingAverage.PrimaryColor = RGB(0,0,255);
        Subgraph_MovingAverage.DrawZeros = false;

        Subgraph_CenterLine.Name = "Line";
        Subgraph_CenterLine.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_CenterLine.PrimaryColor = RGB(128, 128, 128);
        Subgraph_CenterLine.DrawZeros = true;

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

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

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

        Input_MomentumType.Name = "Momentum Type";
        Input_MomentumType.SetCustomInputStrings("Difference;Quotient");
        Input_MomentumType.SetCustomInputIndex(0);
    }
}

```

```

    sc.AutoLoop = 1;

    return;
}

sc.DataStartIndex = Input_MomentumLength.GetInt() + Input_MovAvgLength.GetInt();

if(sc.Index < Input_MomentumLength.GetInt())
    return;

const int SelectedIndex = Input_MomentumType.GetIndex();
switch (SelectedIndex)
{
    case 0:
    {
        Subgraph_Momentum[sc.Index] = sc.BaseDataIn[Input_Data.GetInputDataIndex()][sc.Index] -
sc.BaseDataIn[Input_Data.GetInputDataIndex()][sc.Index - Input_MomentumLength.GetInt()];

        Subgraph_CenterLine[sc.Index] = 0.0;
    }
    break;

    case 1:
    {
        if (sc.BaseDataIn[Input_Data.GetInputDataIndex()][sc.Index - Input_MomentumLength.GetInt()] != 0.0f)
            Subgraph_Momentum[sc.Index] = (sc.BaseDataIn[Input_Data.GetInputDataIndex()][sc.Index] /
sc.BaseDataIn[Input_Data.GetInputDataIndex()][sc.Index - Input_MomentumLength.GetInt()]) * 100;
        else
            Subgraph_Momentum[sc.Index] = 0.0f;

        Subgraph_CenterLine[sc.Index] = 100.0;
    }
    break;
}

sc.SimpleMovAvg(Subgraph_Momentum, Subgraph_MovingAverage, sc.Index, Input_MovAvgLength.GetInt());
}

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

    SCSubgraphRef Subgraph_Temp1 = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Temp2 = sc.Subgraph[2];

    SCInputRef Input_Length = sc.Input[3];

    if(sc.SetDefaults)
    {
        sc.GraphName = "Volatility - Chaikin's";

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

        Subgraph_Volatility.Name = "Volatility";
        Subgraph_Volatility.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Volatility.PrimaryColor = RGB(0,255,0);

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

        sc.AutoLoop = 1;
    }
}

```

```

    return;
}

sc.DataStartIndex = Input_Length.GetInt() + 4;
int i = sc.Index;

Subgraph_Temp2[i] = sc.High[i] - sc.Low[i];

sc.ExponentialMovAvg(Subgraph_Temp2, Subgraph_Temp1, i, Input_Length.GetInt());

Subgraph_Volatility[i] = ((Subgraph_Temp1[i] - Subgraph_Temp1[i - Input_Length.GetInt()])
    / Subgraph_Temp1[i-Input_Length.GetInt()]) * 100;
}

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

    SCInputRef Input_Data = sc.Input[0];
    SCInputRef Input_Length = sc.Input[3];

    if(sc.SetDefaults)
    {
        sc.GraphName = "Rate Of Change - Points";

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

        Subgraph_ROC.Name = "ROC";
        Subgraph_ROC.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_ROC.PrimaryColor = RGB(0,255,0);

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

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

        sc.AutoLoop=1;

        return;
    }
    sc.DataStartIndex = Input_Length.GetInt();

    Subgraph_ROC[sc.Index] = (sc.BaseDataIn[static_cast<int>(Input_Data.GetInputDataIndex())][sc.Index] -
sc.BaseDataIn[static_cast<int>(Input_Data.GetInputDataIndex())][sc.Index - Input_Length.GetInt()]);
}

/*=====*/
SCSFExport scsf_SquareOfNine(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Anchor = sc.Subgraph[0];
    SCSubgraphRef Subgraph_L1Plus = sc.Subgraph[1];
    SCSubgraphRef Subgraph_L1Minus = sc.Subgraph[2];
    SCSubgraphRef Subgraph_L1 = sc.Subgraph[3];

```

```

SCFloatArrayRef Array_BaseUpper = sc.Subgraph[0].Arrays[0];
SCFloatArrayRef Array_BaseLower = sc.Subgraph[0].Arrays[1];
SCFloatArrayRef Array_Offset = sc.Subgraph[0].Arrays[2];

SCInputRef Input_NumberOfDaysToCalculate = sc.Input[0];
SCInputRef Input_AnchorValue = sc.Input[2];
SCInputRef Input_Degree = sc.Input[3];
SCInputRef Input_Level = sc.Input[4];
SCInputRef Input_AutoAdjust = sc.Input[5];
SCInputRef Input_InMultiplier = sc.Input[6];

if(sc.SetDefaults)
{
    sc.GraphName = "Square of 9";

    sc.GraphRegion = 0;

    const int LineLabelFlags = (LL_DISPLAY_VALUE
        | LL_DISPLAY_NAME
        | LL_NAME_ALIGN_FAR_RIGHT
        | LL_VALUE_ALIGN_FAR_RIGHT
        | LL_NAME_ALIGN_ABOVE
        | LL_VALUE_ALIGN_BELOW
    );

    Subgraph_Anchor.Name = "Anchor";
    Subgraph_Anchor.DrawStyle = DRAWSTYLE_DASH;// Set line style to Dash
    Subgraph_Anchor.PrimaryColor = RGB(0,255,255);
    Subgraph_Anchor.DrawZeros = false;
    Subgraph_Anchor.LineLabel = LineLabelFlags;

    Subgraph_L1Plus.Name = "L1+";
    Subgraph_L1Plus.DrawStyle = DRAWSTYLE_DASH;
    Subgraph_L1Plus.PrimaryColor = RGB(0,255,0);
    Subgraph_L1Plus.DrawZeros = false;
    Subgraph_L1Plus.LineLabel = LineLabelFlags;

    Subgraph_L1Minus.Name = "L1-";
    Subgraph_L1Minus.DrawStyle = DRAWSTYLE_DASH;
    Subgraph_L1Minus.PrimaryColor = RGB(255,0,0);
    Subgraph_L1Minus.DrawZeros = false;
    Subgraph_L1Minus.LineLabel = LineLabelFlags;

    Subgraph_L1.Name = "L1";
    Subgraph_L1.DrawStyle = DRAWSTYLE_DASH;
    Subgraph_L1.PrimaryColor = RGB(255,127,0);
    Subgraph_L1.DrawZeros = false;
    Subgraph_L1.LineLabel = LineLabelFlags;

    Input_NumberOfDaysToCalculate.Name = "Number of Days to Calculate";
    Input_NumberOfDaysToCalculate.SetInt(10);
    Input_NumberOfDaysToCalculate.SetIntLimits(1, MAX_STUDY_LENGTH);

    Input_AnchorValue.SetFloat(0);
    Input_AnchorValue.Name = "Anchor Value";
    Input_AnchorValue.SetFloatLimits(0, static_cast<float>(MAX_STUDY_LENGTH));

    Input_Degree.SetFloat(22.5);
    Input_Degree.Name = "Degrees";
    Input_Degree.SetFloatLimits(0,360);

    Input_Level.SetFloat(1);
    Input_Level.Name = "Level";
    Input_Level.SetFloatLimits(1,10);

```



```

Input_AutoAdjust.SetYesNo(false);
Input_AutoAdjust.Name = "Auto Adjust";

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

sc.ScaleRangeType = SCALE_SAMEASREGION;

sc.AutoLoop=1;

return;
}

float MultiplierValue = Input_InMultiplier.GetFloat();
if (MultiplierValue == 0)
    MultiplierValue = 1;

if (Input_AnchorValue.GetFloat() == 0)
{
    Input_AnchorValue.SetFloat(sc.Close[sc.ArraySize - 1]*MultiplierValue);
}

int LevelInput = static_cast<int>(Input_Level.GetFloat() + 0.5f);

if (sc.BaseDateTimeln[sc.Index].GetDate() < sc.BaseDateTimeln[sc.ArraySize - 1].GetDate() -
Input_NumberOfDaysToCalculate.GetInt() + 1)
    return;

if (Input_AutoAdjust.GetYesNo())
{
    float SquareRoot = sqrt(Input_AnchorValue.GetFloat());

    float Offset = 0.0f;
    float Mid = 0.0f;
    float BaseUpper = 0.0f;
    float BaseLower = 0.0f;
    float Upper = 0.0f;
    float Lower = 0.0f;

    if(sc.Index == 0)
    {
        Offset = 0;
        BaseUpper = (SquareRoot + Input_Degree.GetFloat()/180) * (SquareRoot + Input_Degree.GetFloat()/180);
        BaseLower = (SquareRoot - Input_Degree.GetFloat()/180) * (SquareRoot - Input_Degree.GetFloat()/180);
        Mid = Input_AnchorValue.GetFloat();
        Upper = (SquareRoot + LevelInput * Input_Degree.GetFloat()/180) * (SquareRoot + LevelInput *
Input_Degree.GetFloat()/180);
        Lower = (SquareRoot - LevelInput * Input_Degree.GetFloat()/180) * (SquareRoot - LevelInput *
Input_Degree.GetFloat()/180);
    }
    else
    {
        Upper = Subgraph_L1Plus[sc.Index-1] * MultiplierValue;
        Lower = Subgraph_L1Minus[sc.Index-1] * MultiplierValue;
        Mid = Subgraph_L1[sc.Index-1] * MultiplierValue;

        BaseUpper = Array_BaseUpper[sc.Index-1];
        BaseLower = Array_BaseLower[sc.Index-1];
        Offset = Array_Offset[sc.Index-1];
    }
}

```

```

}

SCString TempStr;
TempStr.Format("+%g deg", LevelInput * Input_Degree.GetFloat());

Subgraph_L1Plus.Name = TempStr;

TempStr.Format("-%g deg", LevelInput * Input_Degree.GetFloat());
Subgraph_L1Minus.Name = TempStr;

Subgraph_L1.Name = "Mid";

float PriceInput = sc.Close[sc.Index] * MultiplierValue;

while (PriceInput > BaseUpper)
{
    Offset += Input_Degree.GetFloat();
    BaseUpper = (SquareRoot + (Offset + Input_Degree.GetFloat())/180) * (SquareRoot + (Offset +
Input_Degree.GetFloat())/180);
    BaseLower = (SquareRoot + (Offset - Input_Degree.GetFloat())/180) * (SquareRoot + (Offset -
Input_Degree.GetFloat())/180);
    Upper = (SquareRoot + (Offset + LevelInput * Input_Degree.GetFloat())/180) * (SquareRoot + (Offset +
LevelInput * Input_Degree.GetFloat())/180);
    Lower = (SquareRoot + (Offset - LevelInput * Input_Degree.GetFloat())/180) * (SquareRoot + (Offset - LevelInput
* Input_Degree.GetFloat())/180);
    Mid = (SquareRoot + Offset/180) * (SquareRoot + Offset/180);
}
while (PriceInput < BaseLower)
{
    Offset -= Input_Degree.GetFloat();
    BaseUpper = (SquareRoot + (Offset + Input_Degree.GetFloat())/180) * (SquareRoot + (Offset +
Input_Degree.GetFloat())/180);
    BaseLower = (SquareRoot + (Offset - Input_Degree.GetFloat())/180) * (SquareRoot + (Offset -
Input_Degree.GetFloat())/180);
    Upper = (SquareRoot + (Offset + LevelInput * Input_Degree.GetFloat())/180) * (SquareRoot + (Offset +
LevelInput * Input_Degree.GetFloat())/180);
    Lower = (SquareRoot + (Offset - LevelInput * Input_Degree.GetFloat())/180) * (SquareRoot + (Offset - LevelInput
* Input_Degree.GetFloat())/180);

    Mid = (SquareRoot + Offset/180) * (SquareRoot + Offset/180);
}

Subgraph_Anchor[sc.Index] = Input_AnchorValue.GetFloat() / MultiplierValue;
Subgraph_L1Plus[sc.Index] = Upper / MultiplierValue;
Subgraph_L1Minus[sc.Index] = Lower / MultiplierValue;
Subgraph_L1[sc.Index] = Mid / MultiplierValue;
Array_BaseUpper[sc.Index] = BaseUpper;
Array_BaseLower[sc.Index] = BaseLower;
Array_Offset[sc.Index] = Offset;
}
else
{
    float L1 = 0.0f;
    float L2 = 0.0f;
    SCString PositiveDegrees;
    SCString NegativeDegrees;

    float squareRoot = sqrt(Input_AnchorValue.GetFloat());
    float localDegree = Input_Degree.GetFloat() * LevelInput;

    PositiveDegrees.Format("+%g deg", localDegree);
    NegativeDegrees.Format("-%g deg", localDegree);

```

```

L1 = (squareRoot + localDegree/180) * (squareRoot + localDegree/180);
L2 = (squareRoot - localDegree/180) * (squareRoot - localDegree/180);

Subgraph_Anchor[sc.Index] = Input_AnchorValue.GetFloat() / MultiplierValue;
Subgraph_L1Plus[sc.Index] = L1 / MultiplierValue;
Subgraph_L1Minus[sc.Index] = L2 / MultiplierValue;
}
}

/*=====*/
SCSFExport scsf_LinearRegressionEndChannel(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_LinearRegression = sc.Subgraph[0];
    SCSubgraphRef Subgraph_StdDevHigh = sc.Subgraph[1];
    SCSubgraphRef Subgraph_StdDevLow = sc.Subgraph[2];

    SCInputRef Input_Data = sc.Input[0];
    SCInputRef Input_PeriodType = sc.Input[1];
    SCInputRef Input_NumberOfBarsToCalculate = sc.Input[2];
    SCInputRef Input_NumberDeviations = sc.Input[3];
    SCInputRef Input_UseStartDateTime = sc.Input[4];
    SCInputRef Input_StartDateTime = sc.Input[5];
    SCInputRef Input_CalculateOnBarClose = sc.Input[6];
    SCInputRef Input_Version = sc.Input[7];

    enum PeriodTypeEnum
    { PERIOD_TYPE_NUMBER_OF_BARS = 0
    , PERIOD_TYPE_FROM_START_DATE_TIME = 1
    , PERIOD_TYPE_FROM_START_TIME = 2
    };

    static const int MAX_BARS_TO_CALCULATE = 10000;

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

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

        Subgraph_LinearRegression.Name = "Linear Regression";
        Subgraph_LinearRegression.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_LinearRegression.PrimaryColor = RGB(0,255,0);
        Subgraph_LinearRegression.DrawZeros = false;

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

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

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

        Input_PeriodType.Name = "Period Type";
        Input_PeriodType.SetCustomInputStrings("Number of Bars;From Start Date-Time;From Start Time");
        Input_PeriodType.SetCustomInputIndex(0);
    }
}

```

```

Input_NumberOfBarsToCalculate.Name = "Number of Bars";
Input_NumberOfBarsToCalculate.SetInt(10);
Input_NumberOfBarsToCalculate.SetIntLimits(1, MAX_BARS_TO_CALCULATE);

Input_NumberDeviations.Name = "Number of Deviations";
Input_NumberDeviations.SetFloat(2.0f);
Input_NumberDeviations.SetFloatLimits(0.001f, 2000);

//UseStartDateTime.Name = "Use Start Date-Time instead of Number of Bars";
Input_UseStartDateTime.SetYesNo(false);

Input_StartDateTime.Name = "Start Date and Time";
Input_StartDateTime.SetDateTime(0);

Input_CalculateOnBarClose.Name = "Calculate On Bar Close";
Input_CalculateOnBarClose.SetYesNo(false);

Input_Version.SetInt(1);

return;
}

if (Input_Version.GetInt() < 1)
{
    if (Input_UseStartDateTime.GetYesNo())
        Input_PeriodType.SetCustomInputIndex(1);

    Input_Version.SetInt(1);
}

int NumberOfBarsToCalculate = Input_NumberOfBarsToCalculate.GetInt();
int InputDataOffset = 0;

if (Input_CalculateOnBarClose.GetYesNo())
    InputDataOffset = 1;

const int BaseDataArraySize = sc.ArraySize;
// Formula is  $y = a + bx$ 
//  $a = (\text{sum}Y - (b)\text{sum}X) / n$ 
//  $b = ((n)\text{sum}XY - \text{sum}X * \text{sum}Y) / ((n)\text{sum}XX - (\text{sum}Y)(\text{sum}Y))$ 
float sumY = 0;
float sumX = 0;
float sumXY = 0;
float sumXX = 0;
float StandardDev = 0;
float tempY = 0, a = 0, b = 0;
int NumberOfBars = 0;

int StartIndex = 0;

switch (Input_PeriodType.GetIndex())
{
    case PERIOD_TYPE_NUMBER_OF_BARS:
        StartIndex = max(0, BaseDataArraySize - NumberOfBarsToCalculate - InputDataOffset);
        break;

    case PERIOD_TYPE_FROM_START_DATE_TIME:
    {
        NumberOfBarsToCalculate = MAX_BARS_TO_CALCULATE;

        StartIndex = sc.GetContainingIndexForSCDateTime(sc.ChartNumber,
            Input_StartDateTime.GetDateTime().GetAsDouble());

        if (StartIndex < BaseDataArraySize - MAX_BARS_TO_CALCULATE)
            StartIndex = BaseDataArraySize - MAX_BARS_TO_CALCULATE;
    }
}

```

```

        else
            NumberOfBarsToCalculate = BaseDataArraySize - StartIndex;
    }
    break;

    case PERIOD_TYPE_FROM_START_TIME:
    {
        const int LastBarDate = sc.BaseDateTimeln[sc.ArraySize - 1].GetDate();
        const int StartTime = Input_StartDateTime.GetTime();

        const SCDateTime CurrentDayStartDateTime(LastBarDate, StartTime);

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

        NumberOfBarsToCalculate = BaseDataArraySize - StartIndex;
    }
    break;
};

for (int Index = StartIndex
    ; Index < StartIndex + NumberOfBarsToCalculate && Index < BaseDataArraySize
    ; Index++)
{
    tempY = sc.BaseDataIn[Input_Data.GetInputDataIndex()][Index];
    NumberOfBars++;
    sumY += tempY;
    sumX += NumberOfBars;
    sumXY += (tempY*NumberOfBars);
    sumXX += NumberOfBars*NumberOfBars;
}

b = (NumberOfBars*sumXY - (sumX*sumY))/((NumberOfBars*sumXX) - (sumX*sumX));
a = (sumY - (b*sumX))/NumberOfBars;

// Solve StandardDev sqrt( sum((Xi-Xbar)^2)/n )
float tempX = 0;
for (int Index = StartIndex; Index < StartIndex + NumberOfBarsToCalculate && Index < BaseDataArraySize; Index++)
{
    float diff = sc.BaseDataIn[Input_Data.GetInputDataIndex()][Index] - (a + b * (Index-StartIndex+1));
    tempX += (diff * diff);
}

StandardDev = sqrt(tempX/NumberOfBars);

// Fill in the Linear Regression data

sc.EarliestUpdateSubgraphDataArrayIndex = StartIndex;

for(int BarIndex = 0; BarIndex < NumberOfBars; BarIndex++)
{
    Subgraph_LinearRegression[StartIndex + BarIndex] = a + b * (BarIndex + 1);

    Subgraph_StdDevHigh[StartIndex + BarIndex] = Subgraph_LinearRegression[StartIndex + BarIndex] +
(StandardDev * Input_NumberDeviations.GetFloat());

    Subgraph_StdDevLow[StartIndex + BarIndex] = Subgraph_LinearRegression[StartIndex + BarIndex] - (StandardDev
* Input_NumberDeviations.GetFloat());
}

if (!Input_UseStartDateTime.GetYesNo())
{
    // Zero out points that are not within the scope
    int NumNewElements = sc.Open.GetArraySize() - 1 - sc.UpdateStartIndex;

    int BarIndex = BaseDataArraySize - 1 - NumberOfBarsToCalculate - InputDataOffset;

```

```

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

    for (
        ; BarIndex >= 0 && NumNewElements > 0
        ; BarIndex--)
    {
        Subgraph_LinearRegression[BarIndex] = 0;
        Subgraph_StdDevHigh[BarIndex] = 0;
        Subgraph_StdDevLow[BarIndex] = 0;
        NumNewElements--;
    }
}

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

    if(sc.SetDefaults)
    {
        sc.GraphName = "Price Volume Trend";
        sc.GraphRegion = 1;

        Subgraph_PVT.Name = "PVT";
        Subgraph_PVT.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_PVT.PrimaryColor = RGB(0,255,0);

        sc.AutoLoop = 1;

        return;
    }

    sc.DataStartIndex = 1;
    sc.PriceVolumeTrend(sc.BaseDataIn, Subgraph_PVT);
}

/*=====*/
SCSFExport scsf_StandardDeviationBands(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_TopBand = sc.Subgraph[0];
    SCSubgraphRef Subgraph_BottomBand = sc.Subgraph[1];
    SCSubgraphRef Subgraph_TopMovAvg = sc.Subgraph[2];
    SCSubgraphRef Subgraph_BottomMovAvg = sc.Subgraph[3];
    SCSubgraphRef Subgraph_Temp4 = sc.Subgraph[4];
    SCSubgraphRef Subgraph_Temp5 = sc.Subgraph[5];

    SCInputRef Input_TopBandInputData = sc.Input[0];
    SCInputRef Input_BottomBandInputData = sc.Input[1];
    SCInputRef Input_Length = sc.Input[3];
    SCInputRef Input_MultFactor = sc.Input[5];
    SCInputRef Input_MovAvgType = sc.Input[6];

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

        sc.GraphRegion = 0;

        Subgraph_TopBand.Name = "Top";
        Subgraph_TopBand.DrawStyle = DRAWSTYLE_LINE;

```

```

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

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

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

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

Input_TopBandInputData.Name = "Top Band Input Data";
Input_TopBandInputData.SetInputDataIndex(SC_HIGH);

Input_BottomBandInputData.Name = "Bottom Band Input Data";
Input_BottomBandInputData.SetInputDataIndex(SC_LOW);

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

Input_MultFactor.Name = "Multiplication Factor";
Input_MultFactor.SetFloat(1.8f);
Input_MultFactor.SetFloatLimits(0.1f, 20.0f);

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

sc.AutoLoop = false;

return;
}

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

int CalculationStartIndex = sc.GetCalculationStartIndexForStudy();
if (CalculationStartIndex > sc.UpdateStartIndex)
    CalculationStartIndex = sc.UpdateStartIndex;

sc.EarliestUpdateSubgraphDataArrayIndex = CalculationStartIndex;

for (int BarIndex = CalculationStartIndex; BarIndex < sc.ArraySize; BarIndex++)
{
    SCFloatArrayRef HighArray = sc.BaseDataIn[Input_TopBandInputData.GetInputDataIndex()];
    SCFloatArrayRef LowArray = sc.BaseDataIn[Input_BottomBandInputData.GetInputDataIndex()];

    sc.MovingAverage(HighArray, Subgraph_TopMovAvg, Input_MovAvgType.GetMovAvgType(), BarIndex,
Input_Length.GetInt());
    sc.MovingAverage(LowArray, Subgraph_BottomMovAvg, Input_MovAvgType.GetMovAvgType(), BarIndex,
Input_Length.GetInt());

    sc.StdDeviation(HighArray, Subgraph_Temp4, BarIndex, Input_Length.GetInt());
    sc.StdDeviation(LowArray, Subgraph_Temp5, BarIndex, Input_Length.GetInt());

    Subgraph_TopBand[BarIndex] = (Subgraph_Temp4[BarIndex] * Input_MultFactor.GetFloat()) +
Subgraph_TopMovAvg[BarIndex];
    Subgraph_BottomBand[BarIndex] = (Subgraph_Temp5[BarIndex] * -Input_MultFactor.GetFloat()) +
Subgraph_BottomMovAvg[BarIndex];
}
}

/*=====*/

```

```

SCSFExport scsf_HighestLowestBars(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Highest = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Lowest = sc.Subgraph[1];

    SCInputRef Input_HighestLength = sc.Input[0];
    SCInputRef Input_LowestLength = sc.Input[1];
    SCInputRef Input_Data = sc.Input[2];

    if(sc.SetDefaults)
    {
        sc.GraphName = "NumberOfBarsSinceHighestValue / NumberOfBarsSinceLowestValue";
        sc.GraphRegion = 2;

        Subgraph_Highest.Name = "Highest";
        Subgraph_Highest.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Highest.PrimaryColor = RGB(0,255,0);

        Subgraph_Lowest.Name = "Lowest";
        Subgraph_Lowest.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Lowest.PrimaryColor = RGB(255,0,255);

        Input_HighestLength.Name = "HighestLength";
        Input_HighestLength.SetInt(10);
        Input_HighestLength.SetIntLimits(2,MAX_STUDY_LENGTH);

        Input_LowestLength.Name = "LowestLength";
        Input_LowestLength.SetInt(10);
        Input_LowestLength.SetIntLimits(2,MAX_STUDY_LENGTH);

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

        sc.AutoLoop = 1;

        return;
    }

    SCFloatArrayRef In = sc.BaseDataIn[Input_Data.GetInputDataIndex()];

    sc.DataStartIndex = max(Input_HighestLength.GetInt(), Input_LowestLength.GetInt()) - 1;

    Subgraph_Highest[sc.Index] = static_cast<float>(sc.NumberOfBarsSinceHighestValue(In, sc.Index,
    Input_HighestLength.GetInt()));
    Subgraph_Lowest[sc.Index] = static_cast<float>(sc.NumberOfBarsSinceLowestValue(In, sc.Index,
    Input_LowestLength.GetInt()));
}

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

    SCInputRef Input_CompareType = sc.Input[0];

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

        sc.GraphName = "Volume-Up";

        sc.StudyDescription = "The volume for up bars only.";
    }
}

```



```

sc.AutoLoop = 1;

sc.ValueFormat = 0;

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

Input_CompareType.Name = "Compare Close with: 0=PrevClose, 1=Open";
Input_CompareType.SetInt(0);
Input_CompareType.SetIntLimits(0,1);

    return;
}

// Do data processing

bool IsUpBar = false;
switch (Input_CompareType.GetInt())
{
case 0:
    if (sc.Index > 0)
        IsUpBar = sc.Close[sc.Index] > sc.Close[sc.Index - 1];
    break;

case 1:
    IsUpBar = sc.Close[sc.Index] > sc.Open[sc.Index];
    break;
}

if (IsUpBar)
    Subgraph_Volume[sc.Index] = sc.Volume[sc.Index];
else
    Subgraph_Volume[sc.Index] = 0;
}

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

    SCInputRef Input_CompareType = sc.Input[0];

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

        sc.GraphName = "Volume-Down";

        sc.StudyDescription = "The volume for down bars only.";

        sc.AutoLoop = 1;

        sc.ValueFormat = 0;

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

```

```

Input_CompareType.Name = "Compare Close with: 0=PrevClose, 1=Open";
Input_CompareType.SetInt(0);
Input_CompareType.SetIntLimits(0,1);

    return;
}

// Do data processing

bool IsDownBar = false;
switch (Input_CompareType.GetInt())
{
case 0:
    if (sc.Index > 0)
        IsDownBar = sc.Close[sc.Index] < sc.Close[sc.Index - 1];
    break;

case 1:
    IsDownBar = sc.Close[sc.Index] < sc.Open[sc.Index];
    break;
}

if (IsDownBar)
    Subgraph_Volume[sc.Index] = sc.Volume[sc.Index];
else
    Subgraph_Volume[sc.Index] = 0;
}

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

    if(sc.SetDefaults)
    {
        sc.GraphName = "Bar End Time";

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

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

        sc.AutoLoop = 1;
        sc.MaintainAdditionalChartDataArrays = 1;

        return;
    }

    Subgraph_BET[sc.Index] = static_cast<float>
(sc.BaseDataEndDateTime[sc.Index].GetTimeAsSCDateTimeMS().GetAsDouble());
}

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

    SCInputRef Input_Array1 = sc.Input[0];
    SCInputRef Input_Array2 = sc.Input[1];

```

```

SCInputRef Input_Length = sc.Input[2];

if(sc.SetDefaults)
{
    sc.GraphName = "Correlation Coefficient";

    sc.GraphRegion = 1;

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

    Input_Array1.Name = "Input Array 1";
    Input_Array1.SetStudySubgraphValues(0,0);

    Input_Array2.Name = "Input Array 2";
    Input_Array2.SetStudySubgraphValues(0,0);

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

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

    return;
}
sc.DataStartIndex = Input_Length.GetInt();
SCFloatArray Array1;
SCFloatArray Array2;

sc.GetStudyArrayUsingID(Input_Array1.GetStudyID(),Input_Array1.GetSubgraphIndex(),Array1);
if(Array1.GetArraySize() < sc.ArraySize)
    return;

sc.GetStudyArrayUsingID(Input_Array2.GetStudyID(),Input_Array2.GetSubgraphIndex(),Array2);
if(Array2.GetArraySize() < sc.ArraySize)
    return;

Subgraph_CorrelationCoefficient[sc.Index] = sc.GetCorrelationCoefficient(Array1, Array2, Input_Length.GetInt());
}

/*=====*/
SCSFExport scsf_SierraSqueeze(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_MomentumHist = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SqueezeDots = sc.Subgraph[1];
    SCSubgraphRef Subgraph_MomentumHistUpColors = sc.Subgraph[2];
    SCSubgraphRef Subgraph_MomentumHistDownColors = sc.Subgraph[3];
    SCSubgraphRef Subgraph_Temp4 = sc.Subgraph[4];
    //SCSubgraphRef Temp5 = sc.Subgraph[5];
    SCSubgraphRef Subgraph_SignalValues = sc.Subgraph[6];
    SCSubgraphRef Subgraph_Temp8 = sc.Subgraph[8];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_HistogramLenFirstData = sc.Input[1];
    SCInputRef Input_SqueezeLength = sc.Input[2];
    SCInputRef Input_NK = sc.Input[3];
    SCInputRef Input_NB = sc.Input[4];

```

```

SCInputRef Input_FirstMovAvgType = sc.Input[5];
SCInputRef Input_HistogramLenSecondData = sc.Input[6];
SCInputRef Input_SecondMovAvgType = sc.Input[7];
SCInputRef Input_VersionUpdate = sc.Input[8];

if (sc.SetDefaults)
{
    sc.GraphName = "Squeeze Indicator 2";
    sc.StudyDescription = "Developed by the user Tony C.";

    Subgraph_MomentumHist.Name = "Momentum HISTOGRAM";
    Subgraph_MomentumHist.DrawStyle = DRAWSTYLE_BAR;
    Subgraph_MomentumHist.PrimaryColor = RGB(0,255,0);
    Subgraph_MomentumHist.LineWidth = 5;
    Subgraph_MomentumHist.DrawZeros = true;

    Subgraph_SqueezeDots.Name = "Squeeze Dots";
    Subgraph_SqueezeDots.DrawStyle = DRAWSTYLE_POINT;
    Subgraph_SqueezeDots.PrimaryColor = RGB(255,0,255);
    Subgraph_SqueezeDots.LineWidth = 4;
    Subgraph_SqueezeDots.DrawZeros = true;

    Subgraph_MomentumHistUpColors.Name = "Momentum HISTOGRAM Up Colors";
    Subgraph_MomentumHistUpColors.DrawStyle = DRAWSTYLE_IGNORE;
    Subgraph_MomentumHistUpColors.SecondaryColorUsed = 1;
    Subgraph_MomentumHistUpColors.PrimaryColor = RGB(0, 0, 255);
    Subgraph_MomentumHistUpColors.SecondaryColor = RGB(0, 0, 130);
    Subgraph_MomentumHistUpColors.DrawZeros = true;

    Subgraph_MomentumHistDownColors.Name = "Momentum HISTOGRAM Down Colors";
    Subgraph_MomentumHistDownColors.DrawStyle = DRAWSTYLE_IGNORE;
    Subgraph_MomentumHistDownColors.SecondaryColorUsed = 1;
    Subgraph_MomentumHistDownColors.PrimaryColor = RGB(255, 0, 0);
    Subgraph_MomentumHistDownColors.SecondaryColor = RGB(130, 0, 0);
    Subgraph_MomentumHistDownColors.DrawZeros = true;

    Subgraph_SignalValues.Name = "Signal Values";
    Subgraph_SignalValues.DrawStyle = DRAWSTYLE_IGNORE;
    Subgraph_SignalValues.PrimaryColor = RGB(127,0,255);
    Subgraph_SignalValues.DrawZeros = true;

    Input_InputData.Name = "Input Data";
    Input_InputData.SetInputDataIndex(SC_LAST); // default data field

    Input_HistogramLenFirstData.Name = "Histogram Length First Data";
    Input_HistogramLenFirstData.SetInt(20);
    Input_HistogramLenFirstData.SetIntLimits(1,MAX_STUDY_LENGTH);

    Input_SqueezeLength.Name = "Squeeze Length";
    Input_SqueezeLength.SetFloat(20);

    Input_NK.Name = "NK.GetFloat()";
    Input_NK.SetFloat(1.5);

    Input_NB.Name = "NB.GetFloat()";
    Input_NB.SetFloat(2);

    Input_FirstMovAvgType.Name = "First MA Type";
    Input_FirstMovAvgType.SetMovAvgType(MOVAVGTYPE_EXPONENTIAL);

    Input_HistogramLenSecondData.Name = "Histogram Length Second Data";
    Input_HistogramLenSecondData.SetInt(20);
    Input_HistogramLenSecondData.SetIntLimits(1,MAX_STUDY_LENGTH);

    Input_SecondMovAvgType.Name = "Second MA Type";

```

```

Input_SecondMovAvgType.SetMovAvgType(MOVAVGTYPE_LINEARREGRESSION);

// hidden input for old versions support
Input_VersionUpdate.SetInt(1);

sc.AutoLoop = 1;

return;
}

// upgrading the default settings
if (Input_VersionUpdate.GetInt() != 1)
{
    Subgraph_MomentumHistUpColors.Name = "Momentum Histogram Up Colors";
    Subgraph_MomentumHistUpColors.DrawStyle = DRAWSTYLE_IGNORE;
    Subgraph_MomentumHistUpColors.SecondaryColorUsed = 1;
    Subgraph_MomentumHistUpColors.PrimaryColor = RGB(0, 0, 255);
    Subgraph_MomentumHistUpColors.SecondaryColor = RGB(0, 0, 130);

    Subgraph_MomentumHistDownColors.Name = "Momentum Histogram Down Colors";
    Subgraph_MomentumHistDownColors.DrawStyle = DRAWSTYLE_IGNORE;
    Subgraph_MomentumHistDownColors.SecondaryColorUsed = 1;
    Subgraph_MomentumHistDownColors.PrimaryColor = RGB(255, 0, 0);
    Subgraph_MomentumHistDownColors.SecondaryColor = RGB(130, 0, 0);

    Subgraph_SignalValues.Name = "Signal Values";
    Subgraph_SignalValues.DrawStyle = DRAWSTYLE_IGNORE;

    Input_VersionUpdate.SetInt(1);
}

// Inputs
int i = sc.Index;
const DWORD inside = RGB(255, 0, 0);
const DWORD outside = RGB(0, 255, 0);

Subgraph_MomentumHistUpColors[i] = 0;
Subgraph_MomentumHistDownColors[i] = 0;

// First output elements are not valid
sc.DataStartIndex = Input_HistogramLenSecondData.GetInt();

SCFloatArrayRef close = sc.Close;
sc.ExponentialMovAvg(close, Subgraph_MomentumHistUpColors, Input_HistogramLenSecondData.GetInt()); // Note:
EMA returns close when index is < HistogramLenSecondData.GetInt()
sc.MovingAverage(close, Subgraph_MomentumHistUpColors, Input_FirstMovAvgType.GetMovAvgType(),
Input_HistogramLenFirstData.GetInt());

float hlh = sc.GetHighest(sc.High, Input_HistogramLenSecondData.GetInt());
float llh = sc.GetLowest(sc.Low, Input_HistogramLenSecondData.GetInt());

SCFloatArrayRef price = sc.Open;

Subgraph_MomentumHistDownColors[sc.Index] = price[sc.Index] - ((hlh + llh)/2.0f +
Subgraph_MomentumHistUpColors[sc.Index])/2.0f;
sc.LinearRegressionIndicator(Subgraph_MomentumHistDownColors, Subgraph_MomentumHist,
Input_HistogramLenSecondData.GetInt());
sc.MovingAverage(close, Subgraph_MomentumHistUpColors, Input_SecondMovAvgType.GetMovAvgType(),
Input_HistogramLenSecondData.GetInt());

if(
    (Subgraph_MomentumHist[i]<0)
    &&(Subgraph_MomentumHist[i] < Subgraph_MomentumHist[i-1])

```

```

    )

    {
        Subgraph_MomentumHist.DataColor[sc.Index] = Subgraph_MomentumHistDownColors.PrimaryColor;
    }
    else if(
        (Subgraph_MomentumHist[i]<=0)
        &&(Subgraph_MomentumHist[i] > Subgraph_MomentumHist[i-1])
    )
    {
        Subgraph_MomentumHist.DataColor[sc.Index] = Subgraph_MomentumHistDownColors.SecondaryColor;
    }
    else if(
        (Subgraph_MomentumHist[i]>0)
        &&(Subgraph_MomentumHist[i] > Subgraph_MomentumHist[i-1])
    )
    {
        Subgraph_MomentumHist.DataColor[sc.Index] = Subgraph_MomentumHistUpColors.PrimaryColor;
    }
    else if(
        (Subgraph_MomentumHist[i]>=0)
        &&(Subgraph_MomentumHist[i] < Subgraph_MomentumHist[i-1])
    )
    {
        Subgraph_MomentumHist.DataColor[sc.Index] = Subgraph_MomentumHistUpColors.SecondaryColor;
    }

    //Squeeze
    sc.Keltner(
        sc.BaseDataIn,
        sc.Close,
        Subgraph_Temp8,
        Input_SqueezeLength.GetInt(),
        MOVAVGTYPE_SMOOTHED,
        Input_SqueezeLength.GetInt(),
        MOVAVGTYPE_SMOOTHED,
        Input_NK.GetFloat(),
        Input_NK.GetFloat()
    );

    float TopBandOut = Subgraph_Temp8.Arrays[0][sc.Index];
    float BottomBandOut = Subgraph_Temp8.Arrays[1][sc.Index];

    sc.BollingerBands(sc.Close, Subgraph_Temp4, Input_SqueezeLength.GetInt(), Input_NB.GetFloat(),
    MOVAVGTYPE_SMOOTHED);

    float BU =Subgraph_Temp4.Arrays[0][sc.Index];
    float BL =Subgraph_Temp4.Arrays[1][sc.Index];

    if (
        (BU < TopBandOut)
        || (BL > BottomBandOut)
    )
    {
        Subgraph_SqueezeDots[sc.Index] = 0.0;
        Subgraph_SqueezeDots.DataColor[sc.Index] = inside;
        Subgraph_SignalValues[sc.Index] = 0.0;
        Subgraph_SignalValues.DataColor[sc.Index] = inside;
    }
    else
    {
        Subgraph_SqueezeDots[sc.Index] = 0.0;
        Subgraph_SqueezeDots.DataColor[sc.Index] = outside;
        Subgraph_SignalValues[sc.Index] = 1.0;
    }

```

```

        Subgraph_SignalValues.DataColor[sc.Index] = outside;
    }
}

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

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

    if(sc.SetDefaults)
    {
        sc.GraphName="R Squared";
        sc.StudyDescription="R Squared Indicator by ertrader.";

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

        Subgraph_RSquared.Name="R Squared";
        Subgraph_RSquared.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_RSquared.LineWidth = 2;
        Subgraph_RSquared.PrimaryColor = COLOR_GREEN;
        Subgraph_RSquared.SecondaryColor = COLOR_RED;
        Subgraph_RSquared.DrawZeros = false;

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

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

    // Start plotting after good output elements are valid
    sc.DataStartIndex = Input_Length.GetInt();

    // R^2 indicator: Pwr(Corr(Cum( 1 ),C,14,0),2) or Correlation_Coefficient^2. This method calculates R^ with minimal lag.
    SCFloatArrayRef PriceData = sc.BaseData[Input_Data.GetInputDataIndex()];

    //The following is a cumulative calculation. It sums up a count of 1 for each bar. It is then used to correlate data against
    Subgraph_RSquared.Arrays[0][sc.Index] = static_cast<float>(sc.Index + 1);

    //Get the correlation coefficient. Correlate Price Data against the cumulation
    float coef = sc.GetCorrelationCoefficient(Subgraph_RSquared.Arrays[0], PriceData, sc.Index, Input_Length.GetInt());

    // Calculate and Draw R squared
    //Pwr(Corr(Cum( 1 ),C,14,0),2) or Correlation_Coefficient^2
    Subgraph_RSquared[sc.Index]=coef*coef;
}

/*=====*/
//http://en.wikipedia.org/wiki/Exponential_function
SCSFExport scsf_RSInverseFisherTransform(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_RSI_IFT = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Line1 = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Line2 = sc.Subgraph[2];
    SCSubgraphRef Subgraph_RSI = sc.Subgraph[3];

```

```

SCInputRef Input_RSILength = sc.Input[0];
SCInputRef Input_RSInternalMovAvgType = sc.Input[1];
SCInputRef Input_RSIMovAvgType = sc.Input[2];
SCInputRef Input_RSIMovAvgLength = sc.Input[3];
SCInputRef Input_LineValue = sc.Input[4];

if(sc.SetDefaults)
{
    sc.GraphName="Inverse Fisher Transform RSI";
    sc.StudyDescription="Inverse Fisher Transform RSI by John Ehlers.";

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

    Subgraph_RSI_IFT.Name="RSI IFT";
    Subgraph_RSI_IFT.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_RSI_IFT.LineWidth = 2;
    Subgraph_RSI_IFT.PrimaryColor = COLOR_GREEN;
    Subgraph_RSI_IFT.DrawZeros = true;

    Subgraph_Line1.Name="Line 1";
    Subgraph_Line1.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_Line1.LineWidth = 1;
    Subgraph_Line1.PrimaryColor = COLOR_RED;
    Subgraph_Line1.DrawZeros = true;

    Subgraph_Line2.Name="Line 2";
    Subgraph_Line2.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_Line2.LineWidth = 1;
    Subgraph_Line2.PrimaryColor = COLOR_BLUE;
    Subgraph_Line2.DrawZeros = true;

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

    Input_RSInternalMovAvgType.Name="RSI Internal MovAvg Type";
    Input_RSInternalMovAvgType.SetMovAvgType(MOVAVGTYPE_SIMPLE);

    Input_RSIMovAvgType.Name="RSI MovAvg Type";
    Input_RSIMovAvgType.SetMovAvgType(MOVAVGTYPE_WEIGHTED);

    Input_RSIMovAvgLength.Name="RSI MovAvg Length";
    Input_RSIMovAvgLength.SetInt(9);
    Input_RSIMovAvgLength.SetIntLimits(1,MAX_STUDY_LENGTH);

    Input_LineValue.Name="Line Value";
    Input_LineValue.SetFloat(0.5);

    return;
}

sc.DataStartIndex = Input_RSILength.GetInt()+Input_RSIMovAvgLength.GetInt();

sc.InverseFisherTransformRSI(sc.Close, Subgraph_RSI_IFT, Input_RSILength.GetInt(),
Input_RSInternalMovAvgType.GetMovAvgType(), Input_RSIMovAvgLength.GetInt(),
Input_RSIMovAvgType.GetMovAvgType());

Subgraph_Line1[sc.Index] = Input_LineValue.GetFloat();
Subgraph_Line2[sc.Index] = -Input_LineValue.GetFloat();
}

```



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

```
SCSFExport scsf_InverseFisherTransform(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_InverseFisherTransform = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Line1 = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Line2 = sc.Subgraph[2];
    SCSubgraphRef Subgraph_IFTAvg = sc.Subgraph[3];

    SCInputRef Input_Data = sc.Input[0];
    SCInputRef Input_LineValue = sc.Input[1];
    SCInputRef Input_OutputMovAvgLength = sc.Input[2];
    SCInputRef Input_OutputMovAvgType = sc.Input[3];
    SCInputRef Input_MovAvgLength = sc.Input[4];
    SCInputRef Input_MovAvgType = sc.Input[5];
    SCInputRef Input_HighestLowestLength = sc.Input[6];

    if(sc.SetDefaults)
    {
        sc.GraphName="Inverse Fisher Transform";
        sc.StudyDescription="Inverse Fisher Transform.";

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

        Subgraph_InverseFisherTransform.Name="IFT";
        Subgraph_InverseFisherTransform.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_InverseFisherTransform.LineWidth = 2;
        Subgraph_InverseFisherTransform.PrimaryColor = COLOR_GREEN;
        Subgraph_InverseFisherTransform.DrawZeros = true;

        Subgraph_Line1.Name="Line 1";
        Subgraph_Line1.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Line1.LineWidth = 1;
        Subgraph_Line1.PrimaryColor = COLOR_RED;
        Subgraph_Line1.DrawZeros = true;

        Subgraph_Line2.Name="Line 2";
        Subgraph_Line2.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Line2.LineWidth = 1;
        Subgraph_Line2.PrimaryColor = COLOR_BLUE;
        Subgraph_Line2.DrawZeros = true;

        Subgraph_IFTAvg.Name="IFT Avg";
        Subgraph_IFTAvg.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_IFTAvg.LineWidth = 1;
        Subgraph_IFTAvg.PrimaryColor = RGB(255,255,0);
        Subgraph_IFTAvg.DrawZeros = true;

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

        Input_LineValue.Name="Line Value";
        Input_LineValue.SetFloat(0.5);

        Input_OutputMovAvgLength.Name="Output Mov Avg Length";
        Input_OutputMovAvgLength.SetInt(3);

        Input_OutputMovAvgType.Name="Output Mov Avg Type";
        Input_OutputMovAvgType.SetMovAvgType(MOVAVGTYPE_SIMPLE);

        Input_MovAvgLength.Name="Input Data Mov Avg Length";
```

```

Input_MovAvgLength.SetInt(9);

Input_MovAvgType.Name="Input Data Mov Avg Type";
Input_MovAvgType.SetMovAvgType(MOVAVGTYPE_WEIGHTED);

Input_HighestLowestLength.Name = "Highest Value/Lowest Value Length";
Input_HighestLowestLength.SetInt(10);

return;
}

sc.DataStartIndex = Input_MovAvgLength.GetInt()+Input_OutputMovAvgLength.GetInt();

sc.InverseFisherTransform(sc.BaseData[Input_Data.GetInputDataIndex()], Subgraph_InverseFisherTransform,
Input_HighestLowestLength.GetInt(), Input_MovAvgLength.GetInt(), Input_MovAvgType.GetMovAvgType());

sc.MovingAverage(Subgraph_InverseFisherTransform, Subgraph_IFTAvg,
Input_OutputMovAvgType.GetMovAvgType(),Input_OutputMovAvgLength.GetInt());

Subgraph_Line1[sc.Index] = Input_LineValue.GetFloat();
Subgraph_Line2[sc.Index] = -Input_LineValue.GetFloat();
}

/*=====
Sine-Wave Weighted Moving Average
-----*/
SCSFExport scsf_SineWaveWMA(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_WMA = sc.Subgraph[0];

    SCInputRef Input_Data = sc.Input[0];

    if (sc.SetDefaults)
    {
        sc.GraphName="Sine Wave Weighted Moving Average";

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

        Subgraph_WMA.Name="SWA";
        Subgraph_WMA.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_WMA.LineWidth = 1;
        Subgraph_WMA.PrimaryColor = RGB(0,255,0);
        Subgraph_WMA.DrawZeros = true;

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

        return;
    }

    //PI:=3.1415926;
    //SD:=180/6;
    //S1:=Sin(1*180/6)*C;
    //S2:=Sin(2*180/6)*Ref(C,-1);
    //S3:=Sin(3*180/6)*Ref(C,-2);
    //S4:=Sin(4*180/6)*Ref(C,-3);
    //S5:=Sin(5*180/6)*Ref(C,-4);
    //Num:=S1+S2+S3+S4+S5;
    //Den:=Sin(SD)+Sin(2*SD)+Sin(3*SD)+Sin(4*SD)+Sin(5*SD);
    //Num/Den

    sc.DataStartIndex = 5;

```

```

if (sc.Index < 4)
    return;

int InputDataIndex = Input_Data.GetInputDataIndex();

const float SD = static_cast<float>(M_PI/6); // Note: the sin function takes radians, not degrees
float S1 = sin(1*SD) * sc.BaseData[InputDataIndex][sc.Index - 0];
float S2 = sin(2*SD) * sc.BaseData[InputDataIndex][sc.Index - 1];
float S3 = sin(3*SD) * sc.BaseData[InputDataIndex][sc.Index - 2];
float S4 = sin(4*SD) * sc.BaseData[InputDataIndex][sc.Index - 3];
float S5 = sin(5*SD) * sc.BaseData[InputDataIndex][sc.Index - 4];

float Num = S1 + S2 + S3 + S4 + S5;
float Den = sin(1*SD) + sin(2*SD) + sin(3*SD) + sin(4*SD) + sin(5*SD);

Subgraph_WMA[sc.Index] = Num / Den;
}

/*=====*/
/*
Bollinger bands squeeze.

Proportion = (kUpper - kLower) / (bbUpper - bbLower);

*/
SCSFExport scsf_BollingerSqueeze(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_BandsRatio = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SqueezeIndicator = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Keltner = sc.Subgraph[2];
    SCSubgraphRef Subgraph_BollingerBands = sc.Subgraph[3];

    SCInputRef Input_Data = sc.Input[0];
    SCInputRef Input_InternalMovAvgType = sc.Input[1];
    SCInputRef Input_KeltnerBandsLength = sc.Input[2];
    SCInputRef Input_KeltnerTrueRangeMALength = sc.Input[3];
    SCInputRef Input_KeltnerBandsMultiplier = sc.Input[4];
    SCInputRef Input_BollingerBandsLength = sc.Input[5];
    SCInputRef Input_BollingerBandsMultiplier = sc.Input[6];

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

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

        Subgraph_BandsRatio.Name = "Bands Ratio";
        Subgraph_BandsRatio.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_BandsRatio.PrimaryColor = RGB(0,255,0);
        Subgraph_BandsRatio.SecondaryColor = RGB(255,0,0);
        Subgraph_BandsRatio.SecondaryColorUsed = true;
        Subgraph_BandsRatio.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_Data.Name = "Input Data";
    }
}

```

```

Input_Data.SetInputDataIndex(SC_LAST);

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

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

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

Input_KeltnerBandsMultiplier.Name="Keltner Bands Multiplier";
Input_KeltnerBandsMultiplier.SetFloat(2.0f);

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

Input_BollingerBandsMultiplier.Name="Bollinger Bands Multiplier";
Input_BollingerBandsMultiplier.SetFloat(2.0f);

return;
}

sc.DataStartIndex = max(Input_BollingerBandsLength.GetInt(), max(Input_KeltnerBandsLength.GetInt(),
Input_KeltnerTrueRangeMALength.GetInt())) - 1;

// calculate Bollinger Bands

sc.BollingerBands(sc.BaseData[Input_Data.GetInputDataIndex()],Subgraph_BollingerBands,Input_BollingerBandsLength.G

// calculate Keltner
sc.Keltner(
sc.BaseData,sc.BaseData[Input_Data.GetInputDataIndex()],Subgraph_Keltner,Input_KeltnerBandsLength.GetInt(),
Input_InternalMovAvgType.GetMovAvgType(),Input_BollingerBandsLength.GetInt(),Input_InternalMovAvgType.GetMovAvg
Input_KeltnerBandsMultiplier.GetFloat(),Input_KeltnerBandsMultiplier.GetFloat());

float KUp = Subgraph_Keltner.Arrays[0][sc.CurrentIndex];
float KDown = Subgraph_Keltner.Arrays[1][sc.CurrentIndex];

float UBB = Subgraph_BollingerBands.Arrays[0][sc.CurrentIndex];
float LBB = Subgraph_BollingerBands.Arrays[1][sc.CurrentIndex];

if ((UBB > KUp) && (LBB < KDown))
    Subgraph_SqueezeIndicator.DataColor[sc.CurrentIndex] = Subgraph_SqueezeIndicator.PrimaryColor;
else
    Subgraph_SqueezeIndicator.DataColor[sc.CurrentIndex] = Subgraph_SqueezeIndicator.SecondaryColor;

Subgraph_BandsRatio[sc.CurrentIndex] = (KUp-KDown)/(UBB - LBB) - 1.0f;

if (Subgraph_BandsRatio[sc.CurrentIndex] >= 0)
    Subgraph_BandsRatio.DataColor[sc.CurrentIndex] = Subgraph_BandsRatio.PrimaryColor;
else
    Subgraph_BandsRatio.DataColor[sc.CurrentIndex] = Subgraph_BandsRatio.SecondaryColor;
}

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

```

```

SCInputRef Input_StudySubgraph1 = sc.Input[5];
SCInputRef Input_StudySubgraph2 = sc.Input[6];
SCInputRef Input_Study2SubgraphOffset = sc.Input[7];
SCInputRef Input_VersionUpdate = sc.Input[8];

SCInputRef Input_DrawZeros = sc.Input[9];
SCInputRef Input_PerformSubtractWithZeroValue = sc.Input[10];

if (sc.SetDefaults)
{
    sc.GraphName = "Study Subgraphs Subtract/Difference";
    sc.StudyDescription = "Calculates the difference between the two selected study Subgraphs.";

    sc.AutoLoop = 0;//manual looping
    sc.CalculationPrecedence = LOW_PREC_LEVEL;

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

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

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

    Input_Study2SubgraphOffset.Name = "Study 2 Subgraph Offset";// This is always converted to a positive value and
is the number of bars back
    Input_Study2SubgraphOffset.SetInt(0);

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

    Input_VersionUpdate.SetInt(2);

    Input_PerformSubtractWithZeroValue.Name = "Perform Subtract With Zero Value";
    Input_PerformSubtractWithZeroValue.SetYesNo(true);

    return;
}

if (Input_VersionUpdate.GetInt() < 2)
{
    Input_PerformSubtractWithZeroValue.SetYesNo(true);
    Input_VersionUpdate.SetInt(2);
}

if (sc.IsFullRecalculation && sc.UpdateStartIndex == 0)
    sc.GraphName = "Study Subgraphs Difference";

Subgraph_Diff.DrawZeros = Input_DrawZeros.GetYesNo();

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

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

int CalculationStartIndex = sc.GetCalculationStartIndexForStudy();

```

```

for (int Index = CalculationStartIndex; Index < sc.ArraySize; Index++)
{
    float Value1 = Study1Array[Index];
    float Value2 = Study2Array[Index - labs(Input_Study2SubgraphOffset.GetInt())];

    if (Input_PerformSubtractWithZeroValue.GetYesNo()
        || (Value1 != 0.0 && Value2 != 0.0))
    {
        Subgraph_Diff[Index] = Value1 - Value2;
    }
    else
    {
        Subgraph_Diff[Index] = 0.0;
    }
}

sc.EarliestUpdateSubgraphDataArrayIndex = CalculationStartIndex;

}

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

    SCInputRef Input_Data = sc.Input[0];
    SCInputRef Input_Length = sc.Input[3];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Linear Regressive Slope";

        sc.AutoLoop = 1;

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

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

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

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

        return;
    }

    sc.DataStartIndex = Input_Length.GetInt();

    int Len=Input_Length.GetInt();
    double SumBars = Len*(Len-1)/2.0;
    double SumSqrBars = (Len-1)*Len*(2*Len-1)/6.0;

    double Sum1      = 0;
    double SumY=0;
    for(int Index=sc.Index; Index > (sc.Index - Len); Index--)

```

```

{
    int x = sc.Index - Index;
    Sum1 = Sum1 + x * sc.BaseData[Input_Data.GetInputDataIndex()][Index];
    SumY += sc.BaseData[Input_Data.GetInputDataIndex()][Index];
}

double Sum2 = SumBars * SumY;
double Num1 = Len * Sum1 - Sum2;
double Num2 = SumBars * SumBars - Len * SumSqrBars;
Subgraph_LRS[sc.Index] = static_cast<float>(Num1 / Num2);

return;
}

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

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

    if (sc.SetDefaults)
    {
        sc.GraphName = "Summation - Periodic";
        sc.StudyDescription = "The cumulative sum over the number of elements specified by the Length input.";

        sc.AutoLoop = 1;

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

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

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

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

        return;
    }

    sc.Summation(sc.BaseData[Input_Data.GetInputDataIndex()], Subgraph_Sum, Input_Length.GetInt());
}

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

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

    if (sc.SetDefaults)
    {
        sc.GraphName = "Summation of Study Subgraph - Periodic";
    }
}

```

```

sc.AutoLoop = 1;

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

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

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

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

return;
}

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

if(StudyArray.GetArraySize() == 0)
{
    SCString Message;
    Message = sc.GraphName;
    Message += " Study being referenced does not exist.";
    sc.AddMessageToLog(Message, 1);
    return;
}

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

sc.Summation(StudyArray, Subgraph_Sum, Input_Length.GetInt());
}

/*=====*/
SCSFExport scsf_PeriodOHLCVariablePeriod(SCStudyInterfaceRef sc)
{
    SCInputRef Input_TimePeriodType = sc.Input[0];
    SCInputRef Input_TimePeriodLength = sc.Input[1];
    SCInputRef Input_UseCurrentPeriod = sc.Input[2];
    SCInputRef Input_MinimumRequiredTimePeriodAsPercent = sc.Input[3];
    SCInputRef Input_NumberOfForwardBarsToProject = sc.Input[4];
    SCInputRef Input_DisplayDebuggingOutput = sc.Input[5];
    SCInputRef Input_ForwardProjectLines = sc.Input[6];
    SCInputRef Input_NumberOfDaysToCalculate = sc.Input[7];
    SCInputRef Input_AutoSkipPeriodOfNoTrading = sc.Input[8];
    SCInputRef Input_GraphHighLowLinesHistorically = sc.Input[9];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Period OHLC-Variable Period";
        sc.AutoLoop = 0; // Manual Looping

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

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

```



```

sc.Subgraph[SC_OPEN].LineWidth = 2;
sc.Subgraph[SC_OPEN].DrawZeros = false;

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

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

sc.Subgraph[SC_LAST].Name = "Close";
sc.Subgraph[SC_LAST].DrawStyle = DRAWSTYLE_DASH;
sc.Subgraph[SC_LAST].PrimaryColor = RGB(255,127,0);
sc.Subgraph[SC_LAST].LineWidth = 2;
sc.Subgraph[SC_LAST].DrawZeros = false;

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

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

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

for (int SubgraphIndex = SC_OPEN; SubgraphIndex <= SC_HL_AVG; SubgraphIndex++)
{
    sc.Subgraph[SubgraphIndex].LineLabel =
        LL_DISPLAY_NAME | LL_NAME_ALIGN_CENTER | LL_NAME_ALIGN_FAR_RIGHT |
        LL_DISPLAY_VALUE | LL_VALUE_ALIGN_CENTER | LL_VALUE_ALIGN_VALUES_SCALE;
}

unsigned short DisplayOrder = 1;

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

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

Input_UseCurrentPeriod.Name = "Use Current Period";
Input_UseCurrentPeriod.SetYesNo(false);
Input_UseCurrentPeriod.DisplayOrder = DisplayOrder++;

Input_AutoSkipPeriodOfNoTrading.Name = "Auto Skip Period of No Trading";
Input_AutoSkipPeriodOfNoTrading.SetYesNo(false);
Input_AutoSkipPeriodOfNoTrading.DisplayOrder = DisplayOrder++;

Input_MinimumRequiredTimePeriodAsPercent.Name = "Minimum Required Time Period as %";
Input_MinimumRequiredTimePeriodAsPercent.SetFloat(5.0f);
Input_MinimumRequiredTimePeriodAsPercent.SetFloatLimits(1.0f, 100.0f);

```

```

Input_MinimumRequiredTimePeriodAsPercent.DisplayOrder = DisplayOrder++;

Input_ForwardProjectLines.Name = "Forward Project OHLC Lines";
Input_ForwardProjectLines.SetYesNo(0);
Input_ForwardProjectLines.DisplayOrder = DisplayOrder++;

Input_NumberOfForwardBarsToProject.Name = "Number of Forward Bars to Project";
Input_NumberOfForwardBarsToProject.SetInt(10);
Input_NumberOfForwardBarsToProject.DisplayOrder = DisplayOrder++;

Input_NumberOfDaysToCalculate.Name = "Number of Days to Calculate";
Input_NumberOfDaysToCalculate.SetInt(250);
Input_NumberOfDaysToCalculate.SetIntLimits(1, MAX_STUDY_LENGTH);
Input_NumberOfDaysToCalculate.DisplayOrder = DisplayOrder++;

Input_GraphHighLowLinesHistorically.Name = "Graph High/Low Lines Historically (Use Current Period = yes)";
Input_GraphHighLowLinesHistorically.SetYesNo(0);
Input_GraphHighLowLinesHistorically.DisplayOrder = DisplayOrder++;

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

return;
}

int PeriodLength = Input_TimePeriodLength.GetInt();

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

int NumberOfForwardBars = 0;

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

    for (int SubgraphIndex = SC_OPEN; SubgraphIndex <= SC_HL_AVG; SubgraphIndex++)
        sc.Subgraph[SubgraphIndex].ExtendedArrayElementsToGraph = NumberOfForwardBars;
}
else
{
    for (int SubgraphIndex = SC_OPEN; SubgraphIndex <= SC_HL_AVG; SubgraphIndex++)
        sc.Subgraph[SubgraphIndex].ExtendedArrayElementsToGraph = 0;
}

if (Input_NumberOfDaysToCalculate.GetInt() < 1)
    Input_NumberOfDaysToCalculate.SetInt(250);

if (Input_UseCurrentPeriod.GetYesNo() == 0)
    Input_GraphHighLowLinesHistorically.SetYesNo(false);

SCDateTimeMS FirstDateToCalculate = sc.GetTradingDayDate(sc.BaseDateTimeIn[sc.ArraySize - 1]);
FirstDateToCalculate.SubtractDays(Input_NumberOfDaysToCalculate.GetInt() - 1);

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

SCDateTimeMS BeginOfRefDateTime;
SCDateTimeMS EndOfRefDateTime;
SCDateTimeMS CurrentPeriodBeginDateTime;
SCDateTimeMS CurrentPeriodEndDateTime;

SCDateTimeMS PriorCurrentPeriodStartDateTime;

```

```

for (int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize + NumberOfForwardBars; BarIndex++)
{
    const SCDatetimeMS CurrentBarDateTime = sc.BaseDateTimeIn[BarIndex];

    if (sc.GetTradingDayDate(CurrentBarDateTime) < FirstDateToCalculate)
        continue;

    bool GetReferenceData = true;

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

    if (CurrentPeriodBeginDateTime == PriorCurrentPeriodStartDateTime)
    {
        GetReferenceData = false;
    }

    PriorCurrentPeriodStartDateTime = CurrentPeriodBeginDateTime;

    bool IsStartOfNewPeriod = false;

    if (Input_GraphHighLowLinesHistorically.GetYesNo())
    {
        if (BarIndex == 0)
            IsStartOfNewPeriod = true;
        else
        {
            SCDatetimeMS PriorBarPeriodStartDateTime =
sc.GetStartOfPeriodForDateTime(sc.BaseDateTimeIn[BarIndex - 1], Input_TimePeriodType.GetTimePeriodType(),
PeriodLength, 0);
            IsStartOfNewPeriod = PriorBarPeriodStartDateTime != CurrentPeriodBeginDateTime;
        }
    }

    SCDatetimeMS TimeIncrement = sc.TimePeriodSpan(Input_TimePeriodType.GetTimePeriodType(),
Input_TimePeriodLength.GetInt());

    //If there are insufficient bars for a reference period, then we will walk back 1 block/period at a time.

    if(GetReferenceData)
    {
        CurrentPeriodEndDateTime = sc.GetStartOfPeriodForDateTime(CurrentPeriodBeginDateTime,
Input_TimePeriodType.GetTimePeriodType(), PeriodLength, 1);

        CurrentPeriodEndDateTime.SubtractMicroseconds(1);

        if (!Input_UseCurrentPeriod.GetYesNo())
        {
            BeginOfRefDateTime = sc.GetStartOfPeriodForDateTime(CurrentPeriodBeginDateTime,
Input_TimePeriodType.GetTimePeriodType(), PeriodLength, -1);

            EndOfRefDateTime = CurrentPeriodBeginDateTime - SCDatetime::MICROSECONDS(1);
        }
        else
        {
            BeginOfRefDateTime = CurrentPeriodBeginDateTime;

            EndOfRefDateTime = CurrentPeriodEndDateTime;
        }

        if (Input_DisplayDebuggingOutput.GetYesNo() != 0)
    {

```

```

SCString Message;

Message.Format("Current Bar: %s, BeginOfRefDateTime: %s, EndOfRefDateTime: %s,
CurrentPeriodBeginDateTime: %s, CurrentPeriodEndDateTime: %s",
    sc.FormatDateTimeMS(CurrentBarDateTime).GetChars()
    , sc.FormatDateTimeMS(BeginOfRefDateTime).GetChars()
    , sc.FormatDateTimeMS(EndOfRefDateTime).GetChars()
    , sc.FormatDateTimeMS(CurrentPeriodBeginDateTime).GetChars()
    , sc.FormatDateTimeMS(CurrentPeriodEndDateTime).GetChars());

sc.AddMessageToLog(Message,0);

}

int MaxPeriodsToGoBack = 1;

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

for (int WalkBack = 0; WalkBack < MaxPeriodsToGoBack; WalkBack++)
{
    if (!Input_UseCurrentPeriod.GetYesNo() && WalkBack >= 1) //Walk back 1 period.
    {
        SCDateTimeMS PriorBeginOfRefDateTime = BeginOfRefDateTime;

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

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

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

            sc.AddMessageToLog(Message,0);
        }
    }
}

int NumberOfBars = 0;
SCDateTimeMS TotalTimeSpan;
int Result = 0;

if (Input_UseCurrentPeriod.GetYesNo())
{
    Result = sc.GetOHLCOfTimePeriod(CurrentPeriodBeginDateTime, CurrentPeriodEndDateTime, Open,
High, Low, Close, NextOpen, NumberOfBars, TotalTimeSpan);
}
else
{
    Result = sc.GetOHLCOfTimePeriod(BeginOfRefDateTime, EndOfRefDateTime, Open, High, Low, Close,
NextOpen, NumberOfBars, TotalTimeSpan);
}

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

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

```

```

        sc.AddMessageToLog(Message,0);
    }

    if (Input_UseCurrentPeriod.GetYesNo())
        break;

    if (!Result)
        continue;

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

    if (TotalTimeSpan >= MinimumTimeSpan)
        break;
    }
}

if (!sc.IsFullRecalculation && Input_UseCurrentPeriod.GetYesNo())
{
    int StartingBackIndex
        = sc.GetContainingIndexForSCDateTime(sc.ChartNumber, CurrentPeriodBeginDateTime);

    //Adjust for incorrect date/time match from sc.GetContainingIndexForSCDateTime
    const SCDateTimeMS StartingBackIndexPeriodStartDateTime =
sc.GetStartOfPeriodForDateTime(sc.BaseDateTimeIn[StartingBackIndex], Input_TimePeriodType.GetTimePeriodType(),
PeriodLength, 0);

    if (StartingBackIndexPeriodStartDateTime < CurrentPeriodBeginDateTime
        && StartingBackIndex < sc.ArraySize - 1)
    {
        StartingBackIndex++;
    }

    for (int PriorBarIndex = StartingBackIndex; PriorBarIndex <= BarIndex; ++PriorBarIndex)
    {
        sc.Subgraph[SC_OPEN][PriorBarIndex] = Open;

        if (!Input_GraphHighLowLinesHistorically.GetYesNo())
        {
            sc.Subgraph[SC_HIGH][PriorBarIndex] = High;
            sc.Subgraph[SC_LOW][PriorBarIndex] = Low;
        }

        sc.Subgraph[SC_LAST][PriorBarIndex] = Close;
        sc.CalculateOHLCAverages(PriorBarIndex);
    }
}
else
{
    sc.Subgraph[SC_OPEN][BarIndex] = Open;

    if (!Input_GraphHighLowLinesHistorically.GetYesNo())
    {
        sc.Subgraph[SC_HIGH][BarIndex] = High;
        sc.Subgraph[SC_LOW][BarIndex] = Low;
    }

    sc.Subgraph[SC_LAST][BarIndex] = Close;
    sc.CalculateOHLCAverages(BarIndex);
}

if (Input_GraphHighLowLinesHistorically.GetYesNo())
{
    if (IsStartOfNewPeriod)

```

```

    {
        sc.Subgraph[SC_HIGH][BarIndex] = sc.High[BarIndex];
        sc.Subgraph[SC_LOW][BarIndex] = sc.Low[BarIndex];
    }
    else
    {
        sc.Subgraph[SC_HIGH][BarIndex] = sc.Subgraph[SC_HIGH][BarIndex - 1];

        if (sc.Subgraph[SC_HIGH][BarIndex] < sc.High[BarIndex])
            sc.Subgraph[SC_HIGH][BarIndex] = sc.High[BarIndex];

        sc.Subgraph[SC_LOW][BarIndex] = sc.Subgraph[SC_LOW][BarIndex - 1];

        if (sc.Subgraph[SC_LOW][BarIndex] > sc.Low[BarIndex])
            sc.Subgraph[SC_LOW][BarIndex] = sc.Low[BarIndex];
    }
}
} //for
}

/*=====*/

SCSFExport scsf_RenkoChart(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_NumTrades = 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];
    SCSubgraphRef Subgraph_RenkoUpDownTrend = sc.Subgraph[11];
    SCSubgraphRef Subgraph_RenkoLast = sc.Subgraph[12];

    SCInputRef Input_RenkoBoxSize = sc.Input[0];
    SCInputRef Input_UseHighLowInsteadOfLast = sc.Input[1];
    SCInputRef Input_IgnoreLatestBarUntilClose = sc.Input[2];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Renko Chart (unsupported)";

        sc.GraphRegion = 0;
        sc.StandardChartHeader = 1;
        sc.IsCustomChart = 1;
        sc.GraphDrawType = GDT_RENKO_BRICK;

        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, 127, 0);
        Subgraph_High.DrawZeros = false;

        Subgraph_Low.Name = "Low";
        Subgraph_Low.DrawStyle = DRAWSTYLE_LINE;
    }
}

```

```

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

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

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

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

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

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

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

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

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

Input_RenkoBoxSize.Name = "Renko Box Size";
Input_RenkoBoxSize.SetFloat(0.0f);
Input_RenkoBoxSize.SetFloatLimits(0.0f, static_cast<float>(MAX_STUDY_LENGTH));

Input_UseHighLowInsteadOfLast.Name = "Use High/Low Values instead of Last Value";
Input_UseHighLowInsteadOfLast.SetYesNo(0);

Input_IgnoreLatestBarUntilClose.Name = "Ignore Latest Underlying Bar until Close";
Input_IgnoreLatestBarUntilClose.SetYesNo(0);

return;
}

sc.ValueFormat = sc.BaseGraphValueFormat;
int ValueFormat = sc.BaseGraphValueFormat;

if (Input_RenkoBoxSize.GetFloat() == 0.0f)
{
    Input_RenkoBoxSize.SetFloat(sc.TickSize*4);
}

```

```

sc.GraphName.Format("%s Renko - BoxSize: %s",
    sc.GetStudyName(0).GetChars(),
    sc.FormatGraphValue(Input_RenkoBoxSize.GetFloat(), ValueFormat).GetChars());

int BaseGraphIndex = sc.UpdateStartIndex;

if (Input_IgnoreLatestBarUntilClose.GetYesNo() != false && (sc.UpdateStartIndex == sc.ArraySize - 1))
    return; // No need to process last underlying bar. However, during a full recalculation, the last bar will be processed.
This is OK.

float UpperBox = 0;
float LowerBox = 0;
float CumulativeVolume = 0;
float CumulativeOIOrNT = 0;
float CumulativeAskVolume = 0;
float CumulativeBidVolume = 0;

if (BaseGraphIndex == 0)
{
    sc.ResizeArrays(0);

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

    if (sc.ArraySize < 1)
        return; // nothing to display

    UpperBox = sc.Close[BaseGraphIndex];
    LowerBox = sc.Close[BaseGraphIndex];

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

    Subgraph_RenkoLast[0] = sc.Close[BaseGraphIndex];
    Subgraph_Open[0] = sc.Close[BaseGraphIndex];
    Subgraph_Last[0] = sc.Close[BaseGraphIndex];
    Subgraph_High[0] = sc.Close[BaseGraphIndex];
    Subgraph_Low[0] = sc.Close[BaseGraphIndex];
}
else
{
    int RenkoIndex = sc.OutArraySize - 1;

    // When new data is updated within the same bar
    if (sc.DateTimeOut[RenkoIndex] == sc.BaseDateTimeln[BaseGraphIndex])
    {
        // Decrement BaseGraphIndex to the first matching date time
        for (; BaseGraphIndex >= 0; BaseGraphIndex--)
        {
            if (sc.DateTimeOut[RenkoIndex] != sc.BaseDateTimeln[BaseGraphIndex])
            {
                BaseGraphIndex++;
                break;
            }
        }
    }

    int LastIndex = RenkoIndex;
    // Delete the Renko Elements with that date time
    for (; RenkoIndex >= 0; RenkoIndex--)
    {
        if (sc.DateTimeOut[LastIndex] != sc.DateTimeOut[RenkoIndex])
        {
            RenkoIndex++;
            break;
        }
    }
}

```



```

    }
}
sc.ResizeArrays(RenkoIndex);
RenkoIndex = sc.OutArraySize - 1;
}

if (Subgraph_RenkoUpDownTrend[RenkoIndex] > 0) // Uptrend Box
{
    UpperBox = Subgraph_RenkoLast[RenkoIndex];
    LowerBox = UpperBox - Input_RenkoBoxSize.GetFloat();
    LowerBox = static_cast<float>(sc.RoundToTickSize(LowerBox, Input_RenkoBoxSize.GetFloat()));
}
else
{
    LowerBox = Subgraph_RenkoLast[RenkoIndex];
    UpperBox = LowerBox + Input_RenkoBoxSize.GetFloat();
    UpperBox = static_cast<float>(sc.RoundToTickSize(UpperBox, Input_RenkoBoxSize.GetFloat()));
}

// Calculate Cumulative Volume/OpenInterest/Number Of Trades for current index
// Must calculate from Base Graph since current index could
// potentially have grown in volume since last calculation.

int i = BaseGraphIndex;
for (; i >= 0; i--)
{
    if (sc.BaseDateTimeIn[i] <= sc.DateTimeOut[RenkoIndex])
        break;
}

for (int n = i; n < BaseGraphIndex; n++)
{
    CumulativeVolume += sc.Volume[n];
    CumulativeOIOOrNT += sc.OpenInterest[n];
    CumulativeAskVolume += sc.AskVolume[n];
    CumulativeBidVolume += sc.BidVolume[n];
}

}

int OriginalRenkoIndex = sc.OutArraySize - 1;
int BaseGraphEnd = sc.ArraySize;
if (Input_IgnoreLatestBarUntilClose.GetYesNo() != 0)
    BaseGraphEnd--;

for (; BaseGraphIndex < BaseGraphEnd; BaseGraphIndex++)
{
    CumulativeVolume += sc.Volume[BaseGraphIndex];
    CumulativeOIOOrNT += sc.OpenInterest[BaseGraphIndex];
    CumulativeAskVolume += sc.AskVolume[BaseGraphIndex];
    CumulativeBidVolume += sc.BidVolume[BaseGraphIndex];

    int NewEntry = 0;
    int UpperOrLowerTrend = 0;
    float HigherPrice = sc.Close[BaseGraphIndex];
    float LowerPrice = sc.Close[BaseGraphIndex];

    if (Input_UseHighLowInsteadOfLast.GetYesNo() != 0)
    {
        HigherPrice = sc.High[BaseGraphIndex];
        LowerPrice = sc.Low[BaseGraphIndex];
    }

    if (sc.FormattedEvaluateUsingDoubles(static_cast<double>(HigherPrice), ValueFormat,
    GREATER_EQUAL_OPERATOR, static_cast<double>(UpperBox + Input_RenkoBoxSize.GetFloat()), ValueFormat))

```

```

{
    NewEntry = 1;
    UpperOrLowerTrend = 1;
}
else if (sc.FormattedEvaluateUsingDoubles(static_cast<double>(LowerPrice), ValueFormat,
LESS_EQUAL_OPERATOR, static_cast<double>(LowerBox - Input_RenkoBoxSize.GetFloat()), ValueFormat))
{
    NewEntry = 1;
    UpperOrLowerTrend = -1;
}

int NewEntryCounter = 0;

while (NewEntry != 0)
{
    NewEntryCounter++;
    if (NewEntryCounter >= 500)
    {
        sc.AddMessageToLog("Renko Chart study: Renko error due to either data error or Box Size too small.
Stopping processing and skipping until real time updates. Try increasing the Box Size input.", 1);
        return;
    }

    sc.CalculateOHLCAverages(sc.OutArraySize - 1);

    NewEntry = 0;
    sc.AddElements(1);
    int NewBoxIndex = sc.OutArraySize - 1;
    sc.DateTimeOut[NewBoxIndex] = sc.BaseDateTimeln[BaseGraphIndex];

    Subgraph_Volume[NewBoxIndex-1] = CumulativeVolume - sc.Volume[BaseGraphIndex];
    Subgraph_NumTrades[NewBoxIndex-1] = CumulativeOIOOrNT - sc.OpenInterest[BaseGraphIndex];
    Subgraph_AskVol[NewBoxIndex-1] = CumulativeAskVolume - sc.AskVolume[BaseGraphIndex];
    Subgraph_BidVol[NewBoxIndex-1] = CumulativeBidVolume - sc.BidVolume[BaseGraphIndex];

    CumulativeVolume = sc.Volume[BaseGraphIndex];
    CumulativeOIOOrNT = sc.OpenInterest[BaseGraphIndex];
    CumulativeAskVolume = sc.AskVolume[BaseGraphIndex];
    CumulativeBidVolume = sc.BidVolume[BaseGraphIndex];

    float NewLast = 0;
    float NewOtherBound = 0;

    if (UpperOrLowerTrend == 1) // Upper Trend
    {
        LowerBox = UpperBox;
        UpperBox += Input_RenkoBoxSize.GetFloat();
        UpperBox = static_cast<float>(sc.RoundToTickSize(UpperBox, Input_RenkoBoxSize.GetFloat()));

        NewLast = UpperBox;
        NewOtherBound = LowerBox;

        if (sc.FormattedEvaluateUsingDoubles(static_cast<double>(HigherPrice), ValueFormat,
GREATER_EQUAL_OPERATOR, static_cast<double>(UpperBox + Input_RenkoBoxSize.GetFloat()), ValueFormat))
            NewEntry = 1;
    }
    else
    {
        UpperBox = LowerBox;
        LowerBox -= Input_RenkoBoxSize.GetFloat();
        LowerBox = static_cast<float>(sc.RoundToTickSize(LowerBox, Input_RenkoBoxSize.GetFloat()));

        NewLast = LowerBox;
        NewOtherBound = UpperBox;
    }
}

```

```

        if (sc.FormattedEvaluateUsingDoubles(static_cast<double>(LowerPrice), ValueFormat,
LESS_EQUAL_OPERATOR, static_cast<double>(LowerBox - Input_RenkoBoxSize.GetFloat()), ValueFormat))
            NewEntry = 1;
    }

    Subgraph_Low[NewBoxIndex] = LowerBox;
    Subgraph_High[NewBoxIndex] = UpperBox;
    Subgraph_RenkoLast[NewBoxIndex] = NewLast;
    Subgraph_RenkoUpDownTrend[NewBoxIndex] = static_cast<float>(UpperOrLowerTrend);
    if (UpperOrLowerTrend == 1)
    {
        Subgraph_Open[NewBoxIndex] = LowerBox;
        Subgraph_Last[NewBoxIndex] = UpperBox;
    }
    else
    {
        Subgraph_Open[NewBoxIndex] = UpperBox;
        Subgraph_Last[NewBoxIndex] = LowerBox;
    }
}

sc.CalculateOHLCAverages(sc.OutArraySize - 1);
}

int RenkoIndex = sc.OutArraySize - 1;
if (RenkoIndex >= 0)
{
    Subgraph_Volume[RenkoIndex] = CumulativeVolume;
    Subgraph_NumTrades[RenkoIndex] = CumulativeOIOrNT;
    Subgraph_AskVol[RenkoIndex] = CumulativeAskVolume;
    Subgraph_BidVol[RenkoIndex] = CumulativeBidVolume;
}
}

/*=====*/
SCSFExport scsf_MillisecondsExample(SCStudyInterfaceRef sc)
{
    if (sc.SetDefaults)
    {
        // Set the configuration and defaults

        sc.GraphName = "Milliseconds Example";

        sc.AutoLoop = 1;

        return;
    }
    //When working with milliseconds it is necessary to define variables of type SCDateTimeMS
    SCDateTimeMS DateTimeMS , DateTimeMSRefChart;

    // Get the milliseconds for the current bar. This will be 0 if the chart bars are based upon a specific time by using the
'Days-Minutes-Seconds' Bar Period Type since the start time for a bar is always evenly aligned to a second based upon
the Session Start Time and the time period of the bar. Otherwise, the milliseconds can be obtained for other Bar Period
Types.
    DateTimeMS = sc.BaseDateTimeIn[sc.Index];
    int Millisecond = DateTimeMS.GetMillisecond();

    //Get the Date-Times from another chart.
    SCDateTimeArray RefChartDateTime;
    sc.GetChartDateTimeArray(2, RefChartDateTime );

    if(RefChartDateTime.GetArraySize() != 0)

```

```

{
    //Get milliseconds for last bar in the other chart being referenced
    DateTimeMSRefChart = RefChartDateTime[RefChartDateTime.GetArraySize() - 1];
    int Millisecond = DateTimeMSRefChart.GetMillisecond();
}

//Compare the Date-Times from the 2 charts with millisecond precision
if(DateTimeMS == DateTimeMSRefChart)
{
    //There is a match down to the millisecond
}

}

/*=====*/
SCSFExport scsf_RelativeVigorIndex1(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_RVI = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SmoothedRVI = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Signal = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Buy = sc.Subgraph[3];
    SCSubgraphRef Subgraph_Sell = sc.Subgraph[4];

    SCInputRef Input_SmoothedRVILength = sc.Input[0];
    SCInputRef Input_SmoothedRVIAvgType = sc.Input[1];
    SCInputRef Input_SignalLength = sc.Input[2];
    SCInputRef Input_SignalAvgType = sc.Input[3];
    SCInputRef Input_OffsetPercentInput = sc.Input[4];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Relative Vigor Index 1";

        sc.AutoLoop = 1;

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

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

        Subgraph_SmoothedRVI.Name = "Smoothed RVI Line";
        Subgraph_SmoothedRVI.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_SmoothedRVI.PrimaryColor = RGB(0, 255, 0);
        Subgraph_SmoothedRVI.DrawZeros = true;

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

        Subgraph_Buy.Name = "Buy";
        Subgraph_Buy.PrimaryColor = RGB(0, 255, 0); // green
        Subgraph_Buy.DrawStyle = DRAWSTYLE_ARROW_UP;
        Subgraph_Buy.LineWidth = 2; //Width of arrow
        Subgraph_Buy.DrawZeros = 0;

        Subgraph_Sell.Name = "Sell";

```

```

Subgraph_Sell.DrawStyle = DRAWSTYLE_ARROW_DOWN;
Subgraph_Sell.PrimaryColor = RGB(255, 0, 0); // red
Subgraph_Sell.LineWidth = 2; //Width of arrow
Subgraph_Sell.DrawZeros = 0;

Input_SmoothedRVILength.Name = "Smoothed RVI Length";
Input_SmoothedRVILength.SetInt(10);
Input_SmoothedRVILength.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_SmoothedRVIAvgType.Name = "Smoothed RVI Average Type";
Input_SmoothedRVIAvgType.SetMovAvgType(MOVAVGTYPE_SIMPLE);

Input_SignalLength.Name = "Signal Length";
Input_SignalLength.SetInt(4);
Input_SignalLength.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_SignalAvgType.Name = "Signal Average Type";
Input_SignalAvgType.SetMovAvgType(MOVAVGTYPE_SIMPLE);

Input_OffsetPercentInput.Name = "Arrow Offset Percentage";
Input_OffsetPercentInput.SetFloat(8);

return;
}

sc.DataStartIndex = max(Input_SmoothedRVILength.GetInt(), Input_SignalLength.GetInt()) - 1;

float BarRange = (sc.High[sc.Index] - sc.Low[sc.Index]);
if (BarRange != 0)
    Subgraph_RVI[sc.Index] = (sc.Close[sc.Index] - sc.Open[sc.Index]) / BarRange;
else
    Subgraph_RVI[sc.Index] = Subgraph_RVI.Arrays[0][sc.Index - 1];

    sc.MovingAverage(Subgraph_RVI, Subgraph_SmoothedRVI, Input_SmoothedRVIAvgType.GetMovAvgType(),
Input_SmoothedRVILength.GetInt());
    sc.MovingAverage(Subgraph_RVI, Subgraph_Signal, Input_SignalAvgType.GetMovAvgType(),
Input_SignalLength.GetInt());

float OffsetPercent = Input_OffsetPercentInput.GetFloat() * 0.01f;

if (sc.CrossOver(Subgraph_SmoothedRVI, Subgraph_Signal) == CROSS_FROM_BOTTOM)
{
    Subgraph_Buy[sc.Index] = Subgraph_Signal[sc.Index] - OffsetPercent * Subgraph_Signal[sc.Index];

    Subgraph_Sell[sc.Index] = 0;
}
else if (sc.CrossOver(Subgraph_SmoothedRVI, Subgraph_Signal) == CROSS_FROM_TOP)
{
    Subgraph_Sell[sc.Index] = Subgraph_Signal[sc.Index] + OffsetPercent * Subgraph_Signal[sc.Index];
    Subgraph_Buy[sc.Index] = 0;
}
else
{
    Subgraph_Buy[sc.Index] = 0;
    Subgraph_Sell[sc.Index] = 0;
}
}
/*=====*/

SCSFExport scsf_RelativeVigorIndex2(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_RVI = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SmoothedRVI = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Signal = sc.Subgraph[2];

```

```

SCSubgraphRef Subgraph_Buy = sc.Subgraph[3];
SCSubgraphRef Subgraph_Sell = sc.Subgraph[4];

SCFloatArrayRef Array_CloseOpenAvg = Subgraph_RVI.Arrays[0];
SCFloatArrayRef Array_HighLowAvg = Subgraph_RVI.Arrays[1];
SCFloatArrayRef Array_SmoothedRVINumerator = Subgraph_RVI.Arrays[2];
SCFloatArrayRef Array_SmoothedRVIDenominator = Subgraph_RVI.Arrays[3];

SCInputRef Input_SmoothedRVILength = sc.Input[0];
SCInputRef Input_SmoothedRVIAvgType = sc.Input[1];
SCInputRef Input_OffsetPercentInput = sc.Input[2];

if (sc.SetDefaults)
{
    sc.GraphName = "Relative Vigor Index 2";

    sc.AutoLoop = 1;

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

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

    Subgraph_SmoothedRVI.Name = "Smoothed RVI Line";
    Subgraph_SmoothedRVI.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_SmoothedRVI.PrimaryColor = RGB(0, 255, 0);
    Subgraph_SmoothedRVI.DrawZeros = true;

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

    Subgraph_Buy.Name = "Buy";
    Subgraph_Buy.PrimaryColor = RGB(0, 255, 0); // green
    Subgraph_Buy.DrawStyle = DRAWSTYLE_ARROW_UP;
    Subgraph_Buy.LineWidth = 2; //Width of arrow
    Subgraph_Buy.DrawZeros = 0;

    Subgraph_Sell.Name = "Sell";
    Subgraph_Sell.DrawStyle = DRAWSTYLE_ARROW_DOWN;
    Subgraph_Sell.PrimaryColor = RGB(255, 0, 0); // red
    Subgraph_Sell.LineWidth = 2; //Width of arrow
    Subgraph_Sell.DrawZeros = 0;

    Input_SmoothedRVILength.Name = "Smoothed RVI Length";
    Input_SmoothedRVILength.SetInt(10);
    Input_SmoothedRVILength.SetIntLimits(1, MAX_STUDY_LENGTH);

    Input_SmoothedRVIAvgType.Name = "Smoothed RVI Average Type";
    Input_SmoothedRVIAvgType.SetMovAvgType(MOVAVGTYPE_SIMPLE);

    Input_OffsetPercentInput.Name = "Arrow Offset Percentage";
    Input_OffsetPercentInput.SetFloat(8);

    return;
}

sc.DataStartIndex = Input_SmoothedRVILength.GetInt() + 5;

```

```
Array_CloseOpenAvg[sc.Index] = ((sc.Close[sc.Index - 3] - sc.Open[sc.Index - 3]) + 2 * (sc.Close[sc.Index - 2] - sc.Open[sc.Index - 2]) + 2 * (sc.Close[sc.Index - 1] - sc.Open[sc.Index - 1]) + (sc.Close[sc.Index] - sc.Open[sc.Index])) / 6;
```

```
Array_HighLowAvg[sc.Index] = ((sc.High[sc.Index - 3] - sc.Low[sc.Index - 3]) + 2 * (sc.High[sc.Index - 2] - sc.Low[sc.Index - 2]) + 2 * (sc.High[sc.Index - 1] - sc.Low[sc.Index - 1]) + (sc.High[sc.Index] - sc.Low[sc.Index])) / 6;
```

```
float BarRange = (sc.High[sc.Index] - sc.Low[sc.Index]);
if (BarRange != 0)
    Subgraph_RVI[sc.Index] = (sc.Close[sc.Index] - sc.Open[sc.Index]) / BarRange;
else
    Subgraph_RVI[sc.Index] = Array_CloseOpenAvg[sc.Index - 1];
```

```
sc.MovingAverage(Array_CloseOpenAvg, Array_SmoothedRVINumerator,
Input_SmoothedRVIAvgType.GetMovAvgType(), Input_SmoothedRVILength.GetInt());
```

```
sc.MovingAverage(Array_HighLowAvg, Array_SmoothedRVIDenominator,
Input_SmoothedRVIAvgType.GetMovAvgType(), Input_SmoothedRVILength.GetInt());
```

```
Subgraph_SmoothedRVI[sc.Index] = Array_SmoothedRVINumerator[sc.Index] /
Array_SmoothedRVIDenominator[sc.Index];
```

```
Subgraph_Signal[sc.Index] = (Subgraph_SmoothedRVI[sc.Index - 3] + 2 * Subgraph_SmoothedRVI[sc.Index - 2] + 2 *
Subgraph_SmoothedRVI[sc.Index - 1] + Subgraph_SmoothedRVI[sc.Index]) / 6;
```

```
float OffsetPercent = Input_OffsetPercentInput.GetFloat() * 0.01f;
```

```
if (sc.CrossOver(Subgraph_SmoothedRVI, Subgraph_Signal) == CROSS_FROM_BOTTOM)
{
    Subgraph_Buy[sc.Index] = Subgraph_Signal[sc.Index] - OffsetPercent * Subgraph_Signal[sc.Index];

    Subgraph_Sell[sc.Index] = 0;
}
else if (sc.CrossOver(Subgraph_SmoothedRVI, Subgraph_Signal) == CROSS_FROM_TOP)
{
    Subgraph_Sell[sc.Index] = Subgraph_Signal[sc.Index] + OffsetPercent * Subgraph_Signal[sc.Index];
    Subgraph_Buy[sc.Index] = 0;
}
else
{
    Subgraph_Buy[sc.Index] = 0;
    Subgraph_Sell[sc.Index] = 0;
}
}
```

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

```
SCSFExport scsf_GetStudyPersistentVariableFromChartExample(SCStudyInterfaceRef sc)
```

```
{
    SCInputRef Input_ChartStudyReference = sc.Input[0];

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

        sc.GraphName = "Get Study Persistent Variable from Chart Example";
        sc.AutoLoop = 1;

        Input_ChartStudyReference.Name = "Chart Study Reference";
        Input_ChartStudyReference.SetChartStudyValues(1, 0);

        return;
```

```

}

//Set persistent float variable with a value of 100 using integer key 1.
float& r_PersistentFloat = sc.GetPersistentFloat(1);
r_PersistentFloat = 100;

// Get a reference to a persistent variable with key value 1 in the chart
// and study specified by the "Chart Study Reference" input. Assuming the
// "Chart Study Reference" input is referencing this particular chart and
// study instance, then this function will return 100.
const float PersistentValue = sc.GetPersistentFloatFromChartStudy(Input_ChartStudyReference.GetChartNumber(),
Input_ChartStudyReference.GetStudyID(), 1);

if (sc.Index == sc.ArraySize - 1)
{
    SCString OutputText;
    OutputText.Format("%f", PersistentValue);

    sc.AddMessageToLog(OutputText, 0);
}
}

/*=====
sc.ResetAllScales example function
-----*/
SCSFExport scsf_ResetScalesExample(SCStudyInterfaceRef sc)
{
    if (sc.SetDefaults)
    {
        // Set the configuration and defaults

        sc.GraphName = "sc.ResetAllScales Example";

        sc.AutoLoop = 0;

        return;
    }

    sc.ResetAllScales = 1;

}

/*=====*/
SCSFExport scsf_BidAndAskPrices(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_BidPrices = sc.Subgraph[0];
    SCSubgraphRef Subgraph_AskPrices = sc.Subgraph[1];

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

        sc.GraphName = "Bid & Ask Prices";

        sc.StudyDescription = "This study requires 1-tick data in the intraday data file.";

        sc.AutoLoop = 1;
        sc.MaintainAdditionalChartDataArrays = 1;
    }
}

```



```

sc.GraphRegion = 0;

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

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

return;
}

// Do data processing
if ((sc.Index < sc.ArraySize - 1) || sc.IsReplayRunning())
{
    Subgraph_BidPrices[sc.Index] = sc.BaseData[SC_BID_PRICE][sc.Index];
    Subgraph_AskPrices[sc.Index] = sc.BaseData[SC_ASK_PRICE][sc.Index];
}
else
{
    Subgraph_BidPrices[sc.Index] = sc.Bid;
    Subgraph_AskPrices[sc.Index] = sc.Ask;
}
}

/*=====
Study function to close a single specified chart in the chartbook.
-----*/
SCSFExport scsf_CloseChart(SCStudyInterfaceRef sc)
{
    SCInputRef Input_InChartNumber = sc.Input[0];

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

        sc.GraphName = "Close Chart";

        sc.StudyDescription
            = "Study function to close a single specified chart in the chartbook.";

        sc.AutoLoop = 0;

        sc.GraphRegion = 0;

        Input_InChartNumber.Name = "Chart To Close";
        Input_InChartNumber.SetChartNumber(0);

        return;
    }

    int ChartNumberToClose = Input_InChartNumber.GetChartNumber();

    if (ChartNumberToClose == 0)
        return;

    SCString Message;
    Message.Format("Closing chart #%.d.", ChartNumberToClose);
    sc.AddMessageToLog(Message, 0);

```

```

    sc.CloseChart(ChartNumberToClose);

    Input_InChartNumber.SetChartNumber(0);
}

/*=====
   Study function to close a single specified chartbook.
-----*/
SCSFExport scsf_CloseChartbook(SCStudyInterfaceRef sc)
{
    SCString& r_ChartbookFileName = sc.TextInput;

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

        sc.GraphName = "Close Chartbook";

        sc.StudyDescription
            = "Study function to close a single specified chartbook.";

        sc.AutoLoop = 0;


        sc.GraphRegion = 0;

        sc.TextInputName = "Chartbook File Name To Close";

        return;
    }

    if (r_ChartbookFileName.GetLength() == 0)
        return;

    SCString Message;
    Message.Format("Closing chartbook \"%s\".", r_ChartbookFileName.GetChars());
    sc.AddMessageToLog(Message, 0);

    sc.CloseChartbook(r_ChartbookFileName);

    r_ChartbookFileName = "";
}

/*=====*/
//
// PRC - Polynomial Regression Channel
//
// 20110329 - written by aslan
// 20100330 - added redraw option
//

SCSFExport scsf_PolynomialRegressionChannel(SCStudyInterfaceRef sc)
{
    SCInputRef Input_PolyDegree = sc.Input[0];
    SCInputRef Input_Period     = sc.Input[1];
    SCInputRef Input_StdDev1    = sc.Input[2];
    SCInputRef Input_StdDev2    = sc.Input[3];
    SCInputRef Input_Redraw     = sc.Input[4];

    SCSubgraphRef Subgraph_Fx   = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Upper2 = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Upper1 = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Lower1 = sc.Subgraph[3];

```

```

SCSubgraphRef Subgraph_Lower2 = sc.Subgraph[4];

if (sc.SetDefaults)
{
    sc.GraphName = "Polynomial Regression Channel";
    sc.StudyDescription = "Polynomial Regression Channel";

    Input_PolyDegree.Name = "Polynomial Degree (1-4)";
    Input_PolyDegree.SetInt(3);
    Input_PolyDegree.SetIntLimits(1, 4);

    Input_Period.Name = "PRC Period";
    Input_Period.SetInt(150);
    Input_Period.SetIntLimits(1, 10000);

    Input_StdDev1.Name = "Standard Deviation Level 1";
    Input_StdDev1.SetFloat(1.618f);
    Input_StdDev1.SetFloatLimits(0, 1000);

    Input_StdDev2.Name = "Standard Deviation Level 2";
    Input_StdDev2.SetFloat(2.0);
    Input_StdDev2.SetFloatLimits(0, 1000);

    Input_Redraw.Name = "Use Redraw";
    Input_Redraw.SetYesNo(1);

    Subgraph_Fx.Name = "Mid";
    Subgraph_Fx.DrawZeros = false;
    Subgraph_Fx.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_Fx.LineWidth = 1;
    Subgraph_Fx.LineStyle = LINESTYLE_DOT;
    Subgraph_Fx.PrimaryColor = COLOR_GRAY;

    Subgraph_Upper2.Name = "Upper 2";
    Subgraph_Upper2.DrawZeros = false;
    Subgraph_Upper2.DrawStyle = DRAWSTYLE_TRANSPARENT_FILL_TOP;
    Subgraph_Upper2.PrimaryColor = COLOR_RED;

    Subgraph_Upper1.Name = "Upper 1";
    Subgraph_Upper1.DrawZeros = false;
    Subgraph_Upper1.DrawStyle = DRAWSTYLE_TRANSPARENT_FILL_BOTTOM;
    Subgraph_Upper1.PrimaryColor = COLOR_RED;

    Subgraph_Lower1.Name = "Lower 1";
    Subgraph_Lower1.DrawZeros = false;
    Subgraph_Lower1.DrawStyle = DRAWSTYLE_TRANSPARENT_FILL_TOP;
    Subgraph_Lower1.PrimaryColor = COLOR_BLUE;

    Subgraph_Lower2.Name = "Lower 2";
    Subgraph_Lower2.DrawZeros = false;
    Subgraph_Lower2.DrawStyle = DRAWSTYLE_TRANSPARENT_FILL_BOTTOM;
    Subgraph_Lower2.PrimaryColor = COLOR_BLUE;

    sc.AutoLoop = true;
    sc.DrawZeros = false;
    sc.GraphRegion = 0;

    sc.GlobalDisplayStudySubgraphsNameAndValue = 0;
    sc.DisplayStudyName = 0;
    sc.DisplayStudyInputValues = 0;

    return;
}

```

```

struct s_PRCData
{
    double ai[10][10];
    double b[10];
    double x[10];
    double sx[20];
};

int& lastIndex = sc.GetPersistentInt(1);
int& nn      = sc.GetPersistentInt(2);
s_PRCData *p_PRCMemory = (s_PRCData *) sc.GetPersistentPointer(1);

if (sc.LastCallToFunction)
{
    // This study is being removed from the chart or the chart is being closed - cleanup
    if (p_PRCMemory != 0)
    {
        sc.FreeMemory(p_PRCMemory);
        p_PRCMemory = 0;
        sc.SetPersistentPointer(1, p_PRCMemory);
    }

    return;
}

if (sc.Index == 0)
{
    lastIndex = -1;
    sc.ValueFormat = sc.BaseGraphValueFormat;

    if (p_PRCMemory != 0)
        sc.FreeMemory(p_PRCMemory);

    p_PRCMemory = (s_PRCData*)sc.AllocateMemory(sizeof(s_PRCData));

    sc.SetPersistentPointer(1,p_PRCMemory);

    if (p_PRCMemory == 0)
        return;

    // set up static vars
    s_PRCData& PRCData = *p_PRCMemory;

    nn = Input_PolyDegree.GetInt() + 1;

    //-----SX-----
    PRCData.sx[1] = Input_Period.GetInt() + 1;
    for(int i=1;i<=nn*2-2;i++) {
        double sum = 0.0;
        for(int n=0;n<=Input_Period.GetInt();n++) {
            sum+=pow(static_cast<double>(n),i);
        }
        PRCData.sx[i+1]=sum;
    }
}

// make sure we have enough bars to start
if (sc.Index < Input_Period.GetInt())
    return;

// erase old data on new bar
if (sc.Index != lastIndex)
{

```

```

lastIndex = sc.Index;
if (Input_Redraw.GetYesNo())
{
    Subgraph_Fx[sc.Index-Input_Period.GetInt()-1] = 0;
    Subgraph_Upper2[sc.Index-Input_Period.GetInt()-1] = 0;
    Subgraph_Upper1[sc.Index-Input_Period.GetInt()-1] = 0;
    Subgraph_Lower1[sc.Index-Input_Period.GetInt()-1] = 0;
    Subgraph_Lower2[sc.Index-Input_Period.GetInt()-1] = 0;
}
}

// now lets do the real work
if (p_PRCMemory == 0)
    return;

s_PRCData& PRCData = *p_PRCMemory;

//-----syx-----
for(int i=1;i<=nn;i++) {
    double sum=0.0;
    for(int n=0;n<=Input_Period.GetInt();n++) {
        if(i==1) sum+=sc.Close[sc.Index-n];
        else sum+=sc.Close[sc.Index-n]*pow(static_cast<double>(n),i-1);
    }
    PRCData.b[i]=sum;
}

//=====Matrix=====
for(int j=1;j<=nn;j++) {
    for(int i=1; i<=nn; i++) {
        int k=i+j-1;
        PRCData.ai[i][j]=PRCData.sx[k]; // reset ai matrix
    }
}

//=====Gauss=====
for(int k=1; k<=nn-1; k++) {
    int l=0;
    double m=0.0;
    for(int i=k; i<=nn; i++) {
        if(abs(PRCData.ai[i][k])>m) {
            m=abs(PRCData.ai[i][k]);
            l=i;
        }
    }
    if(l==0) return;
    if (l!=k) {
        double tt = 0.0;
        for(int j=1; j<=nn; j++) {
            tt=PRCData.ai[k][j];
            PRCData.ai[k][j]=PRCData.ai[l][j];
            PRCData.ai[l][j]=tt;
        }
        tt=PRCData.b[k];
        PRCData.b[k]=PRCData.b[l];
        PRCData.b[l]=tt;
    }
    for(int i=k+1;i<=nn;i++) {
        double qq=PRCData.ai[i][k]/PRCData.ai[k][k];
        for(int j=1;j<=nn;j++) {
            if(j==k) PRCData.ai[i][j]=0;
            else PRCData.ai[i][j]=PRCData.ai[i][j]-qq*PRCData.ai[k][j];
        }
        PRCData.b[i]=PRCData.b[i]-qq*PRCData.b[k];
    }
}
PRCData.x[nn]=PRCData.b[nn]/PRCData.ai[nn][nn];

```

```

for(int i=nn-1;i>=1;i--) {
    double tt=0.0;
    for(int j=1;j<=nn-i;j++) {
        tt=tt+PRCData.ai[i][j]*PRCData.x[i+j];
        PRCData.x[i]=(1/PRCData.ai[i][i])*(PRCData.b[i]-tt);
    }
}
//=====
for(int n=0;n<=Input_Period.GetInt();n++) {
    double sum=0.0;
    for(int k=1;k<=Input_PolyDegree.GetInt();k++) {
        sum+=PRCData.x[k+1]*pow(static_cast<double>(n),k);
    }
    if (n==0 || Input_Redraw.GetYesNo())
        Subgraph_Fx[sc.Index-n] = static_cast<float>(PRCData.x[1]+sum);
}
//-----Std-----
double sq=0.0, sq1=0.0, sq2=0.0;
for(int n=0;n<=Input_Period.GetInt();n++) {
    sq+=pow(sc.Close[sc.Index-n]-Subgraph_Fx[sc.Index-n],2);
}
sq=sqrt(sq/(Input_Period.GetInt()+1));
sq1=sq*Input_StdDev1.GetFloat();
sq2=sq*Input_StdDev2.GetFloat();

for(int n=0;n<=Input_Period.GetInt();n++) {
    if (n==0 || Input_Redraw.GetYesNo())
    {
        Subgraph_Upper1[sc.Index-n] = static_cast<float>(Subgraph_Fx[sc.Index-n]+sq1);
        Subgraph_Lower1[sc.Index-n] = static_cast<float>(Subgraph_Fx[sc.Index-n]-sq1);
        Subgraph_Upper2[sc.Index-n] = static_cast<float>(Subgraph_Fx[sc.Index-n]+sq2);
        Subgraph_Lower2[sc.Index-n] = static_cast<float>(Subgraph_Fx[sc.Index-n]-sq2);
    }
}
}

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

    SCInputRef Input_AverageFormula = sc.Input[0];

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

        sc.GraphName = "Average Price For Bar";
        sc.GraphRegion = 0;
        sc.AutoLoop = 1;

        Subgraph_Average.Name = "Average";
        Subgraph_Average.DrawStyle = DRAWSTYLE_DASH;
        Subgraph_Average.LineWidth = 2;
        Subgraph_Average.PrimaryColor = RGB(255, 128, 0);

        Input_AverageFormula.Name = "Average Formula";
        Input_AverageFormula.SetCustomInputStrings
            ( "(High+Low)/2"
              ";(High+Low+Close)/3"
              ";(Open+High+Low+Close)/4"
              ";(High+Low+Close+Close)/4"
            );
        Input_AverageFormula.SetCustomInputIndex(0);
    }
}

```

```

    return;
}

// Do data processing.

const int SelectedIndex = Input_AverageFormula.GetIndex();
switch (SelectedIndex)
{
    case 0:
    {
        Subgraph_Average.Data[sc.Index]
            = (sc.High[sc.Index] + sc.Low[sc.Index]) / 2.0f;
    }
    break;

    case 1:
    {
        Subgraph_Average.Data[sc.Index]
            = (sc.High[sc.Index] + sc.Low[sc.Index] + sc.Close[sc.Index])
              / 3.0f;
    }
    break;

    case 2:
    {
        Subgraph_Average.Data[sc.Index]
            = (sc.Open[sc.Index]
              + sc.High[sc.Index]
              + sc.Low[sc.Index]
              + sc.Close[sc.Index]
              )
              / 4.0f;
    }
    break;

    case 3:
    {
        Subgraph_Average.Data[sc.Index]
            = (sc.High[sc.Index]
              + sc.Low[sc.Index]
              + sc.Close[sc.Index] * 2
              )
              / 4.0f;
    }
    break;
}
}

/*=====
-----*/
SCSFExport scsf_ActionWhenTimeEncountered(SCStudyInterfaceRef sc)
{
    if (sc.SetDefaults)
    {
        // Set the configuration and defaults

        sc.GraphName = "Action When Time is Encountered";

        sc.StudyDescription = "This study function demonstrates performing an action when a certain time is encountered in the most recent chart bar.";

        sc.AutoLoop = 1;
    }
}

```

```

    return;
}

// Do data processing

SCDateTime TimeToCheckFor;

//The first step is to get the current date.
int CurrentDate = sc.BaseDateTimeln[sc.ArraySize - 1].GetDate();

//Apply the time. For this example we will use 12 PM
TimeToCheckFor.SetDate(CurrentDate);
TimeToCheckFor.SetTimeHMS(12, 0, 0);

// TimeToCheckFor is contained within the current bar.
if (sc.IsDateTimeContainedInBarIndex(TimeToCheckFor, sc.Index))
{
    //perform the action here
}

}

/*=====
Symbol Display on chart
-----*/
SCSFExport scsf_SymbolDisplay(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_SymbolText = sc.Subgraph[0];

    SCInputRef Input_HorizontalPosition = sc.Input[0];
    SCInputRef Input_VerticalPosition = sc.Input[1];
    SCInputRef Input_LockPosition = sc.Input[2];
    SCInputRef Input_TransparentLabelBackground = sc.Input[3];
    SCInputRef Input_UseSymbolDescription = sc.Input[5];
    SCInputRef Input_DrawAboveMainPriceGraph = sc.Input[6];
    SCInputRef Input_AlsoDisplayTradeAndCurrentQuoteSymbol = sc.Input[7];

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

        sc.GraphName = "Symbol Display";
        sc.AutoLoop = 0; //Manual looping
        sc.GraphRegion = 0;
        sc.ValueFormat = 0;

        Subgraph_SymbolText.Name = "Symbol Text";
        Subgraph_SymbolText.LineWidth = 20;
        Subgraph_SymbolText.DrawStyle = DRAWSTYLE_CUSTOM_TEXT;
        Subgraph_SymbolText.PrimaryColor = RGB(0, 0, 0); //black
        Subgraph_SymbolText.SecondaryColor = RGB(255, 127, 0); //Orange
        Subgraph_SymbolText.SecondaryColorUsed = true;
        Subgraph_SymbolText.DisplayNameValueInWindowsFlags = 1;

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

        Input_VerticalPosition.Name.Format("Initial Vertical Position From Bottom (1-%d)", static_cast<int>

```



```

(CHART_DRAWING_MAX_VERTICAL_AXIS_RELATIVE_POSITION));
    Input_VerbalPosition.SetInt(90);
    Input_VerbalPosition.SetIntLimits(1, static_cast<int>
(CHART_DRAWING_MAX_VERTICAL_AXIS_RELATIVE_POSITION));

    Input_LockPosition.Name = "Lock Symbol Position";
    Input_LockPosition.SetYesNo(false);

    Input_TransparentLabelBackground.Name = "Transparent Label Background";
    Input_TransparentLabelBackground.SetYesNo(false);

    Input_UseSymbolDescription.Name = "Use Symbol Description";
    Input_UseSymbolDescription.SetYesNo(false);

    Input_DrawAboveMainPriceGraph.Name = "Draw Above Main Price Graph";
    Input_DrawAboveMainPriceGraph.SetYesNo(false);

    Input_AlsoDisplayTradeAndCurrentQuoteSymbol.Name = "Also Display Trade and Current Quote Symbol";
    Input_AlsoDisplayTradeAndCurrentQuoteSymbol.SetYesNo(false);

    sc.TextInputName = "Alternate Symbol Text";

    return;
}

SCString& PriorTextToDisplay = sc.GetPersistentSCString(1);

SCString SymbolDescription;
sc.GetSymbolDescription(SymbolDescription);

if (SymbolDescription.IsEmpty())
    SymbolDescription = "(none)";

SCString TextToDisplay;
if (Input_UseSymbolDescription.GetYesNo())
    TextToDisplay = SymbolDescription;
else if (sc.TextInput.GetLength() > 0)
    TextToDisplay = sc.TextInput;
else
    TextToDisplay = sc.Symbol;

if (Input_AlsoDisplayTradeAndCurrentQuoteSymbol.GetYesNo() && sc.TradeAndCurrentQuoteSymbol.GetLength() >
0)
{
    TextToDisplay += " ";
    TextToDisplay += sc.TradeAndCurrentQuoteSymbol;
}

bool TextHasChanged = PriorTextToDisplay != TextToDisplay;

if (!TextHasChanged && !sc.IsFullRecalculation && !sc.LastCallToFunction)
    return;

PriorTextToDisplay = TextToDisplay;

int HorizontalPosition = Input_HorizontalPosition.GetInt();
int VerticalPosition = Input_VerbalPosition.GetInt();

int DrawAboveMainPriceGraph = Input_DrawAboveMainPriceGraph.GetYesNo();
int TransparentLabelBackground = Input_TransparentLabelBackground.GetYesNo();
int LockDrawing = Input_LockPosition.GetYesNo();

sc.AddAndManageSingleTextUserDrawnDrawingForStudy(sc, false, HorizontalPosition, VerticalPosition,
Subgraph_SymbolText, TransparentLabelBackground, TextToDisplay, DrawAboveMainPriceGraph, LockDrawing);

```

```

}

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

    SCInputRef Input_HorizontalPosition = sc.Input[0];
    SCInputRef Input_VerticalPosition = sc.Input[1];
    SCInputRef Input_DrawAboveMainPriceGraph = sc.Input[2];
    SCInputRef Input_TransparentLabelBackground = sc.Input[3];

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

        sc.GraphName = "Trading Profit Management Status";
        sc.AutoLoop = 0; //Manual looping
        sc.GraphRegion = 0;
        sc.ValueFormat = 0;

        Subgraph_StatusText.Name = "Status Text";
        Subgraph_StatusText.LineWidth = 10;
        Subgraph_StatusText.DrawStyle = DRAWSTYLE_CUSTOM_TEXT;
        Subgraph_StatusText.PrimaryColor = RGB(255, 255, 255);
        Subgraph_StatusText.SecondaryColor = RGB(0, 0, 160);
        Subgraph_StatusText.SecondaryColorUsed = true;
        Subgraph_StatusText.DisplayNameValueInWindowsFlags = 1;

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

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

        Input_DrawAboveMainPriceGraph.Name = "Draw Above Main Price Graph";
        Input_DrawAboveMainPriceGraph.SetYesNo(false);

        Input_TransparentLabelBackground.Name = "Transparent Label Background";
        Input_TransparentLabelBackground.SetYesNo(false);

        return;
    }

    // Do data processing
    SCString TextToDisplay;
    sc.GetProfitManagementStringForTradeAccount(TextToDisplay);

    int HorizontalPosition = Input_HorizontalPosition.GetInt();
    int VerticalPosition = Input_VerticalPosition.GetInt();

    int DrawAboveMainPriceGraph = Input_DrawAboveMainPriceGraph.GetYesNo();
    int TransparentLabelBackground = Input_TransparentLabelBackground.GetYesNo();

    sc.AddAndManageSingleTextUserDrawnDrawingForStudy(sc, false, HorizontalPosition, VerticalPosition,

```

```
Subgraph_StatusText, TransparentLabelBackground, TextToDisplay, DrawAboveMainPriceGraph, 0);
```

```
}
```

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

```
SCSFExport scsf_TimeAndSalesIterationExample(SCStudyInterfaceRef sc)
```

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

        sc.GraphName = "Time and Sales Iteration Example";

        sc.StudyDescription = "This is an example of iterating through the time and sales records which have been added
since the last study function call.";

        sc.AutoLoop = 0; //Use manual looping

        sc.GraphRegion = 0;

        return;
    }
}
```

```
//This code depends upon manual looping being set
```

```
int64_t& LastProcessedSequence = sc.GetPersistentInt64(1);
```

```
//reset the sequence number on a full recalculation so we start fresh for each full recalculation.
```

```
if (sc.IsFullRecalculation && sc.UpdateStartIndex == 0)
    LastProcessedSequence = 0;
```

```
// Get the Time and Sales
```

```
c_SCTimeAndSalesArray TimeSales;
```

```
sc.GetTimeAndSales(TimeSales);
```

```
if (TimeSales.Size() == 0)
    return; // No Time and Sales data available for the symbol
```

```
//Set the initial sequence number
```

```
if (LastProcessedSequence == 0)
    LastProcessedSequence = TimeSales[TimeSales.Size() - 1].Sequence;
```

```
// Loop through the Time and Sales.
```

```
for (int TSIndex = 0; TSIndex < TimeSales.Size(); ++TSIndex)
{
    //do not reprocess previously processed sequence numbers.
    if (TimeSales[TSIndex].Sequence <= LastProcessedSequence)
        continue;
```

```
//only interested in trade records
```

```
if (TimeSales[TSIndex].Type == SC_TS_BID || TimeSales[TSIndex].Type == SC_TS_ASK)
{
```

```
    float TradePrice = TimeSales[TSIndex].Price;
```

```
    float BidPrice = TimeSales[TSIndex].Bid;
```

```
    float AskPrice = TimeSales[TSIndex].Ask;
```

```
    SCDateTime RecordAdjustedDateTime = TimeSales[TSIndex].DateTime;
```

```
    // Apply the time zone offset for the chart. This will result in the actual date-time of the record in the charts time
    zone.
```

```
    RecordAdjustedDateTime += sc.TimeScaleAdjustment;
```

```

    }
}

}

/*=====*/
SCSFExport scsf_ServerConnectionStateExample(SCStudyInterfaceRef sc)
{
    if (sc.SetDefaults)
    {
        // Set the configuration and defaults

        sc.GraphName = "ServerConnectionState Example";

        sc.AutoLoop = 0;

        return;
    }

    SCString MessageText = "Current Data/Trade Service connection state: ";

    switch (sc.ServerConnectionState)
    {
    case SCS_DISCONNECTED:
        MessageText += "SCS_DISCONNECTED";
        break;

    case SCS_CONNECTING:
        MessageText += "SCS_CONNECTING";
        break;

    case SCS_CONNECTED:
        MessageText += "SCS_CONNECTED";
        break;

    case SCS_CONNECTION_LOST:
        MessageText += "SCS_CONNECTION_LOST";
        break;

    case SCS_DISCONNECTING:
        MessageText += "SCS_DISCONNECTING";
        break;

    case SCS_RECONNECTING:
        MessageText += "SCS_RECONNECTING";
        break;

    default:
        MessageText += "INVALID";
        break;
    }

    sc.AddMessageToLog(MessageText, 0);
}

/*=====*/
SCSFExport scsf_StudySubgraphDailyHighLow(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_HighOfDay = sc.Subgraph[0];
    SCSubgraphRef Subgraph_LowOfDay = sc.Subgraph[1];
    SCInputRef Input_StudySubgraphReference = sc.Input[0];

```

```

if (sc.SetDefaults)
{
    sc.GraphName          = "Study Subgraph Daily High Low";
    sc.GraphRegion        = 1;
    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;

    sc.CalculationPrecedence = LOW_PREC_LEVEL;

    Input_StudySubgraphReference.Name = "Study Subgraph Reference";

    Input_StudySubgraphReference.SetStudySubgraphValues(1, 0);

    return;
}

float& r_CurrentDailyHigh = sc.GetPersistentFloat(1);
float& r_CurrentDailyLow = sc.GetPersistentFloat(2);

//Loop
for (int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize; BarIndex++)
{

    if (BarIndex > 0)
    {
        SCDateTime PreviousTradingDayStartDateTimeOfBar = sc.GetTradingDayStartDateTimeOfBar
(sc.BaseDateTimeln[BarIndex - 1]);
        SCDateTime TradingDayStartDateTimeOfBar = sc.GetTradingDayStartDateTimeOfBar
(sc.BaseDateTimeln[BarIndex]);

        if (PreviousTradingDayStartDateTimeOfBar != TradingDayStartDateTimeOfBar)
        {
            r_CurrentDailyHigh = 0;
            r_CurrentDailyLow = 0;
        }
    }
    else
    {
        r_CurrentDailyHigh = 0;
        r_CurrentDailyLow = 0;
    }

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

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

    if (r_CurrentDailyHigh == 0 || r_CurrentDailyHigh < StudySubgraph[BarIndex])
        r_CurrentDailyHigh = StudySubgraph[BarIndex];

    if (r_CurrentDailyLow == 0 || r_CurrentDailyLow > StudySubgraph[BarIndex])
        r_CurrentDailyLow = StudySubgraph[BarIndex];
}

```

```

        Subgraph_HighOfDay[BarIndex] = r_CurrentDailyHigh;
        Subgraph_LowOfDay[BarIndex] = r_CurrentDailyLow;
    }
}

/*=====*/
SCSFExport scsf_TimeSeriesForecast(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_TimeSeriesForecast = sc.Subgraph[0];
    SCFloatArrayRef Array_LinearRegressionSlope = sc.Subgraph[0].Arrays[0];
    SCFloatArrayRef Array_LinearRegressionIndicator = sc.Subgraph[0].Arrays[1];

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

    if (sc.SetDefaults)
    {
        sc.GraphName = "Time Series Forecast";

        sc.AutoLoop = 1;

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

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

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

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

        return;
    }

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

    int CalculationLength=Input_Length.GetInt();
    double SumBars = CalculationLength*(CalculationLength-1)/2.0;
    double SumSqrBars = (CalculationLength-1)*CalculationLength*(2*CalculationLength-1)/6.0;

    double Sum1 = 0;
    double SumY = 0;
    for(int Index=sc.Index; Index > (sc.Index - CalculationLength); Index--)
    {
        int IndexDifference = sc.Index - Index;
        Sum1 = Sum1 + IndexDifference * sc.BaseData[Input_Data.GetInputDataIndex()][Index];
        SumY += sc.BaseData[Input_Data.GetInputDataIndex()][Index];
    }

    double Sum2 = SumBars * SumY;
    double Num1 = CalculationLength*Sum1 - Sum2;
    double Num2 = SumBars*SumBars - CalculationLength*SumSqrBars;
    Array_LinearRegressionSlope[sc.Index] = static_cast<float>(Num1 / Num2);

    sc.LinearRegressionIndicator(sc.BaseData[Input_Data.GetInputDataIndex()], Array_LinearRegressionIndicator,
CalculationLength);

    Subgraph_TimeSeriesForecast[sc.Index] = Array_LinearRegressionIndicator[sc.Index] +
Array_LinearRegressionSlope[sc.Index];

```

```

    return;
}

/*=====*/
SCSFExport scsf_ExtendClosesUntilFutureIntersection(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_LineProperties = sc.Subgraph[0];
    SCInputRef Input_DisplayValueLabel = sc.Input[0];
    SCInputRef Input_NameLabel = sc.Input[1];

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

        sc.GraphName = "Extend Closes Until Future Intersection";

        Subgraph_LineProperties.Name = "Line Properties";
        Subgraph_LineProperties.DrawStyle = DRAWSTYLE_SUBGRAPH_NAME_AND_VALUE_LABELS_ONLY;
        Subgraph_LineProperties.LineWidth = 1;
        Subgraph_LineProperties.PrimaryColor = COLOR_RED;
        Subgraph_LineProperties.DrawZeros = false;

        Input_DisplayValueLabel.Name = "Display Value Label";
        Input_DisplayValueLabel.SetYesNo(true);

        Input_NameLabel.Name = "Name Label";
        Input_NameLabel.SetString("");

        sc.ValueFormat = VALUEFORMAT_INHERITED;

        sc.GraphRegion = 0;

        sc.AutoLoop = 1;

        return;
    }

    if (sc.Index == 0)
        return;

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

    const SCDateTime PriorBarStartDateTimeForDay
        = sc.GetTradingDayStartDateTimeOfBar(sc.BaseDateTimeIn[sc.Index - 1]);

    const bool IsNewDay = (CurrentBarStartDateTimeForDay != PriorBarStartDateTimeForDay);

    if (!IsNewDay)
        return;

    const float ClosePriceOfPriorBar = sc.Close[sc.Index - 1];
    const char* NameLabelString = Input_NameLabel.GetString();

    bool DisplayNameLabel = false;

    if (NameLabelString != NULL && strlen(NameLabelString) > 0)
        DisplayNameLabel = true;
    else
        NameLabelString = "";

```

```

sc.AddLineUntilFutureIntersection
( sc.Index - 1
, 0 // LineIDForBar
, ClosePriceOfPriorBar
, Subgraph_LineProperties.PrimaryColor
, Subgraph_LineProperties.LineWidth
, Subgraph_LineProperties.LineStyle
, Input_DisplayValueLabel.GetYesNo()
, DisplayNameLabel
, NameLabelString
);
}

/*=====*/
SCSFExport scsf_ExtendBarValueUntilFutureIntersection(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_LineProperties = sc.Subgraph[0];
    SCInputRef Input_Data = sc.Input[0];
    SCInputRef Input_DisplayValueLabel = sc.Input[1];
    SCInputRef Input_NameLabel = sc.Input[2];

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

        sc.GraphName = "Extend Bar Value Until Future Intersection";

        Subgraph_LineProperties.Name = "Line Properties";
        Subgraph_LineProperties.DrawStyle = DRAWSTYLE_SUBGRAPH_NAME_AND_VALUE_LABELS_ONLY;
        Subgraph_LineProperties.LineWidth = 1;
        Subgraph_LineProperties.PrimaryColor = COLOR_RED;
        Subgraph_LineProperties.DrawZeros = false;

        Input_Data.Name = "Input Data from Bar for Extension Lines";
        Input_Data.SetInputDataIndex(SC_LAST);

        Input_DisplayValueLabel.Name = "Display Value Label";
        Input_DisplayValueLabel.SetYesNo(true);

        Input_NameLabel.Name = "Name Label";
        Input_NameLabel.SetString("");

        sc.ValueFormat = VALUEFORMAT_INHERITED;

        sc.GraphRegion = 0;

        sc.AutoLoop = 1;

        return;
    }

    const char* NameLabelString = Input_NameLabel.GetString();

    bool DisplayNameLabel = false;

    if (NameLabelString != NULL && strlen(NameLabelString) > 0)
        DisplayNameLabel = true;
    else
        NameLabelString = "";

    sc.AddLineUntilFutureIntersection

```



```

    ( sc.Index
    , 0 // LineIDForBar
    , sc.BaseData[ Input_Data.GetInputDataIndex()][sc.Index]
    , Subgraph_LineProperties.PrimaryColor
    , Subgraph_LineProperties.LineWidth
    , Subgraph_LineProperties.LineStyle
    , Input_DisplayValueLabel.GetYesNo()
    , DisplayNameLabel
    , NameLabelString
    );
}

/*=====*/
SCSFExport scsf_ExtendBarGapUntilFutureIntersection(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_LinePropertiesSubgraph = sc.Subgraph[0];
    SCInputRef Input_DisplayValueLabel = sc.Input[0];
    SCInputRef Input_NameLabel = sc.Input[1];
    SCInputRef Input_CompareToOpentoCloseRangeOnly = sc.Input[2];

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

        sc.GraphName = "Extend Bar Gap Until Future Intersection";

        Subgraph_LinePropertiesSubgraph.Name = "Line Properties";
        Subgraph_LinePropertiesSubgraph.DrawStyle =
DRAWSTYLE_SUBGRAPH_NAME_AND_VALUE_LABELS_ONLY;
        Subgraph_LinePropertiesSubgraph.LineWidth = 1;
        Subgraph_LinePropertiesSubgraph.PrimaryColor = COLOR_RED;
        Subgraph_LinePropertiesSubgraph.DrawZeros = false;

        Input_DisplayValueLabel.Name = "Display Value Label";
        Input_DisplayValueLabel.SetYesNo(true);

        Input_NameLabel.Name = "Name Label";
        Input_NameLabel.SetString("");

        Input_CompareToOpentoCloseRangeOnly.Name = "Compare to Open to Close Range Only";
        Input_CompareToOpentoCloseRangeOnly.SetYesNo(false);

        sc.ValueFormat = VALUEFORMAT_INHERITED;

        sc.GraphRegion = 0;

        sc.AutoLoop = 1;

        return;
    }

    if (sc.Index == 0)
        return;

    float ExtendPrice = 0.0;

    if (!Input_CompareToOpentoCloseRangeOnly.GetYesNo())
    {
        //Check for up gap
        if (sc.FormattedEvaluate(sc.High[sc.Index - 1], sc.BaseGraphValueFormat, LESS_OPERATOR, sc.Low[sc.Index],
sc.BaseGraphValueFormat))
        {

```

```

        ExtendPrice = sc.High[sc.Index - 1];
    }
    //Check for down gap
    else if (sc.FormattedEvaluate(sc.Low[sc.Index - 1], sc.BaseGraphValueFormat, GREATER_OPERATOR,
sc.High[sc.Index], sc.BaseGraphValueFormat))
    {
        ExtendPrice = sc.Low[sc.Index - 1];
    }
    else
    {
        return;
    }
}
else
{
    float PriorMaxValue = max(sc.Open[sc.Index - 1], sc.Close[sc.Index - 1]);
    float PriorMinValue = min(sc.Open[sc.Index - 1], sc.Close[sc.Index - 1]);
    float MaxValue = max(sc.Open[sc.Index], sc.Close[sc.Index]);
    float MinValue = min(sc.Open[sc.Index], sc.Close[sc.Index]);

    //Check for up gap
    if (sc.FormattedEvaluate(PriorMaxValue, sc.BaseGraphValueFormat, LESS_OPERATOR, MinValue,
sc.BaseGraphValueFormat))
    {
        ExtendPrice = PriorMaxValue;
    }
    //Check for down gap
    else if (sc.FormattedEvaluate(PriorMinValue, sc.BaseGraphValueFormat, GREATER_OPERATOR, MaxValue,
sc.BaseGraphValueFormat))
    {
        ExtendPrice = PriorMinValue;
    }
    else
    {
        return;
    }
}

SCString NameLabelString = Input_NameLabel.GetString();

bool DisplayNameLabel = false;

if (NameLabelString != NULL && strlen(NameLabelString) > 0)
    DisplayNameLabel = true;
else
    NameLabelString.Clear();

sc.AddLineUntilFutureIntersection
( sc.Index - 1
, 0 // LineIDForBar. Use 0 since there is only one line per bar.
, ExtendPrice
, Subgraph_LinePropertiesSubgraph.PrimaryColor
, Subgraph_LinePropertiesSubgraph.LineWidth
, Subgraph_LinePropertiesSubgraph.LineStyle
, Input_DisplayValueLabel.GetYesNo()
, DisplayNameLabel
, NameLabelString
);
}
/*=====*/
SCSFExport scsf_MarketDepthPullingStackingValuesExample(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_BidPullingStacking = sc.Subgraph[0];
    SCSubgraphRef Subgraph_AskPullingStacking = sc.Subgraph[1];

```

```

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

    sc.GraphName = "Market Depth Pulling/Stacking Values Example";

    sc.GraphRegion = 1;

    sc.AutoLoop = 0;


    Subgraph_BidPullingStacking.Name = "Bid";
    Subgraph_BidPullingStacking.DrawStyle = DRAWSTYLE_LINE;

    Subgraph_AskPullingStacking.Name = "Ask";
    Subgraph_AskPullingStacking.DrawStyle = DRAWSTYLE_LINE;

    return;
}


//The following two functions will return 0 if there is no pulling and stacking value at the specified price
Subgraph_BidPullingStacking[sc.ArraySize - 1] = static_cast<float>
(sc.GetBidMarketDepthStackPullValueAtPrice(sc.Bid));
Subgraph_AskPullingStacking[sc.ArraySize - 1] = static_cast<float>
(sc.GetAskMarketDepthStackPullValueAtPrice(sc.Ask));
}

/*=====*/

SCSFExport scsf_Repulse(SCStudyGraphRef sc)
{
    SCSubgraphRef Subgraph_Repulse = sc.Subgraph[0];
    SCInputRef Input_Length = sc.Input[0];

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

        sc.GraphName = "Repulse";

        sc.AutoLoop = 1;

        sc.GraphRegion = 1;

        Subgraph_Repulse.Name = "Repulse";
        Subgraph_Repulse.PrimaryColor = RGB(0, 255, 0);
        Subgraph_Repulse.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Repulse.LineWidth = 2;

        Input_Length.Name = "Length";
        Input_Length.SetInt(10);

        return;
    }

    // Do data processing
    /*
    lo = LOWEST(LOW)
    hi = HIGHEST(HIGH)
    a = 100 * ( 3 * CLOSE - 2 * lo - OPEN[p-1] ) / CLOSE
    b = 100 * ( OPEN[p-1] + 2 * hi - 3 * CLOSE ) / CLOSE
    d = EXPONENTIALAVERAGE[5*p](a) - EXPONENTIALAVERAGE[5*p](b)

```

```

RETURN d*/
int Length = Input_Length.GetInt();

sc.Highest(sc.High, Subgraph_Repulse.Arrays[0], Length);
sc.Lowest(sc.Low, Subgraph_Repulse.Arrays[1], Length);

Subgraph_Repulse.Arrays[2][sc.Index] = 100 * (3 * sc.Close[sc.Index] - 2 * Subgraph_Repulse.Arrays[1][sc.Index] -
sc.Open[sc.Index - Length + 1]) / sc.Close[sc.Index];

Subgraph_Repulse.Arrays[3][sc.Index] = 100 * (sc.Open[sc.Index - Length + 1] + 2 * Subgraph_Repulse.Arrays[0]
[sc.Index] - 3 * sc.Close[sc.Index]) / sc.Close[sc.Index];


sc.MovingAverage(Subgraph_Repulse.Arrays[2], Subgraph_Repulse.Arrays[4], MOVAVGTYPE_EXPONENTIAL,
5*Length);
sc.MovingAverage(Subgraph_Repulse.Arrays[3], Subgraph_Repulse.Arrays[5], MOVAVGTYPE_EXPONENTIAL,
5*Length);

Subgraph_Repulse.Data[sc.Index] = Subgraph_Repulse.Arrays[4][sc.Index] - Subgraph_Repulse.Arrays[5][sc.Index];
}

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

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

    if (sc.SetDefaults)
    {
        sc.GraphName = "Full Contract Value";
        sc.ValueFormat = VALUEFORMAT_2_DIGITS;
        sc.AutoLoop = 0; // Needed when using sc.GetCalculationStartIndexForStudy

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

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

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

        return;
    }

    Subgraph_Result.DrawZeros = Input_DrawZeros.GetYesNo();

    int CalculationStartIndex = sc.GetCalculationStartIndexForStudy();

    for (int Index = CalculationStartIndex; Index < sc.ArraySize; Index++)
    {
        Subgraph_Result[Index] = sc.BaseData[Input_Data.GetInputDataIndex()][Index] / sc.TickSize *
max(sc.CurrencyValuePerTick, sc.TickSize);
    }

    sc.EarliestUpdateSubgraphDataArrayIndex = CalculationStartIndex;
}

```

```

/*=====*/
SCSFExport scsf_KeyboardModifierStatesExample(SCStudyInterfaceRef sc)
{
    if (sc.SetDefaults)
    {
        // Set the configuration and defaults

        sc.GraphName = "Keyboard Modifier States Example";

        sc.AutoLoop = 0;///Manual looping

        sc.GraphRegion = 0;

        sc.UpdateAlways = 1;
        sc.SupportKeyboardModifierStates = 1;

        return;
    }

    if (sc.IsKeyPressed_Alt)
    {
        sc.AddMessageToLog("Alt key pressed", 0);
    }

    if (sc.IsKeyPressed_Shift)
    {
        sc.AddMessageToLog("Shift key pressed", 0);
    }

    if (sc.IsKeyPressed_Control)
    {
        sc.AddMessageToLog("Control key pressed", 0);
    }
}
/*=====*/

SCSFExport scsf_BalanceOfPower(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_BalanceOfPower = sc.Subgraph[0];
    SCSubgraphRef Subgraph_BalanceOfPowerAverage = sc.Subgraph[1];

    SCInputRef Input_MovingAverageLength = sc.Input[0];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Balance of Power";
        sc.GraphRegion = 1;
        sc.ValueFormat = 2;

        Subgraph_BalanceOfPower.Name = "BOP";
        Subgraph_BalanceOfPower.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_BalanceOfPower.PrimaryColor = RGB(0, 255, 0);
        Subgraph_BalanceOfPower.LineWidth = 2;

        Subgraph_BalanceOfPowerAverage.Name = "BOP Average";
        Subgraph_BalanceOfPowerAverage.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_BalanceOfPowerAverage.PrimaryColor = RGB(0, 0, 255);
        Subgraph_BalanceOfPowerAverage.LineWidth = 2;

        Input_MovingAverageLength.Name = "Moving Average Length";
        Input_MovingAverageLength.SetInt(10);
        Input_MovingAverageLength.SetIntLimits(1, MAX_STUDY_LENGTH);
        sc.AutoLoop = 1;
    }
}

```

```

    return;
}

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

Subgraph_BalanceOfPower.Data[sc.Index] = (sc.BaseData[SC_LAST][sc.Index] - sc.BaseData[SC_OPEN][sc.Index]) /
(sc.BaseData[SC_HIGH][sc.Index] - sc.BaseData[SC_LOW][sc.Index]);

sc.SimpleMovAvg(Subgraph_BalanceOfPower, Subgraph_BalanceOfPowerAverage,
Input_MovingAverageLength.GetInt());
}
/*=====*/
SCSFExport scsf_MACDBollingerBandsImproved(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_MACD = sc.Subgraph[0];
    SCSubgraphRef Subgraph_MovAvgOfMACD = sc.Subgraph[1];
    SCSubgraphRef Subgraph_MACDDiff = sc.Subgraph[2];
    SCSubgraphRef Subgraph_TopBand = sc.Subgraph[3];
    SCSubgraphRef Subgraph_BottomBand = sc.Subgraph[4];
    SCSubgraphRef Subgraph_RefLine = sc.Subgraph[5];

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

    if (sc.SetDefaults)
    {
        sc.GraphName = "MACD Bollinger Bands - Improved";
        sc.AutoLoop = 1;

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

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

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

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

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

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

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

```

```

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

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

Input_NumberOfStdDevs.Name = "Number of Standard Deviations";
Input_NumberOfStdDevs.SetInt(1);
Input_NumberOfStdDevs.SetIntLimits(1, MAX_STUDY_LENGTH);

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

return;
}

sc.DataStartIndex = 2 * Input_MACDLength.GetInt() + max(Input_FastLength.GetInt(), Input_SlowLength.GetInt());

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

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

Subgraph_MACD.Arrays[6][sc.Index] = fabs(Subgraph_MACDDiff[sc.Index]);

sc.MovingAverage(Subgraph_MACD.Arrays[6], Subgraph_MACD.Arrays[7],
Input_MovingAverageType.GetMovAvgType(), Input_MACDLength.GetInt());

float MovAvgAbsValMACDDiff = Subgraph_MACD.Arrays[7][sc.Index];

sc.StdDeviation(Subgraph_MACD.Arrays[6], Subgraph_MACD.Arrays[8], Input_MACDLength.GetInt());

float StdDevAbsValMACDDiff = Subgraph_MACD.Arrays[8][sc.Index];

Subgraph_TopBand[sc.Index] = Subgraph_MovAvgOfMACD[sc.Index] + MovAvgAbsValMACDDiff +
Input_NumberOfStdDevs.GetInt()*StdDevAbsValMACDDiff;
Subgraph_BottomBand[sc.Index] = Subgraph_MovAvgOfMACD[sc.Index] - MovAvgAbsValMACDDiff -
Input_NumberOfStdDevs.GetInt()*StdDevAbsValMACDDiff;

Subgraph_RefLine[sc.Index] = 0;
}
/*=====*/
SCSFExport scsf_MACDBollingerBandsStandard(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_MACD = sc.Subgraph[0];
    SCSubgraphRef Subgraph_MovAvgOfMACD = sc.Subgraph[1];
    SCSubgraphRef Subgraph_TopBand = sc.Subgraph[2];
    SCSubgraphRef Subgraph_BottomBand = sc.Subgraph[3];
    SCSubgraphRef Subgraph_RefLine = sc.Subgraph[4];

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

    if (sc.SetDefaults)
    {
        sc.GraphName = "MACD Bollinger Bands - Standard";
        sc.AutoLoop = 1;
    }
}

```

```

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

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

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

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

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

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

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

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

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

Input_NumberOfStdDevs.Name = "Number of Standard Deviations";
Input_NumberOfStdDevs.SetInt(1);
Input_NumberOfStdDevs.SetIntLimits(1, MAX_STUDY_LENGTH);

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

return;
}

sc.DataStartIndex = 2 * Input_MACDLength.GetInt() + max(Input_FastLength.GetInt(), Input_SlowLength.GetInt());

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

Subgraph_MovAvgOfMACD[sc.Index] = Subgraph_MACD.Arrays[2][sc.Index];

sc.StdDeviation(Subgraph_MACD, Subgraph_MACD.Arrays[6], Input_MACDLength.GetInt());
float StdDevOfMACD = Subgraph_MACD.Arrays[6][sc.Index];

Subgraph_TopBand[sc.Index] = Subgraph_MovAvgOfMACD[sc.Index] +
Input_NumberOfStdDevs.GetInt()*StdDevOfMACD;
Subgraph_BottomBand[sc.Index] = Subgraph_MovAvgOfMACD[sc.Index] -
Input_NumberOfStdDevs.GetInt()*StdDevOfMACD;

Subgraph_RefLine[sc.Index] = 0;
}

```



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

```
SCSFExport scsf_GreatestSwingValue(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_BuySwing = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SellSwing = sc.Subgraph[1];
    SCSubgraphRef Subgraph_AvgBuySwing = sc.Subgraph[2];
    SCSubgraphRef Subgraph_AvgSellSwing = sc.Subgraph[3];
    SCSubgraphRef Subgraph_BuyPrice = sc.Subgraph[4];
    SCSubgraphRef Subgraph_SellPrice = sc.Subgraph[5];

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

    if (sc.SetDefaults)
    {
        sc.GraphName = "Greatest Swing Value";

        sc.AutoLoop = 1;

        Subgraph_BuySwing.Name = "Buy Swing";
        Subgraph_BuySwing.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_SellSwing.Name = "Sell Swing";
        Subgraph_SellSwing.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_AvgBuySwing.Name = "Average Buy Swing";
        Subgraph_AvgBuySwing.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_AvgSellSwing.Name = "Average Sell Swing";
        Subgraph_AvgSellSwing.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_BuyPrice.Name = "Buy Price";
        Subgraph_BuyPrice.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_BuyPrice.PrimaryColor = RGB(0, 0, 255);

        Subgraph_SellPrice.Name = "Sell Price";
        Subgraph_SellPrice.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_SellPrice.PrimaryColor = RGB(255, 0, 0);

        Input_Length.Name = "Length";
        Input_Length.SetInt(4);
        Input_Length.SetIntLimits(1, MAX_STUDY_LENGTH);

        Input_Multiplier.Name = "Multiplier";
        Input_Multiplier.SetFloat(1.8f);

        return;
    }

    int Length = Input_Length.GetInt();
    float Multiplier = Input_Multiplier.GetFloat();

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

    if (sc.Close[sc.Index] < sc.Open[sc.Index])
        Subgraph_BuySwing[sc.Index] = sc.High[sc.Index] - sc.Open[sc.Index];
    else
        Subgraph_BuySwing[sc.Index] = 0;

    if (sc.Close[sc.Index] > sc.Open[sc.Index])
        Subgraph_SellSwing[sc.Index] = sc.Open[sc.Index] - sc.Low[sc.Index];
    else
        Subgraph_SellSwing[sc.Index] = 0;
}
```

```

sc.MovingAverage(Subgraph_BuySwing, Subgraph_AvgBuySwing, MOVAVGTYPE_SIMPLE_SKIP_ZEROS, Length);
sc.MovingAverage(Subgraph_SellSwing, Subgraph_AvgSellSwing, MOVAVGTYPE_SIMPLE_SKIP_ZEROS, Length);

Subgraph_BuyPrice[sc.Index] = sc.Open[sc.Index] + Multiplier * Subgraph_AvgBuySwing[sc.Index - 1];
Subgraph_SellPrice[sc.Index] = sc.Open[sc.Index] - Multiplier * Subgraph_AvgSellSwing[sc.Index - 1];
}

/*=====*/
SCSFExport scsf_KDJ(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_StochasticOutput = sc.Subgraph[4];
    SCSubgraphRef Subgraph_JLine = sc.Subgraph[5];

    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 = "KDJ";

        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;

        Subgraph_JLine.Name = "J Line";
        Subgraph_JLine.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_JLine.PrimaryColor = RGB(255, 0, 0);
        Subgraph_JLine.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_StochasticOutput, // Data member is Fast %K
    Input_FastKLength.GetInt(),
    Input_FastDLength.GetInt(),
    Input_SlowDLength.GetInt(),
    Input_MovAvgType.GetMovAvgType()
);

Subgraph_SlowK[sc.Index] = Subgraph_StochasticOutput.Arrays[0][sc.Index];
Subgraph_SlowD[sc.Index] = Subgraph_StochasticOutput.Arrays[1][sc.Index];
Subgraph_JLine[sc.Index] = 3 * Subgraph_SlowK[sc.Index] - 2 * Subgraph_SlowD[sc.Index];

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

return;
}

/*=====*/
SCSFExport scsf_NormalizedVolume(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_NormalizedVolume = sc.Subgraph[0];
    SCSubgraphRef Subgraph_HighVolume = sc.Subgraph[1];
    SCSubgraphRef Subgraph_LowVolume = sc.Subgraph[2];

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

    SCInputRef Input_Length = sc.Input[0];
    SCInputRef Input_MovingAverageType = sc.Input[1];
    SCInputRef Input_HighVolume = sc.Input[2];
    SCInputRef Input_LowVolume = sc.Input[3];

```

```

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

    sc.ValueFormat = 0;
    sc.AutoLoop = true;
    sc.ScaleRangeType = SCALE_ZEROBASED;

    Subgraph_NormalizedVolume.Name = "Normalized Volume";
    Subgraph_NormalizedVolume.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_NormalizedVolume.PrimaryColor = RGB(0, 0, 255);
    Subgraph_NormalizedVolume.DrawZeros = true;

    Subgraph_HighVolume.Name = "High Volume";
    Subgraph_HighVolume.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_HighVolume.PrimaryColor = RGB(0, 255, 0);
    Subgraph_HighVolume.DrawZeros = true;

    Subgraph_LowVolume.Name = "Low Volume";
    Subgraph_LowVolume.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_LowVolume.PrimaryColor = RGB(255, 0, 0);
    Subgraph_LowVolume.DrawZeros = true;

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

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

    Input_HighVolume.Name = "High Volume";
    Input_HighVolume.SetFloat(150.0f);

    Input_LowVolume.Name = "Low Volume";
    Input_LowVolume.SetFloat(75.0f);

    return;
}

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

sc.MovingAverage(sc.BaseData[SC_VOLUME], Array_AverageVolume, Input_MovingAverageType.GetMovAvgType(),
Input_Length.GetInt());

if (Array_AverageVolume[sc.Index] == 0)
    Subgraph_NormalizedVolume[sc.Index] = Subgraph_NormalizedVolume[sc.Index - 1];
else
    Subgraph_NormalizedVolume[sc.Index] = 100.0f * sc.BaseData[SC_VOLUME][sc.Index] /
Array_AverageVolume[sc.Index];

    Subgraph_HighVolume[sc.Index] = Input_HighVolume.GetFloat();
    Subgraph_LowVolume[sc.Index] = Input_LowVolume.GetFloat();
}

/*=====*/
SCSFExport scsf_DeMarkerOscillatorTypel(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_DeMaxI = sc.Subgraph[0];
    SCSubgraphRef Subgraph_DeMinI = sc.Subgraph[1];
    SCSubgraphRef Subgraph_DeMI = sc.Subgraph[2];
    SCSubgraphRef Subgraph_OverboughtLine = sc.Subgraph[3];
    SCSubgraphRef Subgraph_OversoldLine = sc.Subgraph[4];
    SCSubgraphRef Subgraph_DurationExceededOverbought = sc.Subgraph[5];
    SCSubgraphRef Subgraph_DurationExceededOversold = sc.Subgraph[6];

```

```
SCFloatArrayRef Array_Temp1 = sc.Subgraph[2].Arrays[0];
SCFloatArrayRef Array_Temp2 = sc.Subgraph[2].Arrays[1];
```

```
SCInputRef Input_InputDataHigh = sc.Input[0];
SCInputRef Input_InputDataLow = sc.Input[1];
SCInputRef Input_DeMLength1 = sc.Input[2];
SCInputRef Input_DeMLength2 = sc.Input[3];
SCInputRef Input_UseSmoothing = sc.Input[4];
SCInputRef Input_SmoothingLength = sc.Input[5];
SCInputRef Input_SmoothingMAType = sc.Input[6];
SCInputRef Input_UseMovingAverage = sc.Input[7];
SCInputRef Input_AverageDeMMALength = sc.Input[8];
SCInputRef Input_AverageDeMMAType = sc.Input[9];
SCInputRef Input_DurationLength = sc.Input[10];
SCInputRef Input_OffsetPercentage = sc.Input[11];
SCInputRef Input_OverboughtLineValue = sc.Input[12];
SCInputRef Input_OversoldLineValue = sc.Input[13];
```

```
if (sc.SetDefaults)
{
    sc.GraphName = "DeMarker Oscillator Type I";

    sc.AutoLoop = 1;

    Subgraph_DeMaxI.Name = "Max DeMarker I";
    Subgraph_DeMaxI.DrawStyle = DRAWSTYLE_IGNORE;

    Subgraph_DeMinI.Name = "Min DeMarker I";
    Subgraph_DeMinI.DrawStyle = DRAWSTYLE_IGNORE;

    Subgraph_DeMI.Name = "DeMarker I";
    Subgraph_DeMI.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_DeMI.PrimaryColor = RGB(0, 0, 255);
    Subgraph_DeMI.DrawZeros = true;

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

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

    Subgraph_DurationExceededOverbought.Name = "Duration Exceeded in Overbought Area";
    Subgraph_DurationExceededOverbought.PrimaryColor = RGB(0, 255, 0);
    Subgraph_DurationExceededOverbought.DrawStyle = DRAWSTYLE_ARROW_DOWN;
    Subgraph_DurationExceededOverbought.LineWidth = 2;
    Subgraph_DurationExceededOverbought.DrawZeros = 0;

    Subgraph_DurationExceededOversold.Name = "Duration Exceeded in Oversold Area";
    Subgraph_DurationExceededOversold.PrimaryColor = RGB(255, 0, 0);
    Subgraph_DurationExceededOversold.DrawStyle = DRAWSTYLE_ARROW_UP;
    Subgraph_DurationExceededOversold.LineWidth = 2;
    Subgraph_DurationExceededOversold.DrawZeros = 0;

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

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

    Input_DeMLength2.Name = "DeM Length 2";
```

```

Input_DeMLength2.SetInt(8);
Input_DeMLength2.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_DeMLength1.Name = "DeM Length 1";
Input_DeMLength1.SetInt(1);
Input_DeMLength1.SetIntLimits(1, Input_DeMLength2.GetInt());

Input_UseSmoothing.Name = "Use Smoothing?";
Input_UseSmoothing.SetYesNo(0);

Input_SmoothingLength.Name = "Smoothing Length";
Input_SmoothingLength.SetInt(3);
Input_SmoothingLength.SetIntLimits(1, Input_DeMLength2.GetInt());

Input_SmoothingMAType.Name = "Smoothing MA Type";
Input_SmoothingMAType.SetMovAvgType(MOVAVGTYPE_SIMPLE);

Input_UseMovingAverage.Name = "Use Moving Average?";
Input_UseMovingAverage.SetYesNo(0);

Input_AverageDeMMALength.Name = "Average DeM MA Length";
Input_AverageDeMMALength.SetInt(8);
Input_AverageDeMMALength.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_AverageDeMMAType.Name = "Average DeM MA Type";
Input_AverageDeMMAType.SetMovAvgType(MOVAVGTYPE_SIMPLE);

Input_DurationLength.Name = "Duration Length";
Input_DurationLength.SetInt(16);
Input_DurationLength.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_OffsetPercentage.Name = "Arrow Offset Percentage";
Input_OffsetPercentage.SetFloat(1.0f);

Input_OverboughtLineValue.Name = "Overbought Line Value";
Input_OverboughtLineValue.SetFloat(60.0f);

Input_OversoldLineValue.Name = "Oversold Line Value";
Input_OversoldLineValue.SetFloat(40.0f);

return;
}

if (Input_UseSmoothing.GetYesNo() == 0 && Input_UseMovingAverage.GetYesNo() == 0)
    sc.DataStartIndex = Input_DeMLength1.GetInt() + Input_DeMLength2.GetInt() - 1;
else if (Input_UseSmoothing.GetYesNo() == 1 && Input_UseMovingAverage.GetYesNo() == 0)
    sc.DataStartIndex = Input_DeMLength1.GetInt() + Input_DeMLength2.GetInt() + Input_SmoothingLength.GetInt() -
2;
else if (Input_UseSmoothing.GetYesNo() == 0 && Input_UseMovingAverage.GetYesNo() == 1)
    sc.DataStartIndex = Input_DeMLength1.GetInt() + Input_DeMLength2.GetInt() +
Input_AverageDeMMALength.GetInt() - 2;
else
    sc.DataStartIndex = Input_DeMLength1.GetInt() + Input_DeMLength2.GetInt() + Input_SmoothingLength.GetInt() +
Input_AverageDeMMALength.GetInt() - 3;

if (sc.Index >= Input_DeMLength1.GetInt())
{
    // Compute Max DeMarker I and Min Demarker I.
    float High = sc.BaseData[Input_InputDataHigh.GetInputDataIndex()][sc.Index];
    float PrevHigh = sc.BaseData[Input_InputDataHigh.GetInputDataIndex()][sc.Index - Input_DeMLength1.GetInt()];
    float Low = sc.BaseData[Input_InputDataLow.GetInputDataIndex()][sc.Index];
    float PrevLow = sc.BaseData[Input_InputDataLow.GetInputDataIndex()][sc.Index - Input_DeMLength1.GetInt()];

    if (High > PrevHigh)

```

```

        Subgraph_DeMaxI[sc.Index] = High - PrevHigh;

        if (Low < PrevLow)
            Subgraph_DeMinI[sc.Index] = PrevLow - Low;
    }

    if (sc.Index > 0)
    {
        // Compute DeM-I.
        float SumDeMaxI = 0.0f;
        float SumDeMinI = 0.0f;

        for (int Index = sc.Index - Input_DeMLength2.GetInt() + 1; Index <= sc.Index; Index++)
        {
            SumDeMaxI += Subgraph_DeMaxI[Index];
            SumDeMinI += Subgraph_DeMinI[Index];
        }

        if (SumDeMaxI + SumDeMinI != 0.0f)
            Array_Temp1[sc.Index] = 100.0f * SumDeMaxI / (SumDeMaxI + SumDeMinI);
        else
            Array_Temp1[sc.Index] = Array_Temp1[sc.Index - 1];
    }

    if (Input_UseSmoothing.GetYesNo() == 1)
        sc.MovingAverage(Array_Temp1, Array_Temp2, Input_SmoothingMAType.GetMovAvgType(),
        Input_SmoothingLength.GetInt());

    if (Input_UseSmoothing.GetYesNo() == 0 && Input_UseMovingAverage.GetYesNo() == 0)
        Subgraph_DeMI[sc.Index] = Array_Temp1[sc.Index];
    else if (Input_UseSmoothing.GetYesNo() == 1 && Input_UseMovingAverage.GetYesNo() == 0)
        Subgraph_DeMI[sc.Index] = Array_Temp2[sc.Index];
    else if (Input_UseSmoothing.GetYesNo() == 0 && Input_UseMovingAverage.GetYesNo() == 1)
        sc.MovingAverage(Array_Temp1, Subgraph_DeMI, Input_AverageDeMMAType.GetMovAvgType(),
        Input_AverageDeMMALength.GetInt());
    else
        sc.MovingAverage(Array_Temp2, Subgraph_DeMI, Input_AverageDeMMAType.GetMovAvgType(),
        Input_AverageDeMMALength.GetInt());

    Subgraph_DurationExceededOverbought[sc.Index] = 0.0f;
    Subgraph_DurationExceededOversold[sc.Index] = 0.0f;

    // Perform Duration Analysis.
    if (Subgraph_DeMI[sc.Index] > Input_OverboughtLineValue.GetFloat())
    {
        // Check beginning of chart.
        if (sc.Index == sc.DataStartIndex + Input_DurationLength.GetInt() - 1)
        {
            int OverboughtSum = 1;

            for (int Index = sc.Index - Input_DurationLength.GetInt() + 1; Index <= sc.Index - 1; Index++)
            {
                if (Subgraph_DeMI[Index] > Input_OverboughtLineValue.GetFloat())
                    OverboughtSum += 1;
            }

            if (OverboughtSum == Input_DurationLength.GetInt())
            {
                Subgraph_DurationExceededOverbought[sc.Index] =
                    Subgraph_DeMI[sc.Index] + (Input_OffsetPercentage.GetFloat() / 100.0f) *
                    Subgraph_DeMI[sc.Index];
            }
        }
    }

    int OverboughtIndex = 1;

```



```

while (Subgraph_DeMI[sc.Index - OverboughtIndex] > Input_OverboughtLineValue.GetFloat() && OverboughtIndex
<= Input_DurationLength.GetInt())
    OverboughtIndex++;

if (OverboughtIndex == Input_DurationLength.GetInt() && sc.Index >= sc.DataStartIndex +
Input_DurationLength.GetInt() - 1)
{
    Subgraph_DurationExceededOverbought[sc.Index] =
        Subgraph_DeMI[sc.Index] + (Input_OffsetPercentage.GetFloat() / 100.0f) *
        Subgraph_DeMI[sc.Index];
}
}

if (Subgraph_DeMI[sc.Index] < Input_OversoldLineValue.GetFloat())
{
    // Check beginning of chart.
    if (sc.Index == sc.DataStartIndex + Input_DurationLength.GetInt() - 1)
    {
        int OversoldSum = 1;

        for (int Index = sc.Index - Input_DurationLength.GetInt() + 1; Index <= sc.Index - 1; Index++)
        {
            if (Subgraph_DeMI[Index] < Input_OversoldLineValue.GetFloat())
                OversoldSum += 1;
        }

        if (OversoldSum == Input_DurationLength.GetInt())
        {
            Subgraph_DurationExceededOversold[sc.Index] =
                Subgraph_DeMI[sc.Index] - (Input_OffsetPercentage.GetFloat() / 100.0f) *
                Subgraph_DeMI[sc.Index];
            float MyNum = Subgraph_DurationExceededOversold[sc.Index];
        }
    }

    int OversoldIndex = 1;

    while (Subgraph_DeMI[sc.Index - OversoldIndex] < Input_OversoldLineValue.GetFloat() && OversoldIndex <=
Input_DurationLength.GetInt())
        OversoldIndex++;

    if (OversoldIndex == Input_DurationLength.GetInt() && sc.Index >= sc.DataStartIndex +
Input_DurationLength.GetInt() - 1)
    {
        Subgraph_DurationExceededOversold[sc.Index] =
            Subgraph_DeMI[sc.Index] - (Input_OffsetPercentage.GetFloat() / 100.0f) *
            Subgraph_DeMI[sc.Index];
    }
}

// Set horizontal lines.
Subgraph_OverboughtLine[sc.Index] = Input_OverboughtLineValue.GetFloat();
Subgraph_OversoldLine[sc.Index] = Input_OversoldLineValue.GetFloat();
}

/*=====*/
SCSFExport scsf_DeMarkerOscillatorTypeII(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_DeMaxII = sc.Subgraph[0];
    SCSubgraphRef Subgraph_DeMinII = sc.Subgraph[1];
    SCSubgraphRef Subgraph_DeMII = sc.Subgraph[2];
    SCSubgraphRef Subgraph_OverboughtLine = sc.Subgraph[3];
    SCSubgraphRef Subgraph_OversoldLine = sc.Subgraph[4];
}

```



```
SCSubgraphRef Subgraph_DurationExceededOverbought = sc.Subgraph[5];
SCSubgraphRef Subgraph_DurationExceededOversold = sc.Subgraph[6];
```

```
SCFloatArrayRef Array_Temp1 = sc.Subgraph[2].Arrays[0];
SCFloatArrayRef Array_Temp2 = sc.Subgraph[2].Arrays[1];
```

```
SCInputRef Input_InputDataHigh = sc.Input[0];
SCInputRef Input_InputDataLow = sc.Input[1];
SCInputRef Input_InputDataClose = sc.Input[2];
SCInputRef Input_DeMLength1 = sc.Input[3];
SCInputRef Input_DeMLength2 = sc.Input[4];
SCInputRef Input_UseSmoothing = sc.Input[5];
SCInputRef Input_SmoothingLength = sc.Input[6];
SCInputRef Input_SmoothingMAType = sc.Input[7];
SCInputRef Input_UseMovingAverage = sc.Input[8];
SCInputRef Input_AverageDeMMALength = sc.Input[9];
SCInputRef Input_AverageDeMMAType = sc.Input[10];
SCInputRef Input_DurationLength = sc.Input[11];
SCInputRef Input_OffsetPercentage = sc.Input[12];
SCInputRef Input_OverboughtLineValue = sc.Input[13];
SCInputRef Input_OversoldLineValue = sc.Input[14];
```

```
if (sc.SetDefaults)
{
    sc.GraphName = "DeMarker Oscillator Type II";

    sc.AutoLoop = 1;

    Subgraph_DeMaxII.Name = "Max DeMarker II";
    Subgraph_DeMaxII.DrawStyle = DRAWSTYLE_IGNORE;

    Subgraph_DeMinII.Name = "Min DeMarker II";
    Subgraph_DeMinII.DrawStyle = DRAWSTYLE_IGNORE;

    Subgraph_DeMII.Name = "DeMarker II";
    Subgraph_DeMII.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_DeMII.PrimaryColor = RGB(0, 0, 255);
    Subgraph_DeMII.DrawZeros = true;

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

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

    Subgraph_DurationExceededOverbought.Name = "Duration Exceeded in Overbought Area";
    Subgraph_DurationExceededOverbought.PrimaryColor = RGB(0, 255, 0);
    Subgraph_DurationExceededOverbought.DrawStyle = DRAWSTYLE_ARROW_DOWN;
    Subgraph_DurationExceededOverbought.LineWidth = 2;
    Subgraph_DurationExceededOverbought.DrawZeros = 0;

    Subgraph_DurationExceededOversold.Name = "Duration Exceeded in Oversold Area";
    Subgraph_DurationExceededOversold.PrimaryColor = RGB(255, 0, 0);
    Subgraph_DurationExceededOversold.DrawStyle = DRAWSTYLE_ARROW_UP;
    Subgraph_DurationExceededOversold.LineWidth = 2;
    Subgraph_DurationExceededOversold.DrawZeros = 0;

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

    Input_InputDataLow.Name = "Input Data Low";
```

```

Input_InputDataLow.SetInputDataIndex(SC_LOW);

Input_InputDataClose.Name = "Input Data Close";
Input_InputDataClose.SetInputDataIndex(SC_LAST);

Input_DeMLength2.Name = "DeM Length 2";
Input_DeMLength2.SetInt(8);
Input_DeMLength2.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_DeMLength1.Name = "DeM Length 1";
Input_DeMLength1.SetInt(1);
Input_DeMLength1.SetIntLimits(1, Input_DeMLength2.GetInt());

Input_UseSmoothing.Name = "Use Smoothing?";
Input_UseSmoothing.SetYesNo(0);

Input_SmoothingLength.Name = "Smoothing Length";
Input_SmoothingLength.SetInt(3);
Input_SmoothingLength.SetIntLimits(1, Input_DeMLength2.GetInt());

Input_SmoothingMAType.Name = "Smoothing MA Type";
Input_SmoothingMAType.SetMovAvgType(MOVAVGTYPE_SIMPLE);

Input_UseMovingAverage.Name = "Use Moving Average?";
Input_UseMovingAverage.SetYesNo(0);

Input_AverageDeMMALength.Name = "Average DeM MA Length";
Input_AverageDeMMALength.SetInt(8);
Input_AverageDeMMALength.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_AverageDeMMAType.Name = "Average DeM MA Type";
Input_AverageDeMMAType.SetMovAvgType(MOVAVGTYPE_SIMPLE);

Input_DurationLength.Name = "Duration Length";
Input_DurationLength.SetInt(16);
Input_DurationLength.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_OffsetPercentage.Name = "Arrow Offset Percentage";
Input_OffsetPercentage.SetFloat(1.0f);

Input_OverboughtLineValue.Name = "Overbought Line Value";
Input_OverboughtLineValue.SetFloat(60.0f);

Input_OversoldLineValue.Name = "Oversold Line Value";
Input_OversoldLineValue.SetFloat(40.0f);

return;
}

if (Input_UseSmoothing.GetYesNo() == 0 && Input_UseMovingAverage.GetYesNo() == 0)
    sc.DataStartIndex = Input_DeMLength1.GetInt() + Input_DeMLength2.GetInt() - 1;
else if (Input_UseSmoothing.GetYesNo() == 1 && Input_UseMovingAverage.GetYesNo() == 0)
    sc.DataStartIndex = Input_DeMLength1.GetInt() + Input_DeMLength2.GetInt() + Input_SmoothingLength.GetInt() -
2;
else if (Input_UseSmoothing.GetYesNo() == 0 && Input_UseMovingAverage.GetYesNo() == 1)
    sc.DataStartIndex = Input_DeMLength1.GetInt() + Input_DeMLength2.GetInt() +
Input_AverageDeMMALength.GetInt() - 2;
else
    sc.DataStartIndex = Input_DeMLength1.GetInt() + Input_DeMLength2.GetInt() + Input_SmoothingLength.GetInt() +
Input_AverageDeMMALength.GetInt() - 3;

if (sc.Index >= Input_DeMLength1.GetInt())
{
    // Compute Max DeMarker II and Min Demarker II.
    float High = sc.BaseData[Input_InputDataHigh.GetInputDataIndex()][sc.Index];

```

```

float Low = sc.BaseData[Input_InputDataLow.GetInputDataIndex()][sc.Index];
float Close = sc.BaseData[Input_InputDataClose.GetInputDataIndex()][sc.Index];
float PrevClose = sc.BaseData[Input_InputDataClose.GetInputDataIndex()][sc.Index - Input_DeMLength1.GetInt()];

if (High - PrevClose + Close - Low > 0)
    Subgraph_DeMaxII[sc.Index] = High - PrevClose + Close - Low;

if (PrevClose - Low + High - Close > 0)
    Subgraph_DeMinII[sc.Index] = PrevClose - Low + High - Close;
}

if (sc.Index > 0)
{
    // Compute DeM-II.
    float SumDeMaxII = 0.0f;
    float SumDeMinII = 0.0f;

    for (int Index = sc.Index - Input_DeMLength2.GetInt() + 1; Index <= sc.Index; Index++)
    {
        SumDeMaxII += Subgraph_DeMaxII[Index];
        SumDeMinII += Subgraph_DeMinII[Index];
    }

    if (SumDeMaxII + SumDeMinII != 0.0f)
        Array_Temp1[sc.Index] = 100.0f * SumDeMaxII / (SumDeMaxII + SumDeMinII);
    else
        Array_Temp1[sc.Index] = Array_Temp1[sc.Index - 1];
}

if (Input_UseSmoothing.GetYesNo() == 1)
    sc.MovingAverage(Array_Temp1, Array_Temp2, Input_SmoothingMAType.GetMovAvgType(),
Input_SmoothingLength.GetInt());

if (Input_UseSmoothing.GetYesNo() == 0 && Input_UseMovingAverage.GetYesNo() == 0)
    Subgraph_DeMII[sc.Index] = Array_Temp1[sc.Index];
else if (Input_UseSmoothing.GetYesNo() == 1 && Input_UseMovingAverage.GetYesNo() == 0)
    Subgraph_DeMII[sc.Index] = Array_Temp2[sc.Index];
else if (Input_UseSmoothing.GetYesNo() == 0 && Input_UseMovingAverage.GetYesNo() == 1)
    sc.MovingAverage(Array_Temp1, Subgraph_DeMII, Input_AverageDeMMAType.GetMovAvgType(),
Input_AverageDeMMALength.GetInt());
else
    sc.MovingAverage(Array_Temp2, Subgraph_DeMII, Input_AverageDeMMAType.GetMovAvgType(),
Input_AverageDeMMALength.GetInt());

Subgraph_DurationExceededOverbought[sc.Index] = 0.0f;
Subgraph_DurationExceededOversold[sc.Index] = 0.0f;

// Perform Duration Analysis.
if (Subgraph_DeMII[sc.Index] > Input_OverboughtLineValue.GetFloat())
{
    // Check beginning of chart.
    if (sc.Index == sc.DataStartIndex + Input_DurationLength.GetInt() - 1)
    {
        int OverboughtSum = 1;

        for (int Index = sc.Index - Input_DurationLength.GetInt() + 1; Index <= sc.Index - 1; Index++)
        {
            if (Subgraph_DeMII[Index] > Input_OverboughtLineValue.GetFloat())
                OverboughtSum += 1;
        }

        if (OverboughtSum == Input_DurationLength.GetInt())
        {
            Subgraph_DurationExceededOverbought[sc.Index] =
                Subgraph_DeMII[sc.Index] + (Input_OffsetPercentage.GetFloat() / 100.0f) *

```

```

        Subgraph_DeMII[sc.Index];
    }
}

int OverboughtIndex = 1;

while (Subgraph_DeMII[sc.Index - OverboughtIndex] > Input_OverboughtLineValue.GetFloat() && OverboughtIndex
<= Input_DurationLength.GetInt())
    OverboughtIndex++;

if (OverboughtIndex == Input_DurationLength.GetInt() && sc.Index >= sc.DataStartIndex +
Input_DurationLength.GetInt() - 1)
{
    Subgraph_DurationExceededOverbought[sc.Index] =
        Subgraph_DeMII[sc.Index] + (Input_OffsetPercentage.GetFloat() / 100.0f) *
        Subgraph_DeMII[sc.Index];
}
}

if (Subgraph_DeMII[sc.Index] < Input_OversoldLineValue.GetFloat())
{
    // Check beginning of chart.
    if (sc.Index == sc.DataStartIndex + Input_DurationLength.GetInt() - 1)
    {
        int OversoldSum = 1;

        for (int Index = sc.Index - Input_DurationLength.GetInt() + 1; Index <= sc.Index - 1; Index++)
        {
            if (Subgraph_DeMII[Index] < Input_OversoldLineValue.GetFloat())
                OversoldSum += 1;
        }

        if (OversoldSum == Input_DurationLength.GetInt())
        {
            Subgraph_DurationExceededOversold[sc.Index] =
                Subgraph_DeMII[sc.Index] - (Input_OffsetPercentage.GetFloat() / 100.0f) *
                Subgraph_DeMII[sc.Index];
        }
    }
}

int OversoldIndex = 1;

while (Subgraph_DeMII[sc.Index - OversoldIndex] < Input_OversoldLineValue.GetFloat() && OversoldIndex <=
Input_DurationLength.GetInt())
    OversoldIndex++;

if (OversoldIndex == Input_DurationLength.GetInt() && sc.Index >= sc.DataStartIndex +
Input_DurationLength.GetInt() - 1)
{
    Subgraph_DurationExceededOversold[sc.Index] =
        Subgraph_DeMII[sc.Index] - (Input_OffsetPercentage.GetFloat() / 100.0f) *
        Subgraph_DeMII[sc.Index];
}
}

// Set horizontal lines.
Subgraph_OverboughtLine[sc.Index] = Input_OverboughtLineValue.GetFloat();
Subgraph_OversoldLine[sc.Index] = Input_OversoldLineValue.GetFloat();
}

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

    SCInputRef Input_InputData = sc.Input[0];

```

```

SCInputRef Input_Length = sc.Input[1];

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

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

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

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

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

    return;
}

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

sc.ExponentialRegressionIndicator(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_ERMA, sc.Index,
Input_Length.GetInt());
}

/*=====*/
SCSFExport scsf_InstantaneousTrendline(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_IT = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Trigger = sc.Subgraph[1];

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

    if (sc.SetDefaults)
    {
        sc.GraphName = "Instantaneous Trendline";

        sc.GraphRegion = 0;
        sc.ValueFormat = 2;
        sc.AutoLoop = 1;

        Subgraph_IT.Name = "Instantaneous Trendline";
        Subgraph_IT.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_IT.PrimaryColor = RGB(0, 255, 0);
        Subgraph_IT.LineWidth = 1;
        Subgraph_IT.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;

        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);

    return;
}

sc.InstantaneousTrendline(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_IT, sc.Index,
Input_Length.GetInt());

Subgraph_Trigger[sc.Index] = 2.0f * Subgraph_IT[sc.Index] - Subgraph_IT[sc.Index - 2];
}

/*=====*/
SCSFExport scsf_CyberCycle(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_CC = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Trigger = sc.Subgraph[1];
    SCSubgraphRef Subgraph_CenterLine = sc.Subgraph[2];

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

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_Length = sc.Input[1];
    SCInputRef Input_Lag = sc.Input[2];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Cyber Cycle";

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

        Subgraph_CC.Name = "Cyber Cycle";
        Subgraph_CC.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_CC.PrimaryColor = RGB(0, 255, 0);
        Subgraph_CC.LineWidth = 1;
        Subgraph_CC.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_Lag.Name = "Lag";
        Input_Lag.SetInt(9);
        Input_Lag.SetIntLimits(1, MAX_STUDY_LENGTH);

        return;
    }

    sc.FourBarSymmetricalFIRFilter(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Array_SmoothedData,

```

```

sc.Index);

    sc.CyberCycle(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Array_SmoothedData, Subgraph_CC, sc.Index,
Input_Length.GetInt());

    float lag = static_cast<float>(Input_Lag.GetInt());
    float b = 1 / (lag + 1);

    Subgraph_Trigger[sc.Index] = b * Subgraph_CC[sc.Index] + (1 - b) * Subgraph_Trigger[sc.Index - 1];

    Subgraph_CenterLine[sc.Index] = 0.0f;
}
/*=====*/
SCSFExport scsf_StochasticCyberCycle(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_CC = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SCC = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Trigger = sc.Subgraph[2];
    SCSubgraphRef Subgraph_CenterLine = sc.Subgraph[3];

    SCFloatArrayRef Array_SmoothedData = sc.Subgraph[0].Arrays[0];
    SCFloatArrayRef Array_Ratio = sc.Subgraph[0].Arrays[1];

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

    if (sc.SetDefaults)
    {
        sc.GraphName = "Stochastic Cyber Cycle";

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

        Subgraph_CC.Name = "Cyber Cycle";
        Subgraph_CC.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_SCC.Name = "Stochastic Cyber Cycle";
        Subgraph_SCC.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_SCC.PrimaryColor = RGB(0, 255, 0);
        Subgraph_SCC.LineWidth = 1;
        Subgraph_SCC.DrawZeros = true;

        Subgraph_Trigger.Name = "Trigger";
        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);

        return;
    }
}

```

```

//Compute Cyber Cycle
sc.FourBarSymmetricalFIRFilter(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Array_SmoothedData,
sc.Index);

sc.CyberCycle(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Array_SmoothedData, Subgraph_CC, sc.Index,
Input_Length.GetInt());

// Compute Stochastic Cyber Cycle
float MaxCC = sc.GetHighest(Subgraph_CC, sc.Index, Input_Length.GetInt());
float MinCC = sc.GetLowest(Subgraph_CC, sc.Index, Input_Length.GetInt());

if (MaxCC - MinCC != 0.0f)
    Array_Ratio[sc.Index] = (Subgraph_CC[sc.Index] - MinCC) / (MaxCC - MinCC);
else
    Array_Ratio[sc.Index] = 0.0f;

float SmoothedRatio = (4.0f * Array_Ratio[sc.Index] + 3.0f * Array_Ratio[sc.Index - 1] + 2.0f * Array_Ratio[sc.Index - 2]
+ Array_Ratio[sc.Index - 3]) / 10.0f;

Subgraph_SCC[sc.Index] = 2.0f * (SmoothedRatio - 0.5f);

Subgraph_Trigger[sc.Index] = 0.96f * (Subgraph_SCC[sc.Index - 1] + 0.02f);

Subgraph_CenterLine[sc.Index] = 0.0f;
}
/*=====*/
SCSFExport scsf_FisherCyberCycle(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_CC = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SCC = sc.Subgraph[1];
    SCSubgraphRef Subgraph_FCC = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Trigger = sc.Subgraph[3];
    SCSubgraphRef Subgraph_CenterLine = sc.Subgraph[4];

    SCFloatArrayRef Array_SmoothedData = sc.Subgraph[0].Arrays[0];
    SCFloatArrayRef Array_Ratio = sc.Subgraph[0].Arrays[1];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_Length = sc.Input[1];
    SCInputRef Input_UseAbsoluteValueWhenLogArgumentIsZero = sc.Input[2];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Fisher Cyber Cycle";

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

        Subgraph_CC.Name = "Cyber Cycle";
        Subgraph_CC.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_SCC.Name = "Stochastic Cyber Cycle";
        Subgraph_SCC.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_FCC.Name = "Fisher Cyber Cycle";
        Subgraph_FCC.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_FCC.PrimaryColor = RGB(0, 255, 0);
        Subgraph_FCC.LineWidth = 1;
        Subgraph_FCC.DrawZeros = true;

        Subgraph_Trigger.Name = "Trigger";
        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_UseAbsoluteValueWhenLogArgumentIsZero.Name = "Use Absolute Value When Log Argument Is Zero";
Input_UseAbsoluteValueWhenLogArgumentIsZero.SetYesNo(true);

return;
}

// Compute Cyber Cycle
sc.FourBarSymmetricalFIRFilter(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Array_SmoothedData,
sc.Index);

sc.CyberCycle(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Array_SmoothedData, Subgraph_CC, sc.Index,
Input_Length.GetInt());

// Compute Stochastic Cyber Cycle
float MaxCC = sc.GetHighest(Subgraph_CC, sc.Index, Input_Length.GetInt());
float MinCC = sc.GetLowest(Subgraph_CC, sc.Index, Input_Length.GetInt());

if (MaxCC - MinCC != 0.0f)
    Array_Ratio[sc.Index] = (Subgraph_CC[sc.Index] - MinCC) / (MaxCC - MinCC);
else
    Array_Ratio[sc.Index] = 0.0f;

float SmoothedRatio = (4.0f * Array_Ratio[sc.Index] + 3.0f * Array_Ratio[sc.Index - 1] + 2.0f * Array_Ratio[sc.Index - 2]
+ Array_Ratio[sc.Index - 3]) / 10.0f;

Subgraph_SCC[sc.Index] = 2.0f * (SmoothedRatio - 0.5f);

// Compute Fisher Cyber Cycle
float LogArgument = (1 + 1.98f * (Subgraph_SCC[sc.Index] - 0.5f)) / (1 - 1.98f * (Subgraph_SCC[sc.Index] - 0.5f));
float LogArgumentDenom = 1 - 1.98f * (Subgraph_SCC[sc.Index] - 0.5f);

const int SelectedIndex = Input_UseAbsoluteValueWhenLogArgumentIsZero.GetYesNo();

switch (SelectedIndex)
{
case 0:
{
    if (LogArgument > 0.0f && LogArgumentDenom != 0.0f)
        Subgraph_FCC[sc.Index] = 0.5f * logf(LogArgument);
    else
        Subgraph_FCC[sc.Index] = 0.0f;
}
break;

case 1:
{
    if (LogArgument != 0.0f && LogArgumentDenom != 0.0f)
    {
        Subgraph_FCC[sc.Index] = 0.5f * logf(fabs(LogArgument));
    }
}
}

```

```

        else
            Subgraph_FCC[sc.Index] = 0.0f;
    }
    break;
}

Subgraph_Trigger[sc.Index] = Subgraph_FCC[sc.Index - 1];

Subgraph_CenterLine[sc.Index] = 0.0f;
}

/*=====*/
SCSFExport scsf_CGOscillator(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_CG = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Trigger = sc.Subgraph[1];
    SCSubgraphRef Subgraph_CenterLine = sc.Subgraph[2];

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

    if (sc.SetDefaults)
    {
        sc.GraphName = "Center of Gravity Oscillator";

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

        Subgraph_CG.Name = "CG";
        Subgraph_CG.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_CG.PrimaryColor = RGB(0, 255, 0);
        Subgraph_CG.LineWidth = 1;
        Subgraph_CG.DrawZeros = true;

        Subgraph_Trigger.Name = "Trigger";
        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(10);
        Input_Length.SetIntLimits(1, MAX_STUDY_LENGTH);

        return;
    }

    float Num = 0;
    float Den = 0;

    for (int j = 0; j < Input_Length.GetInt(); j++)
    {
        Num += (1 + j) * sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - j];
        Den += sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - j];
    }

```

```

float Length = static_cast<float>(Input_Length.GetInt());

if (Den != 0.0f)
    Subgraph_CG[sc.Index] = -1.0f * Num / Den + (Length + 1.0f) / 2.0f;
else
    Subgraph_CG[sc.Index] = 0.0f;

Subgraph_Trigger[sc.Index] = Subgraph_CG[sc.Index - 1];

Subgraph_CenterLine[sc.Index] = 0.0f;
}

/*=====*/
SCSFExport scsf_StochasticCGOscillator(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_CG = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SCG = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Trigger = sc.Subgraph[2];
    SCSubgraphRef Subgraph_CenterLine = sc.Subgraph[3];

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

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

    if (sc.SetDefaults)
    {
        sc.GraphName = "Stochastic Center of Gravity Oscillator";

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

        Subgraph_CG.Name = "CG";
        Subgraph_CG.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_SCG.Name = "Stoch CG";
        Subgraph_SCG.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_SCG.PrimaryColor = RGB(0, 255, 0);
        Subgraph_SCG.LineWidth = 1;
        Subgraph_SCG.DrawZeros = true;

        Subgraph_Trigger.Name = "Trigger";
        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(10);
        Input_Length.SetIntLimits(1, MAX_STUDY_LENGTH);

        return;
    }
}

```

```

// Compute Center of Gravity Oscillator

float Num = 0;
float Den = 0;

for (int j = 0; j < Input_Length.GetInt(); j++)
{
    Num += (1.0f + j) * sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - j];
    Den += sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - j];
}

float Length = static_cast<float>(Input_Length.GetInt());

if (Den != 0.0f)
    Subgraph_CG[sc.Index] = -1.0f * Num / Den + (Length + 1.0f) / 2.0f;
else
    Subgraph_CG[sc.Index] = 0.0f;

// Compute Stochastic Center of Gravity Oscillator
float MaxCG = sc.GetHighest(Subgraph_CG, sc.Index, Input_Length.GetInt());
float MinCG = sc.GetLowest(Subgraph_CG, sc.Index, Input_Length.GetInt());

if (MaxCG - MinCG != 0.0f)
    Array_Ratio[sc.Index] = (Subgraph_CG[sc.Index] - MinCG) / (MaxCG - MinCG);
else
    Array_Ratio[sc.Index] = 0.0f;

float SmoothedRatio = (4.0f * Array_Ratio[sc.Index] + 3.0f * Array_Ratio[sc.Index - 1] + 2.0f * Array_Ratio[sc.Index - 2]
+ Array_Ratio[sc.Index - 3]) / 10.0f;

Subgraph_SCG[sc.Index] = 2.0f * (SmoothedRatio - 0.5f);

Subgraph_Trigger[sc.Index] = 0.96f * (Subgraph_SCG[sc.Index - 1] + 0.02f);

Subgraph_CenterLine[sc.Index] = 0.0f;
}

/*=====*/
SCSFExport scsf_FisherCGOscillator(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_CG = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SCG = sc.Subgraph[1];
    SCSubgraphRef Subgraph_FCG = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Trigger = sc.Subgraph[3];
    SCSubgraphRef Subgraph_CenterLine = sc.Subgraph[4];

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

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_Length = sc.Input[1];
    SCInputRef Input_UseAbsoluteValueWhenLogArgumentIsZero = sc.Input[2];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Fisher Center of Gravity Oscillator";

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

        Subgraph_CG.Name = "CG";
        Subgraph_CG.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_SCG.Name = "Stoch CG";
        Subgraph_SCG.DrawStyle = DRAWSTYLE_IGNORE;
    }
}

```

```

Subgraph_FCG.Name = "Fisher CG";
Subgraph_FCG.DrawStyle = DRAWSTYLE_LINE;
Subgraph_FCG.PrimaryColor = RGB(0, 255, 0);
Subgraph_FCG.LineWidth = 1;
Subgraph_FCG.DrawZeros = true;

Subgraph_Trigger.Name = "Trigger";
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(10);
Input_Length.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_UseAbsoluteValueWhenLogArgumentIsZero.Name = "Use Absolute Value When Log Argument Is Zero";
Input_UseAbsoluteValueWhenLogArgumentIsZero.SetYesNo(true);

return;
}

// Compute Center of Gravity Oscillator
float Num = 0;
float Den = 0;

for (int j = 0; j < Input_Length.GetInt(); j++)
{
    Num += (1.0f + j) * sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - j];
    Den += sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - j];
}

float Length = static_cast<float>(Input_Length.GetInt());

if (Den != 0.0f)
    Subgraph_CG[sc.Index] = -1.0f * Num / Den + (Length + 1.0f) / 2.0f;
else
    Subgraph_CG[sc.Index] = 0.0f;

// Compute Stochastic Center of Gravity Oscillator
float MaxCG = sc.GetHighest(Subgraph_CG, sc.Index, Input_Length.GetInt());
float MinCG = sc.GetLowest(Subgraph_CG, sc.Index, Input_Length.GetInt());

if (MaxCG - MinCG != 0.0f)
    Array_Ratio[sc.Index] = (Subgraph_CG[sc.Index] - MinCG) / (MaxCG - MinCG);
else
    Array_Ratio[sc.Index] = 0.0f;

float SmoothedRatio = (4.0f * Array_Ratio[sc.Index] + 3.0f * Array_Ratio[sc.Index - 1] + 2.0f * Array_Ratio[sc.Index - 2]
+ Array_Ratio[sc.Index - 3]) / 10.0f;

Subgraph_SCG[sc.Index] = 2.0f * (SmoothedRatio - 0.5f);

// Compute Fisher Center of Gravity Oscillator
float LogArgument = (1 + 1.98f * (Subgraph_SCG[sc.Index] - 0.5f)) / (1 - 1.98f * (Subgraph_SCG[sc.Index] - 0.5f));

```

```

float LogArgumentDenom = 1 - 1.98f * (Subgraph_SCG[sc.Index] - 0.5f);

const int SelectedIndex = Input_UseAbsoluteValueWhenLogArgumentIsZero.GetYesNo();

switch (SelectedIndex)
{
case 0:
{
    if (LogArgument > 0.0f && LogArgumentDenom != 0.0f)
        Subgraph_FCG[sc.Index] = 0.5f * logf(LogArgument);
    else
        Subgraph_FCG[sc.Index] = 0.0f;
}
break;

case 1:
{
    if (LogArgument != 0.0f && LogArgumentDenom != 0.0f)
    {
        Subgraph_FCG[sc.Index] = 0.5f * logf(fabs(LogArgument));
    }
    else
        Subgraph_FCG[sc.Index] = 0.0f;
}
break;
}

Subgraph_Trigger[sc.Index] = Subgraph_FCG[sc.Index - 1];

Subgraph_CenterLine[sc.Index] = 0.0f;
}
/*=====*/
SCSFExport scsf_RelativeVigorIndex3(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_RVI = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Trigger = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Centerline = sc.Subgraph[2];

    SCFloatArrayRef Array_CloseOpenAvg = Subgraph_RVI.Arrays[0];
    SCFloatArrayRef Array_HighLowAvg = Subgraph_RVI.Arrays[1];
    SCFloatArrayRef Array_SmoothedRVINumerator = Subgraph_RVI.Arrays[2];
    SCFloatArrayRef Array_SmoothedRVIDenominator = Subgraph_RVI.Arrays[3];

    SCInputRef Input_Length = sc.Input[0];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Relative Vigor Index 3";

        sc.AutoLoop = 1;

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

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

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

        Subgraph_Centerline.Name = "Center Line";
    }
}

```

```

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

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

return;
}

Array_CloseOpenAvg[sc.Index] = ((sc.Close[sc.Index - 3] - sc.Open[sc.Index - 3]) + 2 * (sc.Close[sc.Index - 2] -
sc.Open[sc.Index - 2]) + 2 * (sc.Close[sc.Index - 1] - sc.Open[sc.Index - 1]) + (sc.Close[sc.Index] - sc.Open[sc.Index])) / 6;

Array_HighLowAvg[sc.Index] = ((sc.High[sc.Index - 3] - sc.Low[sc.Index - 3]) + 2 * (sc.High[sc.Index - 2] -
sc.Low[sc.Index - 2]) + 2 * (sc.High[sc.Index - 1] - sc.Low[sc.Index - 1]) + (sc.High[sc.Index] - sc.Low[sc.Index])) / 6;

float Numerator = 0.0f;
float Denominator = 0.0f;

for (int i = 0; i < Input_Length.GetInt(); i++)
{
    Numerator += Array_CloseOpenAvg[sc.Index - i];
    Denominator += Array_HighLowAvg[sc.Index - i];
}

if (Denominator != 0)
    Subgraph_RVI[sc.Index] = Numerator / Denominator;
else
    Subgraph_RVI[sc.Index] = 0.0f;

Subgraph_Trigger[sc.Index] = Subgraph_RVI[sc.Index - 1];

Subgraph_Centerline[sc.Index] = 0.0f;
}

/*=====*/
SCSFExport scsf_StochasticRelativeVigorIndex3(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_RVI = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SRVI = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Trigger = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Centerline = sc.Subgraph[3];

    SCFloatArrayRef Array_CloseOpenAvg = Subgraph_RVI.Arrays[0];
    SCFloatArrayRef Array_HighLowAvg = Subgraph_RVI.Arrays[1];
    SCFloatArrayRef Array_SmoothedRVINumerator = Subgraph_RVI.Arrays[2];
    SCFloatArrayRef Array_SmoothedRVIDenominator = Subgraph_RVI.Arrays[3];
    SCFloatArrayRef Array_Ratio = Subgraph_RVI.Arrays[4];

    SCInputRef Input_Length = sc.Input[0];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Stochastic Relative Vigor Index 3";

        sc.AutoLoop = 1;

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

        Subgraph_RVI.Name = "RVI";
        Subgraph_RVI.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_SRVI.Name = "Stochastic Relative Vigor Index 3";
    }
}

```

```

Subgraph_SRVI.DrawStyle = DRAWSTYLE_LINE;
Subgraph_SRVI.PrimaryColor = RGB(0, 0, 255);
Subgraph_SRVI.LineWidth = 1;
Subgraph_SRVI.DrawZeros = true;

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

Subgraph_Centerline.Name = "Center Line";
Subgraph_Centerline.DrawStyle = DRAWSTYLE_LINE;
Subgraph_Centerline.PrimaryColor = RGB(128, 128, 0);
Subgraph_Centerline.DrawZeros = true;

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

return;
}

// Compute RVI
Array_CloseOpenAvg[sc.Index] = ((sc.Close[sc.Index - 3] - sc.Open[sc.Index - 3]) + 2 * (sc.Close[sc.Index - 2] -
sc.Open[sc.Index - 2]) + 2 * (sc.Close[sc.Index - 1] - sc.Open[sc.Index - 1]) + (sc.Close[sc.Index] - sc.Open[sc.Index])) / 6;

Array_HighLowAvg[sc.Index] = ((sc.High[sc.Index - 3] - sc.Low[sc.Index - 3]) + 2 * (sc.High[sc.Index - 2] -
sc.Low[sc.Index - 2]) + 2 * (sc.High[sc.Index - 1] - sc.Low[sc.Index - 1]) + (sc.High[sc.Index] - sc.Low[sc.Index])) / 6;

float Numerator = 0.0f;
float Denominator = 0.0f;

for (int i = 0; i < Input_Length.GetInt(); i++)
{
    Numerator += Array_CloseOpenAvg[sc.Index - i];
    Denominator += Array_HighLowAvg[sc.Index - i];
}

if (Denominator != 0)
    Subgraph_RVI[sc.Index] = Numerator / Denominator;
else
    Subgraph_RVI[sc.Index] = 0.0f;

// Compute Stochastic RVI
float MaxRVI = sc.GetHighest(Subgraph_RVI, sc.Index, Input_Length.GetInt());
float MinRVI = sc.GetLowest(Subgraph_RVI, sc.Index, Input_Length.GetInt());

if (MaxRVI - MinRVI != 0.0f)
    Array_Ratio[sc.Index] = (Subgraph_RVI[sc.Index] - MinRVI) / (MaxRVI - MinRVI);
else
    Array_Ratio[sc.Index] = 0.0f;

float SmoothedRatio = (4.0f * Array_Ratio[sc.Index] + 3.0f * Array_Ratio[sc.Index - 1] + 2.0f * Array_Ratio[sc.Index - 2]
+ Array_Ratio[sc.Index - 3]) / 10.0f;

Subgraph_SRVI[sc.Index] = 2.0f * (SmoothedRatio - 0.5f);

Subgraph_Trigger[sc.Index] = 0.96f * (Subgraph_SRVI[sc.Index - 1] + 0.02f);

Subgraph_Centerline[sc.Index] = 0.0f;
}

/*=====*/
SCSFExport scsf_FisherRelativeVigorIndex3(SCStudyInterfaceRef sc)
{

```



```

SCSubgraphRef Subgraph_RVI = sc.Subgraph[0];
SCSubgraphRef Subgraph_SRVI = sc.Subgraph[1];
SCSubgraphRef Subgraph_FRVI = sc.Subgraph[2];
SCSubgraphRef Subgraph_Trigger = sc.Subgraph[3];
SCSubgraphRef Subgraph_Centerline = sc.Subgraph[4];

SCFloatArrayRef Array_CloseOpenAvg = Subgraph_RVI.Arrays[0];
SCFloatArrayRef Array_HighLowAvg = Subgraph_RVI.Arrays[1];
SCFloatArrayRef Array_SmoothedRVINumerator = Subgraph_RVI.Arrays[2];
SCFloatArrayRef Array_SmoothedRVIDenominator = Subgraph_RVI.Arrays[3];
SCFloatArrayRef Array_Ratio = Subgraph_RVI.Arrays[4];

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

if (sc.SetDefaults)
{
    sc.GraphName = "Fisher Relative Vigor Index 3";

    sc.AutoLoop = 1;

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

    Subgraph_RVI.Name = "RVI";
    Subgraph_RVI.DrawStyle = DRAWSTYLE_IGNORE;

    Subgraph_SRVI.Name = "Stochastic Relative Vigor Index 3";
    Subgraph_SRVI.DrawStyle = DRAWSTYLE_IGNORE;

    Subgraph_FRVI.Name = "Fisher Relative Vigor Index 3";
    Subgraph_FRVI.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_FRVI.PrimaryColor = RGB(0, 0, 255);
    Subgraph_FRVI.LineWidth = 1;
    Subgraph_FRVI.DrawZeros = true;

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

    Subgraph_Centerline.Name = "Center Line";
    Subgraph_Centerline.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_Centerline.PrimaryColor = RGB(128, 128, 0);
    Subgraph_Centerline.DrawZeros = true;

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

    Input_UseAbsoluteValueWhenLogArgumentIsZero.Name = "Use Absolute Value When Log Argument Is Zero";
    Input_UseAbsoluteValueWhenLogArgumentIsZero.SetYesNo(true);

    return;
}

// Compute RVI
Array_CloseOpenAvg[sc.Index] = ((sc.Close[sc.Index - 3] - sc.Open[sc.Index - 3]) + 2 * (sc.Close[sc.Index - 2] -
sc.Open[sc.Index - 2]) + 2 * (sc.Close[sc.Index - 1] - sc.Open[sc.Index - 1]) + (sc.Close[sc.Index] - sc.Open[sc.Index])) / 6;

Array_HighLowAvg[sc.Index] = ((sc.High[sc.Index - 3] - sc.Low[sc.Index - 3]) + 2 * (sc.High[sc.Index - 2] -
sc.Low[sc.Index - 2]) + 2 * (sc.High[sc.Index - 1] - sc.Low[sc.Index - 1]) + (sc.High[sc.Index] - sc.Low[sc.Index])) / 6;

float Numerator = 0.0f;
float Denominator = 0.0f;

```

```

for (int i = 0; i < Input_Length.GetInt(); i++)
{
    Numerator += Array_CloseOpenAvg[sc.Index - i];
    Denominator += Array_HighLowAvg[sc.Index - i];
}

if (Denominator != 0)
    Subgraph_RVI[sc.Index] = Numerator / Denominator;
else
    Subgraph_RVI[sc.Index] = 0.0f;

// Compute Stochastic RVI
float MaxRVI = sc.GetHighest(Subgraph_RVI, sc.Index, Input_Length.GetInt());
float MinRVI = sc.GetLowest(Subgraph_RVI, sc.Index, Input_Length.GetInt());

if (MaxRVI - MinRVI != 0.0f)
    Array_Ratio[sc.Index] = (Subgraph_RVI[sc.Index] - MinRVI) / (MaxRVI - MinRVI);
else
    Array_Ratio[sc.Index] = 0.0f;

float SmoothedRatio = (4.0f * Array_Ratio[sc.Index] + 3.0f * Array_Ratio[sc.Index - 1] + 2.0f * Array_Ratio[sc.Index - 2]
+ Array_Ratio[sc.Index - 3]) / 10.0f;

Subgraph_SRVI[sc.Index] = 2.0f * (SmoothedRatio - 0.5f);

// Compute Fisher RVI
float LogArgument = (1 + 1.98f * (Subgraph_SRVI[sc.Index] - 0.5f)) / (1 - 1.98f * (Subgraph_SRVI[sc.Index] - 0.5f));
float LogArgumentDenom = 1 - 1.98f * (Subgraph_SRVI[sc.Index] - 0.5f);

const int SelectedIndex = Input_UseAbsoluteValueWhenLogArgumentIsZero.GetYesNo();

switch (SelectedIndex)
{
{
case 0:
{
    if (LogArgument > 0.0f && LogArgumentDenom != 0.0f)
        Subgraph_FRVI[sc.Index] = 0.5f * log(LogArgument);
    else
        Subgraph_FRVI[sc.Index] = 0.0f;
}
break;

case 1:
{
    if (LogArgument != 0.0f && LogArgumentDenom != 0.0f)
    {
        Subgraph_FRVI[sc.Index] = 0.5f * log(fabs(LogArgument));
    }
    else
        Subgraph_FRVI[sc.Index] = 0.0f;
}
break;
}

Subgraph_Trigger[sc.Index] = Subgraph_FRVI[sc.Index - 1];

Subgraph_Centerline[sc.Index] = 0.0f;
}

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

```

```

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

if (sc.SetDefaults)
{
    sc.GraphName = "Kaufman Efficiency Ratio";

    sc.AutoLoop = 1;

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

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

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

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

    return;
}

// Compute Direction
float Direction = fabs(sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index] -
sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - Input_Length.GetInt()]);

// Compute Volatility
float Volatility = 0.0f;

for (int i = 0; i < Input_Length.GetInt(); i++)
    Volatility += fabs(sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - i] -
sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - i - 1]);

// Compute KER
if (Volatility == 0.0f)
    Subgraph_KER[sc.Index] = 0.0f;
else
    Subgraph_KER[sc.Index] = Direction / Volatility;
}

/*=====*/
SCSFExport scsf_StochasticFunction(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Stochastic = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Trigger = sc.Subgraph[1];
    SCSubgraphRef Subgraph_CenterLine = sc.Subgraph[2];

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

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

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

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

```

```

Subgraph_Stochastic.Name = "Stochastic Function";
Subgraph_Stochastic.DrawStyle = DRAWSTYLE_LINE;
Subgraph_Stochastic.PrimaryColor = RGB(0, 255, 0);
Subgraph_Stochastic.LineWidth = 1;
Subgraph_Stochastic.DrawZeros = true;

Subgraph_Trigger.Name = "Trigger";
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(10);
Input_Length.SetIntLimits(1, MAX_STUDY_LENGTH);

return;
}

// Compute Stochastic Function
float MaxOsc = sc.GetHighest(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], sc.Index, Input_Length.GetInt());
float MinOsc = sc.GetLowest(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], sc.Index, Input_Length.GetInt());

if (MaxOsc - MinOsc != 0.0f)
    Array_Ratio[sc.Index] = (sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index] - MinOsc) / (MaxOsc - MinOsc);
else
    Array_Ratio[sc.Index] = 0.0f;

float SmoothedRatio = (4.0f * Array_Ratio[sc.Index] + 3.0f * Array_Ratio[sc.Index - 1] + 2.0f * Array_Ratio[sc.Index - 2]
+ Array_Ratio[sc.Index - 3]) / 10.0f;

Subgraph_Stochastic[sc.Index] = 2.0f * (SmoothedRatio - 0.5f);

Subgraph_Trigger[sc.Index] = 0.96f * (Subgraph_Stochastic[sc.Index - 1] + 0.02f);

Subgraph_CenterLine[sc.Index] = 0.0f;
}

/*=====*/
SCSFExport scsf_FisherFunction(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Stochastic = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Fisher = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Trigger = sc.Subgraph[2];
    SCSubgraphRef Subgraph_CenterLine = sc.Subgraph[3];

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

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_Length = sc.Input[1];
    SCInputRef Input_UseAbsoluteValueWhenLogArgumentIsZero = sc.Input[2];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Fisher Function";
    }
}

```

```

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

Subgraph_Stochastic.Name = "Stochastic Function";
Subgraph_Stochastic.DrawStyle = DRAWSTYLE_IGNORE;

Subgraph_Fisher.Name = "Fisher Function";
Subgraph_Fisher.DrawStyle = DRAWSTYLE_LINE;
Subgraph_Fisher.PrimaryColor = RGB(0, 255, 0);
Subgraph_Fisher.LineWidth = 1;
Subgraph_Fisher.DrawZeros = true;

Subgraph_Trigger.Name = "Trigger";
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(10);
Input_Length.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_UseAbsoluteValueWhenLogArgumentIsZero.Name = "Use Absolute Value When Log Argument Is Zero";
Input_UseAbsoluteValueWhenLogArgumentIsZero.SetYesNo(true);

return;
}

// Compute Stochastic Function
float MaxOsc = sc.GetHighest(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], sc.Index, Input_Length.GetInt());
float MinOsc = sc.GetLowest(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], sc.Index, Input_Length.GetInt());

if (MaxOsc - MinOsc != 0.0f)
    Array_Ratio[sc.Index] = (sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index] - MinOsc) / (MaxOsc - MinOsc);
else
    Array_Ratio[sc.Index] = 0.0f;

float SmoothedRatio = (4.0f * Array_Ratio[sc.Index] + 3.0f * Array_Ratio[sc.Index - 1] + 2.0f * Array_Ratio[sc.Index - 2] + Array_Ratio[sc.Index - 3]) / 10.0f;

Subgraph_Stochastic[sc.Index] = 2.0f * (SmoothedRatio - 0.5f);

// Compute Fisher Function
float LogArgument = (1 + 1.98f * (Subgraph_Stochastic[sc.Index] - 0.5f)) / (1 - 1.98f * (Subgraph_Stochastic[sc.Index] - 0.5f));
float LogArgumentDenom = 1 - 1.98f * (Subgraph_Stochastic[sc.Index] - 0.5f);

const int SelectedIndex = Input_UseAbsoluteValueWhenLogArgumentIsZero.GetYesNo();

switch (SelectedIndex)
{
case 0:
{

```

```

    if (LogArgument > 0.0f && LogArgumentDenom != 0.0f)
        Subgraph_Fisher[sc.Index] = 0.5f * logf(LogArgument);
    else
        Subgraph_Fisher[sc.Index] = 0.0f;
}
break;

case 1:
{
    if (LogArgument != 0.0f && LogArgumentDenom != 0.0f)
    {
        Subgraph_Fisher[sc.Index] = 0.5f * logf(fabs(LogArgument));
    }
    else
        Subgraph_Fisher[sc.Index] = 0.0f;
}
break;
}

Subgraph_Trigger[sc.Index] = Subgraph_Fisher[sc.Index - 1];

Subgraph_CenterLine[sc.Index] = 0.0f;
}

```