

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

```
/*=====*/
SCSFExport scsf_DoubleStochastic(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Bressert = sc.Subgraph[0];
    SCSubgraphRef Subgraph_UpperLine = sc.Subgraph[1];
    SCSubgraphRef Subgraph_LowerLine = sc.Subgraph[2];
    SCSubgraphRef Subgraph_DSTrigger = sc.Subgraph[3];

    SCInputRef Input_HighLowPeriodLength = sc.Input[0];
    SCInputRef Input_EmaLength = sc.Input[1];
    SCInputRef Input_UpperLineValue = sc.Input[2];
    SCInputRef Input_LowerLineValue = sc.Input[3];
    SCInputRef Input_Version = sc.Input[4];
    SCInputRef Input_SmoothingLength = sc.Input[5];

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

        sc.GraphName = "Double Stochastic - Bressert";
        sc.StudyDescription = "";

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

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

        Subgraph_DSTrigger.Name = "DS Trigger";
        Subgraph_DSTrigger.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_DSTrigger.PrimaryColor = RGB(255,0,255);
        Subgraph_DSTrigger.DrawZeros = false;

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

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

        Input_HighLowPeriodLength.Name = "High & Low Period Length";
        Input_HighLowPeriodLength.SetInt(15);
        Input_HighLowPeriodLength.SetIntLimits(1, MAX_STUDY_LENGTH);
        Input_HighLowPeriodLength.DisplayOrder = 1;

        Input_EmaLength.Name = "Stochastic Exponential Mov Avg Length";
        Input_EmaLength.SetInt(18);
        Input_EmaLength.SetIntLimits(1, MAX_STUDY_LENGTH);
        Input_EmaLength.DisplayOrder = 2;

        Input_SmoothingLength.Name = "Smoothing Length";
        Input_SmoothingLength.SetInt(3);
        Input_SmoothingLength.SetIntLimits(1, MAX_STUDY_LENGTH);
        Input_SmoothingLength.DisplayOrder = 3;

        Input_UpperLineValue.Name = "Upper Line Value";
```

```

Input_UpperLineValue.SetFloat(80.0f);
Input_UpperLineValue.DisplayOrder = 4;

Input_LowerLineValue.Name = "Lower Line Value";
Input_LowerLineValue.SetFloat(20.0f);
Input_LowerLineValue.DisplayOrder = 5;

Input_Version.SetInt(1);

return;
}

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

// Do data processing

int Index = sc.Index;
Subgraph_UpperLine[Index] = Input_UpperLineValue.GetFloat();
Subgraph_LowerLine[Index] = Input_LowerLineValue.GetFloat();

int HighLowPeriod = Input_HighLowPeriodLength.GetInt();

float High = sc.GetHighest(sc.High, HighLowPeriod);
float Low = sc.GetLowest(sc.Low, HighLowPeriod);
if (High - Low != 0.0f)
    Subgraph_Bressert.Arrays[0][Index] = ((sc.Close[Index] - Low) / (High - Low)) * 100.0f;
else
    Subgraph_Bressert.Arrays[0][Index] = 0.0f;

sc.ExponentialMovAvg(Subgraph_Bressert.Arrays[0], Subgraph_DSTTrigger, Input_EmaLength.GetInt());

float TriggerHigh = sc.GetHighest(Subgraph_DSTTrigger, HighLowPeriod);
float TriggerLow = sc.GetLowest(Subgraph_DSTTrigger, HighLowPeriod);
if (TriggerHigh - TriggerLow != 0.0f)
    Subgraph_Bressert.Arrays[1][Index] = ((Subgraph_DSTTrigger[Index] - TriggerLow) / (TriggerHigh - TriggerLow)) *
100.0f;
else
    Subgraph_Bressert.Arrays[1][Index] = 0.0f;

sc.ExponentialMovAvg(Subgraph_Bressert.Arrays[1], Subgraph_Bressert.Arrays[2], Input_EmaLength.GetInt());
sc.ExponentialMovAvg(Subgraph_Bressert.Arrays[2], Subgraph_Bressert, Input_SmoothingLength.GetInt());
}

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

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

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

        sc.GraphName = "Previous Bar Close";
        sc.AutoLoop = 1;

        Subgraph_PrevClose.DrawStyle = DRAWSTYLE_STAIR_STEP;
        Subgraph_PrevClose.Name = "PB Close";
    }
}

```

```

Subgraph_PrevClose.LineLabel |= (LL_DISPLAY_VALUE | LL_VALUE_ALIGN_VALUES_SCALE);
Subgraph_PrevClose.PrimaryColor = RGB(0,255,0);

sc.GraphRegion = 0;

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

Input_Version.SetInt(1);

return;
}

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

uint32_t InputDataIndex = Input_InputData.GetInputDataIndex();

// Do data processing
Subgraph_PrevClose[sc.Index] = sc.BaseData[InputDataIndex][sc.Index - 1];
}

/*=====*/
SCSFExport scsf_ChangeForDay(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Change = sc.Subgraph[0];
    SCInputRef Input_TickSize = sc.Input[0];

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

        sc.GraphName = "Change For Day";

        sc.GraphRegion = 1;

        Subgraph_Change.Name = "Change";
        Subgraph_Change.DrawStyle = DRAWSTYLE_DASH;
        Subgraph_Change.SecondaryColorUsed = 1;
        Subgraph_Change.SecondaryColor = RGB(255,0,0);

        Input_TickSize.Name = "Tick Size";
        Input_TickSize.SetFloat(0.25f);

        return;
    }

    // Do data processing
    Subgraph_Change[sc.Index] = sc.Close[sc.ArraySize - 1];

    int OpeningIndex = sc.GetNearestMatchForSCDateTime(sc.ChartNumber,
sc.BaseDateTimeIn.DateAt(sc.ArraySize-1));

    if (sc.BaseDateTimeIn.DateAt(OpeningIndex) != sc.BaseDateTimeIn.DateAt(sc.ArraySize-1))
        OpeningIndex++;

    float ChangeForDay = sc.Close[sc.ArraySize-1] - sc.Open[OpeningIndex];

    for (int i = OpeningIndex; i < sc.ArraySize; i++)
    {
        Subgraph_Change[i] = ChangeForDay/Input_TickSize.GetFloat();
        if (Subgraph_Change[i] >= 0)

```

```

        Subgraph_Change.DataColor[i] = Subgraph_Change.PrimaryColor;
    else
        Subgraph_Change.DataColor[i] = Subgraph_Change.SecondaryColor ;
    }
}

/*=====*/
SCSFExport scsf_JimBergsVolatilitySystem(SCStudyInterfaceRef sc) // scsf_Name
{
    SCSubgraphRef Subgraph_StopL = sc.Subgraph[0];
    SCSubgraphRef Subgraph_StopS = sc.Subgraph[1];

    SCSubgraphRef Subgraph_LLW = sc.Subgraph[6];
    SCSubgraphRef Subgraph_HHV = sc.Subgraph[7];
    SCSubgraphRef Subgraph_LongTrigger = sc.Subgraph[8];
    SCSubgraphRef Subgraph_ShortTrigger = sc.Subgraph[9];
    SCSubgraphRef Subgraph_ATR = sc.Subgraph[11];
    SCSubgraphRef Subgraph_Temp12 = sc.Subgraph[12];
    SCSubgraphRef Subgraph_Temp13 = sc.Subgraph[13];
    SCSubgraphRef Subgraph_LongStopValue = sc.Subgraph[14];
    SCSubgraphRef Subgraph_ShortStopValue = sc.Subgraph[15];

    // Configuration
    if (sc.SetDefaults)
    {
        sc.GraphName = "Jim Berg's Volatility System"; // study name shown in F6 menu and on the chart
        sc.GraphRegion = 0; // zero based region number

        Subgraph_StopL.Name = "Stop L";
        Subgraph_StopL.DrawStyle = DRAWSTYLE_STAIR_STEP;
        Subgraph_StopL.PrimaryColor = RGB(0,185,47); // green

        Subgraph_StopS.Name = "Stop S";
        Subgraph_StopS.DrawStyle = DRAWSTYLE_STAIR_STEP;
        Subgraph_StopS.PrimaryColor = RGB(255,0,128); // red

        sc.AutoLoop = 1;

        return;
    }

    // Data processing

    // LongSignal = C > ( LLV( L, 20 ) + 2 * ATR( 10 ) ); // ATR with Wilder's MA
    // ShortSignal = C < ( HHV( H, 20 ) - 2 * ATR( 10 ) );

    // LongStopValue = HHV( C - 2 * ATR(10), 15 );
    // ShortStopValue = LLV( C + 2 * ATR(10), 15 );

    sc.DataStartIndex = 21; // start drawing subgraph at bar #21 (zero based)

    // LLV(L,20): subgraph #6
    sc.Lowest(sc.Low, Subgraph_LLW, sc.Index, 20);

    // HHV(H,20): subgraph #7
    sc.Highest(sc.High, Subgraph_HHV, sc.Index, 20);

    // ATR(10, Wilder's MA): subgraph #11. Intermediate TR: subgraph #10 (not plotted)
    sc.ATR(sc.BaseDataIn, Subgraph_ATR, sc.Index, 10, MOVAVGTYPE_WILDERS);

    Subgraph_LongTrigger[sc.Index] = Subgraph_LLW[sc.Index] + 2 * Subgraph_ATR[sc.Index]; // Long Signal trigger
    Subgraph_ShortTrigger[sc.Index] = Subgraph_HHV[sc.Index] - 2 * Subgraph_ATR[sc.Index]; // Short Signal trigger

```

```

Subgraph_Temp12[sc.Index] = sc.Close[sc.Index] - 2 * Subgraph_ATR[sc.Index];
Subgraph_Temp13[sc.Index] = sc.Close[sc.Index] + 2 * Subgraph_ATR[sc.Index];

// LongStopValue: subgraph #14
sc.Highest(Subgraph_Temp12, Subgraph_LongStopValue, 15);

// ShortStopValue: subgraph #15
sc.Lowest(Subgraph_Temp13, Subgraph_ShortStopValue, 15);

// If LongSignal=true, plot LongStopValue in Subgraph #0, otherwise plot zero
if (sc.Close[sc.Index] > Subgraph_LongTrigger[sc.Index])
    Subgraph_StopL[sc.Index] = Subgraph_LongStopValue[sc.Index];
else
    Subgraph_StopL[sc.Index] = 0;

// If ShortSignal=true, plot ShortStopValue in Subgraph #1, otherwise plot zero
if (sc.Close[sc.Index] < Subgraph_ShortTrigger[sc.Index])
    Subgraph_StopS[sc.Index] = Subgraph_ShortStopValue[sc.Index];
else
    Subgraph_StopS[sc.Index] = 0;
}

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

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

    // Configuration
    if (sc.SetDefaults)
    {
        sc.GraphName = "Moving Average - Volume Weighted";
        sc.StudyDescription = "Volume Weighted Moving Average";
        sc.GraphRegion = 0; // zero based region number

        Subgraph_VWMA.Name = "VWMA";
        Subgraph_VWMA.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_VWMA.PrimaryColor = RGB(0,185,47); // green

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

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

        sc.ValueFormat = 3;

        sc.AutoLoop = 1;

        return;
    }

    // Data processing

    sc.DataStartIndex = Input_Length.GetInt();
    sc.VolumeWeightedMovingAverage(sc.BaseData[Input_InputData.GetInputDataIndex()], sc.Volume,
    Subgraph_VWMA, Input_Length.GetInt());
}

/*=====*/

```

```

SCSFExport scsf_InsideOrEqualsBar(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_IB = sc.Subgraph[0];

    // Set configuration variables

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

        sc.GraphName = "Inside Or Equals Bar";

        sc.StudyDescription = "";

        sc.GraphRegion = 0;

        Subgraph_IB.Name = "IB";
        Subgraph_IB.DrawStyle = DRAWSTYLE_COLOR_BAR;
        Subgraph_IB.PrimaryColor = RGB(0, 0, 255);
        Subgraph_IB.DrawZeros = false;

        sc.AutoLoop = 1;

        return;
    }

    // Do data processing

    if (sc.Index < 1)
        return;

    // Array references
    SCFloatArrayRef High = sc.High;
    SCFloatArrayRef Low = sc.Low;

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

/*=====*/
SCSFExport scsf_AroonOscillator(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_AroonOscillator = sc.Subgraph[0];
    SCSubgraphRef Subgraph_AroonCalculations = sc.Subgraph[1];

    SCInputRef Input_InputDataHigh = sc.Input[0];
    SCInputRef Input_Length = sc.Input[1];
    SCInputRef Input_InputDataLow = sc.Input[2];

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

        sc.GraphName = "Aroon Oscillator";
        sc.AutoLoop = 1;
        sc.GraphRegion = 1;

        Subgraph_AroonOscillator.Name = "Aroon Oscillator";
        Subgraph_AroonOscillator.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_AroonOscillator.PrimaryColor = RGB(0,255,0);
        Subgraph_AroonOscillator.DrawZeros = true;
    }
}

```

```

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

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

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

return;
}

// Do data processing

sc.AroonIndicator(sc.BaseDataIn[Input_InputDataHigh.GetInputDataIndex()],
sc.BaseDataIn[Input_InputDataLow.GetInputDataIndex()], Subgraph_AroonCalculations, Input_Length.GetInt());

Subgraph_AroonOscillator[sc.Index] = Subgraph_AroonCalculations.Arrays[1][sc.Index];
}

/*=====*/
SCSFExport scsf_AroonIndicator(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_AroonUp = sc.Subgraph[0];
    SCSubgraphRef Subgraph_AroonDown = sc.Subgraph[1];

    SCInputRef Input_InputDataHigh = sc.Input[0];
    SCInputRef Input_Length = sc.Input[1];
    SCInputRef Input_InputDataLow = sc.Input[2];

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

        sc.GraphName = "Aroon Indicator";
        sc.AutoLoop = 1;
        sc.GraphRegion = 1;

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

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

        Input_InputDataHigh.Name = "Input Data High";
        Input_InputDataHigh.SetInputDataIndex(SC_HIGH); // default data field

        Input_Length.Name = "Length";
        Input_Length.SetInt(5); // default integer value
        Input_Length.SetIntLimits(1,MAX_STUDY_LENGTH); // min-max value

        Input_InputDataLow.Name = "Input Data Low";
        Input_InputDataLow.SetInputDataIndex(SC_LOW); // default data field

        return;
    }
}

```

```

    sc.AroonIndicator(sc.BaseDataIn[Input_InputDataHigh.GetInputDataIndex()],
sc.BaseDataIn[Input_InputDataLow.GetInputDataIndex()], Subgraph_AroonUp, Input_Length.GetInt());

    Subgraph_AroonDown[sc.Index] = Subgraph_AroonUp.Arrays[0][sc.Index];
}

/*=====
Study function for Welles Wilder's Plus and Minus Directional Indicators
+DI and -DI.
-----*/
SCSFExport scsf_DMI(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_DIPlus = sc.Subgraph[0];
    SCSubgraphRef Subgraph_DIMinus = sc.Subgraph[1];

    SCInputRef Input_Length = sc.Input[3];

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

        sc.GraphName = "Directional Movement Index";

        sc.StudyDescription = "Welles Wilder's Plus and Minus Directional Indicators +DI and -DI.";

        sc.AutoLoop = 1; // true

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

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

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

        return;
    }

    // Do data processing

    sc.DataStartIndex = Input_Length.GetInt();

    sc.DMI(
        sc.BaseData,
        Subgraph_DIPlus, // DMI+
        Subgraph_DIMinus, // DMI-
        Input_Length.GetInt());

    //sc.CrossOver(DIPlus.Data, DIMinus.Data);
}

/*=====
-----*/
SCSFExport scsf_DMIOscillator(SCStudyInterfaceRef sc)

```



```

{
    SCSubgraphRef Subgraph_DMIDiff = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Line = sc.Subgraph[1];

    SCInputRef Input_Length = sc.Input[3];
    SCInputRef Input_ShowAbsValues = sc.Input[4];

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

        sc.GraphName = "Directional Movement Oscillator";

        sc.StudyDescription = "Calculates the difference between the Directional Indicators -DI and the +DI.";

        sc.AutoLoop = 1; // true

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

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

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

        Input_ShowAbsValues.Name = "Show Absolute Values";
        Input_ShowAbsValues.SetYesNo(0);

        return;
    }

    // Do data processing

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

    sc.DMIDiff(sc.BaseData, Subgraph_DMIDiff, sc.Index, Input_Length.GetInt()); // DMI Diff

    if(Input_ShowAbsValues.GetYesNo())
        Subgraph_DMIDiff[sc.Index] = fabs(Subgraph_DMIDiff[sc.Index]);
}

/*=====
Average Directional Movement Index study. This is calculated according
to the Welles Wilder formulas.
-----*/
SCSFExport scsf_ADX(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_ADX = sc.Subgraph[0];

    SCInputRef Input_DXLength = sc.Input[3];
    SCInputRef Input_DXMALength = sc.Input[4];

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

        sc.GraphName = "ADX";

        sc.StudyDescription = "Average Directional Movement Index. This is calculated according to the Welles Wilder

```

```
formulas.";
```

```
    sc.AutoLoop = 1; // true
    sc.GraphRegion = 1;

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

    Input_DXLength.Name = "DX Length";
    Input_DXLength.SetInt(14);
    Input_DXLength.SetIntLimits(1, INT_MAX);

    Input_DXMALength.Name = "DX Mov Avg Length";
    Input_DXMALength.SetInt(14);
    Input_DXMALength.SetIntLimits(1, INT_MAX);

    return;
}

// Do data processing

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

sc.ADX(
    sc.BaseData,
    Subgraph_ADX,
    Input_DXLength.GetInt(),
    Input_DXMALength.GetInt());
}

/*=====
Average Directional Movement Index Rating study. This is calculated
according to the Welles Wilder formulas.
-----*/
SCSFExport scsf_ADXR(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_ADXR = sc.Subgraph[0];

    SCInputRef Input_DXLength = sc.Input[3];
    SCInputRef Input_DXMALength = sc.Input[4];
    SCInputRef Input_ADXRInterval = sc.Input[5];

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

        sc.GraphName = "ADXR";

        sc.StudyDescription = "Average Directional Movement Index Rating. This is calculated according to the Welles
Wilder formulas.";

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

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

        Input_DXLength.Name = "DX Length";
        Input_DXLength.SetInt(14);
        Input_DXLength.SetIntLimits(1, INT_MAX);
```

```

    Input_DXMALength.Name = "DX Mov Avg Length";
    Input_DXMALength.SetInt(14);
    Input_DXMALength.SetIntLimits(1, INT_MAX);

    Input_ADXRInterval.Name = "ADXR Interval";
    Input_ADXRInterval.SetInt(14);
    Input_ADXRInterval.SetIntLimits(1, INT_MAX);

    return;
}

// Do data processing

sc.DataStartIndex = Input_DXLength.GetInt() + Input_DXMALength.GetInt() + Input_ADXRInterval.GetInt() - 2;

sc.ADXR(
    sc.BaseData,
    Subgraph_ADXR,
    Input_DXLength.GetInt(),
    Input_DXMALength.GetInt(),
    Input_ADXRInterval.GetInt());
}

/*=====
This example code calculates a smoothed moving average (5 period by
default).
-----*/
SCSFExport scsf_SmoothedMovingAverage(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_SMMA = sc.Subgraph[0];
    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_Length = sc.Input[1];

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

        sc.GraphName = "Moving Average - Smoothed";

        sc.StudyDescription = "Function for calculating a Wilders Smooth Moving Average (SMMA).";

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

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

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

        // Make the Length input and default it to 5
        Input_Length.Name = "Length";
        Input_Length.SetInt(5);
        Input_Length.SetIntLimits(2, MAX_STUDY_LENGTH);

```

```

    sc.AutoLoop = 1;

    // Must return before doing any data processing if sc.SetDefaults is set
    return;
}

// Do data processing

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

    sc.SmoothedMovingAverage(sc.BaseData[Input_InputData.GetInputDataIndex()], Subgraph_SMMA, sc.Index,
Input_Length.GetInt());
}

/*=====*/
SCSFExport scsf_PercentagePriceOscillator(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_PPO = sc.Subgraph[0];
    SCSubgraphRef Subgraph_ShortMA = sc.Subgraph[1];
    SCSubgraphRef Subgraph_LongMA = sc.Subgraph[2];

    SCInputRef Input_LongMALength = sc.Input[0];
    SCInputRef Input_ShortMALength = sc.Input[1];
    SCInputRef Input_MAType = sc.Input[2];
    SCInputRef Input_InputData = sc.Input[3];

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

        sc.GraphName = "Percentage Price Oscillator";

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

        Input_LongMALength.Name = "Long Mov Avg Length";
        Input_LongMALength.SetInt(30);
        Input_LongMALength.SetIntLimits(1, INT_MAX);

        Input_ShortMALength.Name = "Short Mov Avg Length";
        Input_ShortMALength.SetInt(10);
        Input_ShortMALength.SetIntLimits(1, INT_MAX);

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

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

        sc.AutoLoop = 1;

        return;
    }

    sc.MovingAverage(sc.BaseData[Input_InputData.GetInputDataIndex()], Subgraph_LongMA,
Input_MAType.GetMovAvgType(),Input_LongMALength.GetInt());

    sc.MovingAverage(sc.BaseData[Input_InputData.GetInputDataIndex()], Subgraph_ShortMA,
Input_MAType.GetMovAvgType(),Input_ShortMALength.GetInt());

    if (Subgraph_LongMA[sc.Index] == 0)
        Subgraph_PPO[sc.Index] = Subgraph_PPO[sc.Index - 1];
    else

```

```

        Subgraph_PPO[sc.Index] = ( (Subgraph_ShortMA[sc.Index] - Subgraph_LongMA[sc.Index]) /
Subgraph_LongMA[sc.Index]) * 100;
    }
    /*=====*/

```

```

SCSFExport scsf_HerrickPayoffIndex(SCStudyInterfaceRef sc)

```

```

{
    SCSubgraphRef Subgraph_HPI = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Zero = sc.Subgraph[1];

    SCInputRef Input_ValueOfMove = sc.Input[0];
    SCInputRef Input_SmoothingMult = sc.Input[1];
    SCInputRef Input_MaximumOrMinimum = sc.Input[2];
    SCInputRef Input_Divisor = sc.Input[3];

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

        sc.GraphName = "Herrick Payoff Index";

        Subgraph_HPI.Name = "HPI";
        Subgraph_HPI.PrimaryColor = RGB(0, 255, 0); // Red
        Subgraph_HPI.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_HPI.DrawZeros = true;

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

        Input_ValueOfMove.Name = "Value of a .01 Move";
        Input_ValueOfMove.SetFloat(0.5f);

        Input_SmoothingMult.Name = "Smoothing multiplier";
        Input_SmoothingMult.SetFloat(0.1f);

        Input_MaximumOrMinimum.Name = "Maximum or Minimum Open Interest: 1= maximum, 2= minimum";
        Input_MaximumOrMinimum.SetInt(1);
        Input_MaximumOrMinimum.SetIntLimits(1, 2);

        Input_Divisor.Name = "Divisor";
        Input_Divisor.SetFloat(100000.0f);

        sc.AutoLoop = 1;

        return;
    }

    if (sc.Index == 0)
        return;

    float CurrentVolume = sc.Volume[sc.Index];

    // current and previous mean price (high + low) / 2
    float AveragePrice = sc.HLAvg[sc.Index];
    float PriorAveragePrice = sc.HLAvg[sc.Index - 1];

    float CurrentOpenInterest = sc.OpenInterest[sc.Index];
    float PriorOpenInterest = sc.OpenInterest[sc.Index-1];

    if (CurrentOpenInterest<=0.0f)

```

```

{
    CurrentOpenInterest = PriorOpenInterest;
}

// Determine sign for calculations
float sign = ( AveragePrice > PriorAveragePrice ) ? 1.0f : -1.0f;

// the absolute value of the difference between current open interest and
// prior open interest
float PositiveOpenInterest = fabs(CurrentOpenInterest - PriorOpenInterest);

float MaximumOrMinimumOpenInterest = (Input_MaximumOrMinimum.GetInt()==1) ? max(CurrentOpenInterest,
PriorOpenInterest) : min(CurrentOpenInterest, PriorOpenInterest);

float PriorHerrickPayoffIndex = Subgraph_HPI[sc.Index - 1];

if (MaximumOrMinimumOpenInterest>0.0f)
{
    float CalculationResult = Input_ValueOfMove.GetFloat() * CurrentVolume * (AveragePrice - PriorAveragePrice) *
(1 + sign*2*PositiveOpenInterest/MaximumOrMinimumOpenInterest)/Input_Divisor.GetFloat();

    if(sc.Index>1)
    {
        Subgraph_HPI[sc.Index] = PriorHerrickPayoffIndex + (CalculationResult-
PriorHerrickPayoffIndex)*Input_SmoothingMult.GetFloat();
    }
    else
    {
        Subgraph_HPI[sc.Index] = CalculationResult;
    }
}
else
{
    Subgraph_HPI[sc.Index] = PriorHerrickPayoffIndex;
}

}

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

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

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

        Subgraph_TRIX.Name = "TRIX";
        Subgraph_TRIX.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_TRIX.PrimaryColor = RGB(0,255,0);

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

        // Make the Length input and default it to 3
        Input_Length.Name = "Length";
        Input_Length.SetInt(3);
        Input_Length.SetIntLimits(1,MAX_STUDY_LENGTH);

        sc.AutoLoop = 1;
    }
}

```

```

    return;
}

// Data start and update start
sc.DataStartIndex = (Input_Length.GetInt() - 1) * 3 + 1;

sc.TRIX(sc.BaseData[Input_InputData.GetInputDataIndex()], Subgraph_TRIX, Input_Length.GetInt());
}

/*=====
* Method      - Momentum Gauging
* Description  -
* Price = Close
* Length = 20
*
* Histo = LinearRegression[Length] (price - ((Highest[Length](High) + Lowest[Length](Low))/2 +
ExponentialAverage[Length](close))/2 )
*
* RETURN Histo as "Histogramme"[/color][/color]
*
=====*/
SCSFExport scsf_MomentumGauging(SCStudyInterfaceRef sc)
{
    const int INDEX_EMA = 1;
    const int INDEX_LREG_SRC = 2;

    SCSubgraphRef Subgraph_Momentum = sc.Subgraph[0];
    SCSubgraphRef Subgraph_EMA = sc.Subgraph[1];
    SCSubgraphRef Subgraph_LREG_SRC = sc.Subgraph[2];

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

    if (sc.SetDefaults)
    {
        sc.GraphName = "Momentum Gauging";

        Subgraph_Momentum.Name = "Momentum";
        Subgraph_Momentum.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_Momentum.PrimaryColor = RGB(0,255,0);
        //Momentum.SecondaryColorUsed = 1;
        Subgraph_Momentum.SecondaryColor = RGB(255,0,0);

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

        // Make the Length input and default it to 3
        Input_Length.Name = "Length";
        Input_Length.SetInt(3);
        Input_Length.SetIntLimits(1,MAX_STUDY_LENGTH);

        sc.AutoLoop = 1;

        return;
    }

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

    sc.ExponentialMovAvg(sc.Close, Subgraph_EMA, sc.Index, Input_Length.GetInt()); // Note: EMA returns close when
index is < Length.GetInt()

    float hlh = sc.GetHighest(sc.High, sc.Index, Input_Length.GetInt());

```

```

float lll = sc.GetLowest(sc.Low, sc.Index, Input_Length.GetInt());

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

// populate the source data for linear regression
Subgraph_LREG_SRC[sc.Index] = price[sc.Index] - ((hlh + lll)/2.0f + Subgraph_EMA[sc.Index])/2.0f;

sc.LinearRegressionIndicator(Subgraph_LREG_SRC, Subgraph_Momentum, sc.Index, Input_Length.GetInt());

/*if (Momentum[sc.Index] >= 0)
    Momentum.DataColor[sc.Index] = Momentum.PrimaryColor;
else
    Momentum.DataColor[sc.Index] = Momentum.SecondaryColor ;*/

}

/*=====

function main(nInputLength)
{
    var vC = close(0);
    var vPH = high(-1);
    var vPL = low(-1);

    if(vC == null || vPH == null || vPL == null) return;

    if (vC > vPH) {
        setPriceBarColor(Color.lime);
    } else if(vC < vPL) {
        setPriceBarColor(Color.red);
    }

    return;
}

=====*/
SCSFExport scsf_HHLLCandles(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_HH = sc.Subgraph[0];
    SCSubgraphRef Subgraph_LL = sc.Subgraph[1];
    SCSubgraphRef Subgraph_N = sc.Subgraph[2];

    if (sc.SetDefaults)
    {
        sc.GraphName = "HH LL Bars";
        sc.StudyDescription = "";

        sc.GraphRegion = 0;

        Subgraph_HH.Name = "HH";
        Subgraph_HH.DrawStyle = DRAWSTYLE_COLOR_BAR_CANDLE_FILL;
        Subgraph_HH.PrimaryColor = RGB(0, 255, 0);
        Subgraph_HH.SecondaryColor = RGB(0, 0, 0);
        Subgraph_HH.SecondaryColorUsed = 1;
        Subgraph_HH.DrawZeros = false;

        Subgraph_LL.Name = "LL";
        Subgraph_LL.DrawStyle = DRAWSTYLE_COLOR_BAR_CANDLE_FILL;
        Subgraph_LL.PrimaryColor = RGB(255, 0, 0);
        Subgraph_LL.SecondaryColor = RGB(0, 0, 0);
        Subgraph_LL.SecondaryColorUsed = 1;
        Subgraph_LL.DrawZeros = false;

        Subgraph_N.Name = "N";
        Subgraph_N.DrawStyle = DRAWSTYLE_COLOR_BAR;
        Subgraph_N.PrimaryColor = RGB(50, 50, 190);
    }
}

```



```

    Subgraph_N.DrawZeros = false;

    sc.AutoLoop = 1;

    return;
}

sc.ValueFormat = sc.BaseGraphValueFormat;

sc.DataStartIndex = 1;
if (sc.Index<1)
    return;

float priorHigh = sc.High[sc.Index - 1];
float priorLow = sc.Low[sc.Index - 1];

float currentClose = sc.Close[sc.Index];

Subgraph_HH[sc.Index] = 0;
Subgraph_LL[sc.Index] = 0;
Subgraph_N[sc.Index] = 0;

if (sc.FormattedEvaluate(currentClose, sc.BaseGraphValueFormat, GREATER_OPERATOR, priorHigh,
sc.BaseGraphValueFormat))
{
    Subgraph_HH[sc.Index] = 1;
    //We need to set these to 0 because conditions can change on the last bar during real-time updating.
    Subgraph_N[sc.Index] = 0;
    Subgraph_LL[sc.Index] = 0;
}
else if (sc.FormattedEvaluate(currentClose, sc.BaseGraphValueFormat, LESS_OPERATOR, priorLow,
sc.BaseGraphValueFormat))
{
    Subgraph_LL[sc.Index] = 1;
    Subgraph_N[sc.Index] = 0;
    Subgraph_HH[sc.Index] = 0;
}
else
{
    Subgraph_N[sc.Index] = 1;
    Subgraph_HH[sc.Index] = 0;
    Subgraph_LL[sc.Index] = 0;
}
}

/*=====*/
SCSFExport scsf_RahulMohindarOscillator(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_ST1 = sc.Subgraph[0];
    SCSubgraphRef Subgraph_ST2 = sc.Subgraph[1];
    SCSubgraphRef Subgraph_ST3 = sc.Subgraph[2];
    SCSubgraphRef Subgraph_RMO = sc.Subgraph[3];
    SCSubgraphRef Subgraph_Refline = sc.Subgraph[4];
    SCSubgraphRef Subgraph_Buy = sc.Subgraph[5];
    SCSubgraphRef Subgraph_Sell = sc.Subgraph[6];

    SCFloatArrayRef Array_MATemp3 = sc.Subgraph[0].Arrays[0];
    SCFloatArrayRef Array_MATemp4 = sc.Subgraph[0].Arrays[1];
    SCFloatArrayRef Array_MATemp5 = sc.Subgraph[0].Arrays[2];
    SCFloatArrayRef Array_MATemp6 = sc.Subgraph[0].Arrays[3];
    SCFloatArrayRef Array_MATemp7 = sc.Subgraph[0].Arrays[4];
    SCFloatArrayRef Array_MATemp8 = sc.Subgraph[0].Arrays[5];
    SCFloatArrayRef Array_MATemp9 = sc.Subgraph[0].Arrays[6];
    SCFloatArrayRef Array_MATemp10 = sc.Subgraph[0].Arrays[7];

```

```

SCFloatArrayRef Array_MATemp11 = sc.Subgraph[0].Arrays[8];
SCFloatArrayRef Array_MATemp12 = sc.Subgraph[0].Arrays[9];
SCFloatArrayRef Array_Highest = sc.Subgraph[0].Arrays[10];
SCFloatArrayRef Array_Lowest = sc.Subgraph[0].Arrays[11];

SCInputRef Input_Len1 = sc.Input[0];
SCInputRef Input_Len2 = sc.Input[1];
SCInputRef Input_Len3 = sc.Input[2];
SCInputRef Input_Len4 = sc.Input[3];
SCInputRef Input_OffsetPercent = sc.Input[4];
SCInputRef Input_InputData = sc.Input[5];
SCInputRef Input_Version = sc.Input[6];

if (sc.SetDefaults)
{
    sc.GraphName = "Rahul Mohindar Oscillator";

    Subgraph_ST1.Name = "Swing Trade 1";
    Subgraph_ST1.DrawStyle = DRAWSTYLE_IGNORE;
    Subgraph_ST1.PrimaryColor = RGB(0, 255, 0);

    Subgraph_ST2.Name = "Swing Trade 2";
    Subgraph_ST2.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_ST2.PrimaryColor = RGB(255, 0, 255);

    Subgraph_ST3.Name = "Swing Trade 3";
    Subgraph_ST3.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_ST3.PrimaryColor = RGB(255, 255, 0);

    Subgraph_RMO.Name = "Rahul Mohindar Oscillator";
    Subgraph_RMO.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_RMO.PrimaryColor = RGB(0, 0, 255);

    Subgraph_Refline.Name = "Reference Line";
    Subgraph_Refline.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_Refline.DrawZeros = true;
    Subgraph_Refline.PrimaryColor = RGB(255, 255, 255);

    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_Len1.Name = "Length 1";
    Input_Len1.SetInt(2);

    Input_Len2.Name = "Length 2";
    Input_Len2.SetInt(10);

    Input_Len3.Name = "Length 3";
    Input_Len3.SetInt(30);

    Input_Len4.Name = "Length 4";
    Input_Len4.SetInt(81);

    Input_OffsetPercent.Name = "Arrow Offset Percentage";
    Input_OffsetPercent.SetInt(8);

```

```

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

Input_Version.SetInt(1);

sc.AutoLoop = 1;

return;
}

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

    Input_Version.SetInt(1);
}

sc.DataStartIndex = max(Input_Len1.GetInt(), max(Input_Len2.GetInt(), max(Input_Len3.GetInt(),
Input_Len4.GetInt())));

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

sc.SimpleMovAvg(BaseDataArray, Array_MATemp3, Input_Len1.GetInt());
sc.SimpleMovAvg(Array_MATemp3, Array_MATemp4, Input_Len1.GetInt());
sc.SimpleMovAvg(Array_MATemp4, Array_MATemp5, Input_Len1.GetInt());
sc.SimpleMovAvg(Array_MATemp5, Array_MATemp6, Input_Len1.GetInt());
sc.SimpleMovAvg(Array_MATemp6, Array_MATemp7, Input_Len1.GetInt());
sc.SimpleMovAvg(Array_MATemp7, Array_MATemp8, Input_Len1.GetInt());
sc.SimpleMovAvg(Array_MATemp8, Array_MATemp9, Input_Len1.GetInt());
sc.SimpleMovAvg(Array_MATemp9, Array_MATemp10, Input_Len1.GetInt());
sc.SimpleMovAvg(Array_MATemp10, Array_MATemp11, Input_Len1.GetInt());
sc.SimpleMovAvg(Array_MATemp11, Array_MATemp12, Input_Len1.GetInt());
sc.Highest(BaseDataArray, Array_Highest, Input_Len2.GetInt());
sc.Lowest(BaseDataArray, Array_Lowest, Input_Len2.GetInt());

float Range = Array_Highest[sc.Index] - Array_Lowest[sc.Index];

if (Range != 0)
{
    Subgraph_ST1[sc.Index] =
        100 * (BaseDataArray[sc.Index]
            - (Array_MATemp3[sc.Index]
                + Array_MATemp4[sc.Index]
                + Array_MATemp5[sc.Index]
                + Array_MATemp6[sc.Index]
                + Array_MATemp7[sc.Index]
                + Array_MATemp8[sc.Index]
                + Array_MATemp9[sc.Index] + Array_MATemp10[sc.Index]
                + Array_MATemp11[sc.Index]
                + Array_MATemp12[sc.Index]) / 10) / Range;
}

sc.ExponentialMovAvg(Subgraph_ST1, Subgraph_ST2, Input_Len3.GetInt());
sc.ExponentialMovAvg(Subgraph_ST2, Subgraph_ST3, Input_Len3.GetInt());
sc.ExponentialMovAvg(Subgraph_ST1, Subgraph_RMO, Input_Len4.GetInt());
Subgraph_Recline[sc.Index] = 0;

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

if (sc.CrossOver(Subgraph_ST3, Subgraph_ST2) == CROSS_FROM_BOTTOM)
{
    Subgraph_Buy[sc.Index] = Subgraph_ST3[sc.Index] - OffsetPercent * Subgraph_ST3[sc.Index];
    Subgraph_Sell[sc.Index] = 0;
}

```

```

    }
    else if (sc.CrossOver(Subgraph_ST3, Subgraph_ST2) == CROSS_FROM_TOP)
    {
        Subgraph_Sell[sc.Index] = Subgraph_ST3[sc.Index] + OffsetPercent * Subgraph_ST3[sc.Index];
        Subgraph_Buy[sc.Index] = 0;
    }
    else
    {
        Subgraph_Buy[sc.Index] = 0;
        Subgraph_Sell[sc.Index] = 0;
    }
}

```

```

/* *****
*
* Calculation:
* The value of the DeMarker for the "i" interval is calculated as follows:
*   The DeMax(i) is calculated:
*
* If high(i) > high(i-1) , then DeMax(i) = high(i)-high(i-1), otherwise DeMax(i) = 0
*   The DeMin(i) is calculated:
*
* If low(i) < low(i-1) , then DeMin(i) = low(i)-low(i-1), otherwise DeMin(i) = 0
*   The value of the DeMarker is calculated as:
*
* DMark(i) = SMA(DeMax, N)/(SMA(DeMax, N)+SMA(DeMin, N))
*
*****

```

```

SCSFExport scsf_Demarker(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Demarker = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Demax = sc.Subgraph[1];
    SCSubgraphRef Subgraph_SMA_Demax = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Demin = sc.Subgraph[3];
    SCSubgraphRef Subgraph_SMA_Demin = sc.Subgraph[4];

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

        Subgraph_Demarker.Name="Demarker";
        Subgraph_Demarker.DrawStyle=DRAWSTYLE_LINE;
        Subgraph_Demarker.PrimaryColor = RGB(0,255,0);

        sc.Input[0].Name="Length";
        sc.Input[0].SetInt(13);
        sc.Input[0].SetIntLimits(1,MAX_STUDY_LENGTH);

        sc.AutoLoop = 1;

        return;
    }

    int length = sc.Input[0].GetInt();

    sc.DataStartIndex = length;

    if (sc.Index > 0)
    {
        // If high(i) > high(i-1) , then DeMax(i) = high(i)-high(i-1), otherwise DeMax(i) = 0

        float high = sc.High[sc.Index];
        float highOld = sc.High[sc.Index-1];
    }
}

```

```

    if (high > highOld)
    {
        Subgraph_Demax[sc.Index] = high - highOld;
    }
    else
    {
        Subgraph_Demax[sc.Index] = 0.0f;
    }

    // If low(i) < low(i-1), then DeMin(i) = low(i-1)-low(i), otherwise DeMin(i) = 0

    float low = sc.Low[sc.Index];
    float lowOld = sc.Low[sc.Index-1];

    if (low < lowOld)
    {
        Subgraph_Demin[sc.Index] = lowOld - low;
    }
    else
    {
        Subgraph_Demin[sc.Index] = 0.0f;
    }
}
else
{
    Subgraph_Demax[sc.Index] = 0.0f;
    Subgraph_Demin[sc.Index] = 0.0f;
}

// DMark(i) = SMA(DeMax, N)/(SMA(DeMax, N)+SMA(DeMin, N))

sc.SimpleMovAvg(Subgraph_Demax, Subgraph_SMA_Demax, length);
sc.SimpleMovAvg(Subgraph_Demin, Subgraph_SMA_Demin, length);

float summ = Subgraph_SMA_Demax[sc.Index] + Subgraph_SMA_Demin[sc.Index];
if (summ != 0.0f)
{
    Subgraph_Demarker[sc.Index] = Subgraph_SMA_Demax[sc.Index] / summ;
}
else
{
    Subgraph_Demarker[sc.Index] = Subgraph_Demarker[sc.Index-1];
}
}

/*=====*/
SCSFExport scsf_StochasticRSI(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_StochasticRSI = sc.Subgraph[0];
    SCSubgraphRef Subgraph_RSI = sc.Subgraph[1];

    SCInputRef Input_RSILength = sc.Input[0];
    SCInputRef Input_RSIAvgType = sc.Input[1];
    SCInputRef Input_RSIHighestLowestLength = sc.Input[2];

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

        Subgraph_StochasticRSI.Name="Stochastic RSI";
        Subgraph_StochasticRSI.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_StochasticRSI.PrimaryColor = RGB(0,255,0);
        Subgraph_StochasticRSI.DrawZeros = true;
    }
}

```

```

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

Input_RSIAvgType.Name="RSI Average Type";
Input_RSIAvgType.SetMovAvgType(MOVAVGTYPE_SIMPLE);

Input_RSIIighestLowestLength.Name = "RSI Highest Lowest Length";
Input_RSIIighestLowestLength.SetInt(14);
Input_RSIIighestLowestLength.SetIntLimits(1, MAX_STUDY_LENGTH);

sc.AutoLoop = 1;

return;
}

sc.DataStartIndex = Input_RSILength.GetInt() + Input_RSIIighestLowestLength.GetInt() - 1;

if (Input_RSIIighestLowestLength.GetInt() == 1)
    Input_RSIIighestLowestLength.SetInt(Input_RSILength.GetInt());

if (sc.Index<1)
{
    return;
}

sc.RSI(sc.Close, Subgraph_RSI, Input_RSIAvgType.GetMovAvgType(), Input_RSILength.GetInt());

// computing Stochastic of RSI
float MaximumRSI = sc.GetHighest(Subgraph_RSI, sc.Index, Input_RSIIighestLowestLength.GetInt());
float MinimumRSI = sc.GetLowest(Subgraph_RSI, sc.Index, Input_RSIIighestLowestLength.GetInt());

if (MaximumRSI-MinimumRSI != 0.0f)
{
    Subgraph_StochasticRSI[sc.Index] = (Subgraph_RSI[sc.Index]-MinimumRSI)/(MaximumRSI - MinimumRSI);
}
else
{
    Subgraph_StochasticRSI[sc.Index] = Subgraph_StochasticRSI[sc.Index-1];
}
}
/*=====*/
SCSFExport scsf_MinerDTOScillator(SCStudyInterfaceRef sc)
{
    SCSubgraphRef SK      = sc.Subgraph[0];
    SCSubgraphRef SD      = sc.Subgraph[1];
    SCSubgraphRef UpperLine = sc.Subgraph[2];
    SCSubgraphRef ZeroLine = sc.Subgraph[3];
    SCSubgraphRef LowerLine = sc.Subgraph[4];

    SCFloatArrayRef RSI      = SK.Arrays[0];
    SCFloatArrayRef RsiUpSums  = SK.Arrays[1];
    SCFloatArrayRef RsiDownSums  = SK.Arrays[2];
    SCFloatArrayRef RsiSmoothedUpSums = SK.Arrays[3];
    SCFloatArrayRef RsiSmoothedDownSums = SK.Arrays[4];
    SCFloatArrayRef StochRSI    = SK.Arrays[5];

    SCInputRef RSILength  = sc.Input[0];
    SCInputRef RSIAvgType = sc.Input[1];
    SCInputRef StochLength = sc.Input[2];
    SCInputRef SKLength   = sc.Input[3];
    SCInputRef SKAvgType  = sc.Input[4];
    SCInputRef SDLength   = sc.Input[5];
    SCInputRef SDAvgType  = sc.Input[6];

```

```

SCInputRef UpperValue = sc.Input[7];
SCInputRef LowerValue = sc.Input[8];

if(sc.SetDefaults)
{
    sc.GraphName="DT Oscillator";
    sc.StudyDescription="DT Oscillator - Robert Miner";

    SK.Name="DT Osc SK";
    SK.DrawStyle = DRAWSTYLE_LINE;
    SK.PrimaryColor = COLOR_RED;
    SK.DrawZeros = true;

    SD.Name="DT Osc SD";
    SD.DrawStyle = DRAWSTYLE_LINE;
    SD.PrimaryColor = COLOR_GREEN;
    SD.DrawZeros = true;

    UpperLine.Name="Upper";
    UpperLine.DrawStyle = DRAWSTYLE_LINE;
    UpperLine.LineStyle = LINESTYLE_DOT;
    UpperLine.PrimaryColor = COLOR_GRAY;
    UpperLine.DisplayNameValueInWindowsFlags = 0;
    UpperLine.DrawZeros = true;

    ZeroLine.Name="Zero";
    ZeroLine.DrawStyle = DRAWSTYLE_IGNORE;
    ZeroLine.LineStyle = LINESTYLE_DOT;
    ZeroLine.PrimaryColor = COLOR_GRAY;
    ZeroLine.DisplayNameValueInWindowsFlags = 0;
    ZeroLine.DrawZeros = true;

    LowerLine.Name="Lower";
    LowerLine.DrawStyle = DRAWSTYLE_LINE;
    LowerLine.LineStyle = LINESTYLE_DOT;
    LowerLine.PrimaryColor = COLOR_GRAY;
    LowerLine.DisplayNameValueInWindowsFlags = 0;
    LowerLine.DrawZeros = true;

    RSILength.Name="RSI Length";
    RSILength.SetInt(13);

    RSIAvgType.Name="RSI Average Type";
    RSIAvgType.SetMovAvgType(MOVAVGTYPE_SIMPLE);

    StochLength.Name="Stochastic Length";
    StochLength.SetInt(8);

    SKLength.Name="SK Length";
    SKLength.SetInt(5);

    SKAvgType.Name="SK Average Type";
    SKAvgType.SetMovAvgType(MOVAVGTYPE_SIMPLE);

    SDLength.Name="SD Length";
    SDLength.SetInt(3);

    SDAvgType.Name="SD Average Type";
    SDAvgType.SetMovAvgType(MOVAVGTYPE_SIMPLE);

    UpperValue.Name="Upper Line Value";
    UpperValue.SetFloat(75.0f);

    LowerValue.Name="Lower Line Value";
    LowerValue.SetFloat(25.0f);

```

```

    sc.AutoLoop = 1;

    return;
}

sc.DataStartIndex = RSILength.GetInt();

UpperLine[sc.Index] = UpperValue.GetFloat();
ZeroLine[sc.Index] = 0;
LowerLine[sc.Index] = LowerValue.GetFloat();

if (sc.Index<1)
{
    return;
}

sc.RSI(sc.Close, RSI, RsiUpSums, RsiDownSums, RsiSmoothedUpSums, RsiSmoothedDownSums,
RSIAvgType.GetMovAvgType(), RSILength.GetInt());

// computing Stochastic of RSI
float maxRSI = sc.GetHighest(RSI, sc.Index, StochLength.GetInt());
float minRSI = sc.GetLowest(RSI, sc.Index, StochLength.GetInt());

if (maxRSI-minRSI != 0.0f)
{
    StochRSI[sc.Index] = 100.0f * (RSI[sc.Index]-minRSI)/(maxRSI - minRSI);
}
else
{
    StochRSI[sc.Index] = StochRSI[sc.Index-1];
}

sc.MovingAverage(StochRSI, SK, SKAvgType.GetMovAvgType(), SKLength.GetInt());
sc.MovingAverage(SK, SD, SDAvgType.GetMovAvgType(), SDLength.GetInt());
}

/*=====*/
SCSFExport scsf_VericalTimeLine(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_TimeLine = sc.Subgraph[0];
    SCSubgraphRef Subgraph_TimeLineBottom = sc.Subgraph[1];
    SCSubgraphRef Subgraph_CustomLabel = sc.Subgraph[2];

    SCInputRef Input_TimeOfLine = sc.Input[0];
    SCInputRef Input_DisplayCustomLabel = sc.Input[1];
    SCInputRef Input_VericalPositionPercent = sc.Input[2];
    SCInputRef Input_Version = sc.Input[3];
    SCInputRef Input_ForwardProject = sc.Input[4];
    SCInputRef Input_AllowedRangeForMatchInMinutes = sc.Input[5];
    SCInputRef Input_MatchingMethod = sc.Input[6];
    SCInputRef Input_DayOfWeek = sc.Input[7];

    if(sc.SetDefaults)
    {
        sc.GraphName="Time Line";
        sc.StudyDescription="Draws a vertical line at the time specified with the Time of Line Input.";

        sc.AutoLoop = 0;//Manual looping
        sc.GraphRegion = 0;
        sc.ScaleRangeType = SCALE_USERDEFINED;
        sc.ScaleRangeTop = 2;
        sc.ScaleRangeBottom = 1;
    }
}

```



```

Subgraph_TimeLine.Name = "TimeLine";
Subgraph_TimeLine.DrawStyle = DRAWSTYLE_BAR_TOP;
Subgraph_TimeLine.PrimaryColor = RGB(127,127,127);
Subgraph_TimeLine.DrawZeros = false;
Subgraph_TimeLine.LineWidth = 1;

Subgraph_TimeLineBottom.Name = "TimeLine";
Subgraph_TimeLineBottom.DrawStyle = DRAWSTYLE_BAR_BOTTOM;
Subgraph_TimeLineBottom.PrimaryColor = RGB(127,127,127);
Subgraph_TimeLineBottom.DrawZeros = false;
Subgraph_TimeLineBottom.LineWidth = 1;

Subgraph_CustomLabel.Name = "Custom Label Color and Text Size";
Subgraph_CustomLabel.DrawStyle=DRAWSTYLE_CUSTOM_TEXT;
Subgraph_CustomLabel.PrimaryColor = RGB(127,127,127);
Subgraph_CustomLabel.LineWidth = 0;

Input_TimeOfLine.Name = "Time of Line";
Input_TimeOfLine.SetTime(HMS_TIME(12, 00, 0));

Input_DisplayCustomLabel.Name = "Display Custom Label";
Input_DisplayCustomLabel.SetYesNo(false);

Input_VerticalPositionPercent.Name = "Label Vertical Position %.";
Input_VerticalPositionPercent.SetInt(5);
Input_VerticalPositionPercent.SetIntLimits(0,100);

sc.TextInputName = "Custom Label Text";

Input_ForwardProject.Name = "Display in Forward Projection Area";
Input_ForwardProject.SetYesNo(false);

Input_AllowedRangeForMatchInMinutes.Name = "Allowed Range for Match in Minutes";
Input_AllowedRangeForMatchInMinutes.SetInt(10);

Input_MatchingMethod.Name = "Matching Method";
Input_MatchingMethod.SetCustomInputStrings("Nearest;Containing");
Input_MatchingMethod.SetCustomInputIndex(0);

Input_DayOfWeek.Name = "Allowed Day of Week";
Input_DayOfWeek.SetCustomInputStrings("All
Days;Monday;Tuesday;Wednesday;Thursday;Friday;Saturday;Sunday");
Input_DayOfWeek.SetCustomInputIndex(0);

Input_Version.SetInt(3);

return;
}

if (Input_Version.GetInt() < 2)
{
    Subgraph_TimeLine.DrawStyle = DRAWSTYLE_BAR_TOP;
    Subgraph_TimeLineBottom.DrawStyle = DRAWSTYLE_BAR_BOTTOM;
    Input_Version.SetInt(3);
}

if (Input_Version.GetInt() < 3)
{
    Input_AllowedRangeForMatchInMinutes.SetInt(10);
    Input_Version.SetInt(3);
}

int NumberForwardBars = 0;
if(Input_ForwardProject.GetYesNo())
{

```

```

//Subtracting 1 since the study references one bar ahead
NumberForwardBars = sc.NumberOfForwardColumns-1;
Subgraph_TimeLine.ExtendedArrayElementsToGraph = NumberForwardBars;
Subgraph_TimeLineBottom.ExtendedArrayElementsToGraph = NumberForwardBars;
}

const int AllowedDayOfWeek = Input_DayOfWeek.GetIndex() - 1;

int NearestDateTimeIndex = -1;

for (int BarIndex = sc.UpdateStartIndex; BarIndex < (sc.ArraySize + NumberForwardBars); BarIndex++)
{
    if (AllowedDayOfWeek >= 0)
    {
        if (sc.BaseDateTimeIn[BarIndex].GetDayOfWeekMondayBased() != AllowedDayOfWeek)
            continue;
    }

    const SCDatetimeMS TimeLineDateTime(sc.BaseDateTimeIn[BarIndex].GetDate(), Input_TimeOfLine.GetTime());

    if(sc.BaseDateTimeIn[BarIndex - 1].GetDate() != sc.BaseDateTimeIn[BarIndex].GetDate())
        NearestDateTimeIndex = -1;

    if(NearestDateTimeIndex == -1)
    {
        if (Input_MatchingMethod.GetIndex() == 0)
        {
            NearestDateTimeIndex
                = sc.GetNearestMatchForSCDateTimeExtended(sc.ChartNumber, TimeLineDateTime);
        }
        else
        {
            if (TimeLineDateTime > sc.BaseDateTimeIn[BarIndex - 1]
                && TimeLineDateTime <= sc.BaseDateTimeIn[BarIndex])
            {
                NearestDateTimeIndex = BarIndex;
            }
        }
    }
}

SCDatetimeMS TimeMargin;
TimeMargin.SetTimeHMS(0, Input_AllowedRangeForMatchInMinutes.GetInt(), 0);
SCDatetimeMS EarlierTime = TimeLineDateTime - TimeMargin;
SCDatetimeMS LaterTime = TimeLineDateTime + TimeMargin;

if (BarIndex == NearestDateTimeIndex
    && sc.BaseDateTimeIn[BarIndex] >= EarlierTime
    && sc.BaseDateTimeIn[BarIndex] <= LaterTime)
{
    Subgraph_TimeLine[BarIndex] = 100;
    Subgraph_TimeLineBottom[BarIndex] = -100;
}
else
{
    Subgraph_TimeLine[BarIndex] = 0;
    Subgraph_TimeLineBottom[BarIndex] = 0;
}

if (Subgraph_TimeLine[BarIndex] == 0)
    continue;

if(Input_DisplayCustomLabel.GetYesNo()
    && !sc.HideStudy
    && BarIndex < sc.ArraySize)

```

```

{
    int &r_LastUsedBarIndex = sc.GetPersistentInt(1);
    int &r_RememberedLineNumber = sc.GetPersistentInt(2);

    if(sc.IsFullRecalculation)
    {
        r_LastUsedBarIndex = -1;
        r_RememberedLineNumber = 0;
    }

    if(BarIndex == r_LastUsedBarIndex)
        continue;

    //If there still is a timeline at r_LastUsedBarIndex, then we need to not reuse the prior line number because this is
    //a new one. If got to this point in the code, then we are at an index after r_LastUsedBarIndex.
    if (r_LastUsedBarIndex != -1 && Subgraph_TimeLine[r_LastUsedBarIndex] != 0)
        r_RememberedLineNumber = 0;

    r_LastUsedBarIndex = BarIndex;

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

    Tool.BeginIndex = BarIndex;
    Tool.UseRelativeVerticalValues = true;
    Tool.BeginValue = static_cast<float>(Input_VerticalPositionPercent.GetInt());
    Tool.Text = sc.TextInput;

    Tool.Color = Subgraph_CustomLabel.PrimaryColor;
    Tool.FontBold = true;
    Tool.FontSize = Subgraph_CustomLabel.LineWidth;
    if (r_RememberedLineNumber != 0)
    {
        Tool.LineNumber = r_RememberedLineNumber;
    }

    Tool.AddMethod = UTAM_ADD_OR_ADJUST;

    sc.UseTool(Tool);
    r_RememberedLineNumber = Tool.LineNumber;
}
}

return;
}

/*=====*/
SCSFExport scsf_VerticalDateTimeLine(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_TimeLine = sc.Subgraph[0];
    SCSubgraphRef Subgraph_TimeLineBottom = sc.Subgraph[1];
    SCSubgraphRef Subgraph_CustomLabel = sc.Subgraph[2];

    SCInputRef Input_DateTimeOfLine = sc.Input[0];
    SCInputRef Input_DisplayCustomLabel = sc.Input[1];
    SCInputRef Input_VerticalPositionPercent = sc.Input[2];
    SCInputRef Input_Version = sc.Input[3];
    SCInputRef Input_ForwardProject = sc.Input[4];
    SCInputRef Input_AllowedRangeForMatchInMinutes = sc.Input[5];
    SCInputRef Input_MatchingMethod = sc.Input[6];

```

```

if(sc.SetDefaults)
{
    sc.GraphName="Date-Time Line";
    sc.StudyDescription="Draws a vertical line at the date-time specified with the Date-Time of Line input.";

    sc.AutoLoop = 0;//Manual looping
    sc.GraphRegion = 0;
    sc.ScaleRangeType = SCALE_USERDEFINED;
    sc.ScaleRangeTop = 2;
    sc.ScaleRangeBottom = 1;

    Subgraph_TimeLine.Name = "TimeLine";
    Subgraph_TimeLine.DrawStyle = DRAWSTYLE_BAR_TOP;
    Subgraph_TimeLine.PrimaryColor = RGB(127,127,127);
    Subgraph_TimeLine.DrawZeros = false;
    Subgraph_TimeLine.LineWidth = 1;

    Subgraph_TimeLineBottom.Name = "TimeLine";
    Subgraph_TimeLineBottom.DrawStyle = DRAWSTYLE_BAR_BOTTOM;
    Subgraph_TimeLineBottom.PrimaryColor = RGB(127,127,127);
    Subgraph_TimeLineBottom.DrawZeros = false;
    Subgraph_TimeLineBottom.LineWidth = 1;

    Subgraph_CustomLabel.Name = "Custom Label Color and Text Size";
    Subgraph_CustomLabel.DrawStyle=DRAWSTYLE_CUSTOM_TEXT;
    Subgraph_CustomLabel.PrimaryColor = RGB(127,127,127);
    Subgraph_CustomLabel.LineWidth = 0;

    Input_DateTimeOfLine.Name = "Date-Time of Line";
    Input_DateTimeOfLine.SetDateTime(sc.CurrentSystemDateTime);

    Input_DisplayCustomLabel.Name = "Display Custom Label";
    Input_DisplayCustomLabel.SetYesNo(false);

    Input_VerticalPositionPercent.Name = "Label Vertical Position %.";
    Input_VerticalPositionPercent.SetInt(5);
    Input_VerticalPositionPercent.SetIntLimits(0,100);

    sc.TextInputName = "Custom Label Text";

    Input_ForwardProject.Name = "Display in Forward Projection Area";
    Input_ForwardProject.SetYesNo(false);

    Input_AllowedRangeForMatchInMinutes.Name = "Allowed Range for Match in Minutes";
    Input_AllowedRangeForMatchInMinutes.SetInt(10);

    Input_MatchingMethod.Name = "Matching Method";
    Input_MatchingMethod.SetCustomInputStrings("Nearest;Containing");
    Input_MatchingMethod.SetCustomInputIndex(0);

    Input_Version.SetInt(1);

    return;
}

int NumberForwardBars = 0;
if(Input_ForwardProject.GetYesNo())
{
    // Subtracting 1 since the study references one bar ahead
    NumberForwardBars = sc.NumberOfForwardColumns -1;

    Subgraph_TimeLine.ExtendedArrayElementsToGraph = NumberForwardBars;
}

```

```

    Subgraph_TimeLineBottom.ExtendedArrayElementsToGraph = NumberForwardBars;
}

int& ChartDrawingLastUsedIndex = sc.GetPersistentInt(1);

if(sc.UpdateStartIndex == 0)
    ChartDrawingLastUsedIndex = -1;

int NearestDateTimeIndex = -1;

for (int Index = sc.UpdateStartIndex; Index < (sc.ArraySize + NumberForwardBars); Index++)
{
    SCDateTime TimeLineDateTime(Input_DateTimeOfLine.GetDateTime());

    if(NearestDateTimeIndex == -1)
    {
        if (Input_MatchingMethod.GetIndex() == 0)
            NearestDateTimeIndex = sc.GetNearestMatchForSCDateTimeExtended(sc.ChartNumber,
TimeLineDateTime);
        else
        {
            if (TimeLineDateTime > sc.BaseDateTimeIn[Index - 1]
                && TimeLineDateTime <= sc.BaseDateTimeIn[Index])
                NearestDateTimeIndex = Index;
        }
    }

    SCDateTime TimeMargin;
    TimeMargin.SetTimeHMS(0, Input_AllowedRangeForMatchInMinutes.GetInt(), 0);
    SCDateTime EarlierTime = TimeLineDateTime - TimeMargin;
    SCDateTime LaterTime = TimeLineDateTime + TimeMargin;

    if (Index == NearestDateTimeIndex
        && sc.BaseDateTimeIn[Index] >= EarlierTime
        && sc.BaseDateTimeIn[Index] <= LaterTime)
    {
        Subgraph_TimeLine[Index] = 100;
        Subgraph_TimeLineBottom[Index] = -100;
    }
    else
    {
        Subgraph_TimeLine[Index] = 0;
        Subgraph_TimeLineBottom[Index] = 0;
    }

    if (Subgraph_TimeLine[Index] == 0)
        continue;

    if(Input_DisplayCustomLabel.GetYesNo() && !sc.HideStudy )
    {
        if(Index == ChartDrawingLastUsedIndex)
            continue;

        ChartDrawingLastUsedIndex = Index;

        s_UseTool Tool;
        Tool.ChartNumber = sc.ChartNumber;
        Tool.DrawingType = DRAWING_TEXT;
        Tool.Region = sc.GraphRegion;
        //Tool.LineNumber //Automatically set
        Tool.AddMethod = UTAM_ADD_OR_ADJUST;
    }
}

```

```

Tool.BeginIndex = Index;
Tool.UseRelativeVerticalValues = true;
Tool.BeginValue = static_cast<float>(Input_VerticalPositionPercent.GetInt());
Tool.Text = sc.TextInput;

Tool.Color = Subgraph_CustomLabel.PrimaryColor;
Tool.FontBold = true;
Tool.FontSize = Subgraph_CustomLabel.LineWidth;
sc.UseTool(Tool);

}
}

return;
}

/*=====*/
SCSFExport scsf_RSITradestation(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_RSIT = sc.Subgraph[0];
    SCSubgraphRef Subgraph_LineUpperLimit = sc.Subgraph[1];
    SCSubgraphRef Subgraph_LineLowerLimit = sc.Subgraph[2];
    SCSubgraphRef Subgraph_RSIAvg = sc.Subgraph[3];
    SCSubgraphRef Subgraph_NetChangeAvg = sc.Subgraph[4];
    SCSubgraphRef Subgraph_TotalChangeAvg = sc.Subgraph[5];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_RSILength = sc.Input[1];
    SCInputRef Input_Line1 = sc.Input[3];
    SCInputRef Input_Line2 = sc.Input[4];
    SCInputRef Input_RSIMovAvgLength = sc.Input[5];

    if(sc.SetDefaults)
    {
        sc.GraphName="RSI-TS";
        sc.StudyDescription="RSI implemented as in TradeStation";

        // Outputs
        Subgraph_RSIT.Name="RSI-T";
        Subgraph_RSIT.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_RSIT.PrimaryColor = RGB(0,255,0);
        Subgraph_RSIT.DrawZeros = false;

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

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

        Subgraph_RSIAvg.Name = "RSI Avg";
        Subgraph_RSIAvg.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_RSIAvg.PrimaryColor = RGB(255,127,0);
        Subgraph_RSIAvg.DrawZeros = false;

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

        Input_RSILength.Name="RSI Length";
        Input_RSILength.SetInt(10);
    }
}

```

```

Input_Line1.Name="Line 1";
Input_Line1.SetFloat(70);
Input_Line1.SetFloatLimits(50, 100);

Input_Line2.Name="Line 2";
Input_Line2.SetFloat(30);
Input_Line2.SetFloatLimits(0, 50);

Input_RSIMovAvgLength.Name = "RSI Mov Avg Length";
Input_RSIMovAvgLength.SetInt(3);
Input_RSIMovAvgLength.SetIntLimits(1, MAX_STUDY_LENGTH);

sc.AutoLoop = 1;

return;
}

// start charting after Length of elements is available
sc.DataStartIndex = Input_RSILength.GetInt();
if (sc.Index<Input_RSILength.GetInt())
    return;

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

// smoothing factor
float smoothingFactor = 1.0f / Input_RSILength.GetInt();

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

// compute net and total changes

float totalChange;
float netChange;

if (sc.Index==Input_RSILength.GetInt())
{
    // NetChgAvg = (Price - Price[Length] ) / Length
    netChange = (price[sc.Index] - price[sc.Index - Input_RSILength.GetInt()]) / Input_RSILength.GetInt();

    // TotChgAvg = Average( AbsValue( Price - Price[1] ), Length )
    float average = 0.0;
    for (int i=1; i<=sc.Index; i++)
    {
        average += abs(price[i] - price[i - 1]);
    }
    totalChange = average / Input_RSILength.GetInt();
}
else
{
    // Change = Price - Price[1]
    float change = price[sc.Index] - price[sc.Index - 1];

    // NetChgAvg = NetChgAvg[1] + SF * ( Change - NetChgAvg[1] )
    float previousNetChange = Subgraph_NetChangeAvg[sc.Index - 1];
    netChange = previousNetChange + smoothingFactor * ( change - previousNetChange );

    // TotChgAvg = TotChgAvg[1] + SF * ( AbsValue(Change) - TotChgAvg[1] )
    float previousTotalChangeAverage = Subgraph_TotalChangeAvg[sc.Index - 1];
    totalChange = previousTotalChangeAverage + smoothingFactor * ( abs(change) - previousTotalChangeAverage );
}

Subgraph_TotalChangeAvg[sc.Index] = totalChange;
Subgraph_NetChangeAvg[sc.Index] = netChange;

```

```

// compute the RSI value
float changeRatio = 0;
if (totalChange!=0.0f)
{
    changeRatio = netChange / totalChange;
}
else
{
    changeRatio = 0.0f;
}

// RSI = 50 * ( ChgRatio + 1)
Subgraph_RSIT[sc.Index] = 50 * (changeRatio + 1.0f);

// Draw overbought / oversold indicators
Subgraph_LineUpperLimit[sc.Index] = Input_Line1.GetFloat();
Subgraph_LineLowerLimit[sc.Index] = Input_Line2.GetFloat();

if (sc.Index >= Input_RSIMovAvgLength.GetInt() + Input_RSILength.GetInt())
{
    // Draw smoothed RSI
    sc.MovingAverage(Subgraph_RSIT, Subgraph_RSIAvg, MOVAVGTYPE_SIMPLE, sc.Index,
Input_RSIMovAvgLength.GetInt());
}
}

/*=====*/
/*
Triple Exponential Moving Average, by Patrick Mulloy and published in January 1994
in the Stocks & Commodities magazine

Algorithm is also described in http://www.visualchart.com/enxx/strategies/indicators/ayuda.asp?Id=2857
*/
SCSFExport scsf_MovingAverageTEMA(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_TEMA = sc.Subgraph[0];

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

    if(sc.SetDefaults)
    {
        sc.GraphName="Moving Average - Triple Exp";
        sc.StudyDescription="Triple Exponential Moving Average";

        sc.GraphRegion = 0;

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

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

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

        sc.AutoLoop = 1;

        return;
    }
}

```



```

sc.DataStartIndex = Input_Length.GetInt();

SCFloatArrayRef data = sc.BaseData[Input_InputData.GetInputDataIndex()];
sc.TEMA(data,Subgraph_TEMA,sc.Index,Input_Length.GetInt());
}

/*=====*/

SCSFExport scsf_HistoricVolatilityStudy(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Volatility = sc.Subgraph[0];
    SCSubgraphRef Subgraph_LogPriceChange = sc.Subgraph[1];
    SCSubgraphRef Subgraph_SMALog = sc.Subgraph[2];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_AlgorithmType = sc.Input[1];
    SCInputRef Input_Length = sc.Input[2];
    SCInputRef Input_NumberBarsPerYear = sc.Input[3];

    if(sc.SetDefaults)
    {
        sc.GraphName="Volatility - Historical";

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

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

        Input_Length.Name = "Length";
        Input_Length.SetIntLimits(2,MAX_STUDY_LENGTH);
        Input_Length.SetInt(20);

        Input_NumberBarsPerYear.Name = "Number of Bars Per Year";
        Input_NumberBarsPerYear.SetInt(262);

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

        return;
    }

    sc.DataStartIndex = Input_Length.GetInt();

    // calculate the logarithm of the price change
    Subgraph_LogPriceChange[sc.Index] = log(sc.BaseData[Input_InputData.GetInputDataIndex()][sc.Index] /
    sc.BaseData[Input_InputData.GetInputDataIndex()][sc.Index - 1]);

    // calculate the average of the logs
    sc.SimpleMovAvg(Subgraph_LogPriceChange, Subgraph_SMALog, sc.Index, Input_Length.GetInt());

    // sum the squares of the difference between the individual logarithms for each period and the average logarithm
    float SumSquareOfDifference = 0;
    for (int SumIndex = max(0, sc.Index - Input_Length.GetInt() + 1); SumIndex <= sc.Index; SumIndex++)
    {
        float diff = Subgraph_LogPriceChange[SumIndex] - Subgraph_SMALog[sc.Index];
        SumSquareOfDifference += diff * diff;
    }

    // Final volatility calculation
    Subgraph_Volatility[sc.Index] = sqrt(SumSquareOfDifference / float(Input_Length.GetInt() - 1)) *

```

```
sqrt(float(Input_NumberBarsPerYear.GetInt())) * 100;
```

```
}
```

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

Synthetic VIX indicator as described in the December 2007 issue of "Active Trader" magazine. The indicator is also named the "Williams VIX Fix" since Larry Williams is credited with the formula's discovery.

Formula:

$$\text{SynthVIX} = (\text{Highest}(\text{Close}, 22) - \text{Low}) / (\text{Highest}(\text{Close}, 22)) * 100;$$

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

```
SCSFExport scsf_SyntheticVIX(SCStudyInterfaceRef sc)
```

```
{
```

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

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

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

```
    {
```

```
        sc.GraphName="Synthetic VIX";
```

```
        sc.StudyDescription="Synthetic VIX";
```

```
        Subgraph_SynthVIX.Name = "SynthVIX";
```

```
        Subgraph_SynthVIX.DrawStyle = DRAWSTYLE_LINE;
```

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

```
        Subgraph_SynthVIX.DrawZeros = false;
```

```
        Input_Length.Name = "Length";
```

```
        Input_Length.SetIntLimits(1, MAX_STUDY_LENGTH);
```

```
        Input_Length.SetInt(22);
```

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

```
        return;
```

```
    }
```

```
    float HighestClose = sc.GetHighest(sc.Close, sc.Index, Input_Length.GetInt());
```

```
    Subgraph_SynthVIX[sc.Index] = 100.0f * (HighestClose - sc.Low[sc.Index]) / HighestClose;
```

```
}
```

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

```
void PassingSCStructureExampleFunction(SCStudyInterfaceRef sc)
```

```
{
```

```
    //The "sc" structure can be used anywhere within this function.
```

```
}
```

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

```
"An example of calling a function that receives the Sierra Chart ACSIL structure (sc)."
```

```
-----*/
```

```
SCSFExport scsf_PassingSCStructureExample(SCStudyInterfaceRef sc)
```

```
{
```

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

```
    {
```

```
        // Set the configuration and defaults
```

```
        sc.GraphName = "Passing sc Structure Example Function";
```

```
        sc.StudyDescription = "An example of calling a function that receives the Sierra Chart ACSIL structure (sc).";
```

```

    return;
}

// Do data processing

// The definition of the function called below must be above this function.
PassingSCStrutureExampleFunction(sc);
}

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

    SCSubgraphRef Subgraph_Open = sc.Subgraph[SC_OPEN];
    SCSubgraphRef Subgraph_High = sc.Subgraph[SC_HIGH];
    SCSubgraphRef Subgraph_Low = sc.Subgraph[SC_LOW];
    SCSubgraphRef Subgraph_Last = sc.Subgraph[SC_LAST];
    SCSubgraphRef Subgraph_Volume = sc.Subgraph[SC_VOLUME];
    SCSubgraphRef Subgraph_OHLCAvg = sc.Subgraph[SC_OHLC_AVG];
    SCSubgraphRef Subgraph_HLCAvg = sc.Subgraph[SC_HLC_AVG];
    SCSubgraphRef Subgraph_HLAvG = sc.Subgraph[SC_HL_AVG];

    SCSubgraphRef Subgraph_NumTrades = sc.Subgraph[SC_NUM_TRADES];
    SCSubgraphRef Subgraph_BidVol = sc.Subgraph[SC_BIDVOL];
    SCSubgraphRef Subgraph_AskVol = sc.Subgraph[SC_ASKVOL];

    if(sc.SetDefaults)
    {
        sc.GraphName="Multiply Bars By -1";

        sc.ValueFormat = VALUEFORMAT_INHERITED;
        sc.AutoLoop = 1;
        sc.GraphDrawType = GDT_OHLCBAR;
        sc.StandardChartHeader = true;
        sc.GraphUsesChartColors = true;

        return;
    }

    if (sc.IsFullRecalculation && sc.Index == 0)
    {
        sc.GraphName.Format("%s Inverted", sc.GetStudyName(0).GetChars());

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

    Subgraph_Open[sc.Index] = sc.Open[sc.Index] * -1.0f;
    Subgraph_Last[sc.Index] = sc.Close[sc.Index] * -1.0f;
    Subgraph_High[sc.Index] = sc.High[sc.Index] * -1.0f;
    Subgraph_Low[sc.Index] = sc.Low[sc.Index] * -1.0f;
    Subgraph_OHLCAvg[sc.Index] = sc.OHLCAvg[sc.Index] * -1.0f;
    Subgraph_HLCAvg[sc.Index] = sc.HLCAvg[sc.Index] * -1.0f;
    Subgraph_HLAvG[sc.Index] = sc.HLAvG[sc.Index] * -1.0f;

```

```

Subgraph_Volume[sc.Index] = sc.Volume[sc.Index];
Subgraph_NumTrades[sc.Index] = sc.NumberOfTrades[sc.Index];
Subgraph_BidVol[sc.Index] = sc.BidVolume[sc.Index];
Subgraph_AskVol[sc.Index] = sc.AskVolume[sc.Index];
}

/*=====*/
SCSFExport scsf_BarsInTicks(SCStudyInterfaceRef sc)
{
    if (sc.SetDefaults)
    {
        sc.GraphName = "Bars in Ticks";
        //sc.GraphRegion = 0;
        sc.ValueFormat = VALUEFORMAT_WHOLE_NUMBER;
        sc.AutoLoop = 0;
        sc.GraphDrawType = GDT_OHLCBAR;
        sc.StandardChartHeader = true;
        sc.GraphUsesChartColors = true;

        return;
    }

    n_ACSIL::s_BarPeriod BarPeriod;
    sc.GetBarPeriodParameters(BarPeriod);
    int RenkoTicksPerBar = BarPeriod.IntradayChartBarPeriodParameter1;

    if (sc.IsFullRecalculation && sc.UpdateStartIndex == 0)
    {
        sc.GraphName.Format("%s in Ticks", sc.GetStudyName(0).GetChars());

        for (int SubgraphIndex = 0; SubgraphIndex <= NUM_BASE_GRAPH_ARRAYS; ++SubgraphIndex)
        {
            sc.Subgraph[SubgraphIndex].Name = sc.GetStudySubgraphName(0, SubgraphIndex);
        }
    }

    for(int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize; BarIndex++)
    {
        for (int SubGraph = 0; SubGraph <= NUM_BASE_GRAPH_ARRAYS; SubGraph++)
        {
            if (SubGraph == SC_VOLUME || SubGraph == SC_NUM_TRADES || SubGraph == SC_BIDVOL || SubGraph ==
SC_ASKVOL)
                sc.Subgraph[SubGraph][BarIndex] = sc.BaseDataIn[SubGraph][BarIndex];
            else
            {
                if (RenkoTicksPerBar != 0 && SubGraph == SC_OPEN)
                    sc.Subgraph[SubGraph][BarIndex] = sc.BaseDataIn[SC_RENKO_OPEN][BarIndex] / sc.TickSize;
                else if (RenkoTicksPerBar != 0 && SubGraph == SC_LAST)
                    sc.Subgraph[SubGraph][BarIndex] = sc.BaseDataIn[SC_RENKO_CLOSE][BarIndex] / sc.TickSize;
                else
                    sc.Subgraph[SubGraph][BarIndex] = sc.BaseDataIn[SubGraph][BarIndex] / sc.TickSize;
            }
        }
    }
}

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

```

```

SCInputRef Input_InputData = sc.Input[0];

if (sc.SetDefaults)
{
    sc.GraphName = "Study Subgraph in Ticks";
    sc.GraphRegion = 1;
    sc.ValueFormat = VALUEFORMAT_WHOLE_NUMBER;
    sc.GraphDrawType = GDT_CUSTOM;

    sc.AutoLoop = 0;

    Subgraph_Result.Name = "Ticks";
    Subgraph_Result.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_Result.PrimaryColor = RGB( 255, 165, 0);

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

    return;
}

int CalculationStartIndex = sc.GetCalculationStartIndexForStudy();

for (int Index = CalculationStartIndex; Index < sc.ArraySize; Index++)
{
    Subgraph_Result[Index] = static_cast<float>(sc.Round(sc.BaseData[Input_InputData.GetInputDataIndex()][Index] /
sc.TickSize));
}

sc.EarliestUpdateSubgraphDataArrayIndex = CalculationStartIndex;

}

/*=====*/

SCSFExport scsf_StudySubgraphAsCurrencyValue(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Result = sc.Subgraph[0];

    SCInputRef Input_InputData = sc.Input[0];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Study Subgraph as Currency Value";
        sc.GraphRegion = 1;
        sc.ValueFormat = VALUEFORMAT_2_DIGITS;
        sc.GraphDrawType = GDT_CUSTOM;

        sc.AutoLoop = 0;

        Subgraph_Result.Name = "Currency Value";
        Subgraph_Result.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Result.PrimaryColor = RGB(255, 165, 0);

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

        return;
    }
}

```

```

int CalculationStartIndex = sc.GetCalculationStartIndexForStudy();

for (int Index = CalculationStartIndex; Index < sc.ArraySize; Index++)
{
    Subgraph_Result[Index] = static_cast<float>(sc.Round(sc.BaseData[Input_InputData.GetInputDataIndex()][Index] /
sc.TickSize)* sc.CurrencyValuePerTick);
}

sc.EarliestUpdateSubgraphDataArrayIndex = CalculationStartIndex;

}

/*=====*/
SCSFExport scsf_ChandeMomentumOscillator(SCStudyInterfaceRef sc)
{
    enum InputIndexes
    {
        INPUT_DATA= 0,
        INPUT_CMO_LENGTH,
        INPUT_LINE_LEVEL2,
        INPUT_LINE_LEVEL1,
        INPUT_LINE_LEVEL3
    };

    SCSubgraphRef Subgraph_CMO = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Line2 = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Line1 = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Line3 = sc.Subgraph[3];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_CMOLength = sc.Input[1];
    SCInputRef Input_Line1Level = sc.Input[2];
    SCInputRef Input_Line2Level = sc.Input[3];
    SCInputRef Input_Line3Level = sc.Input[4];

    if(sc.SetDefaults)
    {
        sc.GraphName="Chande Momentum Oscillator";
        sc.StudyDescription="Chande Momentum Oscillator";

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

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

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

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

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

```

```

Input_CMOLength.Name="CMO Length";
Input_CMOLength.SetInt(14);

Input_Line1Level.Name="Line 1 level";
Input_Line1Level.SetFloat(50.0f);

Input_Line2Level.Name="Line 2 level";
Input_Line2Level.SetFloat(0.0f);

Input_Line3Level.Name="Line 3 level";
Input_Line3Level.SetFloat(-50.0f);

sc.AutoLoop = 1;

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

if (sc.Index < Input_CMOLength.GetInt()-1)
return;

float sum_up = 0;
float sum_down = 0;
for (int j = (sc.Index-Input_CMOLength.GetInt()+1);j<=sc.Index;j++)
{
if (sc.BaseData[Input_InputData.GetInputDataIndex()][j]<sc.BaseData[Input_InputData.GetInputDataIndex()][j-1])
{
sum_down += sc.BaseData[Input_InputData.GetInputDataIndex()][j-1] -
sc.BaseData[Input_InputData.GetInputDataIndex()][j];
}
else if (sc.BaseData[Input_InputData.GetInputDataIndex()][j]>sc.BaseData[Input_InputData.GetInputDataIndex()][j-1])
{
sum_up += sc.BaseData[Input_InputData.GetInputDataIndex()][j] -
sc.BaseData[Input_InputData.GetInputDataIndex()][j-1];
}
}
//CMO(t) = 100 * ((SumUp(t) - SumDown(t)) / (SumUp(t) + SumDown(t)))
float fCMO;

if(sum_up+sum_down == 0)
fCMO = Subgraph_CMO[sc.Index-1];
else
fCMO = 100.0f*((sum_up-sum_down)/(sum_up+sum_down));

Subgraph_CMO[sc.Index] = fCMO;
Subgraph_Line2[sc.Index] = Input_Line2Level.GetFloat();
Subgraph_Line1[sc.Index] = Input_Line1Level.GetFloat();
Subgraph_Line3[sc.Index] = Input_Line3Level.GetFloat();
}

/*=====*/
SCSFExport scsf_ZeroLagEMA(SCStudyInterfaceRef sc)
{
SCSubgraphRef Subgraph_ZeroLagEMA = sc.Subgraph[0];
SCFloatArrayRef Array_DataLagDifference = sc.Subgraph[0].Arrays[0];

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

if(sc.SetDefaults)
{
sc.GraphName="Moving Average - Zero Lag Exponential";

```

```

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

Subgraph_ZeroLagEMA.Name="Zero Lag EMA";
Subgraph_ZeroLagEMA.DrawStyle = DRAWSTYLE_LINE;
Subgraph_ZeroLagEMA.PrimaryColor = RGB(0,255,0);
Subgraph_ZeroLagEMA.DrawZeros = false;

Input_Period.Name="Zero Lag EMA Length";
Input_Period.SetInt(10);
Input_Period.SetIntLimits(1,MAX_STUDY_LENGTH);

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

return;
}

sc.ZeroLagEMA(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_ZeroLagEMA, sc.Index,
Input_Period.GetInt());
}
/*=====*/
SCSFExport scsf_Tenkan_Sen(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_TenkanSen = sc.Subgraph[0];
    SCFloatArrayRef Array_HH = sc.Subgraph[0].Arrays[0];
    SCFloatArrayRef Array_LL = sc.Subgraph[0].Arrays[1];

    SCInputRef Input_Period = sc.Input[0];
    SCInputRef Input_InputDataHigh = sc.Input[1];
    SCInputRef Input_InputDataLow = sc.Input[2];
    SCInputRef Input_Version = sc.Input[3];

    if(sc.SetDefaults)
    {
        sc.GraphName="Tenkan-Sen";

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

        Subgraph_TenkanSen.Name="Tenkan-Sen";
        Subgraph_TenkanSen.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_TenkanSen.LineWidth = 2;
        Subgraph_TenkanSen.PrimaryColor = RGB(255,128,0);
        Subgraph_TenkanSen.DrawZeros = false;

        Input_Period.Name="Tenkan-Sen Length";
        Input_Period.SetInt(9);
        Input_Period.SetIntLimits(1,MAX_STUDY_LENGTH);

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

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

        Input_Version.SetInt(1);

        return;
    }

    if (Input_Version.GetInt()<1)
    {
        Input_InputDataHigh.SetInputDataIndex(SC_HIGH);
        Input_InputDataLow.SetInputDataIndex(SC_LOW);
    }

```



```

    Input_Version. SetInt(1);
}

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

sc.Highest(sc.BaseData[Input_InputDataHigh.GetInputDataIndex()], Array_HH, Input_Period.GetInt());
sc.Lowest(sc.BaseData[Input_InputDataLow.GetInputDataIndex()], Array_LL, Input_Period.GetInt());
Subgraph_TenkanSen[sc.Index] = (Array_HH[sc.Index]+Array_LL[sc.Index])/2.0f;

}
/*=====*/
SCSFExport scsf_Kijun_Sen(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_KijunSen = sc.Subgraph[0];
    SCFloatArrayRef Array_HH = sc.Subgraph[0].Arrays[0];
    SCFloatArrayRef Array_LL = sc.Subgraph[0].Arrays[1];

    SCInputRef Input_Period = sc.Input[0];
    SCInputRef Input_InputDataHigh = sc.Input[1];
    SCInputRef Input_InputDataLow = sc.Input[2];
    SCInputRef Input_Version = sc.Input[3];

    if(sc.SetDefaults)
    {
        sc.GraphName="Kijun-Sen";

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

        Subgraph_KijunSen.Name="Kijun-Sen";
        Subgraph_KijunSen.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_KijunSen.LineWidth = 2;
        Subgraph_KijunSen.PrimaryColor = RGB(255,0,128);
        Subgraph_KijunSen.DrawZeros = false;

        Input_Period.Name="Kijun-Sen Length";
        Input_Period.SetInt(26);
        Input_Period.SetIntLimits(1,MAX_STUDY_LENGTH);

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

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

        Input_Version.SetInt(1);

        return;
    }

    if (Input_Version.GetInt()<1)
    {
        Input_InputDataHigh.SetInputDataIndex(SC_HIGH);
        Input_InputDataLow.SetInputDataIndex(SC_LOW);

        Input_Version. SetInt(1);
    }

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

    sc.Highest(sc.BaseData[Input_InputDataHigh.GetInputDataIndex()],Array_HH, Input_Period.GetInt());
    sc.Lowest(sc.BaseData[Input_InputDataLow.GetInputDataIndex()],Array_LL, Input_Period.GetInt());
    Subgraph_KijunSen[sc.Index] = (Array_HH[sc.Index] + Array_LL[sc.Index])/2.0f;

```

```

}
/*=====*/
SCSFExport scsf_ChikouSpan(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_ChikouSpan = sc.Subgraph[0];

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

    if(sc.SetDefaults)
    {
        sc.GraphName="Chikou Span";

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

        Subgraph_ChikouSpan.Name="Chikou Span";
        Subgraph_ChikouSpan.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_ChikouSpan.LineWidth = 2;
        Subgraph_ChikouSpan.PrimaryColor = RGB(20,180,240);
        Subgraph_ChikouSpan.DrawZeros = false;
        Subgraph_ChikouSpan.GraphicalDisplacement = 0;

        Input_Period.Name="Chikou Span Length";
        Input_Period.SetInt(26);
        Input_Period.SetIntLimits(1,MAX_STUDY_LENGTH);

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

        return;
    }

    //ChikouSpan.GraphicalDisplacement = Period.GetInt()*(-1);

    int OutputIndex = sc.Index - Input_Period.GetInt();

    //Chikou Span (Lagging Span): Close plotted 26 days in the past
    Subgraph_ChikouSpan[OutputIndex] = sc.BaseData[Input_InputData.GetInputDataIndex()][sc.Index];
}

/*=====*/
SCSFExport scsf_SenkouSpanA(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_SpanA = sc.Subgraph[0];
    SCSubgraphRef Subgraph_TenkanHH = sc.Subgraph[1];
    SCSubgraphRef Subgraph_TenkanLL = sc.Subgraph[2];
    SCSubgraphRef Subgraph_KijunHH = sc.Subgraph[3];
    SCSubgraphRef Subgraph_KijunLL = sc.Subgraph[4];

    SCInputRef Input_TenkanPeriod = sc.Input[1];
    SCInputRef Input_KijunPeriod = sc.Input[2];

    if(sc.SetDefaults)
    {
        sc.GraphName="Senkou Span A";
        sc.AutoLoop = 1;
        sc.GraphRegion = 0;

        Subgraph_SpanA.GraphicalDisplacement = 26;
        Subgraph_SpanA.Name="Span A";
        Subgraph_SpanA.DrawStyle = DRAWSTYLE_LINE;
    }
}

```

```

Subgraph_SpanA.LineWidth = 4;
Subgraph_SpanA.PrimaryColor = RGB(0,85,0);
Subgraph_SpanA.DrawZeros = false;

Input_TenkanPeriod.Name="Tenkan-Sen Length";
Input_TenkanPeriod.SetInt(9);
Input_TenkanPeriod.SetIntLimits(1,MAX_STUDY_LENGTH);

Input_KijunPeriod.Name="Kijun-Sen Length";
Input_KijunPeriod.SetInt(26);
Input_KijunPeriod.SetIntLimits(1,MAX_STUDY_LENGTH);

return;
}

sc.Highest(sc.BaseData[SC_HIGH],Subgraph_TenkanHH, Input_TenkanPeriod.GetInt());
sc.Lowest(sc.BaseData[SC_LOW],Subgraph_TenkanLL, Input_TenkanPeriod.GetInt());
sc.Highest(sc.BaseData[SC_HIGH],Subgraph_KijunHH, Input_KijunPeriod.GetInt());
sc.Lowest(sc.BaseData[SC_LOW],Subgraph_KijunLL, Input_KijunPeriod.GetInt());

Subgraph_SpanA[sc.Index] = (
(
(Subgraph_TenkanHH[sc.Index]+Subgraph_TenkanLL[sc.Index])/ 2.0f)+
(Subgraph_KijunHH[sc.Index]+Subgraph_KijunLL[sc.Index])/ 2.0f)
)/2.0f);
}

/*=====*/
SCSFExport scsf_SenkouSpan(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_SpanA = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SpanB = sc.Subgraph[1];
    SCSubgraphRef Subgraph_SpanBHH = sc.Subgraph[2];
    SCSubgraphRef Subgraph_SpanBLL = sc.Subgraph[3];
    SCSubgraphRef Subgraph_TenkanHH = sc.Subgraph[4];
    SCSubgraphRef Subgraph_TenkanLL = sc.Subgraph[5];
    SCSubgraphRef Subgraph_KijunHH = sc.Subgraph[6];
    SCSubgraphRef Subgraph_KijunLL = sc.Subgraph[7];

    SCInputRef Input_InputDataHigh= sc.Input[0];
    SCInputRef Input_SpanBLength = sc.Input[1];
    SCInputRef Input_TenkanLength = sc.Input[2];
    SCInputRef Input_KijunLength = sc.Input[3];
    SCInputRef Input_InputDataLow = sc.Input[4];
    SCInputRef Input_Version = sc.Input[5];

    if(sc.SetDefaults)
    {
        sc.GraphName = "Senkou Span A & B";

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

        Subgraph_SpanA.Name = "Span A";
        Subgraph_SpanA.DrawStyle = DRAWSTYLE_FILL_TOP;
        Subgraph_SpanA.PrimaryColor = RGB(133,205,24);
        Subgraph_SpanA.GraphicalDisplacement = 26;
        Subgraph_SpanA.DrawZeros = false;

        Subgraph_SpanB.Name = "Span B";
        Subgraph_SpanB.DrawStyle = DRAWSTYLE_FILL_BOTTOM;
        Subgraph_SpanB.PrimaryColor = RGB(133,205,24);
        Subgraph_SpanB.GraphicalDisplacement = 26;
    }
}

```

```

Subgraph_SpanB.DrawZeros = false;

int DisplayOrder = 1;
Input_InputDataHigh.Name = "Input Data High";
Input_InputDataHigh.SetInputDataIndex(SC_HIGH);
Input_InputDataHigh.DisplayOrder = DisplayOrder++;

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

Input_SpanBLength.Name="Senkou Span B Length";
Input_SpanBLength.SetInt(52);
Input_SpanBLength.SetIntLimits(1,MAX_STUDY_LENGTH);
Input_SpanBLength.DisplayOrder = DisplayOrder++;

Input_TenkanLength.Name="Tenkan-Sen Length";
Input_TenkanLength.SetInt(9);
Input_TenkanLength.SetIntLimits(1,MAX_STUDY_LENGTH);
Input_TenkanLength.DisplayOrder = DisplayOrder++;

Input_KijunLength.Name="Kijun-Sen Length";
Input_KijunLength.SetInt(26);
Input_KijunLength.SetIntLimits(1,MAX_STUDY_LENGTH);
Input_KijunLength.DisplayOrder = DisplayOrder++;

Input_Version.SetInt(1);

return;
}

if (Input_Version.GetInt() <1)
{
    Input_InputDataHigh.SetInputDataIndex(SC_HIGH);
    Input_InputDataLow.SetInputDataIndex(SC_LOW);
    Input_Version.SetInt(1);
}

// Senkou Span A

sc.Highest(sc.BaseData[Input_InputDataHigh.GetInputDataIndex()],Subgraph_TenkanHH,
Input_TenkanLength.GetInt());
sc.Lowest(sc.BaseData[Input_InputDataLow.GetInputDataIndex()],Subgraph_TenkanLL,
Input_TenkanLength.GetInt());

sc.Highest(sc.BaseData[Input_InputDataHigh.GetInputDataIndex()],Subgraph_KijunHH, Input_KijunLength.GetInt());
sc.Lowest(sc.BaseData[Input_InputDataLow.GetInputDataIndex()],Subgraph_KijunLL, Input_KijunLength.GetInt());

float Tenkan=(Subgraph_TenkanHH[sc.Index]+Subgraph_TenkanLL[sc.Index]) / 2.0f;
float Kijun=(Subgraph_KijunHH[sc.Index]+Subgraph_KijunLL[sc.Index]) / 2.0f;
Subgraph_SpanA[sc.Index] = (Tenkan +Kijun) / 2.0f;

// Senkou Span B

sc.Highest(sc.BaseData[Input_InputDataHigh.GetInputDataIndex()],Subgraph_SpanBHH,
Input_SpanBLength.GetInt());
sc.Lowest(sc.BaseData[Input_InputDataLow.GetInputDataIndex()],Subgraph_SpanBLL, Input_SpanBLength.GetInt());

Subgraph_SpanB[sc.Index] = (Subgraph_SpanBHH[sc.Index]+Subgraph_SpanBLL[sc.Index])/2.0f ;

}

/*=====*/

```

```

SCSFExport scsf_SenkouSpanB(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_SpanB = sc.Subgraph[0];
    SCSubgraphRef Subgraph_HH = sc.Subgraph[1];
    SCSubgraphRef Subgraph_LL = sc.Subgraph[2];

    SCInputRef Input_InputDataHigh = sc.Input[0];
    SCInputRef Input_StudyLength = sc.Input[1];
    SCInputRef Input_InputDataLow = sc.Input[2];
    SCInputRef Input_HiddenFlag = sc.Input[4];

    if(sc.SetDefaults)
    {
        sc.GraphName="Senkou Span B";

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

        Subgraph_SpanB.GraphicalDisplacement = 26;
        Subgraph_SpanB.Name="Span B";
        Subgraph_SpanB.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_SpanB.LineWidth = 4;
        Subgraph_SpanB.PrimaryColor = RGB(133,205,24);
        Subgraph_SpanB.DrawZeros = false;

        Input_StudyLength.Name="Senkou Span B Length";
        Input_StudyLength.SetInt(52);
        Input_StudyLength.SetIntLimits(1,MAX_STUDY_LENGTH);

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

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

        Input_HiddenFlag.SetInt(2);

        return;
    }

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

    sc.Highest(sc.BaseData[Input_InputDataHigh.GetInputDataIndex()],Subgraph_HH, Input_StudyLength.GetInt());
    sc.Lowest(sc.BaseData[Input_InputDataLow.GetInputDataIndex()],Subgraph_LL, Input_StudyLength.GetInt());

    Subgraph_SpanB[sc.Index] = (Subgraph_HH[sc.Index]+Subgraph_LL[sc.Index])/2.0f ;

}
/*=====*/
// This study function colors bars according to the slope of the Study Subgraph it is based on.

SCSFExport scsf_ColorBarBasedOnSlope(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_ColorBar = sc.Subgraph[0];

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

```

```

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

    sc.GraphName = "Color Bar Based On Slope";

    sc.StudyDescription = "Colors the main price graph bars based on the slope of a study Subgraph this study is Based On.";

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

    Subgraph_ColorBar.Name = "Color Bar";
    Subgraph_ColorBar.DrawStyle = DRAWSTYLE_COLOR_BAR;
    Subgraph_ColorBar.SecondaryColorUsed = 1; // true
    Subgraph_ColorBar.PrimaryColor = RGB(0,255,0);
    Subgraph_ColorBar.SecondaryColor = RGB(255,0,0);
    Subgraph_ColorBar.DrawZeros = true;

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

    Input_ModelInput.Name = "Use +1 (+slope) and -1 (-slope) for Color Bar Values";
    Input_ModelInput.SetYesNo(0);

    return;
}

// Do data processing

if (sc.BaseData[Input_InputData.GetInputDataIndex()][sc.Index] > sc.BaseData[Input_InputData.GetInputDataIndex()][sc.Index-1])
{
    Subgraph_ColorBar[sc.Index] = 1;
    Subgraph_ColorBar.DataColor[sc.Index] = Subgraph_ColorBar.PrimaryColor;
}
else if (sc.BaseData[Input_InputData.GetInputDataIndex()][sc.Index] < sc.BaseData[Input_InputData.GetInputDataIndex()][sc.Index-1])
{
    if (Input_ModelInput.GetYesNo() == false)
        Subgraph_ColorBar[sc.Index] = 1;
    else
        Subgraph_ColorBar[sc.Index] = -1;

    Subgraph_ColorBar.DataColor[sc.Index] = Subgraph_ColorBar.SecondaryColor;
}
else
{
    Subgraph_ColorBar[sc.Index] = 0;
}
}

/*=====*/

SCFloatArrayRef MyCCI(SCStudyInterfaceRef sc, SCFloatArrayRef In, SCFloatArrayRef SMAOut, SCFloatArrayRef CCIOut, int Index, int Length, float Multiplier)
{
    if (Length < 1)
        return CCIOut;

    if (Index < 2*Length - 1)
        return CCIOut;

```

```

sc.SimpleMovAvg(In, SMAOut, Index, Length);

float Num0 = 0;
for (int j = Index; j > Index - Length && j >= 0; j--)
    Num0 += fabs(SMAOut[Index] - In[j]);

Num0 /= Length;

CCIOut[Index] = (In[Index] - SMAOut[Index]) / (Num0 * Multiplier);

return CCIOut;
}

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

    SCInputRef Input_Length = sc.Input[0];

    if(sc.SetDefaults)
    {
        sc.GraphName="Commodity Channel Index";
        sc.StudyDescription="This is an example on how to call a function from your study that is not a member of the 'sc'
structure. It calculates the Commodity Channel Index.";

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

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

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

        return;
    }

    sc.DataStartIndex = Input_Length.GetInt();

    //We will call a custom CCI function.

    MyCCI(sc, sc.Close, Subgraph_CCI.Arrays[0], Subgraph_CCI, sc.Index, Input_Length.GetInt(), 0.015f);
}

/*=====*/
SCSFExport scsf_DMI_ADX_ADXR(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_DmiPlus = sc.Subgraph[0];
    SCSubgraphRef Subgraph_DmiMinus = sc.Subgraph[1];
    SCSubgraphRef Subgraph_ADX = sc.Subgraph[2];
    SCSubgraphRef Subgraph_ADXR = sc.Subgraph[3];

    SCInputRef Input_Length = sc.Input[3];
    SCInputRef Input_MALength = sc.Input[4];
    SCInputRef Input_ADXRInterval = sc.Input[5];

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

```

```

sc.GraphName = "DMI & ADX & ADXR";

sc.StudyDescription = "DMI: Welles Wilder's Plus and Minus Directional Indicators +DI and -DI.\n "
    "ADX: Average Directional Movement Index. This is calculated according to the Welles Wilder
formulas.\n"
    "ADXR: Average Directional Movement Index Rating. This is calculated according to the Welles
Wilder formulas.";

sc.AutoLoop = 1; // true


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

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

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

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

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

Input_MALength.Name = "Mov Avg Length";
Input_MALength.SetInt(14);
Input_MALength.SetIntLimits(1, INT_MAX);

Input_ADXRInterval.Name = "ADXR Interval";
Input_ADXRInterval.SetInt(14);
Input_ADXRInterval.SetIntLimits(1, INT_MAX);

return;
}

// Do data processing

sc.DataStartIndex = max(max(Input_Length.GetInt(), Input_Length.GetInt() + Input_MALength.GetInt() +
Input_ADXRInterval.GetInt() - 2), Input_Length.GetInt() + Input_MALength.GetInt() - 1);

sc.DMI(sc.BaseData, Subgraph_DmiPlus, Subgraph_DmiMinus, sc.Index, Input_Length.GetInt());

sc.ADX(sc.BaseData, Subgraph_ADX, sc.Index, Input_Length.GetInt(), Input_MALength.GetInt());

sc.ADXR(sc.BaseData, Subgraph_ADXR, sc.Index, Input_Length.GetInt(), Input_MALength.GetInt(),
Input_ADXRInterval.GetInt());
}

/*=====*/
SCSFExport scsf_ColorBarHHLL(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_ColorBar = sc.Subgraph[0];
    SCSubgraphRef Subgraph_AverageColor = sc.Subgraph[1];

```



```

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

    sc.GraphName = "Color Bar HH/LL";

    sc.StudyDescription = "This study function colors bars based on if they made a higher high or lower low.";

    sc.GraphRegion = 0;

    Subgraph_ColorBar.Name = "Color Bar";
    Subgraph_ColorBar.DrawStyle = DRAWSTYLE_COLOR_BAR;
    Subgraph_ColorBar.SecondaryColorUsed = 1; // true
    Subgraph_ColorBar.PrimaryColor = RGB(0,255,0);
    Subgraph_ColorBar.SecondaryColor = RGB(255,0,0);

    Subgraph_AverageColor.Name = "Average Color";
    Subgraph_AverageColor.DrawStyle = DRAWSTYLE_COLOR_BAR;
    Subgraph_AverageColor.PrimaryColor = RGB(255,128,0);

    sc.AutoLoop = 1;

    return;
}

if (Subgraph_AverageColor.PrimaryColor == 0)
    Subgraph_AverageColor.PrimaryColor = RGB(255, 128, 0);

// Do data processing

float HighValue = sc.High[sc.Index];
float LowValue = sc.Low[sc.Index];
float PrevHighValue = sc.High[sc.Index-1];
float PrevLowValue = sc.Low[sc.Index-1];

if (HighValue > PrevHighValue && LowValue < PrevLowValue)
{
    Subgraph_ColorBar[sc.Index] = 1;

    //uint32_t Pri = ColorBar.PrimaryColor;
    //uint32_t Sec = ColorBar.SecondaryColor;
    Subgraph_ColorBar.DataColor[sc.Index] = Subgraph_AverageColor.PrimaryColor;//sc.RGBInterpolate(Pri, Sec,
0.5f);
}
else if (HighValue > PrevHighValue)
{
    Subgraph_ColorBar[sc.Index] = 1;
    Subgraph_ColorBar.DataColor[sc.Index] = Subgraph_ColorBar.PrimaryColor;
}
else if (LowValue < PrevLowValue)
{
    Subgraph_ColorBar[sc.Index] = 1;
    Subgraph_ColorBar.DataColor[sc.Index] = Subgraph_ColorBar.SecondaryColor;
}
else
{
    Subgraph_ColorBar[sc.Index] = 0;
}
}

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

```

```

SCSubgraphRef Subgraph_ColorVolume = sc.Subgraph[0];

SCInputRef Input_InputData = sc.Input[0];

if (sc.SetDefaults)
{
    // Set the configuration and defaults
    sc.GraphName = "Color Volume Based On Slope";
    sc.StudyDescription = "Colors Volume bars based on the slope of the study subgraph it is Based On.";

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

    Subgraph_ColorVolume.Name = "Color Volume";
    Subgraph_ColorVolume.DrawStyle = DRAWSTYLE_BAR;
    Subgraph_ColorVolume.PrimaryColor = RGB(0,255,0);
    Subgraph_ColorVolume.SecondaryColorUsed = 1; // true
    Subgraph_ColorVolume.SecondaryColor = RGB(0,127,0);

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

    return;
}

// Do data processing

if (sc.BaseData[Input_InputData.GetInputDataIndex()][sc.Index] > sc.BaseData[Input_InputData.GetInputDataIndex()][sc.Index-1])
{
    Subgraph_ColorVolume[sc.Index] = sc.Volume[sc.Index];
    Subgraph_ColorVolume.DataColor[sc.Index] = Subgraph_ColorVolume.PrimaryColor;
}
else if (sc.BaseData[Input_InputData.GetInputDataIndex()][sc.Index] <
sc.BaseData[Input_InputData.GetInputDataIndex()][sc.Index-1])
{
    Subgraph_ColorVolume[sc.Index] = sc.Volume[sc.Index];
    Subgraph_ColorVolume.DataColor[sc.Index] = Subgraph_ColorVolume.SecondaryColor;
}
else
{
    Subgraph_ColorVolume[sc.Index] = 0;
}
}

/*=====*/
SCSFExport scsf_DetrendedOsc(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Oscillator = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Line1 = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Line2 = sc.Subgraph[2];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_MALength = sc.Input[3];
    SCInputRef Input_Line1Value = sc.Input[4];
    SCInputRef Input_Line2Value = sc.Input[5];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Detrended Oscillator - Di Napoli";

        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 = true;

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

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

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

Input_MALength.Name = "Mov Avg Length";
Input_MALength.SetInt(7);
Input_MALength.SetIntLimits(1,MAX_STUDY_LENGTH);

Input_Line1Value.Name = "Line 1 Value";
Input_Line1Value.SetFloat(10);

Input_Line2Value.Name = "Line 2 Value";
Input_Line2Value.SetFloat(-10);

sc.DrawZeros = true;

sc.AutoLoop = 1;

return;
}

sc.DataStartIndex = Input_MALength.GetInt();

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

sc.SimpleMovAvg(sc.BaseData[Input_InputData.GetInputDataIndex()],Subgraph_Oscillator.Arrays[0],
Input_MALength.GetInt());

Subgraph_Oscillator[sc.Index] = sc.BaseData[Input_InputData.GetInputDataIndex()][sc.Index] -
Subgraph_Oscillator.Arrays[0][sc.Index];

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

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

    SCInputRef Input_HorizontalPosition = sc.Input[0];
    SCInputRef Input_VerticalPosition = sc.Input[1];
    SCInputRef Input_DisplayValueOnly = sc.Input[3];
    SCInputRef Input_DisplayContinuousTimeCountdown = sc.Input[4];
    SCInputRef Input_DisplayRangeInTicks = sc.Input[5];
    SCInputRef Input_AlertOnNewBar = sc.Input[6];
    SCInputRef Input_TransparentLabelBackground = sc.Input[7];
    SCInputRef Input_AlertForEveryNewBar = sc.Input[8];
    SCInputRef Input_DisplayRemainingTimeAsSecondsOnly = sc.Input[9];
    SCInputRef Input_UseCurrentPriceForVP = sc.Input[11];

```

```

SCInputRef Input_DrawAboveMainPriceGraph = sc.Input[12];

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

    sc.GraphName = "CountDown Timer";

    sc.StudyDescription = "Bar countdown timer";

    sc.AutoLoop = 0;//Manual looping

    sc.GraphRegion = 0;
    sc.ValueFormat = 2;
    Subgraph_Countdown.Name = "CountDown";
    Subgraph_Countdown.LineWidth = 20;
    Subgraph_Countdown.DrawStyle = DRAWSTYLE_CUSTOM_TEXT;
    Subgraph_Countdown.PrimaryColor = RGB(0, 0, 0); //black
    Subgraph_Countdown.SecondaryColor = RGB(255, 127, 0); //Orange
    Subgraph_Countdown.SecondaryColorUsed = true;
    Subgraph_Countdown.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_DisplayValueOnly.Name = "Display Value Only";
    Input_DisplayValueOnly.SetYesNo(false);

    Input_DisplayContinuousTimeCountdown.Name = "Display Continuous Time Countdown Based on Real-Time
Clock";
    Input_DisplayContinuousTimeCountdown.SetYesNo(false);

    Input_DisplayRangeInTicks.Name = "Display Remaining Range in Ticks";
    Input_DisplayRangeInTicks.SetYesNo(false);

    Input_AlertOnNewBar.Name = "Alert on New Bar";
    Input_AlertOnNewBar.SetYesNo(false);

    Input_AlertForEveryNewBar.Name = "Alert for Every New Bar";
    Input_AlertForEveryNewBar.SetYesNo(false);

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

    Input_DisplayRemainingTimeAsSecondsOnly.Name = "Display Remaining Time as Seconds Only";
    Input_DisplayRemainingTimeAsSecondsOnly.SetYesNo(false);

    Input_UseCurrentPriceForVP.Name = "Use Current Price For Vertical Position";
    Input_UseCurrentPriceForVP.SetYesNo(false);

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

    return;
}

```

```
static const int CountdownTimerLineNumber = COUNTDOWN_TIMER_CHART_DRAWINGNUMBER;
```

```
if ((sc.LastCallToFunction || sc.HideStudy)
    && sc.UserDrawnChartDrawingExists(sc.ChartNumber, CountdownTimerLineNumber) > 0)
{
    // be sure to delete user drawn type drawing when study removed or the study is hidden
    sc.DeleteUserDrawnACSDrawing(sc.ChartNumber, CountdownTimerLineNumber);
    return;
}
```

```
if (sc.HideStudy)
    return;
```

```
int& PriorArraySize = sc.GetPersistentInt(1);
```

```
int DrawAboveMainPriceGraph = Input_DrawAboveMainPriceGraph.GetYesNo();
```

```
// Disable UpdateAlways.
sc.UpdateAlways = 0;
```

```
s_UseTool Tool;
Tool.ChartNumber = sc.ChartNumber;
Tool.DrawingType = DRAWING_TEXT;
Tool.Region = sc.GraphRegion;
Tool.LineNumber = CountdownTimerLineNumber;
Tool.AddMethod = UTAM_ADD_OR_ADJUST;
Tool.AddAsUserDrawnDrawing = 1;
Tool.AllowSaveToChartbook = 1;
Tool.DrawUnderneathMainGraph = DrawAboveMainPriceGraph ? 0 : 1;
```

```
if (sc.UserDrawnChartDrawingExists(sc.ChartNumber, CountdownTimerLineNumber) == 0)
{
    Tool.BeginDateTime = Input_HorizontalPosition.GetInt();
    Tool.BeginValue = static_cast<float>(Input_VerticalPosition.GetInt());
    Tool.UseRelativeVerticalValues = true;
}
```

```
if (Tool.Region == 0 && Input_UseCurrentPriceForVP.GetYesNo() != 0)
{
    int ValueIndex = sc.ArraySize - 1;

    Tool.BeginValue = sc.Close[ValueIndex];
    Tool.UseRelativeVerticalValues = false;
}
else
{
    Tool.UseRelativeVerticalValues = true;
}
```

```
Tool.Color = Subgraph_Countdown.PrimaryColor;
Tool.FontBackColor = Subgraph_Countdown.SecondaryColor;
if (Input_TransparentLabelBackground.GetYesNo() != 0)
    Tool.TransparentLabelBackground = 1;
```

```
Tool.ReverseTextColor = false;
Tool.FontBold = true;
Tool.FontSize = Subgraph_Countdown.LineWidth;
Tool.FontFace = sc.GetChartTextFontFaceName();
```

```
float RemainingAmountForSubgraphValue = 0.0;
```

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

```

//Continuous time countdown.
if (Input_DisplayContinuousTimeCountdown.GetYesNo()
    && BarPeriod.IntradayChartBarPeriodType == IBPT_DAYS_MINS_SECS
    && sc.BaseGraphGraphDrawType != GDT_TPO_PROFILE)
{
    SCDateTime CurrentDateTime = sc.GetCurrentDateTime();

    SCDateTime BarDuration = SCDateTime::SECONDS(sc.SecondsPerBar);

    SCDateTime BarEndingDateTime = sc.BaseDateTimeIn[sc.ArraySize - 1] +
    SCDateTime::SECONDS(BarPeriod.IntradayChartBarPeriodParameter1);

    SCDateTime RemainingTime = BarEndingDateTime - CurrentDateTime;

    if (RemainingTime < 0.0 && BarDuration != 0.0)
    {
        RemainingTime = 0;
        double ElapsedBars = (CurrentDateTime - sc.BaseDateTimeIn[sc.ArraySize-1]).GetAsDouble() /
        BarDuration.GetAsDouble();
        double ElapsedPercentageOfCurrentBar = ElapsedBars - static_cast<int>(ElapsedBars);
        RemainingTime = (1.0 - ElapsedPercentageOfCurrentBar) * BarDuration.GetAsDouble();
    }

    RemainingAmountForSubgraphValue = static_cast<float>(RemainingTime.GetFloatSecondsSinceBaseDate());

    SCString StrRemainingTime;

    int Hour, Minute, Second;
    RemainingTime.GetTimeHMS(Hour, Minute, Second);

    if(Input_DisplayRemainingTimeAsSecondsOnly.GetYesNo())
        StrRemainingTime.Format("%02d", RemainingTime.GetTimeInSeconds());
    else if (Hour && Minute)
        StrRemainingTime.Format("%d:%02d:%02d",Hour, Minute, Second);
    else if (Minute)
        StrRemainingTime.Format("%d:%02d", Minute, Second);
    else
        StrRemainingTime.Format("%02d", Second);

    Tool.Text = StrRemainingTime;

    //Enable update always so our real-time countdown is continuously visible.
    sc.UpdateAlways = 1;
}
else if (BarPeriod.AreRangeBars() && Input_DisplayRangeInTicks.GetYesNo())
{
    SCString CountDownText(sc.GetCountDownText());
    const char* CharPosition = strchr(CountDownText.GetChars(), ':');
    int RangeInTicks = 0;
    if (CharPosition)
    {
        float RR = static_cast<float>(atof(CharPosition+1));
        RangeInTicks = static_cast<int>(RR/sc.TickSize+0.5f);
        RemainingAmountForSubgraphValue = static_cast<float>(RangeInTicks);
        Tool.Text.Format("%d ",RangeInTicks);
    }
    else
        Tool.Text = "";
}
else if (Input_DisplayValueOnly.GetYesNo() || (Input_DisplayRemainingTimeAsSecondsOnly.GetYesNo()
    && BarPeriod.IntradayChartBarPeriodType == IBPT_DAYS_MINS_SECS))

```

```

{
    int CountdownValue = sc.GetLatestBarCountdownAsInteger();
    RemainingAmountForSubgraphValue = static_cast<float>(CountdownValue);

    Tool.Text.Format("%d", CountdownValue);
}
else
    Tool.Text.Format("%s ", sc.GetCountDownText().GetChars());

sc.UseTool(Tool);

if (!sc.ChartIsDownloadingHistoricalData(sc.ChartNumber)
    && PriorArraySize < sc.ArraySize
    && sc.UpdateStartIndex != 0
    && Input_AlertOnNewBar.GetYesNo())
{
    int StartIndex;

    if (Input_AlertForEveryNewBar.GetYesNo())
        StartIndex = sc.UpdateStartIndex;
    else
        StartIndex = sc.ArraySize-2;

    for(int BarIndex = StartIndex; BarIndex < sc.ArraySize - 1; BarIndex++)
        sc.PlaySound( sc.SelectedAlertSound, "Bar has completed", 0);
}

PriorArraySize = sc.ArraySize;

// Zero out prior elements and set the remaining time into the last subgraph element.
for(int Index = sc.UpdateStartIndex; Index < sc.ArraySize; Index++)
{
    Subgraph_Countdown[Index] = 0;
}

Subgraph_Countdown[sc.ArraySize - 1] = RemainingAmountForSubgraphValue;
}

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

    SCInputRef Input_HorizontalPosition = sc.Input[0];
    SCInputRef Input_VerticalPosition = sc.Input[1];
    SCInputRef Input_UseAMPM = sc.Input[3];
    SCInputRef Input_DrawAboveMainPriceGraph = sc.Input[4];
    SCInputRef Input_UseBoldFont = sc.Input[5];
    SCInputRef Input_TransparentLabelBackground = sc.Input[7];
    SCInputRef Input_HourOffset = sc.Input[8];

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

        sc.GraphName = "Clock - RealTime";

        sc.AutoLoop = 0; //Manual looping
        sc.GraphRegion = 0;
        sc.ValueFormat = 0;
        sc.UpdateAlways = 1;
    }
}

```

```

Subgraph_Clock.Name = "Clock";
Subgraph_Clock.LineWidth = 20;
Subgraph_Clock.DrawStyle = DRAWSTYLE_CUSTOM_TEXT;
Subgraph_Clock.PrimaryColor = RGB(0, 0, 0); //black
Subgraph_Clock.SecondaryColor = RGB(255, 127, 0); //Orange
Subgraph_Clock.SecondaryColorUsed = true;
Subgraph_Clock.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_UseAMPM.Name = "Use A.M. and P.M.";
Input_UseAMPM.SetYesNo(false);

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

Input_UseBoldFont.Name = "Use Bold Font";
Input_UseBoldFont.SetYesNo(true);

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

Input_HourOffset.Name = "Hour Offset";
Input_HourOffset.SetInt(0);

return;
}

int Hour = 0, Minute = 0, Second = 0;
if (sc.IsReplayRunning())
    SCDateTimeMS(sc.GetEndingDateTimeForBarIndex(sc.ArraySize - 1)).GetTimeHMS(Hour, Minute, Second);
else
    sc.CurrentSystemDateTimeMS.GetTimeHMS(Hour, Minute, Second);

if (Input_HourOffset.GetInt() != 0)
{
    Hour += Input_HourOffset.GetInt();
    if (Hour > 24)
        Hour -= 24;
    else if (Hour < 0)
        Hour += 24;
}

SCString RemainingTimeString;
if(Input_UseAMPM.GetYesNo())
{
    int AdjustedHour = Hour;
    if (Hour >= 13)
        AdjustedHour -= 12;

    RemainingTimeString.Format("%d:%02d:%02d ", AdjustedHour, Minute, Second);
    if (Hour < 12)
        RemainingTimeString += "am";
}

```



```

    else
        RemainingTimeString += "pm";
    }
    else
        RemainingTimeString.Format("%d:%02d:%02d", Hour, Minute, Second);

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

    int DrawAboveMainPriceGraph = Input_DrawAboveMainPriceGraph.GetYesNo();

    int TransparentLabelBackground = Input_TransparentLabelBackground.GetYesNo();
    int UseBoldFont = Input_UseBoldFont.GetYesNo();

    sc.AddAndManageSingleTextUserDrawnDrawingForStudy(sc, 0, HorizontalPosition, VerticalPosition, Subgraph_Clock,
    TransparentLabelBackground, RemainingTimeString, DrawAboveMainPriceGraph, 0, UseBoldFont);
}

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

    SCInputRef Input_HorizontalPosition = sc.Input[0];
    SCInputRef Input_VerticalPosition = sc.Input[1];
    SCInputRef Input_TransparentLabelBackground = sc.Input[4];

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

        sc.GraphName = "Account Balance Text - External Service";

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

        Subgraph_AccountBalance.Name = "Cash Balance";
        Subgraph_AccountBalance.LineWidth = 20;
        Subgraph_AccountBalance.DrawStyle = DRAWSTYLE_CUSTOM_TEXT;
        Subgraph_AccountBalance.PrimaryColor = RGB(0, 0, 0); //black
        Subgraph_AccountBalance.SecondaryColor = RGB(255, 127, 0); //Orange
        Subgraph_AccountBalance.SecondaryColorUsed = true;
        Subgraph_AccountBalance.DisplayNameValueInWindowsFlags = 0;

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

        return;
    }

    int & IndexOfLastUsedSubgraphElementBalanceData = sc.GetPersistentInt(2);

```

```

Subgraph_AccountBalance[ IndexOfLastUsedSubgraphElementBalanceData ] = 0;

double AccountValue = 0;
n_ACSIL::s_TradeAccountDataFields TradeAccountDataFields;
if (sc.GetTradeAccountData(TradeAccountDataFields, sc.SelectedTradeAccount))
{
    AccountValue = TradeAccountDataFields.m_CurrentCashBalance;
}

SCString ValueText;
ValueText.Format("%.2f", AccountValue);

Subgraph_AccountBalance[sc.ArraySize - 1] = static_cast<float>(AccountValue);

IndexOfLastUsedSubgraphElementBalanceData = sc.ArraySize - 1;

int TransparentLabelBackground = Input_TransparentLabelBackground.GetYesNo();

sc.AddAndManageSingleTextUserDrawnDrawingForStudy(sc, false, Input_HorizontalPosition.GetInt(),
Input_VerticalPosition.GetInt(), Subgraph_AccountBalance, TransparentLabelBackground, ValueText, true, 0);
}

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

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

        sc.GraphName = "Account Balance Graph - External Service";

        sc.StudyDescription = "";

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

        Subgraph_Balance.Name = "Balance";
        Subgraph_Balance.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Balance.DrawZeros = false; // false
        Subgraph_Balance.PrimaryColor = RGB(0, 255, 0); // Green

        return;
    }

    // Do data processing
    double AccountValue = 0;
    n_ACSIL::s_TradeAccountDataFields TradeAccountDataFields;
    if (sc.GetTradeAccountData(TradeAccountDataFields, sc.SelectedTradeAccount))
    {
        AccountValue = TradeAccountDataFields.m_CurrentCashBalance;
    }

    Subgraph_Balance[sc.Index] = static_cast<float>(AccountValue);
}

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

```

```

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

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

    sc.GraphName = "Moving Average - Welles Wilders";

    sc.StudyDescription = "";

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

    // Set the name of the first subgraph
    Subgraph_MovAvg.Name = "MovAvg";

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

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

    // Make the Length input and default it to 5
    Input_Length.Name = "Length";
    Input_Length.SetInt(10);
    Input_Length.SetIntLimits(1,MAX_STUDY_LENGTH);

    sc.AutoLoop = 1;

    return;
}

// Do data processing

sc.WildersMovingAverage(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_MovAvg, sc.Index,
Input_Length.GetInt());
}

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

    SCInputRef Input_DisplayAtBottom = sc.Input[0];
    SCInputRef Input_TextOffset = sc.Input[1];
    SCInputRef Input_NumberingEveryNBars = sc.Input[2];
    SCInputRef Input_DoNotResetDaily = sc.Input[4];

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

```

```

sc.GraphName = "Bar Numbering";

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

// Set the name of the first subgraph
Subgraph_Number.Name = "Number";
Subgraph_Number.DrawStyle = DRAWSTYLE_CUSTOM_VALUE_AT_Y;
Subgraph_Number.PrimaryColor = RGB(0,255,0);
Subgraph_Number.LineWidth = 8;
Subgraph_Number.DrawZeros = false;

Input_DisplayAtBottom.Name = "Display At Bottom";
Input_DisplayAtBottom.SetYesNo(false);

Input_TextOffset.Name = "Text Labels Tick Offset";
Input_TextOffset.SetInt(2);
Input_TextOffset.SetIntLimits(0, 10000);

Input_NumberingEveryNBars.Name = "Number Every N Bars";
Input_NumberingEveryNBars.SetInt(2);
Input_NumberingEveryNBars.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_DoNotResetDaily.Name = "Do Not Reset at Start of New Day";
Input_DoNotResetDaily.SetYesNo(false);

sc.AutoLoop = 1;

return;
}

if (!Input_DisplayAtBottom.GetYesNo())
{
    sc.ScaleRangeType = SCALE_AUTO;
}
else
{
    sc.ScaleRangeType = SCALE_USERDEFINED;
    sc.ScaleRangeTop = 100;
    sc.ScaleRangeBottom = 1;
}

sc.ValueFormat = 0;
Subgraph_Number.DrawStyle = DRAWSTYLE_CUSTOM_VALUE_AT_Y;

int& r_BarCounter = sc.GetPersistentInt(1);
int& r_LastProcessedBarIndex = sc.GetPersistentInt(2);

if (sc.Index == 0)
{
    r_BarCounter = 1;
    r_LastProcessedBarIndex = -1;
}
else if (sc.Index != r_LastProcessedBarIndex)
{
    SCDateTimeMS StartDateTimeOfDay;

    StartDateTimeOfDay.SetTime(sc.StartTimeOfDay);

    StartDateTimeOfDay.SetDate(sc.BaseDateTimeln[sc.Index].GetDate());

    //reset counter if a new session has begun.
    if (!Input_DoNotResetDaily.GetYesNo() &&
        (sc.BaseDateTimeln[sc.Index] == StartDateTimeOfDay ||

```

```

        (sc.BaseDateTimeln[sc.Index-1] < StartDateTimeOfDay && sc.BaseDateTimeln[sc.Index] > StartDateTimeOfDay)
    )
    r_BarCounter = 1;
else
    r_BarCounter++; //increment counter
}

r_LastProcessedBarIndex = sc.Index;

if(r_BarCounter != 1 && r_BarCounter % Input_NumberingEveryNBars.GetInt())
    return;

Subgraph_Number[sc.Index] = static_cast<float>(r_BarCounter);

float Offset = sc.TickSize * Input_TextOffset.GetInt();

float VerticalPositionValue = 0;

if (Input_DisplayAtBottom.GetYesNo())
    VerticalPositionValue = 5;
else
    VerticalPositionValue = sc.Low[sc.Index] - Offset;

Subgraph_Number.Arrays[0][sc.Index] = VerticalPositionValue;
}

/*=====*/

SCSFExport scsf_StudySubgraphsReference(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_PriceOverlay1 = sc.Subgraph[0];
    SCSubgraphRef Subgraph_PriceOverlay2 = sc.Subgraph[1];

    SCInputRef Input_StudySubgraph1 = sc.Input[0];
    SCInputRef Input_StudySubgraph2 = sc.Input[1];
    SCInputRef Input_DrawZeros = sc.Input[2];

    // Set configuration variables

    if (sc.SetDefaults)
    {
        sc.GraphName = "Study Subgraphs Reference";
        sc.GraphRegion = 1;

        sc.ScaleRangeType = SCALE_INDEPENDENT;

        Subgraph_PriceOverlay1.Name = "Study Subgraph 1";
        Subgraph_PriceOverlay1.DrawStyle = DRAWSTYLE_FILL_TOP;
        Subgraph_PriceOverlay1.PrimaryColor = RGB(0,255,0);

        Subgraph_PriceOverlay2.Name = "Study Subgraph 2";
        Subgraph_PriceOverlay2.DrawStyle = DRAWSTYLE_FILL_BOTTOM;
        Subgraph_PriceOverlay2.PrimaryColor = RGB(255,0,255);

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

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

```

```

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

sc.CalculationPrecedence = VERY_LOW_PREC_LEVEL;
sc.DrawStudyUnderneathMainPriceGraph = true;

sc.AutoLoop = 0;

return;
}

// Do data processing

//Set Draw Zeros as defined by user input
Subgraph_PriceOverlay1.DrawZeros = Input_DrawZeros.GetYesNo();
Subgraph_PriceOverlay2.DrawZeros = Input_DrawZeros.GetYesNo();

//This will return sc.UpdateStartIndex or an earlier index if this study is calculated after a study which calculated at an
earlier index and supports setting that earlier index.
int ActualStartIndex = sc.GetCalculationStartIndexForStudy();

// Get the array for the specified Input Data from the specified studies
SCFloatArray Study1Array;
sc.GetStudyArrayUsingID(Input_StudySubgraph1.GetStudyID(),
Input_StudySubgraph1.GetSubgraphIndex(),Study1Array);

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

for(int BarIndex = ActualStartIndex; BarIndex < sc.ArraySize; BarIndex++)
{
    Subgraph_PriceOverlay1[BarIndex] = Study1Array[BarIndex];
    Subgraph_PriceOverlay2[BarIndex] = Study2Array[BarIndex];
}

sc.EarliestUpdateSubgraphDataArrayIndex = ActualStartIndex;
}

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

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

    // Set configuration variables

    if (sc.SetDefaults)
    {
        sc.GraphName = "Study Subgraph Reference";
        sc.GraphRegion = 2;

        sc.ScaleRangeType = SCALE_AUTO;

        Subgraph_StudySubgraphReference.Name = "Study Subgraph";
        Subgraph_StudySubgraphReference.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_StudySubgraphReference.PrimaryColor = RGB(0, 255, 0);

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

        Input_DrawZeros.Name = "Draw Zeros";

```

```

    Input_DrawZeros.SetYesNo(true);

    sc.CalculationPrecedence = VERY_LOW_PREC_LEVEL;

    sc.AutoLoop = 0;

    return;
}

// Do data processing
Subgraph_StudySubgraphReference.DrawZeros = Input_DrawZeros.GetYesNo();

//This will return sc.UpdateStartIndex or an earlier index if this study is calculated after a study which calculated at an
earlier index and supports setting that earlier index.
int ActualStartIndex = sc.GetCalculationStartIndexForStudy();

// Get the array for the specified Input Data from the specified studies
SCFloatArray Study1Array;
sc.GetStudyArrayUsingID(Input_StudySubgraphReference.GetStudyID(),
Input_StudySubgraphReference.GetSubgraphIndex(), Study1Array);

for (int BarIndex = ActualStartIndex; BarIndex < sc.ArraySize; BarIndex++)
{
    Subgraph_StudySubgraphReference[BarIndex] = Study1Array[BarIndex];
}

sc.EarliestUpdateSubgraphDataArrayIndex = ActualStartIndex;
}

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

    if (sc.SetDefaults)
    {
        sc.GraphName = "Volume - Close To Midpoint Colored";

        sc.ValueFormat = 0;

        sc.AutoLoop = true;
        sc.ScaleRangeType = SCALE_ZEROBASED;

        Subgraph_Volume.Name = "Volume";
        Subgraph_Volume.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_Volume.PrimaryColor = RGB(0, 255, 0);
        Subgraph_Volume.SecondaryColor = RGB(255, 0, 0);
        Subgraph_Volume.SecondaryColorUsed = 1;
        Subgraph_Volume.LineWidth = 2;
        Subgraph_Volume.DrawZeros = false;

        sc.DisplayStudyInputValues = false;

        return;
    }

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

    if(sc.Close[sc.Index] >= sc.HLAvg[sc.Index])
        Subgraph_Volume.DataColor[sc.Index] = Subgraph_Volume.PrimaryColor;
    else
        Subgraph_Volume.DataColor[sc.Index] = Subgraph_Volume.SecondaryColor;
}

```

```

/*=====*/
SCSFExport scsf_CumulativeAdjustedValue(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_CumulativeValue = sc.Subgraph[0];
    SCFloatArray Array_AverageValue = Subgraph_CumulativeValue.Arrays[0];
    SCFloatArray Array_AdjustedValue = Subgraph_CumulativeValue.Arrays[1];

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

    if (sc.SetDefaults)
    {
        // Set the configuration and defaults
        sc.GraphName = "Cumulative Adjusted Value";
        sc.StudyDescription = "Input data is smoothed to attain a bias, then adjusted to remove the bias, and then accumulated.";

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

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

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

        Input_SmoothingLength.Name = "Long Term Smoothing Length";
        Input_SmoothingLength.SetInt(10000);
        Input_SmoothingLength.SetIntLimits(1, MAX_STUDY_LENGTH);

        return;
    }

    // Do data processing
    sc.ExponentialMovAvg(sc.BaseData[Input_InputData.GetInputDataIndex()], Array_AverageValue,
    Input_SmoothingLength.GetInt());

    Array_AdjustedValue[sc.Index] = sc.BaseData[Input_InputData.GetInputDataIndex()][sc.Index] -
    Array_AverageValue[sc.Index];

    if (sc.Index == 0)
        Subgraph_CumulativeValue[sc.Index] = Array_AdjustedValue[sc.Index];
    else
        Subgraph_CumulativeValue[sc.Index] = Subgraph_CumulativeValue[sc.Index-1] + Array_AdjustedValue[sc.Index];
}

/*=====*/
SCSFExport scsf_TradersDynamicIndex(SCStudyInterfaceRef sc)
{
    if (sc.SetDefaults)
    {
        sc.GraphName = "Traders Dynamic Index";
        sc.StudyDescription = "<p><strong>Traders Dynamic Index</strong></p><p>with thanks to Dean Malone <a href=http://www.compassfx.com>CompassFX</a></p><p><a href=http://www.earnforex.com/metatrader-indicators/Traders-Dynamic-Index>Earnforex description</a></p>";

        sc.AutoLoop = 1;

        sc.Subgraph[0].Name = "RSI";
        sc.Subgraph[0].DrawStyle = DRAWSTYLE_IGNORE;
        sc.Subgraph[1].Name = "UpVol";
    }
}

```



```

sc.Subgraph[1].DrawStyle = DRAWSTYLE_LINE;
sc.Subgraph[2].Name = "Mid";
sc.Subgraph[2].DrawStyle = DRAWSTYLE_LINE;
sc.Subgraph[3].Name = "DnVol";
sc.Subgraph[3].DrawStyle = DRAWSTYLE_LINE;
sc.Subgraph[4].Name = "Ma";
sc.Subgraph[4].DrawStyle = DRAWSTYLE_LINE;
sc.Subgraph[5].Name = "Mb";
sc.Subgraph[5].DrawStyle = DRAWSTYLE_LINE;

sc.Input[3].Name = "RSI Length";
sc.Input[3].SetInt(13);
sc.Input[3].SetDescription("<strong>RSI_Period</strong> (default = 13) &#151; period in bars for calculation of RSI.");

sc.Input[4].Name = "RSI Price";
sc.Input[4].SetInputDataIndex(SC_LAST);
sc.Input[4].SetDescription("<strong>RSI_Price</strong> (default = MODE_CLOSE) &#151; price type to use in RSI calculation");

sc.Input[5].Name = "Volatility Band Length";
sc.Input[5].SetInt(34);
sc.Input[5].SetDescription("<strong>Volatility_Band</strong> (default = 34) &#151; parameter for volatility band calculation. Can be between 20 and 40. The lower this value is the curvier becomes the band.");

sc.Input[6].Name = "RSI Price Line";
sc.Input[6].SetInt(2);
sc.Input[6].SetDescription("<strong>RSI_Price_Line</strong> (default = 2) &#151; period of the first moving average (fast).");

sc.Input[7].Name = "RSI Price Type";
sc.Input[7].SetMovAvgType(MOVAVGTYPE_SIMPLE);

sc.Input[8].Name = "RSI Trade Signal Line";
sc.Input[8].SetInt(7);
sc.Input[8].SetDescription("<strong>Trade_Signal_Line</strong> (default = 7) &#151; period of the second moving average (slow)");

sc.Input[9].Name = "Trade Signal Type";
sc.Input[9].SetMovAvgType(MOVAVGTYPE_SIMPLE);

sc.DocumentationImageUrl =
"http://codebase.mql4.com/c/codebase/2008/03/indicatori\_gtradersmdynamicmindeuvisualqalerts.gif";

return;
}

```

```

sc.RSI(sc.BaseDataIn[sc.Input[4].GetInputDataIndex()], sc.Subgraph[0], MOVAVGTYPE_SIMPLE,
sc.Input[3].GetInt());

sc.MovingAverage(sc.Subgraph[0], sc.Subgraph[4], sc.Input[7].GetMovAvgType(), sc.Input[6].GetInt());

sc.MovingAverage(sc.Subgraph[0], sc.Subgraph[5], sc.Input[9].GetMovAvgType(), sc.Input[8].GetInt());

sc.BollingerBands(sc.Subgraph[0], sc.Subgraph[2], sc.Input[5].GetInt(), 1.6185f, MOVAVGTYPE_SIMPLE);
sc.Subgraph[1][sc.Index] = sc.Subgraph[2].Arrays[0][sc.Index];
sc.Subgraph[3][sc.Index] = sc.Subgraph[2].Arrays[1][sc.Index];

}

```

```

/*=====

```

```

-----*/
SCSFExport scsf_PointFigureChartReversalMarker(SCStudyInterfaceRef sc)

```

```

{
    SCSubgraphRef Subgraph_Marker = sc.Subgraph[0];

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

        sc.GraphName = "Point and Figure Chart Reversal Marker";
        sc.GraphRegion = 0;
        sc.AutoLoop = 0;
        sc.ValueFormat = sc.BaseGraphValueFormat;

        Subgraph_Marker.Name = "Reversal Mark";
        Subgraph_Marker.DrawStyle = DRAWSTYLE_DASH;
        Subgraph_Marker.LineWidth = 4;
        Subgraph_Marker.PrimaryColor = RGB(255, 0, 128);
        Subgraph_Marker.DrawZeros = false;

        return;
    }

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

    if (BarPeriodData.ChartDataType != INTRADAY_DATA
        || BarPeriodData.IntradayChartBarPeriodType != IBPT_POINT_AND_FIGURE)
        return;

    float ReversalAmountNeeded = BarPeriodData.IntradayChartBarPeriodParameter1 * sc.TickSize *
    BarPeriodData.IntradayChartBarPeriodParameter2;

    for (int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize; BarIndex++)
    {
        Subgraph_Marker[BarIndex] = 0;
        if (BarIndex == sc.ArraySize - 1)
        {
            if (sc.BaseData[PF_DIRECTION_ARRAY][BarIndex] == 1) // Up bar
            {
                Subgraph_Marker[BarIndex] = sc.BaseData[SC_POINT_FIGURE_HIGH][BarIndex] -
                ReversalAmountNeeded;
            }
            else
            {
                Subgraph_Marker[BarIndex] = sc.BaseData[SC_POINT_FIGURE_LOW][BarIndex] + ReversalAmountNeeded;
            }
        }
    }
}

/*=====
-----*/
SCSFExport scsf_PointFigureBoxCount(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_BoxCount = sc.Subgraph[0];

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

        sc.GraphName = "Point and Figure Box Count";
        sc.GraphRegion = 1;
        sc.AutoLoop = 0;
        sc.ValueFormat = 0;
    }
}

```

```

    Subgraph_BoxCount.Name = "P&F Box Count";
    Subgraph_BoxCount.DrawStyle = DRAWSTYLE_BAR;
    Subgraph_BoxCount.LineWidth = 2;
    Subgraph_BoxCount.PrimaryColor = RGB(255, 0, 128);
    Subgraph_BoxCount.DrawZeros = false;

    return;
}

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

if (BarPeriod.IntradayChartBarPeriodType != IBPT_POINT_AND_FIGURE)
    return;

for (int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize; BarIndex++)
{
    Subgraph_BoxCount[BarIndex] = sc.BaseData[PF_NUM_BOXES_ARRAY][BarIndex];
}

}

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

        sc.GraphName = "Read from Underlying Intraday File Example";

        sc.AutoLoop = 0;

        return;
    }

    if (sc.LastCallToFunction)
        return;

    s_IntradayRecord IntradayRecord;

    for (int FileIndex = sc.FileRecordIndexOfLastDataRecordInChart; FileIndex >=
sc.FileRecordIndexOfLastDataRecordInChart - 10; FileIndex--)
    {
        bool FirstIteration = FileIndex == sc.FileRecordIndexOfLastDataRecordInChart;
        bool LastIteration = FileIndex <= sc.FileRecordIndexOfLastDataRecordInChart - 10;
        IntradayFileLockActionEnum IntradayFileLockAction = IFLA_NO_CHANGE;

        if (FirstIteration && LastIteration)
            IntradayFileLockAction = IFLA_LOCK_READ_RELEASE;
        else if (FirstIteration)
            IntradayFileLockAction = IFLA_LOCK_READ_HOLD;
        else if (LastIteration)
            IntradayFileLockAction = IFLA_RELEASE_AFTER_READ;

        sc.ReadIntradayFileRecordAtIndex(FileIndex, IntradayRecord, IntradayFileLockAction);

        SCDateTimeMS TradeTimestampInChartTimeZone = IntradayRecord.DateTime;

```

```

}

// Could also use this line of code to release the lock if not sure when will be complete with reading and want to do it as
a separate operation. An index of -1 signifies not actually to perform a read.
//sc.ReadIntradayFileRecordAtIndex( -1, IntradayRecord, IFLA_RELEASE_AFTER_READ);

}

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

        sc.GraphName = "Read Chart Bar Records from Underlying Intraday File Example";

        sc.AutoLoop = 0;

        //this must be set to 1 in order to use the sc.ReadIntradayFileRecordForBarIndexAndSubIndex function.
        sc.MaintainAdditionalChartDataArrays = 1;

        return;
    }

    if (sc.LastCallToFunction)
        return;

    s_IntradayRecord IntradayRecord;

    if (sc.GetBarHasClosedStatus(sc.UpdateStartIndex) == BHCS_BAR_HAS_NOT_CLOSED)//Only execute on updating
of last bar.
    {
        int ReadSuccess = true;
        bool FirstIteration = true;
        uint32_t TotalVolume = 0;
        int SubIndex = 0;//Start at first record within bar

        //Read records until sc.ReadIntradayFileRecordForBarIndexAndSubIndex returns 0
        while (ReadSuccess)
        {
            IntradayFileLockActionEnum IntradayFileLockAction = IFLA_NO_CHANGE;

            if (FirstIteration)
            {
                IntradayFileLockAction = IFLA_LOCK_READ_HOLD;

                FirstIteration = false;
            }

            ReadSuccess = sc.ReadIntradayFileRecordForBarIndexAndSubIndex(sc.ArraySize - 1, SubIndex,
IntradayRecord, IntradayFileLockAction);

            if (ReadSuccess)
            {
                TotalVolume += IntradayRecord.TotalVolume;
                ++SubIndex;
            }
        }
    }
}

```

```

        sc.ReadIntradayFileRecordForBarIndexAndSubIndex(-1, -1, IntradayRecord, IFLA_RELEASE_AFTER_READ);

        SCString TotalVolumeString;
        TotalVolumeString.Format("Total volume: %u", TotalVolume);

        sc.AddMessageToLog(TotalVolumeString, 0);
    }
}

/*=====*/
SCSFExport scsf_ChangeStudyInputExample(SCStudyInterfaceRef sc)
{
    // Configuration
    if (sc.SetDefaults)
    {
        sc.GraphName = "Change Study Input Example";
        sc.GraphRegion = 0; // zero based region number

        sc.AutoLoop = 0;

        return;
    }

    // Data processing
    int IntegerInput = 0;
    sc.GetChartStudyInputInt(sc.ChartNumber, 1, 36, IntegerInput);

    double DoubleInput = 0;
    sc.GetChartStudyInputFloat(sc.ChartNumber, 1, 36, DoubleInput);

    if (DoubleInput != 20)
    {
        sc.SetChartStudyInputFloat(sc.ChartNumber, 1, 36, 20);
        sc.RecalculateChart(sc.ChartNumber);
    }

    //SCString InputString;
    //sc.GetChartStudyInputString(sc.ChartNumber, 1, 90, InputString);
}

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

        sc.GraphName = "GetStudyPeakValleyLine Example";

        sc.AutoLoop = 0;

        sc.Input[0].SetStudyID(1);
        sc.Input[0].Name = "Volume by Price Study";

        return;
    }

    // Do data processing
    float PeakValleyLinePrice= 0;

```

```

int PeakValleyType = 0;
int StartIndex = 0;
int PeakValleyExtensionChartColumnEndIndex = 0;

// get the first Peak/Valley line from the last volume by price profile
if (sc.GetStudyPeakValleyLine(sc.ChartNumber, sc.Input[0].GetStudyID(), PeakValleyLinePrice, PeakValleyType,
StartIndex, PeakValleyExtensionChartColumnEndIndex, -1, 0))
{
    // Peak/Valley line found
    int Test = 1;
}
}

/*=====
This study periodically recalculates the studies on the chart.
-----*/
SCSFExport scsf_PeriodicChartRecalculation(SCStudyInterfaceRef sc)
{
    SCInputRef Input_RecalculationTimeInSeconds = sc.Input[0];
    SCInputRef Input_UpdateAlways = sc.Input[1];

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

        sc.GraphName = "Chart Recalculation - Periodic";

        sc.AutoLoop = 0;

        sc.GraphRegion = 0;

        Input_RecalculationTimeInSeconds.Name = "Recalculation Time in Seconds";
        Input_RecalculationTimeInSeconds.SetInt(SECONDS_PER_HOUR);// 1 hour
        Input_RecalculationTimeInSeconds.SetIntLimits(1, SECONDS_PER_DAY);

        Input_UpdateAlways.Name = "Update Always";
        Input_UpdateAlways.SetYesNo(false);

        return;
    }

    sc.UpdateAlways = Input_UpdateAlways.GetYesNo();

    SCDateTime& r_LastCalculationDateTime = sc.GetPersistentSCDateTime(1);

    const SCDateTimeMS CurrentDateTime = sc.GetCurrentDateTime();
    if ((CurrentDateTime - r_LastCalculationDateTime).GetAbsoluteValue()
> SCDateTime::SECONDS(Input_RecalculationTimeInSeconds.GetInt()))
    {
        r_LastCalculationDateTime = CurrentDateTime;
        sc.FlagFullRecalculate = 1;
        sc.AddMessageToLog("Performing full recalculation of chart", 0);
    }

}

/*=====
SCSFExport scsf_HLVolatility(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_HLDiff = sc.Subgraph[0];
    SCSubgraphRef Subgraph_AvgHLDiff = sc.Subgraph[1];
    SCSubgraphRef Subgraph_AvgClose = sc.Subgraph[2];

```

```

SCSubgraphRef Subgraph_HLVolatility = sc.Subgraph[3];

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

if (sc.SetDefaults)
{
    sc.GraphName = "High Low Volatility";

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

    Subgraph_HLDiff.Name = "HL Difference";
    Subgraph_HLDiff.DrawStyle = DRAWSTYLE_IGNORE;

    Subgraph_AvgHLDiff.Name = "Average HL Difference";
    Subgraph_AvgHLDiff.DrawStyle = DRAWSTYLE_IGNORE;

    Subgraph_AvgClose.Name = "Average Close";
    Subgraph_AvgClose.DrawStyle = DRAWSTYLE_IGNORE;

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

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

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

    return;
}

Subgraph_HLDiff[sc.Index] = sc.GetHighest(sc.BaseDataIn[SC_HIGH], Input_Length.GetInt()) -
sc.GetLowest(sc.BaseDataIn[SC_LOW], Input_Length.GetInt());

sc.MovingAverage(Subgraph_HLDiff, Subgraph_AvgHLDiff, Input_MAType.GetMovAvgType(), Input_Length.GetInt());
sc.MovingAverage(sc.BaseDataIn[SC_LAST], Subgraph_AvgClose, Input_MAType.GetMovAvgType(),
Input_Length.GetInt());

if (Subgraph_AvgClose[sc.Index] == 0.0f)
    Subgraph_HLVolatility[sc.Index] = 0.0f;
else
    Subgraph_HLVolatility[sc.Index] = 100.0f*Subgraph_AvgHLDiff[sc.Index] / Subgraph_AvgClose[sc.Index];
}

/*=====*/
SCSFExport scsf_GetVolumeAtPriceDataForStudyProfileExample(SCStudyInterfaceRef sc)
{
    SCInputRef Input_StudyReference = sc.Input[0];

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

        sc.GraphName = "GetVolumeAtPriceDataForStudyProfile Example";

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

        Input_StudyReference.Name = "Profile Study Reference";
        Input_StudyReference.SetStudyID(1);
    }
}

```

```

    return;
}

int PricesCount = sc.GetNumPriceLevelsForStudyProfile(Input_StudyReference.GetStudyID(), 0);

for (int PriceIndex = 0; PriceIndex < PricesCount; PriceIndex++)
{
    s_VolumeAtPriceV2 VolumeAtPrice;

    int Result = sc.GetVolumeAtPriceDataForStudyProfile
        (Input_StudyReference.GetStudyID()
        , 0
        , PriceIndex
        , VolumeAtPrice
        );

    int Volume = VolumeAtPrice.Volume;

}

}

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

    if (sc.SetDefaults)
    {
        sc.GraphName = "Rotation Factor";

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

        Subgraph_RotationFactor.Name = "RF";
        Subgraph_RotationFactor.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_RotationFactor.AutoColoring = AUTOCOLOR_POSNEG;
        Subgraph_RotationFactor.LineWidth = 2;
        Subgraph_RotationFactor.DrawZeros = 1;

        sc.DataStartIndex = 1;

        return;
    }

    if (sc.Index == 0)
        return;

    int RotationFactor = 0;

    if (sc.High[sc.Index] > sc.High[sc.Index - 1])
        RotationFactor++;

    if (sc.Low[sc.Index] > sc.Low[sc.Index - 1])
        RotationFactor++;

    if (sc.High[sc.Index] < sc.High[sc.Index - 1])
        RotationFactor--;

    if (sc.Low[sc.Index] < sc.Low[sc.Index - 1])
        RotationFactor--;

```



```

Subgraph_RotationFactor.Arrays[0][sc.Index] = static_cast<float>(RotationFactor);

if (sc.IsNewTradingDay(sc.Index))
    Subgraph_RotationFactor.Data[sc.Index] = 0;
else
    Subgraph_RotationFactor.Data[sc.Index] = Subgraph_RotationFactor.Arrays[0][sc.Index] +
Subgraph_RotationFactor.Data[sc.Index - 1];

}

/*=====*/
SCSFExport scsf_HurstBands(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_DisplacedPrice = sc.Subgraph[0];
    SCSubgraphRef Subgraph_MovingAverage = sc.Subgraph[1];
    SCSubgraphRef Subgraph_TopInnerBand = sc.Subgraph[2];
    SCSubgraphRef Subgraph_BottomInnerBand = sc.Subgraph[3];
    SCSubgraphRef Subgraph_TopOuterBand = sc.Subgraph[4];
    SCSubgraphRef Subgraph_BottomOuterBand = sc.Subgraph[5];
    SCSubgraphRef Subgraph_TopExtremeBand = sc.Subgraph[6];
    SCSubgraphRef Subgraph_BottomExtremeBand = sc.Subgraph[7];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_Length = sc.Input[1];
    SCInputRef Input_MovAvgType = sc.Input[2];
    SCInputRef Input_InnerMultiplier = sc.Input[3];
    SCInputRef Input_OuterMultiplier = sc.Input[4];
    SCInputRef Input_ExtremeMultiplier = sc.Input[5];

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

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

        Subgraph_DisplacedPrice.Name = "Displaced Price";
        Subgraph_DisplacedPrice.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_MovingAverage.Name = "Moving Average";
        Subgraph_MovingAverage.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_MovingAverage.PrimaryColor = RGB(128, 0, 128);
        Subgraph_MovingAverage.DrawZeros = true;

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

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

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

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

        Subgraph_TopExtremeBand.Name = "Top Extreme Band";

```

```

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

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

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

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

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

Input_InnerMultiplier.Name = "Inner Band Multiplier";
Input_InnerMultiplier.SetFloat(0.05f);

Input_OuterMultiplier.Name = "Outer Band Multiplier";
Input_OuterMultiplier.SetFloat(0.025f);

Input_ExtremeMultiplier.Name = "Extreme Band Multiplier";
Input_ExtremeMultiplier.SetFloat(0.0125f);

return;
}

int Displacement = Input_Length.GetInt() / 2 + 1;

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

Subgraph_DisplacedPrice[sc.Index] = sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - Displacement];

sc.MovingAverage(Subgraph_DisplacedPrice, Subgraph_MovingAverage, Input_MovAvgType.GetMovAvgType(),
Input_Length.GetInt());

Subgraph_TopInnerBand[sc.Index] = Subgraph_MovingAverage[sc.Index] + (Input_InnerMultiplier.GetFloat() / 100.0f) *
Subgraph_MovingAverage[sc.Index];

Subgraph_BottomInnerBand[sc.Index] = Subgraph_MovingAverage[sc.Index] - (Input_InnerMultiplier.GetFloat() /
100.0f) * Subgraph_MovingAverage[sc.Index];

Subgraph_TopOuterBand[sc.Index] = Subgraph_MovingAverage[sc.Index] + (Input_OuterMultiplier.GetFloat() / 100.0f)
* Subgraph_MovingAverage[sc.Index];

Subgraph_BottomOuterBand[sc.Index] = Subgraph_MovingAverage[sc.Index] - (Input_OuterMultiplier.GetFloat() /
100.0f) * Subgraph_MovingAverage[sc.Index];

Subgraph_TopExtremeBand[sc.Index] = Subgraph_MovingAverage[sc.Index] + (Input_ExtremeMultiplier.GetFloat() /
100.0f) * Subgraph_MovingAverage[sc.Index];

Subgraph_BottomExtremeBand[sc.Index] = Subgraph_MovingAverage[sc.Index] - (Input_ExtremeMultiplier.GetFloat() /
100.0f) * Subgraph_MovingAverage[sc.Index];
}

/*=====*/
SCSFExport scsf_GannHiLoActivator(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_HiAvg = sc.Subgraph[0];
    SCSubgraphRef Subgraph_LoAvg = sc.Subgraph[1];

```

```

SCSubgraphRef Subgraph_HiLo = sc.Subgraph[2];
SCSubgraphRef Subgraph_GannHiLoActivator = sc.Subgraph[3];

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

if (sc.SetDefaults)
{
    sc.GraphName = "Gann HiLo Activator";

    sc.GraphRegion = 0;
    sc.ValueFormat = 2;
    sc.AutoLoop = 1;

    Subgraph_HiAvg.Name = "Average High Price";
    Subgraph_HiAvg.DrawStyle = DRAWSTYLE_IGNORE;

    Subgraph_LoAvg.Name = "Average Low Price";
    Subgraph_LoAvg.DrawStyle = DRAWSTYLE_IGNORE;

    Subgraph_HiLo.Name = "HiLo";
    Subgraph_HiLo.DrawStyle = DRAWSTYLE_IGNORE;

    Subgraph_GannHiLoActivator.Name = "Gann HiLo Activator";
    Subgraph_GannHiLoActivator.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_GannHiLoActivator.PrimaryColor = RGB(0, 255, 0);
    Subgraph_GannHiLoActivator.SecondaryColor = RGB(255, 0, 0);
    Subgraph_GannHiLoActivator.SecondaryColorUsed = true;
    Subgraph_GannHiLoActivator.DrawZeros = true;

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

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

    return;
}

sc.DataStartIndex = Input_Length.GetInt();

sc.MovingAverage(sc.High, Subgraph_HiAvg, Input_MovAvgType.GetMovAvgType(), Input_Length.GetInt());
sc.MovingAverage(sc.Low, Subgraph_LoAvg, Input_MovAvgType.GetMovAvgType(), Input_Length.GetInt());

if (sc.Close[sc.Index] > Subgraph_HiAvg[sc.Index - 1])
    Subgraph_HiLo[sc.Index] = 1.0f;

else if (sc.Close[sc.Index] < Subgraph_LoAvg[sc.Index - 1])
    Subgraph_HiLo[sc.Index] = -1.0f;

else
    Subgraph_HiLo[sc.Index] = 0.0f;

if (Subgraph_HiLo[sc.Index] == 1.0f)
{
    Subgraph_GannHiLoActivator.DataColor[sc.Index] = Subgraph_GannHiLoActivator.PrimaryColor;
    Subgraph_GannHiLoActivator[sc.Index] = Subgraph_LoAvg[sc.Index - 1];
}

else if (Subgraph_HiLo[sc.Index] == -1.0f)
{
    Subgraph_GannHiLoActivator.DataColor[sc.Index] = Subgraph_GannHiLoActivator.SecondaryColor;
    Subgraph_GannHiLoActivator[sc.Index] = Subgraph_HiAvg[sc.Index - 1];
}

```

```

else
{
    Subgraph_GannHiLoActivator.DataColor[sc.Index] = Subgraph_GannHiLoActivator.DataColor[sc.Index - 1];
    Subgraph_GannHiLoActivator[sc.Index] = Subgraph_GannHiLoActivator[sc.Index - 1];
}
}

/*=====*/
SCSFExport scsf_RangeExpansionIndex(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_REI = sc.Subgraph[0];
    SCSubgraphRef Subgraph_ConditionalMultiplier = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Sum1 = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Sum2 = sc.Subgraph[3];
    SCSubgraphRef Subgraph_OverboughtLine = sc.Subgraph[4];
    SCSubgraphRef Subgraph_OversoldLine = sc.Subgraph[5];
    SCSubgraphRef Subgraph_ZeroLine = sc.Subgraph[6];
    SCSubgraphRef Subgraph_DurationExceededOverbought = sc.Subgraph[7];
    SCSubgraphRef Subgraph_DurationExceededOversold = sc.Subgraph[8];

    SCInputRef Input_Length = sc.Input[0];
    SCInputRef Input_LookbackLength1 = sc.Input[1];
    SCInputRef Input_LookbackLength2 = sc.Input[2];
    SCInputRef Input_LookbackLength3 = sc.Input[3];
    SCInputRef Input_DurationLength = sc.Input[4];
    SCInputRef Input_OffsetPercentage = sc.Input[5];
    SCInputRef Input_InputDataForHighComparison = sc.Input[6];
    SCInputRef Input_InputDataForLowComparison = sc.Input[7];
    SCInputRef Input_BasicOrAdvanced = sc.Input[8];
    SCInputRef Input_UseStrictInequalities = sc.Input[9];
    SCInputRef Input_OverboughtLineValue = sc.Input[10];
    SCInputRef Input_OversoldLineValue = sc.Input[11];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Range Expansion Index";

        sc.AutoLoop = 1;

        Subgraph_REI.Name = "Range Expansion Index";
        Subgraph_REI.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_REI.PrimaryColor = RGB(0, 0, 255);
        Subgraph_REI.DrawZeros = true;

        Subgraph_ConditionalMultiplier.Name = "Conditional Multiplier";
        Subgraph_ConditionalMultiplier.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_Sum1.Name = "Sum 1";
        Subgraph_Sum1.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_Sum2.Name = "Sum 2";
        Subgraph_Sum2.DrawStyle = DRAWSTYLE_IGNORE;

        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_ZeroLine.Name = "Zero Line";
        Subgraph_ZeroLine.DrawStyle = DRAWSTYLE_LINE;
    }
}

```

```

Subgraph_ZeroLine.PrimaryColor = RGB(255, 255, 255);
Subgraph_ZeroLine.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_Length.Name = "Length";
Input_Length.SetInt(8);
Input_Length.SetIntLimits(2, MAX_STUDY_LENGTH);

Input_LookbackLength1.Name = "Lookback Length 1";
Input_LookbackLength1.SetInt(2);
Input_LookbackLength1.SetIntLimits(1, Input_Length.GetInt());

Input_LookbackLength2.Name = "Lookback Length 2";
Input_LookbackLength2.SetInt(5);
Input_LookbackLength2.SetIntLimits(1, Input_Length.GetInt() - 1);

Input_LookbackLength3.Name = "Lookback Length 3";
Input_LookbackLength3.SetInt(7);
Input_LookbackLength3.SetIntLimits(1, Input_Length.GetInt() - 1);

Input_DurationLength.Name = "Duration Length";
Input_DurationLength.SetInt(6);
Input_DurationLength.SetIntLimits(1, Input_Length.GetInt() - 1);

Input_OffsetPercentage.Name = "Arrow Offset Percentage";
Input_OffsetPercentage.SetFloat(8.0f);

Input_InputDataForHighComparison.Name = "Input Data for Comparison with High";
Input_InputDataForHighComparison.SetInputDataIndex(SC_LOW);

Input_InputDataForLowComparison.Name = "Input Data for Comparison with Low";
Input_InputDataForLowComparison.SetInputDataIndex(SC_HIGH);

Input_BasicOrAdvanced.Name = "Basic or Advanced?";
Input_BasicOrAdvanced.SetCustomInputStrings
("Basic";Advanced");
Input_BasicOrAdvanced.SetCustomInputIndex(0);

Input_UseStrictInequalities.Name = "Use Strict Inequalities?";
Input_UseStrictInequalities.SetCustomInputStrings
("No";Yes");
Input_UseStrictInequalities.SetCustomInputIndex(0);

Input_OverboughtLineValue.Name = "Overbought Line Value";
Input_OverboughtLineValue.SetFloat(45.0f);

Input_OversoldLineValue.Name = "Oversold Line Value";
Input_OversoldLineValue.SetFloat(-45.0f);

return;
}

sc.DataStartIndex = Input_Length.GetInt() + max(Input_LookbackLength1.GetInt(),
max(Input_LookbackLength2.GetInt(), max(Input_LookbackLength3.GetInt(), Input_DurationLength.GetInt())));

```

```

// Evaluate logical conditions that determine the value of the Conditional Multiplier
bool Condition1 = ((sc.High[sc.Index] >= sc.BaseData[Input_InputDataForHighComparison.GetInputDataIndex()]
[sc.Index - Input_LookbackLength2.GetInt()]) ||
(sc.High[sc.Index] >= sc.BaseData[Input_InputDataForHighComparison.GetInputDataIndex()][sc.Index -
Input_LookbackLength2.GetInt() - 1])) &&
((sc.Low[sc.Index] <= sc.BaseData[Input_InputDataForLowComparison.GetInputDataIndex()][sc.Index -
Input_LookbackLength2.GetInt()]) ||
(sc.Low[sc.Index] <= sc.BaseData[Input_InputDataForLowComparison.GetInputDataIndex()][sc.Index -
Input_LookbackLength2.GetInt() - 1]));

bool Condition2 = ((sc.High[sc.Index - Input_LookbackLength1.GetInt()] >= sc.Close[sc.Index -
Input_LookbackLength3.GetInt()]) ||
(sc.High[sc.Index - Input_LookbackLength1.GetInt()] >= sc.Close[sc.Index - Input_LookbackLength3.GetInt() - 1]))
&&
((sc.Low[sc.Index - Input_LookbackLength1.GetInt()] <= sc.Close[sc.Index - Input_LookbackLength3.GetInt()]) ||
(sc.Low[sc.Index - Input_LookbackLength1.GetInt()] <= sc.Close[sc.Index - Input_LookbackLength3.GetInt() - 1]));

bool Condition3 = ((sc.High[sc.Index] > sc.BaseData[Input_InputDataForHighComparison.GetInputDataIndex()]
[sc.Index - Input_LookbackLength2.GetInt()]) ||
(sc.High[sc.Index] > sc.BaseData[Input_InputDataForHighComparison.GetInputDataIndex()][sc.Index -
Input_LookbackLength2.GetInt() - 1])) &&
((sc.Low[sc.Index] < sc.BaseData[Input_InputDataForLowComparison.GetInputDataIndex()][sc.Index -
Input_LookbackLength2.GetInt()]) ||
(sc.Low[sc.Index] < sc.BaseData[Input_InputDataForLowComparison.GetInputDataIndex()][sc.Index -
Input_LookbackLength2.GetInt() - 1]));

bool Condition4 = ((sc.High[sc.Index - Input_LookbackLength1.GetInt()] > sc.Close[sc.Index -
Input_LookbackLength3.GetInt()]) ||
(sc.High[sc.Index - Input_LookbackLength1.GetInt()] > sc.Close[sc.Index - Input_LookbackLength3.GetInt() - 1])) &&
((sc.Low[sc.Index - Input_LookbackLength1.GetInt()] < sc.Close[sc.Index - Input_LookbackLength3.GetInt()]) ||
(sc.Low[sc.Index - Input_LookbackLength1.GetInt()] < sc.Close[sc.Index - Input_LookbackLength3.GetInt() - 1]));

// Switch between Basic and Advanced versions of the REI.
float TempPrevConditionalMultiplier = Subgraph_ConditionalMultiplier[sc.Index - 1];

const int SelectedIndex1 = Input_BasicOrAdvanced.GetIndex();
switch (SelectedIndex1)
{
case 0:
{
if (Condition1 || Condition2)
Subgraph_ConditionalMultiplier[sc.Index] = 1.0f;
else
Subgraph_ConditionalMultiplier[sc.Index] = 0.0f;
}
break;

case 1:
{
const int SelectedIndex2 = Input_UseStrictInequalities.GetIndex();
switch (SelectedIndex2)
{
case 0:
{
if (Condition1 || Condition2)
Subgraph_ConditionalMultiplier[sc.Index] = 1.0f;
else
{
Subgraph_ConditionalMultiplier[sc.Index - 1] = 0.0f;
Subgraph_ConditionalMultiplier[sc.Index] = 0.0f;
}
}
}
break;
}

```

```

    case 1:
    {
        if (Condition3 || Condition4)
            Subgraph_ConditionalMultiplier[sc.Index] = 1.0f;
        else
        {
            Subgraph_ConditionalMultiplier[sc.Index - 1] = 0.0f;
            Subgraph_ConditionalMultiplier[sc.Index] = 0.0f;
        }
    }
    break;
}
break;
}

// Compute sums.
float Sum1 = 0;
float Sum2 = 0;

for (int Index = sc.Index - Input_Length.GetInt() + 1; Index <= sc.Index; Index++)
{
    Sum1 += Subgraph_ConditionalMultiplier[Index] * (sc.High[Index] - sc.High[Index - Input_LookbackLength1.GetInt()]
+ sc.Low[Index] - sc.Low[Index - Input_LookbackLength1.GetInt()]);

    Sum2 += fabs(sc.High[Index] - sc.High[Index - Input_LookbackLength1.GetInt()]) + fabs(sc.Low[Index] -
sc.Low[Index - Input_LookbackLength1.GetInt()]);
}

Subgraph_Sum1[sc.Index] = Sum1;

Subgraph_Sum2[sc.Index] = Sum2;

// Compute REI.
if (Sum2 != 0)
    Subgraph_REI[sc.Index] = 100.0f * Subgraph_Sum1[sc.Index] / Subgraph_Sum2[sc.Index];
else
    Subgraph_REI[sc.Index] = Subgraph_REI[sc.Index - 1];

// Recompute Sum1 and REI if previous value of Conditional Multiplier changes.
if (Subgraph_ConditionalMultiplier[sc.Index - 1] != TempPrevConditionalMultiplier)
{
    Subgraph_Sum1[sc.Index - 1] = Subgraph_Sum1[sc.Index - 1] - (sc.High[sc.Index - 1] - sc.High[sc.Index - 1 -
Input_LookbackLength1.GetInt()] + sc.Low[sc.Index - 1] - sc.Low[sc.Index - 1 - Input_LookbackLength1.GetInt()]);

    Subgraph_REI[sc.Index - 1] = 100.0f * Subgraph_Sum1[sc.Index - 1] / Subgraph_Sum2[sc.Index - 1];
}

Subgraph_DurationExceededOverbought[sc.Index] = 0.0f;
Subgraph_DurationExceededOversold[sc.Index] = 0.0f;

// Perform Duration Analysis
if (Subgraph_REI[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_REI[Index] > Input_OverboughtLineValue.GetFloat())
                OverboughtSum += 1;
        }
    }
}

```



```

    if (OverboughtSum == Input_DurationLength.GetInt())
    {
        Subgraph_DurationExceededOverbought[sc.Index] =
            Subgraph_REI[sc.Index] + (Input_OffsetPercentage.GetFloat() / 100.0f) *
            Subgraph_REI[sc.Index];
    }
}

int OverboughtIndex = 1;

while (Subgraph_REI[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_REI[sc.Index] +
        (Input_OffsetPercentage.GetFloat() / 100.0f) *
        Subgraph_REI[sc.Index];
}

if (Subgraph_REI[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_REI[Index] < Input_OversoldLineValue.GetFloat())
                OversoldSum += 1;
        }

        if (OversoldSum == Input_DurationLength.GetInt())
        {
            Subgraph_DurationExceededOversold[sc.Index] =
                Subgraph_REI[sc.Index] - (Input_OffsetPercentage.GetFloat() / 100.0f) *
                Subgraph_REI[sc.Index];
        }
    }

    int OversoldIndex = 1;

    while (Subgraph_REI[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_REI[sc.Index] +
            (Input_OffsetPercentage.GetFloat() / 100.0f) *
            Subgraph_REI[sc.Index];
    }
}

// Set horizontal lines.
Subgraph_OverboughtLine[sc.Index] = Input_OverboughtLineValue.GetFloat();
Subgraph_OversoldLine[sc.Index] = Input_OversoldLineValue.GetFloat();

```



```

    Subgraph_ZeroLine[sc.Index] = 0.0f;
}

/*=====*/
SCSFExport scsf_CumulativeSumOverNBars(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Sum = sc.Subgraph[0];
    SCInputRef Input_StudySubgraphReference = sc.Input[1];
    SCInputRef Input_Length = sc.Input[2];

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

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

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

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

        return;
    }

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

    sc.DataStartIndex = max(Input_Length.GetInt(),
    sc.GetStudyDataStartIndexUsingID(Input_StudySubgraphReference.GetStudyID()));

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

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

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

    for (int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize; BarIndex++)
    {
        Subgraph_Sum[BarIndex] = 0;

        for (int RefBackIndex = BarIndex; RefBackIndex > max(BarIndex - Input_Length.GetInt(), 0); RefBackIndex--)
        {

```

```

        Subgraph_Sum[BarIndex] += StudyArray[RefBackIndex];
    }
}

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

        sc.GraphName = "Bar Period Parameters Test";

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

        return;
    }

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

    if (BarPeriod.ChartDataType == INTRADAY_DATA && BarPeriod.IntradayChartBarPeriodType ==
IBPT_DAYS_MINS_SECS)
    {
        int SecondsPerBar = BarPeriod.IntradayChartBarPeriodParameter1;
    }

    SCString DebugString;
    DebugString.Format
("Bar Period type=%d, parameters={%d, %d, %d, %d}."
, BarPeriod.IntradayChartBarPeriodType
, BarPeriod.IntradayChartBarPeriodParameter1
, BarPeriod.IntradayChartBarPeriodParameter2
, BarPeriod.IntradayChartBarPeriodParameter3
, BarPeriod.IntradayChartBarPeriodParameter4
);

    sc.AddMessageToLog(DebugString, 0);

    if (BarPeriod.ChartDataType != INTRADAY_DATA
    || BarPeriod.IntradayChartBarPeriodType != IBPT_DAYS_MINS_SECS
    || BarPeriod.IntradayChartBarPeriodParameter1 != 300)
    {
        n_ACSIL::s_BarPeriod NewBarPeriod;
        NewBarPeriod.ChartDataType = INTRADAY_DATA;
        NewBarPeriod.IntradayChartBarPeriodType = IBPT_DAYS_MINS_SECS;
        NewBarPeriod.IntradayChartBarPeriodParameter1 = 300; // 300 seconds per bar which is 5 minutes

        //Set the bar period parameters. This will go into effect after the study function returns.
        sc.SetBarPeriodParameters(NewBarPeriod);
    }
}

/*=====*/
SCSFExport scsf_RateOfChangeOscillatorType1(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Temp1 = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Temp2 = sc.Subgraph[1];
    SCSubgraphRef Subgraph_ROC1 = 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];
```

```
SCInputRef Input_InputData = sc.Input[0];
SCInputRef Input_ROCLength = sc.Input[1];
SCInputRef Input_UseSmoothing = sc.Input[2];
SCInputRef Input_SmoothingLength = sc.Input[3];
SCInputRef Input_SmoothingMAType = sc.Input[4];
SCInputRef Input_UseMovingAverage = sc.Input[5];
SCInputRef Input_AverageROCMALength = sc.Input[6];
SCInputRef Input_AverageROCMAType = sc.Input[7];
SCInputRef Input_DurationLength = sc.Input[8];
SCInputRef Input_OffsetPercentage = sc.Input[9];
SCInputRef Input_OverboughtLineValue = sc.Input[10];
SCInputRef Input_OversoldLineValue = sc.Input[11];
```

```
if (sc.SetDefaults)
{
    sc.GraphName = "Rate of Change Oscillator Type I";

    sc.AutoLoop = 1;

    Subgraph_ROCI.Name = "ROC I";
    Subgraph_ROCI.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_ROCI.PrimaryColor = RGB(0, 0, 255);
    Subgraph_ROCI.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_InputData.Name = "Input Data";
    Input_InputData.SetInputDataIndex(SC_LAST);

    Input_ROCLength.Name = "ROC Length";
    Input_ROCLength.SetInt(12);
    Input_ROCLength.SetIntLimits(1, MAX_STUDY_LENGTH);

    Input_UseSmoothing.Name = "Use Smoothing?";
    Input_UseSmoothing.SetCustomInputStrings
("No";"Yes");
    Input_UseSmoothing.SetCustomInputIndex(0);

    Input_SmoothingLength.Name = "Smoothing Length";
    Input_SmoothingLength.SetInt(3);
    Input_SmoothingLength.SetIntLimits(1, Input_ROCLength.GetInt());
```

```

Input_SmoothingMAType.Name = "Smoothing MA Type";
Input_SmoothingMAType.SetMovAvgType(MOVAVGTYPE_SIMPLE);

Input_UseMovingAverage.Name = "Use Moving Average?";
Input_UseMovingAverage.SetCustomInputStrings
("No";"Yes");
Input_UseMovingAverage.SetCustomInputIndex(0);

Input_AverageROCMALength.Name = "Average ROC MA Length";
Input_AverageROCMALength.SetInt(12);
Input_AverageROCMALength.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_AverageROCMAType.Name = "Average ROC MA Type";
Input_AverageROCMAType.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(102.5f);

Input_OversoldLineValue.Name = "Oversold Line Value";
Input_OversoldLineValue.SetFloat(97.5f);

return;
}

if (Input_UseSmoothing.GetIndex() == 0 && Input_UseMovingAverage.GetIndex() == 0)
    sc.DataStartIndex = Input_ROCLength.GetInt();
else if (Input_UseSmoothing.GetIndex() == 1 && Input_UseMovingAverage.GetIndex() == 0)
    sc.DataStartIndex = Input_ROCLength.GetInt() + Input_SmoothingLength.GetInt() - 1;
else if (Input_UseSmoothing.GetIndex() == 0 && Input_UseMovingAverage.GetIndex() == 1)
    sc.DataStartIndex = Input_ROCLength.GetInt() + Input_AverageROCMALength.GetInt() - 1;
else
    sc.DataStartIndex = Input_ROCLength.GetInt() + Input_SmoothingLength.GetInt() +
Input_AverageROCMALength.GetInt() - 2;

// Compute ROC I.
if (sc.Index >= Input_ROCLength.GetInt())
{
    Subgraph_Temp1[sc.Index] = 100.0f * sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index] /
sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - Input_ROCLength.GetInt()];
}

if (Input_UseSmoothing.GetIndex() == 1)
    sc.MovingAverage(Subgraph_Temp1, Subgraph_Temp2, Input_SmoothingMAType.GetMovAvgType(),
Input_SmoothingLength.GetInt());

if (Input_UseSmoothing.GetIndex() == 0 && Input_UseMovingAverage.GetIndex() == 0)
    Subgraph_ROCI[sc.Index] = Subgraph_Temp1[sc.Index];
else if (Input_UseSmoothing.GetIndex() == 1 && Input_UseMovingAverage.GetIndex() == 0)
    Subgraph_ROCI[sc.Index] = Subgraph_Temp2[sc.Index];
else if (Input_UseSmoothing.GetIndex() == 0 && Input_UseMovingAverage.GetIndex() == 1)
    sc.MovingAverage(Subgraph_Temp1, Subgraph_ROCI, Input_AverageROCMAType.GetMovAvgType(),
Input_AverageROCMALength.GetInt());
else
    sc.MovingAverage(Subgraph_Temp2, Subgraph_ROCI, Input_AverageROCMAType.GetMovAvgType(),
Input_AverageROCMALength.GetInt());

Subgraph_DurationExceededOverbought[sc.Index] = 0.0f;

```

```

Subgraph_DurationExceededOversold[sc.Index] = 0.0f;

// Perform Duration Analysis.
if (Subgraph_ROCI[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_ROCI[Index] > Input_OverboughtLineValue.GetFloat())
                OverboughtSum += 1;
        }

        if (OverboughtSum == Input_DurationLength.GetInt())
        {
            Subgraph_DurationExceededOverbought[sc.Index] =
                Subgraph_ROCI[sc.Index] +
                (Input_OffsetPercentage.GetFloat() / 100.0f) *
                Subgraph_ROCI[sc.Index];
        }
    }

    int OverboughtIndex = 1;

    while (Subgraph_ROCI[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_ROCI[sc.Index] +
            (Input_OffsetPercentage.GetFloat() / 100.0f) *
            Subgraph_ROCI[sc.Index];
    }
}

if (Subgraph_ROCI[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_ROCI[Index] < Input_OversoldLineValue.GetFloat())
                OversoldSum += 1;
        }

        if (OversoldSum == Input_DurationLength.GetInt())
        {
            Subgraph_DurationExceededOversold[sc.Index] =
                Subgraph_ROCI[sc.Index] -
                (Input_OffsetPercentage.GetFloat() / 100.0f) *
                Subgraph_ROCI[sc.Index];
        }
    }

    int OversoldIndex = 1;

```

```

while (Subgraph_ROCI[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_ROCI[sc.Index] -
        (Input_OffsetPercentage.GetFloat() / 100.0f) *
        Subgraph_ROCI[sc.Index];
}
}

// Set horizontal lines.
Subgraph_OverboughtLine[sc.Index] = Input_OverboughtLineValue.GetFloat();
Subgraph_OversoldLine[sc.Index] = Input_OversoldLineValue.GetFloat();
}

/*=====*/
SCSFExport scsf_RateOfChangeOscillatorTypeII(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_ROCI = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Temp1 = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Temp2 = sc.Subgraph[2];
    SCSubgraphRef Subgraph_ROCII = sc.Subgraph[3];
    SCSubgraphRef Subgraph_OverboughtLine = sc.Subgraph[4];
    SCSubgraphRef Subgraph_OversoldLine = sc.Subgraph[5];
    SCSubgraphRef Subgraph_DurationExceededOverbought = sc.Subgraph[6];
    SCSubgraphRef Subgraph_DurationExceededOversold = sc.Subgraph[7];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_InputDataHigh = sc.Input[1];
    SCInputRef Input_InputDataLow = sc.Input[2];
    SCInputRef Input_ROCLength = sc.Input[3];
    SCInputRef Input_HighThreshold = sc.Input[4];
    SCInputRef Input_LowThreshold = sc.Input[5];
    SCInputRef Input_UseSmoothing = sc.Input[6];
    SCInputRef Input_SmoothingLength = sc.Input[7];
    SCInputRef Input_SmoothingMAType = sc.Input[8];
    SCInputRef Input_UseMovingAverage = sc.Input[9];
    SCInputRef Input_AverageROCMALength = sc.Input[10];
    SCInputRef Input_AverageROCMAType = sc.Input[11];
    SCInputRef Input_DurationLength = sc.Input[12];
    SCInputRef Input_OffsetPercentage = sc.Input[13];
    SCInputRef Input_OverboughtLineValue = sc.Input[14];
    SCInputRef Input_OversoldLineValue = sc.Input[15];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Rate of Change Oscillator Type II";

        sc.AutoLoop = 1;

        Subgraph_ROCI.Name = "ROC I";
        Subgraph_ROCI.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_ROCII.Name = "ROC II";
        Subgraph_ROCII.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_ROCII.PrimaryColor = RGB(0, 0, 255);
        Subgraph_ROCII.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_InputData.Name = "Input Data";
Input_InputData.SetInputDataIndex(SC_LAST);

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

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

Input_ROCLength.Name = "ROC Length";
Input_ROCLength.SetInt(12);
Input_ROCLength.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_HighThreshold.Name = "High Threshold";
Input_HighThreshold.SetFloat(100.0f);

Input_LowThreshold.Name = "Low Threshold";
Input_LowThreshold.SetFloat(100.0f);

Input_UseSmoothing.Name = "Use Smoothing?";
Input_UseSmoothing.SetCustomInputStrings
("No";Yes");
Input_UseSmoothing.SetCustomInputIndex(0);

Input_SmoothingLength.Name = "Smoothing Length";
Input_SmoothingLength.SetInt(3);
Input_SmoothingLength.SetIntLimits(1, Input_ROCLength.GetInt());

Input_SmoothingMAType.Name = "Smoothing MA Type";
Input_SmoothingMAType.SetMovAvgType(MOAVGTYPE_SIMPLE);

Input_UseMovingAverage.Name = "Use Moving Average?";
Input_UseMovingAverage.SetCustomInputStrings
("No";Yes");
Input_UseMovingAverage.SetCustomInputIndex(0);

Input_AverageROCMALength.Name = "Average ROC MA Length";
Input_AverageROCMALength.SetInt(12);
Input_AverageROCMALength.SetIntLimits(1, MAX_STUDY_LENGTH);

Input_AverageROCMAType.Name = "Average ROC MA Type";
Input_AverageROCMAType.SetMovAvgType(MOAVGTYPE_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(102.5f);

Input_OversoldLineValue.Name = "Oversold Line Value";
Input_OversoldLineValue.SetFloat(97.5f);

return;
}

if (Input_UseSmoothing.GetIndex() == 0 && Input_UseMovingAverage.GetIndex() == 0)
    sc.DataStartIndex = Input_ROCLength.GetInt();
else if (Input_UseSmoothing.GetIndex() == 1 && Input_UseMovingAverage.GetIndex() == 0)
    sc.DataStartIndex = Input_ROCLength.GetInt() + Input_SmoothingLength.GetInt() - 1;
else if (Input_UseSmoothing.GetIndex() == 0 && Input_UseMovingAverage.GetIndex() == 1)
    sc.DataStartIndex = Input_ROCLength.GetInt() + Input_AverageROCMALength.GetInt() - 1;
else
    sc.DataStartIndex = Input_ROCLength.GetInt() + Input_SmoothingLength.GetInt() +
Input_AverageROCMALength.GetInt() - 2;

// Compute ROC I.
if (sc.Index >= Input_ROCLength.GetInt())
{
    Subgraph_ROCI[sc.Index] = 100.0f * sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index] /
sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index - Input_ROCLength.GetInt()];
}

//Compute ROC II.
if (Subgraph_ROCI[sc.Index] > Input_HighThreshold.GetFloat())
    Subgraph_Temp1[sc.Index] = 100.0f * sc.BaseDataIn[Input_InputDataHigh.GetInputDataIndex()][sc.Index] /
sc.BaseDataIn[Input_InputDataHigh.GetInputDataIndex()][sc.Index - Input_ROCLength.GetInt()];
else if (Subgraph_ROCI[sc.Index] < Input_LowThreshold.GetFloat())
    Subgraph_Temp1[sc.Index] = 100.0f * sc.BaseDataIn[Input_InputDataLow.GetInputDataIndex()][sc.Index] /
sc.BaseDataIn[Input_InputDataLow.GetInputDataIndex()][sc.Index - Input_ROCLength.GetInt()];
else
    Subgraph_Temp1[sc.Index] = Subgraph_ROCI[sc.Index];

if (Input_UseSmoothing.GetIndex() == 1)
    sc.MovingAverage(Subgraph_Temp1, Subgraph_Temp2, Input_SmoothingMAType.GetMovAvgType(),
Input_SmoothingLength.GetInt());

if (Input_UseSmoothing.GetIndex() == 0 && Input_UseMovingAverage.GetIndex() == 0)
    Subgraph_ROCII[sc.Index] = Subgraph_Temp1[sc.Index];
else if (Input_UseSmoothing.GetIndex() == 1 && Input_UseMovingAverage.GetIndex() == 0)
    Subgraph_ROCII[sc.Index] = Subgraph_Temp2[sc.Index];
else if (Input_UseSmoothing.GetIndex() == 0 && Input_UseMovingAverage.GetIndex() == 1)
    sc.MovingAverage(Subgraph_Temp1, Subgraph_ROCII, Input_AverageROCMAType.GetMovAvgType(),
Input_AverageROCMALength.GetInt());
else
    sc.MovingAverage(Subgraph_Temp2, Subgraph_ROCII, Input_AverageROCMAType.GetMovAvgType(),
Input_AverageROCMALength.GetInt());

Subgraph_DurationExceededOverbought[sc.Index] = 0.0f;
Subgraph_DurationExceededOversold[sc.Index] = 0.0f;

// Perform Duration Analysis.
if (Subgraph_ROCII[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_ROCII[Index] > Input_OverboughtLineValue.GetFloat())
        OverboughtSum += 1;
}

if (OverboughtSum == Input_DurationLength.GetInt())
{
    Subgraph_DurationExceededOverbought[sc.Index] =
        Subgraph_ROCII[sc.Index] +
        (Input_OffsetPercentage.GetFloat() / 100.0f) *
        Subgraph_ROCII[sc.Index];
}
}

int OverboughtIndex = 1;

while (Subgraph_ROCII[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_ROCII[sc.Index] +
        (Input_OffsetPercentage.GetFloat() / 100.0f) *
        Subgraph_ROCII[sc.Index];
}
}

if (Subgraph_ROCII[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_ROCII[Index] < Input_OversoldLineValue.GetFloat())
                OversoldSum += 1;
        }

        if (OversoldSum == Input_DurationLength.GetInt())
        {
            Subgraph_DurationExceededOversold[sc.Index] =
                Subgraph_ROCII[sc.Index] -
                (Input_OffsetPercentage.GetFloat() / 100.0f) *
                Subgraph_ROCII[sc.Index];
        }
    }
}

int OversoldIndex = 1;

while (Subgraph_ROCII[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_ROCII[sc.Index] -

```

```

        (Input_OffsetPercentage.GetFloat() / 100.0f) *
        Subgraph_ROCII[sc.Index];
    }
}

// Set horizontal lines.
Subgraph_OverboughtLine[sc.Index] = Input_OverboughtLineValue.GetFloat();
Subgraph_OversoldLine[sc.Index] = Input_OversoldLineValue.GetFloat();
}

/*=====*/
SCSFExport scsf_GetStudyDataColorArrayFromChartUsingIDExample(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_ColorBar = sc.Subgraph[0];
    SCInputRef Input_ChartStudySubgraph = sc.Input[0];

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

        sc.GraphName = "GetStudyDataColorArrayFromChartUsingID Example";

        sc.AutoLoop = 0;

        sc.GraphRegion = 0;

        Subgraph_ColorBar.Name = "Bar";
        Subgraph_ColorBar.DrawStyle = DRAWSTYLE_COLOR_BAR;
        Subgraph_ColorBar.PrimaryColor = RGB(200, 200, 0);

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

        sc.CalculationPrecedence = LOW_PREC_LEVEL;

        return;
    }

    // Do data processing

    SCColorArray DataColorArray;
    sc.GetStudyDataColorArrayFromChartUsingID(Input_ChartStudySubgraph.GetChartNumber(),
    Input_ChartStudySubgraph.GetStudyID(), Input_ChartStudySubgraph.GetSubgraphIndex(), DataColorArray);

    if (DataColorArray.GetArraySize() > 0)
    {
        for (int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize; BarIndex++)
        {
            Subgraph_ColorBar[BarIndex] = 1;
            Subgraph_ColorBar.DataColor[BarIndex] = DataColorArray[BarIndex];

        }
    }
}

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

```

```

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

    sc.GraphName = "Apply Study Collection Example";

    sc.AutoLoop = 0;

    return;
}

//if (sc.IsFullRecalculation)
// return;

int& r_IsStudyCollectionApplied = sc.GetPersistentInt(1);

if (r_IsStudyCollectionApplied)
    return;

sc.ApplyStudyCollection(sc.ChartNumber, "Volume", 0);

r_IsStudyCollectionApplied = 1;
}

/*=====
-----*/
SCSFExport scsf_FileFunctionsExample(SCStudyInterfaceRef sc)
{
    SCSubgraphRef FirstSubgraph = sc.Subgraph[0];

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

        sc.GraphName = "File Functions Example";

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

        return;
    }

    if (sc.IsFullRecalculation && sc.UpdateStartIndex == 0)
    {
        int FileHandle = 0;
        SCString PathAndFileName = sc.DataFilesFolder();
        PathAndFileName += "Testing.txt";
        sc.OpenFile(PathAndFileName, n_ACSIL::FILE_MODE_OPEN_TO_APPEND, FileHandle);
        unsigned int BytesWritten;
        const char* TestLine = "Test Line\r\n";
        sc.WriteFile(FileHandle, TestLine, static_cast<int>(strlen(TestLine)), &BytesWritten);
        sc.CloseFile(FileHandle);
    }
}

/*=====
-----*/
SCSFExport scsf_VolumeByPriceResetButton(SCStudyInterfaceRef sc)
{

```

```

SCInputRef Input_VolumeByPriceStudy = sc.Input[0];
SCInputRef Input_ControlBarButtonNumberForEnableDisable = sc.Input[1];

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

    sc.GraphName = "Volume By Price Reset Button";

    sc.AutoLoop = 0; // manual looping
    sc.GraphRegion = 0;

    Input_VolumeByPriceStudy.Name = "Volume By Price Study Reference";
    Input_VolumeByPriceStudy.SetStudyID(0);

    Input_ControlBarButtonNumberForEnableDisable.Name = "ACS Control Bar Button # for Reset";
    Input_ControlBarButtonNumberForEnableDisable.SetInt(1);
    Input_ControlBarButtonNumberForEnableDisable.SetIntLimits(1, MAX_ACS_CONTROL_BAR_BUTTONS);

    return;
}

if (sc.MenuEventID == Input_ControlBarButtonNumberForEnableDisable.GetInt())
{
    const int ButtonState = (sc.PointerEventType == SC_ACS_BUTTON_ON) ? 1 : 0;
    if (ButtonState == 1)
    {
        sc.SetCustomStudyControlBarButtonEnable(Input_ControlBarButtonNumberForEnableDisable.GetInt(), 0);

        sc.SetChartStudyInputFloat(sc.ChartNumber, Input_VolumeByPriceStudy.GetStudyID(), 36,
sc.GetCurrentDateTime().GetDateAsSCDateTime().GetAsDouble());

        sc.SetChartStudyInputFloat(sc.ChartNumber, Input_VolumeByPriceStudy.GetStudyID(), 38,
sc.GetCurrentDateTime().GetTimeAsSCDateTime().GetAsDouble());

        // Set the 'Volume Graph Period Type' to 'From Start Date-Time To End'.
        sc.SetChartStudyInputInt(sc.ChartNumber, Input_VolumeByPriceStudy.GetStudyID(), 32, 5);

        sc.RecalculateChart(sc.ChartNumber);
    }
}

}

/*=====*/
SCSFExport scsf_MESAAdaptiveMovingAverage(SCStudyGraphRef sc)
{
    SCSubgraphRef Subgraph_MAMA = sc.Subgraph[0];
    SCSubgraphRef Subgraph_FAMA = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Smooth = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Detrender = sc.Subgraph[3];
    SCSubgraphRef Subgraph_Q1 = sc.Subgraph[4];
    SCSubgraphRef Subgraph_I1 = sc.Subgraph[5];
    SCSubgraphRef Subgraph_I = sc.Subgraph[6];
    SCSubgraphRef Subgraph_IQ = sc.Subgraph[7];
    SCSubgraphRef Subgraph_I2 = sc.Subgraph[8];
    SCSubgraphRef Subgraph_Q2 = sc.Subgraph[9];
    SCSubgraphRef Subgraph_Re = sc.Subgraph[10];
    SCSubgraphRef Subgraph_Im = sc.Subgraph[11];
    SCSubgraphRef Subgraph_Period = sc.Subgraph[12];
    SCSubgraphRef Subgraph_Phase = sc.Subgraph[13];

```

```

SCSubgraphRef Subgraph_SmoothPeriod = sc.Subgraph[14];

#define GREEN RGB(0,255,0)
#define RED RGB(255,0,0)
#define BLACK RGB(0,0,1)
#define WHITE RGB(255,255,255)
#define PURPLE RGB(255,0,255)
#define GREY RGB(192,192,192)
#define BLUE RGB(0,128,255)
#define ORANGE RGB(255, 127, 0)

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

    sc.GraphName = "MESA Adaptive Moving Average";
    sc.StudyDescription = "MAMA";
    //sg.FreeDLL = 0;

    sc.AutoLoop = 1; // true

    sc.GraphRegion = 0;

    // Set the name of the first subgraph
    sc.Subgraph[0].Name = "MAMA";
    sc.Subgraph[0].PrimaryColor = RED;
    sc.Subgraph[0].DrawStyle = DRAWSTYLE_LINE;
    sc.Subgraph[0].LineWidth = 2;

    sc.Subgraph[1].Name = "FAMA";
    sc.Subgraph[1].PrimaryColor = GREEN;
    sc.Subgraph[1].DrawStyle = DRAWSTYLE_LINE;
    sc.Subgraph[1].LineWidth = 2;

    sc.Subgraph[14].Name = "Period";
    sc.Subgraph[14].PrimaryColor = GREEN;
    sc.Subgraph[14].DrawStyle = DRAWSTYLE_IGNORE;
    sc.Subgraph[14].LineWidth = 2;

    sc.Input[0].Name = "Input Data";
    sc.Input[0].SetInputDataIndex(SC_HL);

    sc.Input[1].Name = "Fast Limit";
    sc.Input[1].SetFloat(0.5);

    sc.Input[2].Name = "Slow Limit";
    sc.Input[2].SetFloat(0.05f);

    return;
}

int BarIndex;
float FastLimit = sc.Input[1].FloatValue;
float SlowLimit = sc.Input[2].FloatValue;
float alpha, DeltaPhase;

SCFloatArrayRef Price = sc.BaseDataIn[sc.Input[0].GetInputDataIndex()];

sc.DataStartIndex = 50;
BarIndex = sc.CurrentIndex;

// smooth
Subgraph_Smooth[BarIndex] = (4 * Price[BarIndex] + 3 * Price[BarIndex - 1] + 2 * Price[BarIndex - 2] + Price[BarIndex - 3]) / 10;

```

```

// detrender
Subgraph_Detrender[BarIndex] = (0.0962f*Subgraph_Smooth[BarIndex] + 0.5769f*Subgraph_Smooth[BarIndex - 2] -
0.5769f*Subgraph_Smooth[BarIndex - 4] - 0.0962f*Subgraph_Smooth[BarIndex - 6])*(0.075f*Subgraph_Period[BarIndex -
1] + 0.54f);

// compute InPhase and Quadrature components
Subgraph_Q1[BarIndex] = (0.0962f*Subgraph_Detrender[BarIndex] + 0.5769f*Subgraph_Detrender[BarIndex - 2] -
0.5769f*Subgraph_Detrender[BarIndex - 4] - 0.0962f*Subgraph_Detrender[BarIndex - 6])*
(0.075f*Subgraph_Period[BarIndex - 1] + 0.54f);
Subgraph_I1[BarIndex] = Subgraph_Detrender[BarIndex - 3];

// Advance the phase of I1 and Q1 by 90 degrees
Subgraph_jI[BarIndex] = (0.0962f*Subgraph_I1[BarIndex] + 0.5769f*Subgraph_I1[BarIndex - 2] -
0.5769f*Subgraph_I1[BarIndex - 4] - 0.0962f*Subgraph_I1[BarIndex - 6])*(0.075f*Subgraph_Period[BarIndex - 1] + 0.54f);
Subgraph_jQ[BarIndex] = (0.0962f*Subgraph_Q1[BarIndex] + 0.5769f*Subgraph_Q1[BarIndex - 2] -
0.5769f*Subgraph_Q1[BarIndex - 4] - 0.0962f*Subgraph_Q1[BarIndex - 6])*(0.075f*Subgraph_Period[BarIndex - 1] +
0.54f);

// Phasor addition for 3 bar averaging
Subgraph_I2[BarIndex] = Subgraph_I1[BarIndex] - Subgraph_jQ[BarIndex];
Subgraph_Q2[BarIndex] = Subgraph_Q1[BarIndex] + Subgraph_jI[BarIndex];

// Smooth the I and Q components before applying the discriminator
Subgraph_I2[BarIndex] = 0.2f*Subgraph_I2[BarIndex] + 0.8f*Subgraph_I2[BarIndex - 1];
Subgraph_Q2[BarIndex] = 0.2f*Subgraph_Q2[BarIndex] + 0.8f*Subgraph_Q2[BarIndex - 1];

// Homodyne Discriminator
Subgraph_Re[BarIndex] = Subgraph_I2[BarIndex] * Subgraph_I2[BarIndex - 1] + Subgraph_Q2[BarIndex] *
Subgraph_Q2[BarIndex - 1];
Subgraph_Im[BarIndex] = Subgraph_I2[BarIndex] * Subgraph_Q2[BarIndex - 1] - Subgraph_Q2[BarIndex] *
Subgraph_I2[BarIndex - 1];
Subgraph_Re[BarIndex] = 0.2f*Subgraph_Re[BarIndex] + 0.8f*Subgraph_Re[BarIndex - 1];
Subgraph_Im[BarIndex] = 0.2f*Subgraph_Im[BarIndex] + 0.8f*Subgraph_Im[BarIndex - 1];

if (Subgraph_Im[BarIndex] != 0.0 && Subgraph_Re[BarIndex] != 0.0)
    Subgraph_Period[BarIndex] = 360 / (57.3f*atan(Subgraph_Im[BarIndex] / Subgraph_Re[BarIndex]));
if (Subgraph_Period[BarIndex] > 1.5f*Subgraph_Period[BarIndex - 1])
    Subgraph_Period[BarIndex] = 1.5f*Subgraph_Period[BarIndex - 1];
if (Subgraph_Period[BarIndex] < 0.67f*Subgraph_Period[BarIndex - 1])
    Subgraph_Period[BarIndex] = 0.67f*Subgraph_Period[BarIndex - 1];
if (Subgraph_Period[BarIndex] < 6)
    Subgraph_Period[BarIndex] = 6;
if (Subgraph_Period[BarIndex] > 50)
    Subgraph_Period[BarIndex] = 50;
Subgraph_Period[BarIndex] = 0.2f*Subgraph_Period[BarIndex] + 0.8f*Subgraph_Period[BarIndex - 1];
Subgraph_SmoothPeriod[BarIndex] = 0.33f*Subgraph_Period[BarIndex] + 0.67f*Subgraph_SmoothPeriod[BarIndex -
1];

if (Subgraph_I1[BarIndex] != 0)
    Subgraph_Phase[BarIndex] = 57.3f*atan(Subgraph_Q1[BarIndex] / Subgraph_I1[BarIndex]);

DeltaPhase = Subgraph_Phase[BarIndex - 1] - Subgraph_Phase[BarIndex];

if (DeltaPhase < 1) DeltaPhase = 1;

alpha = FastLimit / DeltaPhase;
if (alpha < SlowLimit) alpha = SlowLimit;

Subgraph_MAMA[BarIndex] = alpha * Price[BarIndex] + (1 - alpha)*Subgraph_MAMA[BarIndex - 1];
Subgraph_FAMA[BarIndex] = 0.5f*alpha*Subgraph_MAMA[BarIndex] + (1 - 0.5f*alpha)*Subgraph_FAMA[BarIndex -
1];
}

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

```

```

{
    if (sc.SetDefaults)
    {
        sc.GraphName = "Fibonacci Bands";
        sc.GraphRegion = 0;
        sc.Subgraph[0].Name = "MA";
        sc.Subgraph[1].Name = "U1";
        sc.Subgraph[2].Name = "L1";
        sc.Subgraph[3].Name = "U2";
        sc.Subgraph[4].Name = "L2";
        sc.Subgraph[4].DrawStyle = DRAWSTYLE_LINE;
        sc.Subgraph[5].Name = "U3";
        sc.Subgraph[5].DrawStyle = DRAWSTYLE_LINE;
        sc.Subgraph[6].Name = "L3";
        sc.Subgraph[6].DrawStyle = DRAWSTYLE_LINE;

        sc.Input[0].Name = "Study And Subgraph Reference";
        sc.Input[0].SetStudySubgraphValues(0, SC_LAST);

        sc.Input[2].Name = "Moving Average Type";
        sc.Input[2].SetMovAvgType(MOVAVGTYPE_EXPONENTIAL);

        sc.Input[3].Name = "True Range Moving Average Type";
        sc.Input[3].SetMovAvgType(MOVAVGTYPE_EXPONENTIAL);

        sc.Input[4].Name = "Moving Average Length";
        sc.Input[4].SetInt(8);

        sc.Input[5].Name = "ATR Moving Average Length";
        sc.Input[5].SetInt(14);

        sc.Input[6].Name = "Fib 1";
        sc.Input[6].SetFloat(0.382f);

        sc.Input[7].Name = "Fib 2";
        sc.Input[7].SetFloat(0.618f);

        sc.Input[8].Name = "Fib 3";
        sc.Input[8].SetFloat(1.0f);

        sc.CalculationPrecedence = LOW_PREC_LEVEL;
        sc.AutoLoop = 0;
        return;
    }
}

SCFloatArray SubgraphArray;
sc.GetStudyArrayUsingID(sc.Input[0].GetStudyID(), sc.Input[0].GetSubgraphIndex(), SubgraphArray);

for (int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize; ++BarIndex)
{
    sc.MovingAverage(SubgraphArray, sc.Subgraph[0], sc.Input[2].GetMovAvgType(), BarIndex, sc.Input[4].GetInt());
    sc.ATR(sc.BaseDataIn, sc.Subgraph[8], BarIndex, sc.Input[5].GetInt(), sc.Input[3].GetMovAvgType());

    float Gap = sc.Subgraph[8][BarIndex];
    float t1 = sc.Subgraph[0][BarIndex] + sc.Input[6].FloatValue*Gap;
    float b1 = sc.Subgraph[0][BarIndex] - sc.Input[6].FloatValue*Gap;
    float t2 = sc.Subgraph[0][BarIndex] + sc.Input[7].FloatValue*Gap;
    float b2 = sc.Subgraph[0][BarIndex] - sc.Input[7].FloatValue*Gap;
    float t3 = sc.Subgraph[0][BarIndex] + sc.Input[8].FloatValue*Gap;
    float b3 = sc.Subgraph[0][BarIndex] - sc.Input[8].FloatValue*Gap;

    sc.Subgraph[1][BarIndex] = t1;
    sc.Subgraph[2][BarIndex] = b1;
    sc.Subgraph[3][BarIndex] = t2;
    sc.Subgraph[4][BarIndex] = b2;

```

```

        sc.Subgraph[5][BarIndex] = t3;
        sc.Subgraph[6][BarIndex] = b3;
    }
}

/*=====
-----*/
SCSFExport scsf_TradeQuantityGraph(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_TradeQuantity = sc.Subgraph[0];

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

        sc.GraphName = "Trade Quantity Graph";

        sc.AutoLoop = 0;

        sc.GraphRegion = 1;
        sc.UpdateAlways = 1;

        Subgraph_TradeQuantity.Name = "Quantity";
        Subgraph_TradeQuantity.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_TradeQuantity.LineWidth = 2;
        Subgraph_TradeQuantity.PrimaryColor = RGB(200, 200, 0);
        return;
    }

    // Do data processing
    for (int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize; BarIndex++)
    {
        Subgraph_TradeQuantity.Data[BarIndex] = static_cast<float>(sc.TradeWindowOrderQuantity);
    }
}

/*=====
-----*/
SCSFExport scsf_RolloverDateDisplay(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Clock = sc.Subgraph[0];

    SCInputRef Input_HorizontalPosition = sc.Input[0];
    SCInputRef Input_VerticalPosition = sc.Input[1];
    SCInputRef Input_DrawAboveMainPriceGraph = sc.Input[4];
    SCInputRef Input_UseBoldFont = sc.Input[5];
    SCInputRef Input_TransparentLabelBackground = sc.Input[7];

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

        sc.GraphName = "Rollover Date Display";

        sc.AutoLoop = 0; //Manual looping
        sc.GraphRegion = 0;
        sc.ValueFormat = 0;

        Subgraph_Clock.Name = "Date";
        Subgraph_Clock.LineWidth = 10;
    }
}

```



```

Subgraph_Clock.DrawStyle = DRAWSTYLE_CUSTOM_TEXT;
Subgraph_Clock.PrimaryColor = RGB(0, 200, 0);
Subgraph_Clock.SecondaryColor = RGB(0, 0, 0);
Subgraph_Clock.SecondaryColorUsed = true;
Subgraph_Clock.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_UseBoldFont.Name = "Use Bold Font";
Input_UseBoldFont.SetYesNo(true);

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

return;
}

SCString DateString ("Rollover: ");
DateString += sc.DateToString(sc.ContractRolloverDate);

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

int DrawAboveMainPriceGraph = Input_DrawAboveMainPriceGraph.GetYesNo();

int TransparentLabelBackground = Input_TransparentLabelBackground.GetYesNo();
int UseBoldFont = Input_UseBoldFont.GetYesNo();

sc.AddAndManageSingleTextUserDrawnDrawingForStudy(sc, 0, HorizontalPosition, VerticalPosition, Subgraph_Clock,
TransparentLabelBackground, DateString, DrawAboveMainPriceGraph, 0, UseBoldFont);
}

/*=====
-----*/
SCSFExport scsf_VolumePointOfControlForBars(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_VPOC = sc.Subgraph[0];

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

        sc.GraphName = "Volume Point of Control for Bars";

        sc.AutoLoop = 0;
        sc.MaintainVolumeAtPriceData = true;

        sc.GraphRegion = 0;

```

```

Subgraph_VPOC.Name = "VPOC";
Subgraph_VPOC.DrawStyle = DRAWSTYLE_SQUARE_OFFSET_LEFT_FOR_CANDLESTICK;
Subgraph_VPOC.LineWidth = 8;
Subgraph_VPOC.PrimaryColor = RGB(255, 128, 0);
return;
}

// Do data processing
for (int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize; BarIndex++)
{
    s_VolumeAtPriceV2 VolumeAtPrice;
    sc.GetPointOfControlPriceVolumeForBar(BarIndex, VolumeAtPrice);

    if (VolumeAtPrice.PriceInTicks != 0)
    {
        Subgraph_VPOC.Data[BarIndex]
            = static_cast<float>
              ( sc.TicksToPriceValue(VolumeAtPrice.PriceInTicks)
              );
    }
}

}

/*=====
-----*/
SCSFExport scsf_ValueAreaForBars(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_ValueAreaHigh = sc.Subgraph[0];
    SCSubgraphRef Subgraph_ValueAreaLow = sc.Subgraph[1];

    SCInputRef Input_ValueAreaPercentage = sc.Input[0];

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

        sc.GraphName = "Volume Value Area for Bars";

        sc.AutoLoop = 0;
        sc.MaintainVolumeAtPriceData = true;

        sc.GraphRegion = 0;

        Subgraph_ValueAreaHigh.Name = "VAH";
        Subgraph_ValueAreaHigh.DrawStyle = DRAWSTYLE_SQUARE_OFFSET_LEFT_FOR_CANDLESTICK;
        Subgraph_ValueAreaHigh.LineWidth = 8;
        Subgraph_ValueAreaHigh.PrimaryColor = RGB(255, 128, 0);

        Subgraph_ValueAreaLow.Name = "VAL";
        Subgraph_ValueAreaLow.DrawStyle = DRAWSTYLE_SQUARE_OFFSET_LEFT_FOR_CANDLESTICK;
        Subgraph_ValueAreaLow.LineWidth = 8;
        Subgraph_ValueAreaLow.PrimaryColor = RGB(255, 0, 128);

        Input_ValueAreaPercentage.Name = "Value Area Percentage";
        Input_ValueAreaPercentage.SetFloat(70.0f);

        return;
    }

    // Do data processing

```

```

for (int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize; BarIndex++)
{
    double PointOfControl = 0.0f;
    double ValueAreaHigh = 0.0f;
    double ValueAreaLow = 0.0f;

    sc.GetPointOfControlAndValueAreaPricesForBar(BarIndex, PointOfControl, ValueAreaHigh, ValueAreaLow,
Input_ValueAreaPercentage.GetFloat());

    Subgraph_ValueAreaHigh.Data[BarIndex] = static_cast <float> (ValueAreaHigh);
    Subgraph_ValueAreaLow.Data[BarIndex] = static_cast <float> (ValueAreaLow);
}
}

```

```

/*=====

```

```

-----*/
SCSFExport scsf_ElderImpulse(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_MovAvg = sc.Subgraph[0];
    SCSubgraphRef Subgraph_FastMovAvg = sc.Subgraph[1];
    SCSubgraphRef Subgraph_SlowMovAvg = sc.Subgraph[2];
    SCSubgraphRef Subgraph_MACD = sc.Subgraph[3];
    SCSubgraphRef Subgraph_MovAvgOfMACD = sc.Subgraph[4];
    SCSubgraphRef Subgraph_MACDDiff = sc.Subgraph[5];
    SCSubgraphRef Subgraph_ColorBar = sc.Subgraph[6];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_MovAvgLength = sc.Input[1];
    SCInputRef Input_FastLength = sc.Input[2];
    SCInputRef Input_SlowLength = sc.Input[3];
    SCInputRef Input_MACDLength = sc.Input[4];
    SCInputRef Input_MovingAverageType = sc.Input[5];
    SCInputRef Input_MACDMovingAverageType = sc.Input[6];
    SCInputRef Input_Color1 = sc.Input[7];
    SCInputRef Input_Color2 = sc.Input[8];
    SCInputRef Input_Color3 = sc.Input[9];

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

        sc.ValueFormat = VALUEFORMAT_INHERITED;
        sc.GraphRegion = 0;
        sc.AutoLoop = 1;
        sc.GraphDrawType = GDT_CUSTOM;

        Subgraph_MovAvg.Name = "Moving Average of Input Data";
        Subgraph_MovAvg.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_FastMovAvg.Name = "Fast Moving Average";
        Subgraph_FastMovAvg.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_SlowMovAvg.Name = "Slow Moving Average";
        Subgraph_SlowMovAvg.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_MACD.Name = "MACD";
        Subgraph_MACD.DrawStyle = DRAWSTYLE_IGNORE;

        Subgraph_MovAvgOfMACD.Name = "MA of MACD";
        Subgraph_MovAvgOfMACD.DrawStyle = DRAWSTYLE_IGNORE;
    }
}

```

```

Subgraph_MACDDiff.Name = "MACD Diff";
Subgraph_MACDDiff.DrawStyle = DRAWSTYLE_IGNORE;

Subgraph_ColorBar.Name = "Color Bar";
Subgraph_ColorBar.DrawStyle = DRAWSTYLE_COLORBAR;
Subgraph_ColorBar.DrawZeros = false;

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

Input_MovAvgLength.Name = "Moving Average Length";
Input_MovAvgLength.SetInt(13);
Input_MovAvgLength.SetIntLimits(1, MAX_STUDY_LENGTH);

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

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

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

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

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

Input_Color1.Name = "Color 1";
Input_Color1.SetColor(BLUE);

Input_Color2.Name = "Color 2";
Input_Color2.SetColor(GREEN);

Input_Color3.Name = "Color 3";
Input_Color3.SetColor(RED);

return;
}

sc.MovingAverage(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_MovAvg,
Input_MovingAverageType.GetMovAvgType(), Input_MovAvgLength.GetInt());

sc.MovingAverage(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_FastMovAvg,
Input_MACDMovingAverageType.GetMovAvgType(), Input_FastLength.GetInt());
sc.MovingAverage(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_SlowMovAvg,
Input_MACDMovingAverageType.GetMovAvgType(), Input_SlowLength.GetInt());

Subgraph_MACD[sc.Index] = Subgraph_FastMovAvg[sc.Index] - Subgraph_SlowMovAvg[sc.Index];

sc.MovingAverage(Subgraph_MACD, Subgraph_MovAvgOfMACD,
Input_MACDMovingAverageType.GetMovAvgType(), Input_MACDLength.GetInt());

Subgraph_MACDDiff[sc.Index] = Subgraph_MACD[sc.Index] - Subgraph_MovAvgOfMACD[sc.Index];

Subgraph_ColorBar[sc.Index] = 0;

if (Subgraph_MovAvg[sc.Index] > Subgraph_MovAvg[sc.Index - 1] && Subgraph_MACDDiff[sc.Index] >
Subgraph_MACDDiff[sc.Index - 1])
{
    Subgraph_ColorBar[sc.Index] = 1;
}

```

```

        Subgraph_ColorBar.DataColor[sc.Index] = Input_Color2.GetColor();
    }
    else if (Subgraph_MovAvg[sc.Index] < Subgraph_MovAvg[sc.Index - 1] && Subgraph_MACDDiff[sc.Index] <
Subgraph_MACDDiff[sc.Index - 1])
    {
        Subgraph_ColorBar[sc.Index] = 1;
        Subgraph_ColorBar.DataColor[sc.Index] = Input_Color3.GetColor();
    }
    else
    {
        Subgraph_ColorBar[sc.Index] = 1;
        Subgraph_ColorBar.DataColor[sc.Index] = Input_Color1.GetColor();
    }
}

```

```

/*=====*/

```

```

SCSFExport scsf_OneTimeInitializationExample(SCStudyInterfaceRef sc)
{

```

```

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

        sc.GraphName = "One Time Initialization Example";

        sc.AutoLoop = 0;

        return;
    }

```

```

    int& r_IsInitialized = sc.GetPersistentInt(1);

```

```

    if (!r_IsInitialized)
    {
        //Perform initialization here

        r_IsInitialized = 1;
    }

```

```

    if (sc.LastCallToFunction)
    {
        if (r_IsInitialized)
        {
            //Perform uninitialization here

            r_IsInitialized = 0;
        }

        return;
    }

```

```

    //Do standard processing here

```

```

}

```

```

/*=====*/

```

```

-----*/

```

```

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

```

```

sc.GraphName = "IsLastBarClosedBasedOnElapsedTime Example";

sc.StudyDescription = "Example of IsLastBarClosedBasedOnElapsedTime function.";

sc.AutoLoop = 1;
sc.UpdateAlways = 1;
return;
}

// Do data processing
if (sc.Index == sc.ArraySize - 1)
{
    int IsClosed = sc.IsLastBarClosedBasedOnElapsedTime(sc.ChartNumber);
    SCString MessageText("sc.IsLastBarClosedBasedOnElapsedTime=");
    MessageText.AppendFormat("%d", IsClosed);
    sc.AddMessageToLog(MessageText, 0);
}
}

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

        sc.GraphName = "MarketLimitOrdersForPrice Example";

        sc.AutoLoop = 0;
        return;
    }

    // Do data processing
    int BidNumLevels = sc.GetBidMarketDepthNumberOfLevels();
    int AskNumLevels = sc.GetAskMarketDepthNumberOfLevels();

    const int NumberOfMarketOrderDataElements = 20;
    n_ACSIL::s_MarketOrderData MarketOrderData[NumberOfMarketOrderDataElements];

    for (int LevelIndex = 0; LevelIndex < BidNumLevels; LevelIndex++)
    {
        s_MarketDepthEntry MarketDepthEntry;
        sc.GetBidMarketDepthEntryAtLevel(MarketDepthEntry, LevelIndex);

        int ActualLevels = sc.GetBidMarketLimitOrdersForPrice(sc.Round(MarketDepthEntry.AdjustedPrice / sc.TickSize),
        NumberOfMarketOrderDataElements, MarketOrderData);

        for (int OrderDataIndex = 0; OrderDataIndex < ActualLevels; OrderDataIndex++)
        {
            uint64_t OrderID = MarketOrderData[OrderDataIndex].OrderID;
            t_MarketDataQuantity MarketDataQuantity = MarketOrderData[OrderDataIndex].OrderQuantity;
        }
    }

    for (int LevelIndex = 0; LevelIndex < AskNumLevels; LevelIndex++)
    {
        s_MarketDepthEntry MarketDepthEntry;
        sc.GetAskMarketDepthEntryAtLevel(MarketDepthEntry, LevelIndex);
    }
}

```

```
int ActualLevels = sc.GetAskMarketLimitOrdersForPrice(sc.Round(MarketDepthEntry.AdjustedPrice / sc.TickSize),  
NumberOfMarketOrderDataElements, MarketOrderData);
```

```
for (int OrderDataIndex = 0; OrderDataIndex < ActualLevels; OrderDataIndex++)  
{  
    uint64_t OrderID = MarketOrderData[OrderDataIndex].OrderID;  
    t_MarketDataQuantity MarketDataQuantity = MarketOrderData[OrderDataIndex].OrderQuantity;  
}  
  
}  
  
}
```