

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

```
This is a C++ source code file. This file contains example Advanced Custom  
Study functions. It also contains functions for some built-in studies  
in Sierra Chart.
```

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

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

```
SCDLLName("Sierra Chart Custom Studies and Examples")
```

```
enum TimeAndSalesStudyFlagsEnum
```

```
{  
    TIME_AND_SALES_STUDY_UNKNOWN = 0,  
    TIME_AND_SALES_STUDY_PRICE = 1,  
    TIME_AND_SALES_STUDY_VOLUME = 2,  
    TIME_AND_SALES_STUDY_TIME = 3,  
    TIME_AND_SALES_STUDY_BID_AND_ASK_PRICES = 4,  
    TIME_AND_SALES_STUDY_BID_SIZE = 5,  
    TIME_AND_SALES_STUDY_ASK_SIZE = 6,  
    TIME_AND_SALES_STUDY_MILLISECONDS = 7,  
    TIME_AND_SALES_STUDY_TOTAL_BID_SIZE_DEPTH = 8,  
    TIME_AND_SALES_STUDY_TOTAL_ASK_SIZE_DEPTH = 9  
};
```

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

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

```
This example code calculates a simple moving average (30 period by  
default).
```

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

```
SCSFExport scsf_SimpMovAvg(SCStudyInterfaceRef sc)
```

```
{  
    SCSubgraphRef Subgraph_Average = sc.Subgraph[0];  
  
    SCInputRef Input_Length = sc.Input[0];  
  
    // Set configuration variables  
    if (sc.SetDefaults)  
    {  
        // Set the configuration and defaults  
  
        sc.GraphName = "Simple Moving Average";  
  
        sc.StudyDescription = "Example function for calculating a simple moving average.";  
  
        // Set the region to draw the graph in. Region zero is the main  
        // price graph region.  
        sc.GraphRegion = 0;  
  
        sc.ValueFormat = VALUEFORMAT_INHERITED;  
  
        // Set the name of the first subgraph  
        Subgraph_Average.Name = "Average";  
  
        // Set the color, style and line width for the subgraph  
        Subgraph_Average.PrimaryColor = RGB(0,255,0);  
        Subgraph_Average.DrawStyle = DRAWSTYLE_LINE;  
        Subgraph_Average.LineWidth = 2;  
  
        // Set the Length input and default it to 30  
        Input_Length.Name = "Length";  
        Input_Length.SetInt(30);  
        Input_Length.SetIntLimits(1,MAX_STUDY_LENGTH);  
        Input_Length.SetDescription("The number of bars to average.");
```

```

sc.AutoLoop = 1;

sc.AlertOnlyOncePerBar = true;

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

// Do data processing

// Set the index of the first array element to begin drawing at
sc.DataStartIndex = Input_Length.GetInt() - 1;

// Calculate a simple moving average from the base data
sc.SimpleMovAvg(sc.Close, Subgraph_Average, Input_Length.GetInt());

if (sc.CrossOver(Subgraph_Average, sc.Close))
{
    // Since we are using auto-looping we do not specify the Index parameter.
    sc.SetAlert(1, "Moving average has crossed last price.");
}
}

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

    SCInputRef Input_Length = sc.Input[0];

    // Set configuration variables

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

        sc.GraphName = "Simple Moving Average With Dynamic Length";

        sc.StudyDescription = "Example function for calculating a simple moving average with a length based on the time
frame of the bars.";

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

        sc.ValueFormat = VALUEFORMAT_INHERITED;

        // Set the name of the first subgraph
        Subgraph_Average.Name = "Average";

        // Set the color and style of the graph line. If these are not set, the default will be used.
        Subgraph_Average.PrimaryColor = RGB(0,255,0);
        Subgraph_Average.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Average.LineWidth = 3;

        // Make the Length input and default it to 30
        Input_Length.Name = "Length";
        Input_Length.SetInt(30);
        Input_Length.SetIntLimits(1, MAX_STUDY_LENGTH);

        sc.AutoLoop = 1;

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

```

```

    return;
}

Input_Length.SetInt(30 / (sc.SecondsPerBar/60) * 50);

// Set the index of the first array element to begin drawing at
sc.DataStartIndex = Input_Length.GetInt(); - 1;

// Calculate a simple moving average from the base data
sc.SimpleMovAvg(sc.Close,Subgraph_Average,Input_Length.GetInt());
}

/*=====
Parabolic study function.
-----*/
SCSFExport scsf_Parabolic(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Parabolic = sc.Subgraph[0];

    SCInputRef Input_InputDataHigh = sc.Input[0];
    SCInputRef Input_InputDataLow = sc.Input[1];
    SCInputRef Input_StartAccelerationFactor = sc.Input[3];
    SCInputRef Input_AccelerationIncrement = sc.Input[4];
    SCInputRef Input_MaxAccelerationFactor = sc.Input[5];
    SCInputRef Input_AdjustForGap = sc.Input[6];
    SCInputRef Input_Version = sc.Input[7];

    // Set configuration variables

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

        sc.GraphName = "Parabolic";

        sc.StudyDescription = "";
        sc.DrawZeros = false;
        sc.GraphRegion = 0;
        sc.ValueFormat = VALUEFORMAT_INHERITED;

        sc.AutoLoop = 1;

        Subgraph_Parabolic.Name = "Parabolic";
        Subgraph_Parabolic.DrawStyle = DRAWSTYLE_DASH;
        Subgraph_Parabolic.PrimaryColor = RGB(0,255,0);

        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);

        Input_StartAccelerationFactor.Name = "Start Acceleration Factor";
        Input_StartAccelerationFactor.SetFloat(0.02f);
        Input_StartAccelerationFactor.SetFloatLimits(0.000001f, static_cast<float>(MAX_STUDY_LENGTH));

        Input_AccelerationIncrement.Name = "Acceleration Increment";
        Input_AccelerationIncrement.SetFloat(0.02f);
        Input_AccelerationIncrement.SetFloatLimits(0.00001f, 100.0f);

        Input_MaxAccelerationFactor.Name = "Max Acceleration Factor";
        Input_MaxAccelerationFactor.SetFloat(0.2f);
        Input_MaxAccelerationFactor.SetFloatLimits(0.000001f, static_cast<float>(MAX_STUDY_LENGTH));
    }
}

```

```

    Input_AdjustForGap.Name = "Adjust for Gap";
    Input_AdjustForGap.SetYesNo(0); // No

    return;
}

if (Input_Version.GetInt() == 0)
{
    Input_InputDataHigh.SetInputDataIndex(SC_HIGH);
    Input_InputDataLow.SetInputDataIndex(SC_LOW);
}

// Do data processing

sc.DataStartIndex = 1;

sc.Parabolic(
    sc.BaseDataIn,
    sc.BaseDateTimeln,
    Subgraph_Parabolic,
    sc.Index,
    Input_StartAccelerationFactor.GetFloat(),
    Input_AccelerationIncrement.GetFloat(),
    Input_MaxAccelerationFactor.GetFloat(),
    Input_AdjustForGap.GetYesNo(),
    Input_InputDataHigh.GetInputDataIndex(),
    Input_InputDataLow.GetInputDataIndex()
);
}

/*=====
SuperTrend Stop study function.
-----*/
SCSFExport scsf_SuperTrendStop(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Stop = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Median    = sc.Subgraph[1];
    SCSubgraphRef Subgraph_HullATR = sc.Subgraph[2];

    SCFloatArrayRef Array_TrueRange    = Subgraph_Stop.Arrays[0];
    SCFloatArrayRef Array_AvgTrueRange = Subgraph_Stop.Arrays[1];
    SCFloatArrayRef Array_Trend        = Subgraph_Stop.Arrays[2];

    SCInputRef Input_ATRMultiplier = sc.Input[0];
    SCInputRef Input_ATRPeriod     = sc.Input[1];
    SCInputRef Input_MedianPeriod  = sc.Input[2];
    SCInputRef Input_ATRMovAvgType = sc.Input[3];

    // Set configuration variables
    if (sc.SetDefaults)
    {
        // Set the configuration and defaults
        sc.GraphName = "SuperTrend Stop";

        sc.StudyDescription = "";
        sc.DrawZeros = false;
        sc.GraphRegion = 0;
        sc.ValueFormat = sc.BaseGraphValueFormat;

        sc.AutoLoop = 1;

        Subgraph_Stop.Name = "Stop";
        Subgraph_Stop.DrawStyle = DRAWSTYLE_DASH;

```

```

Subgraph_Stop.LineWidth = 2;
Subgraph_Stop.PrimaryColor = COLOR_BLUE;
Subgraph_Stop.SecondaryColor = COLOR_RED;
Subgraph_Stop.SecondaryColorUsed = 1;

Input_ATRMultiplier.Name = "ATR Multiplier";
Input_ATRMultiplier.SetFloat(2);
Input_ATRMultiplier.SetFloatLimits(0.000001f, static_cast<float>(MAX_STUDY_LENGTH));

Input_ATRPeriod.Name = "ATR Period";
Input_ATRPeriod.SetInt(3);
Input_ATRPeriod.SetIntLimits(1,MAX_STUDY_LENGTH);

Input_MedianPeriod.Name = "Median Period";
Input_MedianPeriod.SetInt(3);
Input_MedianPeriod.SetIntLimits(1,MAX_STUDY_LENGTH);

Input_ATRMovAvgType.Name = "ATR Moving Average Type";
Input_ATRMovAvgType.SetCustomInputStrings("Exponential Moving Avg;Linear Regression Moving Avg;Simple
Moving Avg;Weighted Moving Avg;Wilders Moving Avg;Simple Moving Avg SkipZeros;Smoothed Moving Avg;Hull Moving
Avg");
Input_ATRMovAvgType.SetCustomInputIndex(7);

return;
}

// Do data processing
sc.MovingMedian(sc.HLAv, Subgraph_Median, sc.Index, Input_MedianPeriod.GetInt());

sc.TrueRange(sc.BaseDataIn, Array_TrueRange);

if(Input_ATRMovAvgType.GetIndex() == 0)
    sc.ATR(sc.BaseDataIn, Array_TrueRange, Array_AvgTrueRange, sc.Index, Input_ATRPeriod.GetInt(),
MOVAVGTYPE_EXPONENTIAL);
else if(Input_ATRMovAvgType.GetIndex() == 1)
    sc.ATR(sc.BaseDataIn, Array_TrueRange, Array_AvgTrueRange, sc.Index, Input_ATRPeriod.GetInt(),
MOVAVGTYPE_LINEARREGRESSION);
else if (Input_ATRMovAvgType.GetIndex() == 2)
    sc.ATR(sc.BaseDataIn, Array_TrueRange, Array_AvgTrueRange, sc.Index, Input_ATRPeriod.GetInt(),
MOVAVGTYPE_SIMPLE);
else if (Input_ATRMovAvgType.GetIndex() == 3)
    sc.ATR(sc.BaseDataIn, Array_TrueRange, Array_AvgTrueRange, sc.Index, Input_ATRPeriod.GetInt(),
MOVAVGTYPE_WEIGHTED);
else if (Input_ATRMovAvgType.GetIndex() == 4)
    sc.ATR(sc.BaseDataIn, Array_TrueRange, Array_AvgTrueRange, sc.Index, Input_ATRPeriod.GetInt(),
MOVAVGTYPE_WILDERS);
else if (Input_ATRMovAvgType.GetIndex() == 5)
    sc.ATR(sc.BaseDataIn, Array_TrueRange, Array_AvgTrueRange, sc.Index, Input_ATRPeriod.GetInt(),
MOVAVGTYPE_SIMPLE_SKIP_ZEROS);
else if (Input_ATRMovAvgType.GetIndex() == 6)
    sc.ATR(sc.BaseDataIn, Array_TrueRange, Array_AvgTrueRange, sc.Index, Input_ATRPeriod.GetInt(),
MOVAVGTYPE_SMOOTHED);
else
{
    sc.HullMovingAverage(Array_TrueRange, Subgraph_HullATR, Input_ATRPeriod.GetInt());
    Array_AvgTrueRange[sc.Index] = Subgraph_HullATR[sc.Index];
}

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

    Subgraph_Stop[sc.Index] = sc.Close[sc.Index];
    Array_Trend[sc.Index] = 1;
    return;
}

```

```

    }

    if (sc.FormattedEvaluate(sc.Close[sc.Index], sc.ValueFormat, GREATER_OPERATOR, Subgraph_Stop[sc.Index-1],
sc.ValueFormat))
    {
        Array_Trend[sc.Index] = 1;
        float NewStop = Subgraph_Median[sc.Index] - Input_ATRMultiplier.GetFloat()*Array_AvgTrueRange[sc.Index-1];
        if (Array_Trend[sc.Index-1] < 0)
        {
            Subgraph_Stop[sc.Index] = NewStop;
        }
        else
        {
            Subgraph_Stop[sc.Index] = max(NewStop, Subgraph_Stop[sc.Index-1]);
        }
    }
    else if (sc.FormattedEvaluate(sc.Close[sc.Index], sc.ValueFormat, LESS_OPERATOR, Subgraph_Stop[sc.Index-1],
sc.ValueFormat))
    {
        Array_Trend[sc.Index] = -1;
        float NewStop = Subgraph_Median[sc.Index] + Input_ATRMultiplier.GetFloat()*Array_AvgTrueRange[sc.Index-1];
        if (Array_Trend[sc.Index-1] > 0)
        {
            Subgraph_Stop[sc.Index] = NewStop;
        }
        else
        {
            Subgraph_Stop[sc.Index] = min(NewStop, Subgraph_Stop[sc.Index-1]);
        }
    }
    else
    {
        Array_Trend[sc.Index] = Array_Trend[sc.Index-1];
        Subgraph_Stop[sc.Index] = Subgraph_Stop[sc.Index-1];
    }

    Subgraph_Stop.DataColor[sc.Index] = Array_Trend[sc.Index] > 0 ? Subgraph_Stop.PrimaryColor :
Subgraph_Stop.SecondaryColor;

}

/*=====
   This function is an example of how to calculate high - low * 2.5.
   -----*/
SCSFExport scsf_SimpleArithmeticExample(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Line = sc.Subgraph[0];

    // Set configuration variables

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

        sc.GraphName = "SimpleArithmeticExample";

        sc.StudyDescription = "This function is an example of how to calculate (high-low)*10.";

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

        sc.AutoLoop = 1;

        return;
    }

```

```

}

// Do data processing

Subgraph_Line[sc.Index] = (sc.High[sc.Index] - sc.Low[sc.Index]) * 10;
}

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

    // Set configuration variables

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

        sc.GraphName = "Inside Bar";

        sc.GraphRegion = 0;

        Subgraph_IB.Name = "IB";
        Subgraph_IB.DrawStyle = DRAWSTYLE_COLOR_BAR;
        Subgraph_IB.PrimaryColor = RGB(255,128,0);
        Subgraph_IB.DrawZeros = false;

        sc.AutoLoop = 1;

        return;
    }

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

    // Do data processing

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

    // Set configuration variables

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

        sc.GraphName = "Inside Bar - Markttechnik";

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

        Subgraph_IB.Name = "IB";
        Subgraph_IB.DrawStyle = DRAWSTYLE_COLOR_BAR;
        Subgraph_IB.PrimaryColor = RGB(255,128,0);

```

```

Subgraph_IB.DrawZeros = false;

sc.AutoLoop = 1;

return;
}

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

// Do data processing

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_OutsideBar(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_OutsideBar = sc.Subgraph[0];
    SCInputRef Input_IncludeEqualHighLow = sc.Input[0];

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

        sc.GraphName = "Outside Bar";

        sc.GraphRegion = 0;

        Subgraph_OutsideBar.Name = "OB";
        Subgraph_OutsideBar.DrawStyle = DRAWSTYLE_COLOR_BAR;
        Subgraph_OutsideBar.LineWidth = 1;
        Subgraph_OutsideBar.PrimaryColor = RGB(255,128,0);
        Subgraph_OutsideBar.DrawZeros = false;

        Input_IncludeEqualHighLow.Name = "Include Equal High Low";
        Input_IncludeEqualHighLow.SetYesNo(false);

        sc.AutoLoop = 1;

        return;
    }

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

    // Do data processing

    if (Input_IncludeEqualHighLow.GetYesNo() == 0 &&
        High[sc.Index] > High[sc.Index - 1]
        && Low[sc.Index] < Low[sc.Index - 1])
    {
        Subgraph_OutsideBar[sc.Index] = High[sc.Index];
    }
    else if (Input_IncludeEqualHighLow.GetYesNo() != 0 &&
        (
            High[sc.Index] >= High[sc.Index - 1]
            && Low[sc.Index] < Low[sc.Index - 1])
    )

```



```

    || (High[sc.Index] > High[sc.Index - 1]
    && Low[sc.Index] <= Low[sc.Index - 1])
    )
    )
    {
        Subgraph_OutsideBar[sc.Index] = High[sc.Index];
    }
    else
    {
        Subgraph_OutsideBar[sc.Index] = 0;
    }
}

/*=====
This function is an example of how to use the GetCharArray function.
-----*/
SCSFExport scsf_GetCharArrayExample(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Prices = sc.Subgraph[0];
    SCSubgraphRef Subgraph_DateTime = sc.Subgraph[1];

    // Set configuration variables

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

        sc.GraphName = "GetCharArrayExample";

        sc.StudyDescription = "This function is an example of how to use the GetCharArray function.";

        Subgraph_Prices.Name = "Prices";
        Subgraph_Prices.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Prices.PrimaryColor = RGB(0,255,0);

        Subgraph_DateTime.Name = "Date Time";
        Subgraph_DateTime.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_DateTime.PrimaryColor = RGB(255,0,255);

        sc.AutoLoop = 1;

        return;
    }

    // Do data processing

    SCDateTimeArray DateTimeArray;
    SCFloatArray PriceArray;

    // Get the date-time array from chart 1
    sc.GetChartDataTimeArray(1, DateTimeArray);
    if (DateTimeArray.GetArraySize() == 0)
        return; // The DateTimeArray may not exist or is empty -- either way we can't do anything with it

    // Get the close/last array from chart 1
    sc.GetCharArray(1, SC_LAST, PriceArray);
    if (PriceArray.GetArraySize() == 0)
        return; // The PriceArray may not exist or is empty -- either way we can't do anything with it

    // Copy the price array value from chart 1 to subgraph 1. Most likely
    // the array from chart 1 is not the same size as the array this study
    // function is applied to. Therefore there is not going to be a correct
    // column to column correspondence. However, this is just a simple example.
    Subgraph_Prices[sc.Index] = PriceArray[sc.Index];

```

```

//copy the date time array from chart 1 to subgraph 2
Subgraph_DateTime[sc.Index] = static_cast<float>(DateTimeArray[sc.Index].GetAsDouble());
}

/*=====
This function is an example of how to use the GetStudyArray function.
-----*/
SCSFExport scsf_GetStudyArrayExample(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Line = sc.Subgraph[0];

    // Set configuration variables

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

        sc.GraphName = "GetStudyArrayExample";

        sc.StudyDescription = "This function is an example of how to use the GetStudyArray function.";

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

        sc.AutoLoop = 1;

        return;
    }

    // Do data processing

    SCFloatArray StudyArray;

    // Get first Subgraph from study #1.
    sc.GetStudyArray(1, 1, StudyArray);
    if (StudyArray.GetArraySize() == 0)
        return; // The StudyArray may not exist or is empty. Either way we cannot do anything with it

    Subgraph_Line[sc.Index] = StudyArray[sc.Index];
}

/*=====
Calculates the adaptive moving average.
-----*/
SCSFExport scsf_AdaptiveMovingAverage(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Avg = sc.Subgraph[0];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_Length = sc.Input[3];
    SCInputRef Input_FastSmoothingConstant = sc.Input[4];
    SCInputRef Input_SlowSmoothingConstant = sc.Input[5];

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

        sc.GraphName = "Moving Average - Adaptive";

        sc.StudyDescription = "Calculates the adaptive moving average.";
    }
}

```

```

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

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

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

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

Input_FastSmoothingConstant.Name = "Fast Smoothing Constant";
Input_FastSmoothingConstant.SetFloat(2.0f);
Input_FastSmoothingConstant.SetFloatLimits(0.0001f, static_cast<float>(MAX_STUDY_LENGTH));

Input_SlowSmoothingConstant.Name = "Slow Smoothing Constant";
Input_SlowSmoothingConstant.SetFloat(30.0f);
Input_SlowSmoothingConstant.SetFloatLimits(0.0001f, static_cast<float>(MAX_STUDY_LENGTH));

sc.AutoLoop = 1;

return;
}

// Do data processing
sc.AdaptiveMovAvg(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_Avg, Input_Length.GetInt(),
Input_FastSmoothingConstant.GetFloat(), Input_SlowSmoothingConstant.GetFloat());
}

/*=====
Calculates the Moving Average - Adaptive Binary Wave
-----*/
SCSFExport scsf_MovingAverageAdaptiveBinaryWave(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Wave = sc.Subgraph[0];

    SCFloatArrayRef Array_AdaptiveMovingAverage = Subgraph_Wave.Arrays[0];
    SCFloatArrayRef Array_StdDev = Subgraph_Wave.Arrays[1];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_Length = sc.Input[1];
    SCInputRef Input_FastSmoothingConstant = sc.Input[2];
    SCInputRef Input_SlowSmoothingConstant = sc.Input[3];
    SCInputRef Input_FilterPercent = sc.Input[4];

    if (sc.SetDefaults)
    {
        // Set the configuration and defaults
        sc.GraphName = "Moving Average - Adaptive Binary Wave";

        sc.ValueFormat = 0;
        sc.ScaleIncrement = 1.0f;

        sc.AutoLoop = 1;

        // subgraphs
        Subgraph_Wave.Name = "Binary Wave";
        Subgraph_Wave.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Wave.PrimaryColor = RGB(0,0,255);
        Subgraph_Wave.DrawZeros = 1;
    }
}

```

```

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

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

Input_FastSmoothingConstant.Name = "Fast Smoothing Constant";
Input_FastSmoothingConstant.SetFloat(2.0f);
Input_FastSmoothingConstant.SetFloatLimits(0.0001f, static_cast<float>(MAX_STUDY_LENGTH));

Input_SlowSmoothingConstant.Name = "Slow Smoothing Constant";
Input_SlowSmoothingConstant.SetFloat(30.0f);
Input_SlowSmoothingConstant.SetFloatLimits(0.0001f, static_cast<float>(MAX_STUDY_LENGTH));

Input_FilterPercent.Name = "Filter Percent";
Input_FilterPercent.SetFloat(25.0f);
Input_FilterPercent.SetFloatLimits(0.0f, 100.0f);

return;
}

// Do data processing
sc.AdaptiveMovAvg(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Array_AdaptiveMovingAverage,
Input_Length.GetInt(), Input_FastSmoothingConstant.GetFloat(), Input_SlowSmoothingConstant.GetFloat());

sc.StdDeviation(Array_AdaptiveMovingAverage, Array_StdDev, Input_Length.GetInt());

float Filter = Input_FilterPercent.GetFloat() / 100.0f * Array_StdDev[sc.Index];

float& AMALow = sc.GetPersistentFloat(0);
float& AMAHigh = sc.GetPersistentFloat(1);

if (sc.Index == 0)
{
    AMALow = Array_AdaptiveMovingAverage[0];
    AMAHigh = Array_AdaptiveMovingAverage[0];
}
else
{
    if (Array_AdaptiveMovingAverage[sc.Index] < Array_AdaptiveMovingAverage[sc.Index-1])
        AMALow = Array_AdaptiveMovingAverage[sc.Index];
    if (Array_AdaptiveMovingAverage[sc.Index] > Array_AdaptiveMovingAverage[sc.Index-1])
        AMAHigh = Array_AdaptiveMovingAverage[sc.Index];
}

if (Array_AdaptiveMovingAverage[sc.Index] - AMALow > Filter)
    Subgraph_Wave[sc.Index] = 1.0;
else if (AMAHigh - Array_AdaptiveMovingAverage[sc.Index] > Filter)
    Subgraph_Wave[sc.Index] = -1.0;
else
    Subgraph_Wave[sc.Index] = 0;
}

/*=====
This function is only used internally by the other Time and Sales
functions. This needs to be above the other Time and Sales functions
that use this function.

```

This function fills in a study Subgraph array with the Time and Sales Price, Volume, or Time based on what Flag has been set.

```

-----*/
void TimeAndSales(SCStudyInterfaceRef sc, TimeAndSalesStudyFlagsEnum Flag)
{
    SCSubgraphRef FirstSubgraph = sc.Subgraph[0];
    SCSubgraphRef SecondSubgraph = sc.Subgraph[1];
    SCSubgraphRef ThirdSubgraph = sc.Subgraph[2];

    SCInputRef InVolumeMinFilter = sc.Input[2];
    SCInputRef InVolumeMaxFilter = sc.Input[3];

    int& EarliestIndex = sc.GetPersistentInt(1); // This is the earliest index we updated on the last call

    // Get the Time and Sales
    c_SCTimeAndSalesArray TimeSales;
    sc.GetTimeAndSales(TimeSales);
    int TimeSalesSize = TimeSales.Size();

    if (TimeSalesSize == 0)
        return; // No Time and Sales data available for the symbol

    // Loop through the Time and Sales
    int OutputArrayIndex = sc.ArraySize;
    for (int TSIndex = TimeSalesSize - 1; TSIndex >= 0; --TSIndex)
    {
        s_TimeAndSales TimeAndSalesRecord = TimeSales[TSIndex];
        TimeAndSalesRecord *= sc.RealTimePriceMultiplier;
        float Volume = TimeAndSalesRecord.Volume * sc.MultiplierFromVolumeValueFormat();

        if (Flag == TIME_AND_SALES_STUDY_PRICE
            || Flag == TIME_AND_SALES_STUDY_VOLUME
            || Flag == TIME_AND_SALES_STUDY_TIME )
        {
            if (Volume < static_cast<unsigned int>(InVolumeMinFilter.GetInt())
                || (Volume > static_cast<unsigned int>(InVolumeMaxFilter.GetInt()) && InVolumeMaxFilter.GetInt() > 0)
                || TimeAndSalesRecord.Type == SC_TS_BIDASKVALUES
            )
                continue;
        }
        else if (Flag == TIME_AND_SALES_STUDY_BID_AND_ASK_PRICES
            || Flag == TIME_AND_SALES_STUDY_BID_SIZE
            || Flag == TIME_AND_SALES_STUDY_ASK_SIZE
            || Flag == TIME_AND_SALES_STUDY_TOTAL_BID_SIZE_DEPTH
            || Flag == TIME_AND_SALES_STUDY_TOTAL_ASK_SIZE_DEPTH)
        {
            if (TimeAndSalesRecord.Type != SC_TS_BIDASKVALUES && !sc.IsReplayRunning())
                continue;
        }

        if (Flag == TIME_AND_SALES_STUDY_PRICE) // Price
        {
            --OutputArrayIndex;
            if (OutputArrayIndex < 0)
                break;

            if (TimeAndSalesRecord.Type == SC_TS_BID)
            {
                FirstSubgraph[OutputArrayIndex] = TimeAndSalesRecord.Price;
                SecondSubgraph[OutputArrayIndex] = 0.0f;
                ThirdSubgraph[OutputArrayIndex] = 0.0f;
            }
            else if (TimeAndSalesRecord.Type == SC_TS_ASK)
            {
                FirstSubgraph[OutputArrayIndex] = 0.0f;

```

```

        SecondSubgraph[OutputArrayIndex] = TimeAndSalesRecord.Price;
        ThirdSubgraph[OutputArrayIndex] = 0.0f;
    }
    else
    {
        FirstSubgraph[OutputArrayIndex] = 0.0f;
        SecondSubgraph[OutputArrayIndex] = 0.0f;
        ThirdSubgraph[OutputArrayIndex] = TimeAndSalesRecord.Price;
    }
}
else if (Flag == TIME_AND_SALES_STUDY_VOLUME) // Volume
{
    --OutputArrayIndex;
    if (OutputArrayIndex < 0)
        break;

    if (TimeAndSalesRecord.Type == SC_TS_BID)
    {
        FirstSubgraph[OutputArrayIndex] = static_cast<float>(Volume);
        SecondSubgraph[OutputArrayIndex] = 0.0f;
        ThirdSubgraph[OutputArrayIndex] = 0.0f;
    }
    else if (TimeAndSalesRecord.Type == SC_TS_ASK)
    {
        FirstSubgraph[OutputArrayIndex] = 0.0f;
        SecondSubgraph[OutputArrayIndex] = static_cast<float>(Volume);
        ThirdSubgraph[OutputArrayIndex] = 0.0f;
    }
    else
    {
        FirstSubgraph[OutputArrayIndex] = 0.0f;
        SecondSubgraph[OutputArrayIndex] = 0.0f;
        ThirdSubgraph[OutputArrayIndex] = static_cast<float>(Volume);
    }
}
else if (Flag == TIME_AND_SALES_STUDY_TIME) // Time
{
    --OutputArrayIndex;
    if (OutputArrayIndex < 0)
        break;

    // Adjust a time and sales record date-time
    // TimeSales is a s_TimeAndSales record requested with sc.GetTimeAndSales().
    SCDateTime TempDateTime = TimeAndSalesRecord.DateTime;

    //TempDateTime += sc.TimeScaleAdjustment; // Apply the time zone offset for the chart

    TempDateTime = sc.ConvertDateTimeUTCToChartTimeZone(TempDateTime);

    TempDateTime.ClearDate(); // Remove the date part so only the time remains
    FirstSubgraph[OutputArrayIndex] = static_cast<float>(TempDateTime.GetAsDouble()); // Put in the output array
}
else if (Flag == TIME_AND_SALES_STUDY_BID_AND_ASK_PRICES) // Bid & Ask
{
    --OutputArrayIndex;
    if (OutputArrayIndex < 0)
        break;

    FirstSubgraph[OutputArrayIndex] = TimeAndSalesRecord.Bid;
    SecondSubgraph[OutputArrayIndex] = TimeAndSalesRecord.Ask;
}
else if (Flag == TIME_AND_SALES_STUDY_BID_SIZE) // Bid size

```

```

{
    float BidSize = static_cast<float>(TimeAndSalesRecord.BidSize) * sc.MultiplierFromVolumeValueFormat();

    if (BidSize< InVolumeMinFilter.GetInt()
        || (InVolumeMaxFilter.GetInt() > 0 && BidSize > InVolumeMaxFilter.GetInt() )
        )
        continue;

    --OutputArrayIndex;
    if (OutputArrayIndex < 0)
        break;

    FirstSubgraph[OutputArrayIndex] = BidSize;

}
else if (Flag == TIME_AND_SALES_STUDY_ASK_SIZE) // Ask size
{
    float AskSize = static_cast<float>(TimeAndSalesRecord.AskSize) * sc.MultiplierFromVolumeValueFormat();

    if (AskSize< InVolumeMinFilter.GetInt()
        || (InVolumeMaxFilter.GetInt() > 0 && AskSize > InVolumeMaxFilter.GetInt() )
        )
        continue;

    --OutputArrayIndex;
    if (OutputArrayIndex < 0)
        break;

    FirstSubgraph[OutputArrayIndex] = AskSize;
}
else if (Flag == TIME_AND_SALES_STUDY_TOTAL_BID_SIZE_DEPTH)
{
    float TotalBidSizeDepth = static_cast<float>(TimeAndSalesRecord.TotalBidDepth) *
sc.MultiplierFromVolumeValueFormat();

    if (TotalBidSizeDepth < InVolumeMinFilter.GetInt()
        || (InVolumeMaxFilter.GetInt() > 0 && TotalBidSizeDepth > InVolumeMaxFilter.GetInt() )
        )
        continue;

    --OutputArrayIndex;
    if (OutputArrayIndex < 0)
        break;

    FirstSubgraph[OutputArrayIndex] = TotalBidSizeDepth;
}
else if (Flag == TIME_AND_SALES_STUDY_TOTAL_ASK_SIZE_DEPTH)
{
    float TotalAskSizeDepth = static_cast<float>(TimeAndSalesRecord.TotalAskDepth) *
sc.MultiplierFromVolumeValueFormat();

    if (TotalAskSizeDepth < InVolumeMinFilter.GetInt()
        || (InVolumeMaxFilter.GetInt() > 0 && TotalAskSizeDepth > InVolumeMaxFilter.GetInt() )
        )
        continue;

    --OutputArrayIndex;
    if (OutputArrayIndex < 0)
        break;

    FirstSubgraph[OutputArrayIndex] = TotalAskSizeDepth;
}
else if (Flag == TIME_AND_SALES_STUDY_MILLISECONDS) // Milliseconds
{

```

```

--OutputArrayIndex;
if (OutputArrayIndex < 0)
    break;

FirstSubgraph[OutputArrayIndex] = static_cast<float>(TimeAndSalesRecord.DateTime.GetMillisecond());
}
}

// Zero elements from the first index that was written to last time, to the
// first index that was written to this time.
for (int SubgraphIndex = 0; SubgraphIndex < sc.NumberOfArrays; ++SubgraphIndex)
{
    for (int DataIndex = EarliestIndex; DataIndex < OutputArrayIndex; ++DataIndex)
        sc.Subgraph[SubgraphIndex][DataIndex] = 0.0f;
}

// Store the last index that was written to
EarliestIndex = OutputArrayIndex;
}

/*=====*/
SCSFExport scsf_TimeAndSalesPrice(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Bid = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Ask = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Other = sc.Subgraph[2];

    SCInputRef Input_VolumeFilterGE = sc.Input[2];
    SCInputRef Input_VolumeFilterLE = sc.Input[3];

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

        sc.GraphName = "Time and Sales Price";

        sc.StudyDescription = "";

        sc.GraphRegion = 1;
        sc.ValueFormat = VALUEFORMAT_INHERITED;

        Subgraph_Bid.Name = "Bid Trade";
        Subgraph_Bid.DrawStyle = DRAWSTYLE_DASH;
        Subgraph_Bid.PrimaryColor = RGB(255,0,0); // Red
        Subgraph_Bid.LineWidth = 4;
        Subgraph_Bid.DrawZeros = false;

        Subgraph_Ask.Name = "Ask Trade";
        Subgraph_Ask.DrawStyle = DRAWSTYLE_DASH;
        Subgraph_Ask.PrimaryColor = RGB(0,255,0); // Green
        Subgraph_Ask.LineWidth = 4;
        Subgraph_Ask.DrawZeros = false;

        Subgraph_Other.Name = "Other";
        Subgraph_Other.DrawStyle = DRAWSTYLE_DASH;
        Subgraph_Other.PrimaryColor = RGB(255,255,255); // White
        Subgraph_Other.LineWidth = 4;
        Subgraph_Other.DrawZeros = false;

        Input_VolumeFilterGE.Name = "Volume Filter >=";
        Input_VolumeFilterGE.SetInt(1);
        Input_VolumeFilterGE.SetIntLimits(0, INT_MAX);

        Input_VolumeFilterLE.Name = "Volume Filter <= (0 means no limit)";
        Input_VolumeFilterLE.SetInt(0);
    }
}

```



```

        Input_VolumeFilterLE.SetIntLimits(0, INT_MAX);

    return;
}

TimeAndSales(sc, TIME_AND_SALES_STUDY_PRICE);
}

/*=====*/
SCSFExport scsf_TimeAndSalesVolume(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_BidVol = sc.Subgraph[0];
    SCSubgraphRef Subgraph_AskVol = sc.Subgraph[1];
    SCSubgraphRef Subgraph_OtherVol = sc.Subgraph[2];

    SCInputRef Input_VolumeFilterGreaterEqual = sc.Input[2];
    SCInputRef Input_VolumeFilterLessEqual = sc.Input[3];

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

        sc.GraphName = "Time and Sales Volume";

        sc.StudyDescription = "";

        sc.GraphRegion = 1;
        sc.ValueFormat = sc.VolumeValueFormat;

        Subgraph_BidVol.Name = "Bid Vol";
        Subgraph_BidVol.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_BidVol.PrimaryColor = RGB(255,0,0); // Red
        Subgraph_BidVol.LineWidth = 4;
        Subgraph_BidVol.DrawZeros = false;

        Subgraph_AskVol.Name = "Ask Vol";
        Subgraph_AskVol.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_AskVol.PrimaryColor = RGB(0,255,0); // Green
        Subgraph_AskVol.LineWidth = 4;
        Subgraph_AskVol.DrawZeros = false;

        Subgraph_OtherVol.Name = "Other Vol";
        Subgraph_OtherVol.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_OtherVol.PrimaryColor = RGB(255,255,255); // White
        Subgraph_OtherVol.LineWidth = 4;
        Subgraph_OtherVol.DrawZeros = false;

        Input_VolumeFilterGreaterEqual.Name = "Volume Filter >=";
        Input_VolumeFilterGreaterEqual.SetFloat(1);
        Input_VolumeFilterGreaterEqual.SetFloatLimits(0, FLT_MAX);

        Input_VolumeFilterLessEqual.Name = "Volume Filter <= (0 means no limit)";
        Input_VolumeFilterLessEqual.SetFloat(0);
        Input_VolumeFilterLessEqual.SetFloatLimits(0, FLT_MAX);

    return;
    }

    TimeAndSales(sc, TIME_AND_SALES_STUDY_VOLUME);
}

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

```

```

SCInputRef Input_VolumeFilterGE = sc.Input[2];
SCInputRef Input_VolumeFilterLE = sc.Input[3];

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

    sc.GraphName = "Time and Sales Time";

    sc.StudyDescription = "";

    sc.GraphRegion = 1;
    sc.ValueFormat = 20; // Time

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

    Input_VolumeFilterGE.Name = "Volume Filter >=";
    Input_VolumeFilterGE.SetInt(1);
    Input_VolumeFilterGE.SetIntLimits(0, INT_MAX);

    Input_VolumeFilterLE.Name = "Volume Filter <= (0 means no limit)";
    Input_VolumeFilterLE.SetInt(0);
    Input_VolumeFilterLE.SetIntLimits(0, INT_MAX);

    return;
}

TimeAndSales(sc, TIME_AND_SALES_STUDY_TIME);
}

/*=====*/
SCSFExport scsf_TimeAndSalesBidAsk(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Bid = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Ask = sc.Subgraph[1];

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

        sc.GraphName = "Time and Sales Bid & Ask";

        sc.GraphRegion = 1;

        sc.ValueFormat = VALUEFORMAT_INHERITED;

        Subgraph_Bid.Name = "Bid";
        Subgraph_Bid.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Bid.PrimaryColor = RGB(255,0,0); // Red
        Subgraph_Bid.LineWidth = 1;
        Subgraph_Bid.LineLabel = LL_DISPLAY_VALUE | LL_VALUE_ALIGN_VALUES_SCALE |
LL_VALUE_ALIGN_CENTER;
        Subgraph_Bid.DrawZeros = false;

        Subgraph_Ask.Name = "Ask";
        Subgraph_Ask.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Ask.PrimaryColor = RGB(0,255,0); // Green
        Subgraph_Ask.LineWidth = 1;
        Subgraph_Ask.LineLabel = LL_DISPLAY_VALUE | LL_VALUE_ALIGN_VALUES_SCALE |
LL_VALUE_ALIGN_CENTER;
    }
}

```

```

        Subgraph_Ask.DrawZeros = false;

    return;
}

TimeAndSales(sc, TIME_AND_SALES_STUDY_BID_AND_ASK_PRICES);
}

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

    SCInputRef Input_VolumeFilterGreaterEqual = sc.Input[2];
    SCInputRef Input_VolumeFilterLessEqual = sc.Input[3];

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

        sc.GraphName = "Time and Sales Bid Size";

        sc.GraphRegion = 1;
        sc.ValueFormat = sc.VolumeValueFormat;

        Subgraph_BidSize.Name = "Bid Size";
        Subgraph_BidSize.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_BidSize.PrimaryColor = RGB(255,0,0); // Red
        Subgraph_BidSize.LineWidth = 1;
        Subgraph_BidSize.DrawZeros = false;

        Input_VolumeFilterGreaterEqual.Name = "Size Filter >=";
        Input_VolumeFilterGreaterEqual.SetFloat(1);
        Input_VolumeFilterGreaterEqual.SetFloatLimits(0, FLT_MAX);

        Input_VolumeFilterLessEqual.Name = "Size Filter <= (0 means no limit)";
        Input_VolumeFilterLessEqual.SetFloat(0);
        Input_VolumeFilterLessEqual.SetFloatLimits(0, FLT_MAX);

    return;
    }

    TimeAndSales(sc, TIME_AND_SALES_STUDY_BID_SIZE);
}

/*=====*/
SCSFExport scsf_TimeAndSalesAskSize(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_AskSize = sc.Subgraph[0];
    SCInputRef Input_VolumeFilterGreaterEqual = sc.Input[2];
    SCInputRef Input_VolumeFilterLessEqual = sc.Input[3];

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

        sc.GraphName = "Time and Sales Ask Size";

        sc.GraphRegion = 1;
        sc.ValueFormat = sc.VolumeValueFormat;

        Subgraph_AskSize.Name = "Ask Size";

```

```

Subgraph_AskSize.DrawStyle = DRAWSTYLE_BAR;
Subgraph_AskSize.PrimaryColor = RGB(0,255,0); // Green
Subgraph_AskSize.LineWidth = 1;
Subgraph_AskSize.DrawZeros = false;

Input_VolumeFilterGreaterEqual.Name = "Size Filter >=";
Input_VolumeFilterGreaterEqual.SetFloat(1);
Input_VolumeFilterGreaterEqual.SetFloatLimits(0, FLT_MAX);

Input_VolumeFilterLessEqual.Name = "Size Filter <= (0 means no limit)";
Input_VolumeFilterLessEqual.SetFloat(0);
Input_VolumeFilterLessEqual.SetFloatLimits(0, FLT_MAX);

return;
}

TimeAndSales(sc, TIME_AND_SALES_STUDY_ASK_SIZE);
}

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

    SCInputRef Input_VolumeFilterGreaterEqual = sc.Input[2];
    SCInputRef Input_VolumeFilterLessEqual = sc.Input[3];

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

        sc.GraphName = "Time and Sales Total Ask Size Depth";
        sc.GraphRegion = 1;
        sc.ValueFormat = sc.VolumeValueFormat;

        sc.UsesMarketDepthData = 1;

        Subgraph_TotalAskSizeDepth.Name = "Ask Size Depth";
        Subgraph_TotalAskSizeDepth.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_TotalAskSizeDepth.PrimaryColor = RGB(0,255,0); // Green
        Subgraph_TotalAskSizeDepth.LineWidth = 1;
        Subgraph_TotalAskSizeDepth.DrawZeros = false;

        Input_VolumeFilterGreaterEqual.Name = "Size Filter >=";
        Input_VolumeFilterGreaterEqual.SetFloat(1);
        Input_VolumeFilterGreaterEqual.SetFloatLimits(0, FLT_MAX);

        Input_VolumeFilterLessEqual.Name = "Size Filter <= (0 means no limit)";
        Input_VolumeFilterLessEqual.SetFloat(0);
        Input_VolumeFilterLessEqual.SetFloatLimits(0, FLT_MAX);

        return;
    }

    TimeAndSales(sc, TIME_AND_SALES_STUDY_TOTAL_ASK_SIZE_DEPTH);
}

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

```

```

{
    SCSubgraphRef Subgraph_TotalBidSizeDepth = sc.Subgraph[0];
    SCInputRef Input_VolumeFilterGreaterEqual = sc.Input[2];
    SCInputRef Input_VolumeFilterLessEqual = sc.Input[3];

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

        sc.GraphName = "Time and Sales Total Bid Size Depth";
        sc.GraphRegion = 1;
        sc.ValueFormat = sc.VolumeValueFormat;

        sc.UsesMarketDepthData = 1;

        Subgraph_TotalBidSizeDepth.Name = "Bid Size Depth";
        Subgraph_TotalBidSizeDepth.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_TotalBidSizeDepth.PrimaryColor = RGB(255,0,0); // Green
        Subgraph_TotalBidSizeDepth.LineWidth = 1;
        Subgraph_TotalBidSizeDepth.DrawZeros = false;

        Input_VolumeFilterGreaterEqual.Name = "Size Filter >=";
        Input_VolumeFilterGreaterEqual.SetFloat(1);
        Input_VolumeFilterGreaterEqual.SetFloatLimits(0, FLT_MAX);

        Input_VolumeFilterLessEqual.Name = "Size Filter <= (0 means no limit)";
        Input_VolumeFilterLessEqual.SetFloat(0);
        Input_VolumeFilterLessEqual.SetFloatLimits(0, FLT_MAX);

        return;
    }

    TimeAndSales(sc, TIME_AND_SALES_STUDY_TOTAL_BID_SIZE_DEPTH);
}

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

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

        sc.GraphName = "Time and Sales Milliseconds";

        sc.StudyDescription = "";

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

        Subgraph_MillisecondsSubgraph.Name = "Milliseconds";
        Subgraph_MillisecondsSubgraph.DrawStyle = DRAWSTYLE_BAR;
        Subgraph_MillisecondsSubgraph.PrimaryColor = RGB(255,255,255);
        Subgraph_MillisecondsSubgraph.LineWidth = 4;
        Subgraph_MillisecondsSubgraph.DrawZeros = true;

        return;
    }

```

```

}

TimeAndSales(sc, TIME_AND_SALES_STUDY_MILLISECONDS);
}

/*=====
This study graphs a line for the selected Input Data item from another chart on to the chart this study is applied to.

The data is synchronized so that the overlayed bars will match with the correct times.
-----*/
SCSFExport scsf_OverlaySingleLine(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_PriceOverlay = sc.Subgraph[0];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_ChartNumberToOverlay = sc.Input[3];
    SCInputRef Input_Multiplier = sc.Input[4];
    SCInputRef Input_DrawZeroValues = sc.Input[5];

    // Set configuration variables

    if (sc.SetDefaults)
    {
        sc.GraphName = "Overlay (Single Line)";

        sc.ValueFormat = VALUEFORMAT_INHERITED;

        sc.GraphRegion = 1;

        sc.ScaleRangeType = SCALE_INDEPENDENT;

        Subgraph_PriceOverlay.Name = "Price Overlay";
        Subgraph_PriceOverlay.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_PriceOverlay.PrimaryColor = RGB(0,255,0);
        Subgraph_PriceOverlay.DrawZeros = false;

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

        Input_ChartNumberToOverlay.Name = "Chart Number To Overlay";
        Input_ChartNumberToOverlay.SetChartNumber(2);

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

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

        sc.AutoLoop = 1;

        return;
    }

    // Do data processing

    Subgraph_PriceOverlay.DrawZeros = Input_DrawZeroValues.GetYesNo();

    // Get the array for the requested Input Data from the requested chart
    SCFloatArray PriceArray;
    sc.GetChartArray(-Input_ChartNumberToOverlay.GetChartNumber(), Input_InputData.GetInputDataIndex(),
    PriceArray);

```

```

// Make sure we got something
if (PriceArray.GetArraySize() == 0)
    return;

int TheirIndex = sc.GetNearestMatchForDateTimeIndex(Input_ChartNumberToOverlay.GetChartNumber(), sc.Index);
Subgraph_PriceOverlay[sc.Index] = PriceArray[TheirIndex] * Input_Multiplier.GetFloat();

SCString ChartOverlayName = sc.GetStudyNameFromChart(Input_ChartNumberToOverlay.GetChartNumber(), 0);
sc.GraphName.Format("%s Overlay", ChartOverlayName.GetChars());
}

/*=====*/
SCSFExport scsf_UseToolExampleDrawingPitchfork(SCStudyInterfaceRef sc)
{
    // Pitchfork example

    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Pitchfork";
        sc.StudyDescription = "";

        sc.GraphRegion = 0;

        sc.AutoLoop = 0; //No automatic looping

        return;
    }

    s_UseTool Tool;

    Tool.ChartNumber = sc.ChartNumber;
    Tool.DrawingType = DRAWING_PITCHFORK;

    int& r_LineNumber = sc.GetPersistentInt(1);

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

    int BarIndex = max(0, sc.ArraySize - 50);

    //Beginning of middle line
    Tool.BeginDateTime = sc.BaseDateTimeIn[BarIndex];
    Tool.BeginValue = sc.Close[BarIndex];

    //Top line
    Tool.EndDateTime = sc.BaseDateTimeIn[BarIndex+10];
    Tool.EndValue = sc.High[BarIndex];

    //Bottom line
    Tool.ThirdDateTime = sc.BaseDateTimeIn[BarIndex+10];
    Tool.ThirdValue = sc.Low[BarIndex];

    Tool.Color = RGB(0,200,0);
    Tool.LineWidth = 2;
    Tool.RetracementLevels[0] = 100.0f;
    Tool.RetracementLevels[1] = -100.0f;
    Tool.RetracementLevels[2] = 150.0f;
    Tool.RetracementLevels[3] = -150.0f;
    Tool.LevelColor[2] = COLOR_YELLOW;
    Tool.LevelColor[3] = COLOR_MAGENTA;

    Tool.AddMethod = UTAM_ADD_OR_ADJUST;
    sc.UseTool(Tool);
    r_LineNumber = Tool.LineNumber; //Remember line number which has been automatically set

```

```

}

/*=====*/
SCSFExport scsf_UseToolExampleDrawingMarker(SCStudyInterfaceRef sc)
{
    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Marker";
        sc.StudyDescription = "";

        sc.GraphRegion = 0;

        sc.AutoLoop = 0; //No automatic looping

        return;
    }

    // Do data processing
    for (int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize - 1; ++BarIndex)
    {
        //Since the AddMethod = UTAM_ADD_ALWAYS, we do not want to add a marker drawing on the last bar until it is
        //completed. The loop indexing above prevents this, but we also have added this as an extra check.
        if (sc.GetBarHasClosedStatus(BarIndex) != BHCS_BAR_HAS_CLOSED)
            return;

        if (BarIndex % 5)
            continue;

        s_UseTool Tool;

        Tool.ChartNumber = sc.ChartNumber;
        //Tool.LineNumber will be automatically set. No need to set this.
        Tool.DrawingType = DRAWING_MARKER;
        Tool.BeginIndex = BarIndex;
        if (BarIndex % 10 == 0)
        {
            Tool.MarkerType = MARKER_ARROWUP;
            Tool.BeginValue = sc.Low[BarIndex];
        }
        else
        {
            Tool.MarkerType = MARKER_ARROWDOWN;
            Tool.BeginValue = sc.High[BarIndex];
        }

        Tool.Region = 0;
        Tool.Color = RGB(255, 255, 0);
        Tool.MarkerSize = 8;
        Tool.LineWidth = 5;
        Tool.AddMethod = UTAM_ADD_ALWAYS;

        sc.UseTool(Tool); // Here we make the function call to add the marker
    }
}

/*=====*/
SCSFExport scsf_UseToolExampleDrawingLine(SCStudyInterfaceRef sc)
{
    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Line";
        sc.GraphRegion = 0;
    }
}

```



```

    sc.AutoLoop = 0; //No automatic looping

    return;
}

int& r_LineNumber = sc.GetPersistentInt(1);

if (sc.IsFullRecalculation)
{
    //Draw vertical line on prior bar during full recalculation.
    s_UseTool Tool;
    Tool.Clear();
    Tool.ChartNumber = sc.ChartNumber;
    //Tool.LineNumber will be automatically set. No need to set this.
    Tool.DrawingType = DRAWING_LINE;
    int DrawingIndex = sc.ArraySize - 1;
    Tool.BeginDateTime = sc.BaseDateTimeIn[DrawingIndex-20];
    Tool.EndDateTime = sc.BaseDateTimeIn[DrawingIndex];
    Tool.BeginValue = sc.Low[DrawingIndex];
    Tool.EndValue = sc.High[DrawingIndex];
    Tool.Text = "Label 1";
    Tool.TextAlignment = DT_BOTTOM | DT_RIGHT;
    Tool.Region = 0;
    Tool.Color = RGB(255, 255, 0);
    Tool.LineWidth = 5;
    Tool.AddMethod = UTAM_ADD_ALWAYS;

    sc.UseTool(Tool);
    r_LineNumber = Tool.LineNumber;
}
else if (sc.UpdateStartIndex < sc.ArraySize -1)
{
    //Now move the drawing to the last bar when a new bar has been added to the chart
    s_UseTool Tool;
    Tool.Clear();
    Tool.ChartNumber = sc.ChartNumber;
    int DrawingIndex = sc.ArraySize - 1;
    Tool.BeginDateTime = sc.BaseDateTimeIn[DrawingIndex];
    Tool.EndDateTime = sc.BaseDateTimeIn[DrawingIndex];
    Tool.BeginValue = sc.Low[DrawingIndex];
    Tool.EndValue = sc.High[DrawingIndex];
    Tool.AddMethod = UTAM_ADD_OR_ADJUST;
    Tool.LineNumber = r_LineNumber;
    sc.UseTool(Tool);
}

//int Count = sc.DeleteACSCChartDrawing(sc.ChartNumber, TOOL_DELETE_ALL, 0);
}

/*=====*/
SCSFExport scsf_UseToolExampleDrawingPriceRetracement(SCStudyInterfaceRef sc)
{
    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Price Retracement";
        sc.GraphRegion = 0;

        sc.AutoLoop = 0; //No automatic looping

        return;
    }

    int& r_LineNumber = sc.GetPersistentInt(1);

```

```

if (sc.IsFullRecalculation)
{
    s_UseTool Tool;
    //Tool.Clear();
    Tool.ChartNumber = sc.ChartNumber;
    Tool.DrawingType = DRAWING_RETRACEMENT;
    Tool.ExtendLeft = 1; // Extend the horizontal retracement lines to the left of the chart region.

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

    int BarIndex = sc.ArraySize - 100; // 100 bars back and from high to low
    BarIndex = max(BarIndex, 0);
    Tool.BeginDateTime = sc.BaseDateTimeIn[BarIndex];
    Tool.EndDateTime = sc.BaseDateTimeIn[sc.ArraySize - 1];
    Tool.BeginValue = sc.High[sc.ArraySize - 1];
    Tool.EndValue = sc.Low[BarIndex];
    Tool.Color = RGB(255, 0, 255); // Magenta
    Tool.AddMethod = UTAM_ADD_OR_ADJUST;

    for (int LevelIndex = 0; LevelIndex < 16; LevelIndex++)
        Tool.RetractionLevels[LevelIndex] = LevelIndex * 10.0f;

    Tool.ShowPrice = 1;
    Tool.ShowPercent = 1;
    Tool.RoundToTickSize = 0;
    Tool.TextAlignment = DT_VCENTER; // label vertical alignment
    //Tool.AddAsUserDrawnDrawing = 1;

    sc.UseTool(Tool);
    r_LineNumber = Tool.LineNumber; // Remember line number which has been automatically set
}
else if (sc.UpdateStartIndex < sc.ArraySize - 1)
{
    // Update the drawing
    s_UseTool Tool;
    Tool.RetractionLevels[10] = FLT_MAX; // remove the 11th level

    int BarIndex = sc.ArraySize - 100; // 100 bars back and from high to low
    BarIndex = max(BarIndex, 0);
    Tool.BeginDateTime = sc.BaseDateTimeIn[BarIndex];
    Tool.EndDateTime = sc.BaseDateTimeIn[sc.ArraySize - 1];

    Tool.ChartNumber = sc.ChartNumber;
    Tool.AddMethod = UTAM_ADD_OR_ADJUST;
    Tool.LineNumber = r_LineNumber;

    //Tool.UseToolConfigNum = 2;
    //Tool.AddAsUserDrawnDrawing = 1;

    sc.UseTool(Tool);
}
}

/*=====*/
SCSFExport scsf_UseToolExampleDrawingPriceExpansion(SCStudyInterfaceRef sc)
{
    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Price Expansion";
        sc.GraphRegion = 0;
    }
}

```

```

    sc.AutoLoop = 0; //No automatic looping

    return;
}

int& r_LineNumber = sc.GetPersistentInt(1);

if (sc.IsFullRecalculation)
{
    s_UseTool Tool;
    //Tool.Clear();
    Tool.ChartNumber = sc.ChartNumber;
    Tool.DrawingType = DRAWING_PRICE_EXPANSION;
    Tool.ExtendLeft = 1; // Extend the horizontal lines to the left of the chart region.

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

    // Update BarIndex to 40 bars from the end
    int BarIndex = sc.ArraySize - 100; // 100 bars back and from high to low
    BarIndex = max(BarIndex, 0);
    Tool.BeginDateTime = sc.BaseDateTimeIn[BarIndex];
    Tool.EndDateTime = sc.BaseDateTimeIn[BarIndex]; //sc.ArraySize - 1
    Tool.BeginValue = sc.High[BarIndex]; //sc.ArraySize - 1
    Tool.EndValue = sc.Low[BarIndex];
    Tool.Color = RGB(255, 0, 255); // Magenta
    Tool.AddMethod = UTAM_ADD_OR_ADJUST;

    for (int LevelIndex = 0; LevelIndex < min(8, ACSIL_DRAWING_MAX_LEVELS); LevelIndex++)
        Tool.RetacementLevels[LevelIndex] = LevelIndex * 10.0f;

    Tool.ShowPrice = 1;
    Tool.ShowPercent = 1;
    Tool.RoundToTickSize = 0;
    Tool.TextAlignment = DT_VCENTER; // label vertical alignment

    sc.UseTool(Tool);
    r_LineNumber = Tool.LineNumber; //Remember line number which has been automatically set
}
}

/*=====*/
SCSFExport scsf_UseToolExampleDrawingRay(SCStudyInterfaceRef sc)
{
    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Ray";
        sc.GraphRegion = 0;

        sc.AutoLoop = 0; //No automatic looping

        return;
    }

    // Do data processing
    int BarIndex;

    int& r_LineNumber = sc.GetPersistentInt(1);

    s_UseTool Tool;
    //Draw chart drawing on chart study is applied to. It is not possible to draw on another chart unless
    Tool.AddAsUserDrawnDrawing is set to 1.

```

```

Tool.ChartNumber = sc.ChartNumber;

//Tool.AddAsUserDrawnDrawing = 1;

Tool.DrawingType = DRAWING_RAY;
//int &LineNumber2 = sc.GetPersistentInt(2);
if (r_LineNumber != 0)
    Tool.LineNumber = r_LineNumber;

// Set BarIndex to 10 bars from the end
BarIndex = sc.ArraySize - 10;
BarIndex = max(BarIndex, 0);
Tool.BeginDateTime = sc.BaseDateTimeIn[BarIndex];
Tool.EndDateTime = sc.BaseDateTimeIn[sc.ArraySize - 1];
Tool.BeginValue = sc.Low[BarIndex];
Tool.EndValue = sc.High[sc.ArraySize - 1];
Tool.Region = 0;
Tool.Color = RGB(0, 0, 255);
Tool.AddMethod = UTAM_ADD_OR_ADJUST;

sc.UseTool(Tool); // Here we make the function call to add the line

r_LineNumber = Tool.LineNumber; //Remember line number which has been automatically set
}

/*=====*/
SCSFExport scsf_UseToolExampleDrawingText30BarsBack(SCStudyInterfaceRef sc)
{
    // Text example, put 30 bars back

    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Text 30 Bars Back";
        sc.GraphRegion = 0;

        sc.AutoLoop = 0; //No automatic looping

        return;
    }

    // Do data processing
    int BarIndex;

    int& r_LineNumber = sc.GetPersistentInt(1);

    s_UseTool Tool;

    Tool.ChartNumber = sc.ChartNumber;
    Tool.DrawingType = DRAWING_TEXT;
    //int &LineNumber3 = sc.GetPersistentInt(3);
    if (r_LineNumber != 0)
        Tool.LineNumber = r_LineNumber;
    // Set BarIndex to 30 bars from the end
    BarIndex = sc.ArraySize - 30;
    BarIndex = max(BarIndex, 0);
    Tool.BeginDateTime = sc.BaseDateTimeIn[BarIndex];
    Tool.BeginValue = sc.Low[BarIndex];
    Tool.Color = RGB(255, 255, 0); // Yellow
    Tool.Text = "Text Example";
    Tool.FontSize = 16;
    Tool.AddMethod = UTAM_ADD_OR_ADJUST;

    sc.UseTool(Tool);

```

```

    r_LineNumber = Tool.LineNumber;//Remember line number which has been automatically set
}

/*=====*/
SCSFExport scsf_UseToolExampleDrawingVerticalLine(SCStudyInterfaceRef sc)
{
    // Draws a vertical line 20 bars back.

    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Vertical Line";
        sc.GraphRegion = 0;

        sc.AutoLoop = 0;//No automatic looping

        return;
    }

    // Do data processing
    int BarIndex;

    int& r_LineNumber = sc.GetPersistentInt(1);

    s_UseTool Tool;

    Tool.ChartNumber = sc.ChartNumber;
    Tool.DrawingType = DRAWING_VERTICALLINE;
    //int &LineNumber5 = sc.GetPersistentInt(5);
    if (r_LineNumber != 0)
        Tool.LineNumber = r_LineNumber;
    // Update BarIndex to 20 bars from the end
    BarIndex = sc.ArraySize - 20;
    BarIndex = max(BarIndex, 0);
    Tool.BeginDateTime = sc.BaseDateTimeIn[BarIndex];
    Tool.Color = RGB(255, 0, 0); // Red
    Tool.AddMethod = UTAM_ADD_OR_ADJUST;

    sc.UseTool(Tool);

    r_LineNumber = Tool.LineNumber;//Remember line number which has been automatically set

    // If the vertical line exists, add a message to the Message Log
    if (sc.ChartDrawingExists(sc.ChartNumber, Tool.LineNumber))
    {
        sc.AddMessageToLog("Vertical line exists.", 0);
    }
}

/*=====*/
SCSFExport scsf_UseToolExampleDrawingTextFillSpace(SCStudyInterfaceRef sc)
{
    // Display some text in the fill space

    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Text Fill Space";
        sc.GraphRegion = 0;

        sc.AutoLoop = 0;//No automatic looping

        return;
    }
}

```

```

int& r_LineNumber = sc.GetPersistentInt(1);

s_UseTool Tool;

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

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

Tool.BeginDateTime = -1;
//Tool.BeginIndex = sc.ArraySize + 5;
Tool.BeginValue = sc.Low[sc.ArraySize - 1];
Tool.Color = COLOR_WHITE;
Tool.Text = "Example Text";
Tool.AddMethod = UTAM_ADD_OR_ADJUST;

sc.UseTool(Tool);

r_LineNumber = Tool.LineNumber; //Remember line number which has been automatically set
}

/*=====*/
SCSFExport scsf_UseToolExampleDrawingMinus1(SCStudyInterfaceRef sc)
{
    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Minus 1";
        sc.GraphRegion = 0;

        sc.AutoLoop = 0; //No automatic looping

        return;
    }

    int& r_LineNumber = sc.GetPersistentInt(1);

    s_UseTool Tool;

    Tool.ChartNumber = sc.ChartNumber;
    Tool.LineNumber = r_LineNumber;
    Tool.Color = COLOR_CYAN;
    Tool.Text = "Minus 1 - Changed";
    Tool.AddMethod = UTAM_ADD_OR_ADJUST;

    sc.UseTool(Tool);

    r_LineNumber = Tool.LineNumber;
}

/*=====*/
SCSFExport scsf_UseToolExampleDrawingTextAtTheEndOfTheChart(SCStudyInterfaceRef sc)
{
    // Display some text after the end of the chart

    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Text At The End Of The Chart";
        sc.GraphRegion = 0;

        sc.AutoLoop = 0; //No automatic looping

        return;
    }

```

```

}

// Do data processing
int BarIndex;

int& r_LineNumber = sc.GetPersistentInt(1);

s_UseTool Tool;

Tool.ChartNumber = sc.ChartNumber;
Tool.DrawingType = DRAWING_TEXT;
//int &LineNumber7 = sc.GetPersistentInt(7);
if (r_LineNumber != 0)
    Tool.LineNumber = r_LineNumber;
Tool.BeginDateTime = -2;
// Update BarIndex to 20 bars from the end
BarIndex = sc.ArraySize - 20;
BarIndex = max(BarIndex, 0);
Tool.BeginValue = sc.Low[BarIndex];
Tool.Color = RGB(255, 255, 255); // White;
Tool.Text = "Minus 2";
Tool.AddMethod = UTAM_ADD_OR_ADJUST;
Tool.ReverseTextColor = 1; // true

sc.UseTool(Tool);

r_LineNumber = Tool.LineNumber; // Remember line number which has been automatically set
}

/*=====*/
SCSFExport scsf_UseToolExampleDrawingArrow(SCStudyInterfaceRef sc)
{
    // Add an arrow

    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Arrow";
        sc.GraphRegion = 0;

        sc.AutoLoop = 0; // No automatic looping

        return;
    }

    // Do data processing
    int BarIndex;

    int& r_LineNumber = sc.GetPersistentInt(1);

    s_UseTool Tool;

    Tool.ChartNumber = sc.ChartNumber;
    Tool.DrawingType = DRAWING_ARROW;
    //int &LineNumber8 = sc.GetPersistentInt(8);
    if (r_LineNumber != 0)
        Tool.LineNumber = r_LineNumber;
    // Update BarIndex to 30 bars from the end
    BarIndex = sc.ArraySize - 30;
    BarIndex = max(BarIndex, 0);
    Tool.BeginDateTime = sc.BaseDateTimeIn[BarIndex];
    Tool.EndDateTime = sc.BaseDateTimeIn[sc.ArraySize - 11];
    Tool.BeginValue = sc.Low[BarIndex + 10];
    Tool.EndValue = sc.High[sc.ArraySize - 1];
    Tool.Color = RGB(255, 255, 0); // Yellow

```

```

Tool.AddMethod = UTAM_ADD_OR_ADJUST;

Tool.ArrowHeadSize = 5.0; // ratio when not fixed
Tool.FixedSizeArrowHead = 0;

sc.UseTool(Tool);

r_LineNumber = Tool.LineNumber; //Remember line number which has been automatically set
}

/*=====*/
SCSFExport scsf_UseToolExampleDrawingHorizontalLineLastPrice(SCStudyInterfaceRef sc)
{
    // Draw a horizontal line at the last price.

    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Horizontal Line Last Price";
        sc.GraphRegion = 0;

        sc.AutoLoop = 0; //No automatic looping

        return;
    }

    int& r_LineNumber = sc.GetPersistentInt(1);

    s_UseTool Tool;

    Tool.ChartNumber = sc.ChartNumber;
    Tool.DrawingType = DRAWING_HORIZONTALLINE;
    Tool.DisplayHorizontalLineValue = 1;
    //int &LineNumber9 = sc.GetPersistentInt(9);
    if (r_LineNumber != 0)
        Tool.LineNumber = r_LineNumber;
    Tool.BeginValue = sc.Close[sc.ArraySize - 1];
    Tool.Color = RGB(255, 0, 255); // Magenta
    Tool.LineWidth = 4;
    Tool.AddMethod = UTAM_ADD_OR_ADJUST;

    sc.UseTool(Tool);

    r_LineNumber = Tool.LineNumber; //Remember line number which has been automatically set
}

/*=====*/
SCSFExport scsf_UseToolExampleDrawingHorizontalLineAtLowOfLast30Bars(SCStudyInterfaceRef sc)
{
    // Draw a horizontal line at the low of the last 30 bars.

    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Horizontal Line At Low Of Last 30 Bars";
        sc.GraphRegion = 0;

        sc.AutoLoop = 0; //No automatic looping

        return;
    }

    // Do data processing
    int BarIndex;

```



```

int& r_LineNumber = sc.GetPersistentInt(1);

s_UseTool Tool;

Tool.ChartNumber = sc.ChartNumber;
Tool.DrawingType = DRAWING_HORIZONTALLINE;
//int &LineNumber10 = sc.GetPersistentInt(10);
if (r_LineNumber != 0)
    Tool.LineNumber = r_LineNumber;
// Update BarIndex to 30 bars from the end
BarIndex = sc.ArraySize - 30;
BarIndex = max(BarIndex, 0);
Tool.BeginValue = sc.GetLowest(sc.Low, sc.ArraySize - 1, sc.ArraySize - BarIndex);
Tool.Color = RGB(127, 0, 255); // Purple
Tool.LineWidth = 1; // LineWidth must be 1 to use non-solid line styles
Tool.LineStyle = LINESTYLE_DOT;
Tool.AddMethod = UTAM_ADD_OR_ADJUST;

sc.UseTool(Tool);

r_LineNumber = Tool.LineNumber; //Remember line number which has been automatically set
}

/*=====*/
SCSFExport scsf_UseToolExampleDrawingText40BarsBack(SCStudyInterfaceRef sc)
{
    // Numbers as text example. Put 40 bars back

    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Text 40 Bars Back";
        sc.GraphRegion = 0;

        sc.AutoLoop = 0; //No automatic looping

        return;
    }

    // Do data processing
    int BarIndex;

    int& r_LineNumber = sc.GetPersistentInt(1);

    s_UseTool Tool;

    Tool.ChartNumber = sc.ChartNumber;
    Tool.DrawingType = DRAWING_TEXT;
    //int &LineNumber11 = sc.GetPersistentInt(11);
    if (r_LineNumber != 0)
        Tool.LineNumber = r_LineNumber;
    // Update BarIndex to 40 bars from the end
    BarIndex = sc.ArraySize - 40;
    BarIndex = max(BarIndex, 0);
    Tool.BeginDateTime = sc.BaseDateTimeIn[BarIndex];
    Tool.BeginValue = sc.High[BarIndex];
    Tool.Color = RGB(255, 255, 0); // Yellow
    Tool.Text.Format("High Value: %.3f", sc.High[BarIndex]);
    Tool.AddMethod = UTAM_ADD_OR_ADJUST;

    sc.UseTool(Tool);

    r_LineNumber = Tool.LineNumber; //Remember line number which has been automatically set
}

```

```

/*=====*/
SCSFExport scsf_UseToolExampleChangeFontBackColor(SCStudyInterfaceRef sc)
{
    // Change font back color

    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Change Font Back Color";
        sc.GraphRegion = 0;

        sc.AutoLoop = 0; //No automatic looping

        return;
    }

    int& r_LineNumber = sc.GetPersistentInt(1);

    s_UseTool Tool;

    Tool.ChartNumber = sc.ChartNumber;
    Tool.LineNumber = r_LineNumber;
    Tool.FontBackColor = RGB(255, 255, 255);
    Tool.AddMethod = UTAM_ADD_OR_ADJUST;

    sc.UseTool(Tool);

    r_LineNumber = Tool.LineNumber;
}

/*=====*/
SCSFExport scsf_UseToolExampleChangeFontColorBackToTransparent(SCStudyInterfaceRef sc)
{
    // Change font back color back to transparent

    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Change Font Color Back To Transparent";
        sc.GraphRegion = 0;

        sc.AutoLoop = 0; //No automatic looping

        return;
    }

    int& r_LineNumber = sc.GetPersistentInt(1);

    s_UseTool Tool;

    Tool.ChartNumber = sc.ChartNumber;
    Tool.LineNumber = r_LineNumber;
    Tool.TransparentLabelBackground = 1;
    Tool.AddMethod = UTAM_ADD_OR_ADJUST;

    sc.UseTool(Tool);

    r_LineNumber = Tool.LineNumber;
}

/*=====*/
SCSFExport scsf_UseToolExampleRectangleHighlight(SCStudyInterfaceRef sc)
{
    // Draw a rectangle highlight

```

```

// Set configuration variables
if (sc.SetDefaults)
{
    sc.GraphName = "UseTool Example: Rectangle Highlight";
    sc.GraphRegion = 0;

    sc.AutoLoop = 0; //No automatic looping

    return;
}

if (sc.LastCallToFunction)
    return;

// Do data processing
int BarIndex = 0;

int& r_LineNumber = sc.GetPersistentInt(1);

s_UseTool Tool;

//Tool.ChartNumber = sc.ChartNumber;
Tool.DrawingType = DRAWING_RECTANGLEHIGHLIGHT;

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

// Update BarIndex to 30 bars from the end
BarIndex = max(sc.ArraySize - 25, 0);
Tool.BeginDateTime = sc.BaseDateTimeIn[BarIndex];
BarIndex = max(sc.ArraySize - 15, 0);
Tool.EndDateTime = sc.BaseDateTimeIn[BarIndex];
Tool.BeginValue = sc.GetHighest(sc.Low, BarIndex, 10);
Tool.EndValue = sc.GetLowest(sc.Low, BarIndex, 10);
Tool.Color = RGB(255, 0, 0); // Red
Tool.LineWidth = 1; //To see the outline this must be 1 or greater.
Tool.SecondaryColor = RGB(0, 255, 0);
Tool.TransparencyLevel = 50;
Tool.AddMethod = UTAM_ADD_OR_ADJUST;

// Add rectangle drawing to chart number 2.
Tool.AddAsUserDrawnDrawing = 1;
Tool.ChartNumber = 2;

sc.UseTool(Tool);

r_LineNumber = Tool.LineNumber; //Remember line number which has been automatically set
}

/*=====*/
SCSFExport scsf_UseToolExampleChangeRectangleHighlight(SCStudyInterfaceRef sc)
{
    // Change rectangle highlight

    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Change Rectangle Highlight";
        sc.GraphRegion = 0;

        sc.AutoLoop = 0; //No automatic looping

        return;
    }
}

```

```

int& r_LineNumber = sc.GetPersistentInt(1);

s_UseTool Tool;

Tool.ChartNumber = sc.ChartNumber;
Tool.LineNumber = r_LineNumber;
Tool.Color = RGB(255, 200, 0);
Tool.LineWidth = 3;
Tool.SecondaryColor = RGB(0, 0, 255);
Tool.AddMethod = UTAM_ADD_OR_ADJUST;

sc.UseTool(Tool);

r_LineNumber = Tool.LineNumber;
}

/*=====*/
SCSFExport scsf_UseToolExampleDrawingPriceProjection(SCStudyInterfaceRef sc)
{
    // Custom Projection levels example
    // 60 bars back and from high to low, project from 40 bars back

    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Custom Projection Levels";
        sc.GraphRegion = 0;

        sc.AutoLoop = 0; //No automatic looping

        return;
    }

    // Do data processing
    int BarIndex;

    int& r_LineNumber = sc.GetPersistentInt(1);

    s_UseTool Tool;

    Tool.ChartNumber = sc.ChartNumber;
    Tool.DrawingType = DRAWING_PRICE_PROJECTION;
    //int &LineNumber13 = sc.GetPersistentInt(13);
    if (r_LineNumber != 0)
        Tool.LineNumber = r_LineNumber;
    // Update BarIndex to 60 bars from the end
    BarIndex = sc.ArraySize - 60;
    BarIndex = max(BarIndex, 0);
    Tool.BeginDateTime = sc.BaseDateTimeIn[BarIndex];
    Tool.BeginValue = sc.High[BarIndex];
    BarIndex = sc.ArraySize - 50;
    BarIndex = max(BarIndex, 0);
    Tool.EndDateTime = sc.BaseDateTimeIn[BarIndex];
    Tool.EndValue = sc.Low[BarIndex];
    BarIndex = sc.ArraySize - 40;
    BarIndex = max(BarIndex, 0);
    Tool.ThirdDateTime = sc.BaseDateTimeIn[BarIndex];
    Tool.ThirdValue = sc.High[BarIndex];
    Tool.Color = COLOR_LIGHTSTEELBLUE;
    Tool.AddMethod = UTAM_ADD_OR_ADJUST;

    for (int LevelIndex = 0; LevelIndex < 16; LevelIndex++)
        Tool.RetacementLevels[LevelIndex] = LevelIndex * 10.0f;

    sc.UseTool(Tool);
}

```

```

    r_LineNumber = Tool.LineNumber;//Remember line number which has been automatically set
}

/*=====*/
SCSFExport scsf_UseToolExampleDrawingPitchforkV2(SCStudyInterfaceRef sc)
{
    // Pitchfork example

    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Pitchfork V2";
        sc.GraphRegion = 0;

        sc.AutoLoop = 0;//No automatic looping

        return;
    }

    // Do data processing
    int BarIndex;

    int& r_LineNumber = sc.GetPersistentInt(1);

    s_UseTool Tool;

    Tool.ChartNumber = sc.ChartNumber;
    Tool.DrawingType = DRAWING_PITCHFORK;
    //int &LineNumber14 = sc.GetPersistentInt(14);
    if (r_LineNumber != 0)
        Tool.LineNumber = r_LineNumber;

    BarIndex = max(0, sc.ArraySize - 50);

    //Beginning of middle line
    Tool.BeginDateTime = sc.BaseDateTimeIn[BarIndex];
    Tool.BeginValue = sc.Close[BarIndex];

    //Beginning of top line
    Tool.EndDateTime = sc.BaseDateTimeIn[BarIndex];
    Tool.EndValue = sc.High[BarIndex];

    //Beginning of bottom line
    Tool.ThirdDateTime = sc.BaseDateTimeIn[BarIndex];
    Tool.ThirdValue = sc.Low[BarIndex];

    Tool.Color = RGB(0, 200, 0);
    Tool.LineWidth = 5;
    Tool.AddMethod = UTAM_ADD_OR_ADJUST;
    Tool.RetacementLevels[0] = 100.0f;
    Tool.RetacementLevels[1] = -100.0f;
    Tool.RetacementLevels[2] = 150.0f;
    Tool.RetacementLevels[3] = -150.0f;
    Tool.LevelColor[2] = COLOR_YELLOW;
    Tool.LevelColor[3] = COLOR_MAGENTA;

    sc.UseTool(Tool);

    r_LineNumber = Tool.LineNumber;//Remember line number which has been automatically set
}

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

```

```

// Parallel lines example

// Set configuration variables
if (sc.SetDefaults)
{
    sc.GraphName = "UseTool Example: Parallel Lines";
    sc.GraphRegion = 0;

    sc.AutoLoop = 0; //No automatic looping

    return;
}

// Do data processing
int BarIndex;

int& r_LineNumber = sc.GetPersistentInt(1);

int& r_LevelColor = sc.GetPersistentInt(2);
int& r_PrimaryParallelLinesColor = sc.GetPersistentInt(3);

if (sc.IsFullRecalculation)
{
    r_LevelColor = COLOR_GOLDENROD;
    r_PrimaryParallelLinesColor = COLOR_MAGENTA;
}

s_UseTool Tool;

Tool.ChartNumber = sc.ChartNumber;
Tool.DrawingType = DRAWING_PARALLEL_LINES;

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

BarIndex = max(0, sc.ArraySize - 90);
Tool.BeginDateTime = sc.BaseDateTimeIn[BarIndex];
Tool.BeginValue = sc.High[BarIndex];
BarIndex = max(0, sc.ArraySize - 70);
Tool.EndDateTime = sc.BaseDateTimeIn[BarIndex];
Tool.EndValue = sc.High[BarIndex];
BarIndex = max(0, sc.ArraySize - 80);
Tool.ThirdDateTime = sc.BaseDateTimeIn[BarIndex];
Tool.ThirdValue = sc.Low[BarIndex];
Tool.Color = r_PrimaryParallelLinesColor;
Tool.LineWidth = 3;
Tool.AddMethod = UTAM_ADD_OR_ADJUST;

Tool.RetracementLevels[0] = 200.0f;
Tool.RetracementLevels[1] = -200.0f;
Tool.LevelColor[0] = r_LevelColor;
Tool.LevelColor[1] = r_LevelColor;
Tool.LevelWidth[0] = 3;
Tool.LevelWidth[1] = 3;

sc.UseTool(Tool);

r_LineNumber = Tool.LineNumber; //Remember line number which has been automatically set

r_LevelColor += 10;
r_PrimaryParallelLinesColor += 10;
}

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

```

```

{
    // Linear Regression example

    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Linear Regression";
        sc.GraphRegion = 0;

        sc.AutoLoop = 0; //No automatic looping

        return;
    }

    // Do data processing
    int BarIndex;

    int& r_LineNumber = sc.GetPersistentInt(1);

    s_UseTool Tool;

    Tool.ChartNumber = sc.ChartNumber;
    Tool.DrawingType = DRAWING_LINEAR_REGRESSION;
    //int &LineNumber16 = sc.GetPersistentInt(16);
    if (r_LineNumber != 0)
        Tool.LineNumber = r_LineNumber;
    BarIndex = max(0, sc.ArraySize - 90);
    Tool.BeginDateTime = sc.BaseDateTimeIn[BarIndex];
    Tool.BeginValue = sc.Open[BarIndex];
    BarIndex = max(0, sc.ArraySize - 60);
    Tool.EndDateTime = sc.BaseDateTimeIn[BarIndex];
    Tool.EndValue = sc.Open[BarIndex];
    Tool.Color = RGB(0, 128, 0);
    Tool.LineWidth = 3;
    Tool.AddMethod = UTAM_ADD_ONLY_IF_NEW; // UTAM_ADD_OR_ADJUST;
    Tool.RetacementLevels[0] = 2.0f;
    Tool.RetacementLevels[1] = -2.0f;
    Tool.RetacementLevels[2] = 3.0f;
    Tool.RetacementLevels[3] = -3.0f;
    Tool.LevelColor[0] = COLOR_GOLDENROD;
    Tool.LevelColor[1] = COLOR_GOLDENROD;
    Tool.ExtendRight = 1;

    sc.UseTool(Tool);

    r_LineNumber = Tool.LineNumber; //Remember line number which has been automatically set
}

/*=====*/
SCSFExport scsf_UseToolExampleDrawingTimeExpansion(SCStudyInterfaceRef sc)
{
    // Time Expansion example

    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Time Expansion";
        sc.GraphRegion = 0;

        sc.AutoLoop = 0; //No automatic looping

        return;
    }

    // Do data processing

```

```

int BarIndex;

int& r_LineNumber = sc.GetPersistentInt(1);

s_UseTool Tool;

Tool.ChartNumber = sc.ChartNumber;
Tool.DrawingType = DRAWING_TIME_EXPANSION;
//int &LineNumber17 = sc.GetPersistentInt(17);
if (r_LineNumber != 0)
    Tool.LineNumber = r_LineNumber;
BarIndex = max(0, sc.ArraySize - 90);
Tool.BeginDateTime = sc.BaseDateTimeIn[BarIndex];
Tool.BeginValue = sc.Close[BarIndex];
BarIndex = max(0, sc.ArraySize - 80);
Tool.EndDateTime = sc.BaseDateTimeIn[BarIndex];
Tool.EndValue = sc.Close[BarIndex];
Tool.Color = COLOR_LIGHTPINK;
Tool.AddMethod = UTAM_ADD_OR_ADJUST;
Tool.RetacementLevels[0] = 100.0f;
Tool.RetacementLevels[1] = 200.0f;
Tool.RetacementLevels[2] = 300.0f;
Tool.RetacementLevels[3] = 400.0f;
Tool.RetacementLevels[3] = 800.0f;
Tool.RetacementLevels[3] = 1000.0f;

Tool.TimeExpTopLabel1 = TIME_EXP_LABEL_PERCENTBARS;
Tool.TimeExpBottomLabel1 = TIME_EXP_LABEL_TIME;
Tool.TimeExpVerticals = TIME_EXP_COMPRESSED;

Tool.TransparentLabelBackground = 1;

sc.UseTool(Tool);

r_LineNumber = Tool.LineNumber; //Remember line number which has been automatically set
}

/*=====*/
SCSFExport scsf_UseToolExampleDrawingMarkerV2(SCStudyInterfaceRef sc)
{
    // Marker example

    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Marker V2";
        sc.GraphRegion = 0;

        sc.AutoLoop = 0; //No automatic looping

        return;
    }

    // Do data processing
    int BarIndex;

    int& r_LineNumber = sc.GetPersistentInt(1);

    s_UseTool Tool;

    Tool.ChartNumber = sc.ChartNumber;
    Tool.DrawingType = DRAWING_MARKER;
    //int &LineNumber18 = sc.GetPersistentInt(18);
    if (r_LineNumber != 0)
        Tool.LineNumber = r_LineNumber;

```



```

BarIndex = max(0, sc.ArraySize - 35);
Tool.BeginDateTime = sc.BaseDateTimeIn[BarIndex];
Tool.BeginValue = sc.High[BarIndex];
Tool.Color = RGB(0, 200, 200);
Tool.AddMethod = UTAM_ADD_OR_ADJUST;
Tool.MarkerType = MARKER_X;
Tool.MarkerSize = 8;
Tool.LineWidth = 5;

sc.UseTool(Tool);

r_LineNumber = Tool.LineNumber; //Remember line number which has been automatically set
}

/*=====*/
SCSFExport scsf_UseToolExampleDrawingCalculator(SCStudyInterfaceRef sc)
{
    // Draws a chart calculator line over the last 20 bars.

    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Calculator";
        sc.GraphRegion = 0;

        sc.AutoLoop = 0; //No automatic looping

        return;
    }

    // Do data processing
    int BarIndex;

    int& r_LineNumber = sc.GetPersistentInt(1);

    s_UseTool Tool;

    Tool.ChartNumber = sc.ChartNumber;
    Tool.DrawingType = DRAWING_CALCULATOR;
    //int &LineNumber19 = sc.GetPersistentInt(19);
    if (r_LineNumber != 0)
        Tool.LineNumber = r_LineNumber;
    // Set BarIndex to 20 bars from the end
    BarIndex = max(0, sc.ArraySize - 20);
    //Tool.BeginDateTime = sc.BaseDateTimeIn[BarIndex];
    //Tool.EndDateTime = sc.BaseDateTimeIn[sc.ArraySize - 1];
    Tool.BeginIndex = BarIndex;
    Tool.EndIndex = sc.ArraySize - 1;

    Tool.BeginValue = sc.Low[BarIndex];
    Tool.EndValue = sc.High[sc.ArraySize - 1];
    Tool.Region = 0;
    Tool.Color = RGB(255, 255, 255);
    Tool.LineWidth = 4;
    Tool.AddMethod = UTAM_ADD_OR_ADJUST;

    Tool.ShowEndPointPrice = 1;
    Tool.ShowAngle = 1;
    Tool.ShowPriceDifference = 1;
    Tool.ShowNumberOfBars = 1;
    Tool.ShowTimeDifference = 1;
    Tool.MultiLineLabel = 1;

    sc.UseTool(Tool); // Here we make the function call to add the line

```

```

    r_LineNumber = Tool.LineNumber; //Remember line number which has been automatically set
}

/*=====*/
SCSFExport scsf_UseToolExampleDrawingCycle(SCStudyInterfaceRef sc)
{
    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Cycle";
        sc.GraphRegion = 1;

        sc.AutoLoop = 0; //No automatic looping

        return;
    }

    int& r_LineNumber = sc.GetPersistentInt(1);

    if (sc.IsFullRecalculation)
    {
        r_LineNumber = 0;
    }

    //Add or update cycle drawing.
    s_UseTool Tool;
    Tool.Clear();
    Tool.ChartNumber = sc.ChartNumber;
    Tool.Region = 1;
    if (r_LineNumber != 0)
        Tool.LineNumber = r_LineNumber;
    Tool.DrawingType = DRAWING_CYCLE;
    Tool.BeginDateTime = sc.BaseDateTimeIn[sc.ArraySize - 100];
    Tool.EndDateTime = sc.BaseDateTimeIn[sc.ArraySize - 90];
    Tool.Color = RGB(255, 255, 0);
    Tool.LineWidth = 5;
    Tool.NumCycles = 5;
    Tool.AddMethod = UTAM_ADD_OR_ADJUST;

    if (sc.UseTool(Tool) > 0)
        r_LineNumber = Tool.LineNumber;
}

/*=====*/
SCSFExport scsf_UseToolExampleText(SCStudyInterfaceRef sc)
{
    // Use Indexes instead of DateTime Text example.

    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Text";
        sc.GraphRegion = 0;

        sc.AutoLoop = 0; //No automatic looping

        return;
    }

    // Do data processing
    int BarIndex;

    int& r_LineNumber = sc.GetPersistentInt(1);

    s_UseTool Tool;

```

```

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

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

// Update BarIndex to 30 bars from the end
BarIndex = max(0, sc.ArraySize - 30);
//Tool.BeginIndex = BarIndex;
Tool.BeginDateTime = 150;
Tool.BeginValue = sc.High[BarIndex];
Tool.Color = COLOR_WHITE;
Tool.Text.Format("Text On Index: %i", BarIndex);
Tool.TextAlignment = DT_RIGHT ;
Tool.AddMethod = UTAM_ADD_OR_ADJUST;

sc.UseTool(Tool);

r_LineNumber = Tool.LineNumber; //Remember line number which has been automatically set
}

/*=====*/
SCSFExport scsf_UseToolExampleText75BarsBack(SCStudyInterfaceRef sc)
{
    // User Drawn Drawing example, put 75 bars back

    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Text 75 Bars Back";
        sc.GraphRegion = 0;

        sc.AutoLoop = 0; //No automatic looping

        return;
    }

    // Do data processing
    int BarIndex;

    int& r_TextDrawingLineNumber = sc.GetPersistentInt(1);

    if (r_TextDrawingLineNumber != 0 && sc.LastCallToFunction)
    {
        // Be sure to remove the Text drawing added as a user drawn drawing
        sc.DeleteUserDrawnACSDrawing(sc.ChartNumber, r_TextDrawingLineNumber);
        return; //No further processing needed in this case.
    }

    int& r_LineNumber = sc.GetPersistentInt(2);

    s_UseTool Tool;

    Tool.ChartNumber = sc.ChartNumber;
    Tool.DrawingType = DRAWING_TEXT;
    if (r_LineNumber != 0)
        Tool.LineNumber = r_LineNumber;
    // Must set user drawn flag
    Tool.AddAsUserDrawnDrawing = 1;
    BarIndex = max(0, sc.ArraySize - 75);
    Tool.BeginIndex = BarIndex;
    Tool.BeginValue = sc.High[BarIndex];
    Tool.Color = COLOR_KHAKI;
    Tool.Text = "User Drawn Text Example, Move Me";

```

```

Tool.FontSize = 18;
// do not keep adjusting
Tool.AddMethod = UTAM_ADD_ONLY_IF_NEW;

sc.UseTool(Tool);

//Remember the line number which has been automatically assigned
r_LineNumber = Tool.LineNumber;

// If the user drawn line exists, we will add a message to the Message Log
if (sc.UserDrawnChartDrawingExists(sc.ChartNumber, r_TextDrawingLineNumber))
{
    sc.AddMessageToLog("User drawn Text exists.", 0);
}

if (sc.ChartDrawingExists(sc.ChartNumber, r_TextDrawingLineNumber))
{
    sc.AddMessageToLog("Error: User drawn Text should not exist when calling the ChartDrawingExists().", 1);
}
}

/*=====*/
SCSFExport scsf_UseToolExampleDrawingFanFibonacci(SCStudyInterfaceRef sc)
{
    // Fibonacci Fan example

    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Fibonacci Fan";
        sc.GraphRegion = 0;

        sc.AutoLoop = 0; //No automatic looping

        return;
    }

    // Do data processing
    int BarIndex;

    int& r_LineNumber = sc.GetPersistentInt(1);

    s_UseTool Tool;

    Tool.ChartNumber = sc.ChartNumber;
    Tool.DrawingType = DRAWING_FAN_FIBONACCI;
    //int &LineNumber21 = sc.GetPersistentInt(21);
    if (r_LineNumber != 0)
        Tool.LineNumber = r_LineNumber;
    // Update BarIndex to 75 bars from the end
    BarIndex = sc.ArraySize - 75;
    BarIndex = max(BarIndex, 0);
    Tool.BeginDateTime = sc.BaseDateTimeIn[BarIndex];
    Tool.BeginValue = sc.High[BarIndex];
    BarIndex = sc.ArraySize - 70;
    BarIndex = max(BarIndex, 0);
    Tool.EndDateTime = sc.BaseDateTimeIn[BarIndex];
    Tool.EndValue = sc.Low[BarIndex];
    Tool.ShowLabels = 1;
    Tool.ShowLabelsAtEnd = 1;
    //Tool.AllLevelsSameColorStyle = 1;
    Tool.Color = COLOR_RED;
    Tool.AddMethod = UTAM_ADD_OR_ADJUST;
    Tool.RetracementLevels[0] = 50.0f;
    Tool.LevelColor[0] = COLOR_GREEN;

```

```

Tool.RetracementLevels[1] = 61.8f;
Tool.LevelColor[1] = COLOR_LIGHTGREEN;
Tool.RetracementLevels[2] = 100.0f;
Tool.LevelColor[2] = COLOR_LIGHTBLUE;
Tool.RetracementLevels[3] = 127.2f;
Tool.LevelColor[3] = COLOR_LIGHTSTEELBLUE;
Tool.RetracementLevels[4] = 161.8f;
Tool.LevelColor[4] = COLOR_DARKCYAN;
Tool.RetracementLevels[5] = 200.0f;
Tool.LevelColor[5] = COLOR_GOLD;

sc.UseTool(Tool);

r_LineNumber = Tool.LineNumber;//Remember line number which has been automatically set
}

/*=====*/
SCSFExport scsf_UseToolExampleDrawingHorizontalLine(SCStudyInterfaceRef sc)
{
    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Horizontal Line";
        sc.GraphRegion = 0;

        sc.AutoLoop = 0; //No automatic looping

        return;
    }

    // Do data processing
    int BarIndex;

    int& r_LineNumber = sc.GetPersistentInt(1);

    s_UseTool Tool;

    Tool.ChartNumber = sc.ChartNumber;
    Tool.DrawingType = DRAWING_HORIZONTALLINE;
    //int &LineNumber22 = sc.GetPersistentInt(22);
    if (r_LineNumber != 0)
        Tool.LineNumber = r_LineNumber;
    // Set BarIndex to 20 bars from the end
    BarIndex = sc.ArraySize - 20;
    BarIndex = max(BarIndex, 0);
    Tool.BeginValue = sc.Close[BarIndex];
    Tool.Color = RGB(255, 0, 0);
    Tool.LineWidth = 10;
    Tool.Region = 0;
    Tool.UseToolConfigNum = 1;
    Tool.AddMethod = UTAM_ADD_OR_ADJUST;

    sc.UseTool(Tool); // Here we make the function call to add the line

    r_LineNumber = Tool.LineNumber;//Remember line number which has been automatically set
}

/*=====*/
SCSFExport scsf_UseToolExampleDrawingHorizontalLineNonExtended(SCStudyInterfaceRef sc)
{
    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Horizontal Line Non-Extended";
        sc.GraphRegion = 0;
    }
}

```

```

    sc.AutoLoop = 0; //No automatic looping

    return;
}

// Do data processing

int& r_LineNumber = sc.GetPersistentInt(1);

s_UseTool Tool;

Tool.ChartNumber = sc.ChartNumber;
Tool.DrawingType = DRAWING_HORIZONTAL_LINE_NON_EXTENDED;

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

// Set BarIndex to 50 bars from the end
int BarIndex = sc.ArraySize - 50;
BarIndex = max(BarIndex, 0);

Tool.BeginValue = sc.Close[BarIndex];

Tool.BeginDateTime = sc.BaseDateTimeIn[BarIndex];
Tool.EndDateTime = sc.BaseDateTimeIn[sc.ArraySize - 3];

Tool.Color = RGB(255, 0, 0);
Tool.LineWidth = 10;
Tool.Region = 0;
Tool.AddMethod = UTAM_ADD_OR_ADJUST;

sc.UseTool(Tool); // Here we make the function call to add the line

r_LineNumber = Tool.LineNumber; //Remember line number which has been automatically set
}

/*=====*/
SCSFExport scsf_UseToolExampleDrawingHorizontalLineV2(SCStudyInterfaceRef sc)
{
    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Horizontal Line V2";
        sc.GraphRegion = 0;

        sc.AutoLoop = 0; //No automatic looping

        return;
    }

    int& r_LineNumber = sc.GetPersistentInt(1);

    if (sc.IsFullRecalculation)
    {
        if (r_LineNumber != 0)
        {
            sc.DeleteUserDrawnACSDrawing(sc.ChartNumber, r_LineNumber);
            r_LineNumber = 0;
        }

        //Draw horizontal line during full recalculation.
        s_UseTool Tool;

```

```

Tool.ChartNumber = sc.ChartNumber;

Tool.DrawingType = DRAWING_HORIZONTALLINE;
Tool.DisplayHorizontalLineValue = 1;

int DrawingIndex = sc.ArraySize - 4;
Tool.BeginValue = sc.Close[sc.ArraySize - 1];
Tool.Region = 0;
Tool.Color = RGB(255, 0, 255); // Magenta
Tool.LineWidth = 4;
Tool.AddMethod = UTAM_ADD_ALWAYS;
Tool.AddAsUserDrawnDrawing = 1;

sc.UseTool(Tool);

r_LineNumber = Tool.LineNumber;
}
else if (sc.UpdateStartIndex < sc.ArraySize - 1)//When there is a new bar
{
    //Now move the drawing to the current closing price when a new bar has been added to the chart
    s_UseTool Tool;

    Tool.ChartNumber = sc.ChartNumber;
    Tool.BeginValue = sc.Close[sc.ArraySize - 1];
    Tool.AddMethod = UTAM_ADD_OR_ADJUST;
    Tool.LineNumber = r_LineNumber;
    Tool.AddAsUserDrawnDrawing = 1;
    sc.UseTool(Tool);

    r_LineNumber = Tool.LineNumber;
}
}

/*=====*/
SCSFExport scsf_UseToolExampleMultipleDrawingsSameIndex(SCStudyInterfaceRef sc)
{
    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Multiple Drawings At Same Index";
        sc.StudyDescription = "";

        sc.GraphRegion = 0;

        sc.AutoLoop = 1; //Automatic looping

        return;
    }

    //Set up persistent line number variables
    int &LineNumber_1 = sc.GetPersistentInt(1);
    int &LineNumber_2 = sc.GetPersistentInt(2);

    if (sc.Index == sc.ArraySize - 1) //only do this at last bar
    {
        s_UseTool Tool;

        Tool.ChartNumber = sc.ChartNumber;

        if (LineNumber_1 != 0)
            Tool.LineNumber = LineNumber_1;

        Tool.DrawingType = DRAWING_MARKER;
        Tool.BeginIndex = sc.Index;
    }
}

```

```

Tool.MarkerType = MARKER_ARROWRIGHT;
Tool.BeginValue = sc.Low[sc.Index];
Tool.Region = 0;
Tool.Color = RGB(255, 255, 0);
Tool.MarkerSize = 8;
Tool.LineWidth = 5;
Tool.AddMethod = UTAM_ADD_OR_ADJUST;

sc.UseTool(Tool); // Here we make the function call to add the marker

//Remember the Line Number
LineNumber_1 = Tool.LineNumber;

//Now add second marker
if (LineNumber_2 != 0)
    Tool.LineNumber = LineNumber_2;
else
    Tool.LineNumber = -1; //automatically assign number

Tool.BeginValue = sc.High[sc.Index];
Tool.Color = RGB(128, 255, 0);

sc.UseTool(Tool); // Here we make the function call to add the marker

LineNumber_2 = Tool.LineNumber;

}

//int Count = sc.DeleteACSCChartDrawing(0, TOOL_DELETE_CHARTDRAWING, LineNumber_2);
}

/*=====*/
SCSFExport scsf_UseToolExampleUserDrawnText(SCStudyInterfaceRef sc)
{
    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: User Drawn Text";
        sc.StudyDescription = "User drawn text drawing example.";

        sc.GraphRegion = 0;

        sc.AutoLoop = 0; //No automatic looping
        return;
    }

    int &TextDrawingLineNumber = sc.GetPersistentInt(1);

    if (sc.LastCallToFunction)
    {
        if (TextDrawingLineNumber != 0)
        {
            // Be sure to remove the Text drawing added as a user drawn drawing
            sc.DeleteUserDrawnACSDrawing(sc.ChartNumber, TextDrawingLineNumber);
        }
        return;
    }

    int BarIndex = sc.ArraySize - 20;

    if (BarIndex < 0)
        return;

```



```

s_UseTool TextDrawing;
TextDrawing.Clear();//Not necessary but good practice
TextDrawing.ChartNumber = sc.ChartNumber;
TextDrawing.DrawingType = DRAWING_TEXT;

if(TextDrawingLineNumber != 0)
    TextDrawing.LineNumber = TextDrawingLineNumber;

TextDrawing.AddAsUserDrawnDrawing = 1;

TextDrawing.BeginIndex = BarIndex;
TextDrawing.BeginValue = sc.High[BarIndex];
TextDrawing.Color = COLOR_KHAKI;
TextDrawing.Text = "Movable User Drawn Text Example";
TextDrawing.FontSize = 18;
// do not keep adjusting
TextDrawing.AddMethod = UTAM_ADD_ONLY_IF_NEW;

sc.UseTool(TextDrawing);

//Remember the line number which has been automatically assigned
TextDrawingLineNumber = TextDrawing.LineNumber;
}
/*=====*/
SCSFExport scsf_UseToolExampleStationaryText(SCStudyInterfaceRef sc)
{
    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Stationary Text";
        sc.GraphRegion = 0;

        sc.AutoLoop = 0; //No automatic looping
        return;
    }

    int &TextDrawingLineNumber = sc.GetPersistentInt(1);

    if (sc.LastCallToFunction)
    {
        if (TextDrawingLineNumber != 0)
        {
            // Be sure to remove the Text drawing added as a user drawn drawing
            sc.DeleteUserDrawnACSDrawing(sc.ChartNumber, TextDrawingLineNumber);
        }
        return;
    }

    s_UseTool TextDrawing;
    TextDrawing.Clear();//Not necessary but good practice
    TextDrawing.ChartNumber = sc.ChartNumber;
    TextDrawing.DrawingType = DRAWING_STATIONARY_TEXT;

    if (TextDrawingLineNumber != 0)
        TextDrawing.LineNumber = TextDrawingLineNumber;

    TextDrawing.AddAsUserDrawnDrawing = 1; // Optional

    //If this is not set, it will be automatically set to 1 anyway.
    TextDrawing.UseRelativeVerticalValues = 1;

    TextDrawing.BeginDateTime = 50;
    TextDrawing.BeginValue = 50;

```

```

TextDrawing.Color = COLOR_KHAKI;
TextDrawing.Text = "Stationary Text Example";
TextDrawing.FontSize = 18;

TextDrawing.AddMethod = UTAM_ADD_OR_ADJUST;

sc.UseTool(TextDrawing);

//Remember the line number which has been automatically assigned
TextDrawingLineNumber = TextDrawing.LineNumber;

}

/*=====*/
SCSFExport scsf_UseToolExampleMultilineText(SCStudyInterfaceRef sc)
{
    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Multiline Text";
        sc.StudyDescription = "";

        sc.GraphRegion = 0;

        sc.AutoLoop = 0; //No automatic looping

        return;
    }

    // Do data processing
    int BarIndex = sc.ArraySize - 20;

    if (BarIndex < 0)
        return;

    s_UseTool Tool;

    Tool.ChartNumber = sc.ChartNumber;
    int &LineNumber = sc.GetPersistentInt(1);
    if (LineNumber != 0)
        Tool.LineNumber = LineNumber;
    Tool.Region = 0;
    Tool.DrawingType = DRAWING_TEXT;
    Tool.AddMethod = UTAM_ADD_OR_ADJUST;

    // for multi-line text use "\n" as the delimiter and turn on the multi line label flag
    Tool.Text = "Text Example\nSecond Line\nThird Line\nForth Line";
    Tool.MultiLineLabel = 1;

    Tool.BeginIndex = BarIndex;
    Tool.BeginValue = sc.Close[BarIndex];

    Tool.Color = COLOR_YELLOW;
    Tool.FontFace = sc.GetChartTextFontFaceName(); // override chart drawing text font
    Tool.FontSize = 14;
    Tool.FontBold = 1;

    sc.UseTool(Tool);
    LineNumber = Tool.LineNumber; //Remember line number which has been automatically set
}

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

```

```

// Set configuration variables
if (sc.SetDefaults)
{
    sc.GraphName = "UseTool Example: Reward/Risk";
    sc.StudyDescription = "";

    sc.GraphRegion = 0;

    sc.AutoLoop = 0; //No automatically looping

    return;
}

// Do data processing
int BarIndex = sc.ArraySize - 1;

if (BarIndex < 20)
    return;

float Entry = sc.Close[BarIndex-1];
float Stop = Entry - 10 * sc.TickSize;
float Target = Entry + 20 * sc.TickSize;

s_UseTool Tool;

Tool.ChartNumber = sc.ChartNumber;
int &LineNumber = sc.GetPersistentInt(1);
if (LineNumber != 0)
    Tool.LineNumber = LineNumber;
Tool.Region = 0;
Tool.AddMethod = UTAM_ADD_OR_ADJUST;

Tool.DrawingType = DRAWING_REWARD_RISK;

Tool.BeginIndex = BarIndex - 2;
Tool.BeginValue = Stop;
Tool.EndIndex = BarIndex - 1;
Tool.EndValue = Entry;
Tool.ThirdIndex = BarIndex;
Tool.ThirdValue = Target;

Tool.Color = COLOR_YELLOW; // text color
Tool.TransparentLabelBackground = 0; // opaque labels
Tool.TextAlignment = DT_RIGHT; // text alignment (target marker is left/right of text)
Tool.ShowTickDifference = 1; // text options
Tool.ShowPriceDifference = 1;
Tool.ShowCurrencyValue = 1;
Tool.FontFace = sc.GetChartTextFontFaceName(); // override chart drawing text font
Tool.FontSize = 14;
Tool.FontBold = 1;

// remaining options are controlled via the use tool levels
Tool.LevelColor[0] = COLOR_DARKRED; // stop to entry line
Tool.LevelStyle[0] = LINESTYLE_SOLID;
Tool.LevelWidth[0] = 2;

Tool.LevelColor[1] = COLOR_DARKGREEN; // entry to target line
Tool.LevelStyle[1] = LINESTYLE_SOLID;
Tool.LevelWidth[1] = 2;

Tool.LevelColor[2] = COLOR_WHITE; // entry marker
Tool.LevelStyle[2] = MARKER_POINT; // marker type
Tool.RetacementLevels[2] = 8; // marker size
Tool.LevelWidth[2] = 3; // marker width

```

```

Tool.LevelColor[3] = COLOR_RED;           // stop marker
Tool.LevelStyle[3] = MARKER_DASH;         // marker type
Tool.ReplacementLevels[3] = 8;            // marker size
Tool.LevelWidth[3] = 3;                   // marker width

Tool.LevelColor[4] = COLOR_ORANGE;        // target marker
Tool.LevelStyle[4] = MARKER_STAR;         // marker type
Tool.ReplacementLevels[4] = 8;            // marker size
Tool.LevelWidth[4] = 3;                   // marker width

sc.UseTool(Tool); // here we make the function call to add the reward risk tool
LineNumber = Tool.LineNumber; // remember line number which has been automatically set
}

/*=====*/
SCSFExport scsf_UseToolVerticalLineText(SCStudyInterfaceRef sc)
{
    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Vertical Line / Text";
        sc.StudyDescription = "";

        sc.GraphRegion = 0;

        sc.AutoLoop = 0; //No automatic looping

        return;
    }

    // Do data processing
    for (int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize - 1; BarIndex++)
    {
        if (BarIndex < 80)
            continue;

        s_UseTool Tool;

        Tool.ChartNumber = sc.ChartNumber;
        Tool.DrawingType = DRAWING_VERTICALLINE;
        Tool.Region = 0;
        Tool.AddMethod = UTAM_ADD_OR_ADJUST;

        Tool.Color = COLOR_DARKMAGENTA;
        Tool.LineWidth = 2;

        Tool.ShowEndPointTime = 1;
        Tool.MultiLineLabel = 1;
        Tool.DrawWithinRegion = 1;
        Tool.FontSize = 7;
        Tool.TextColor = COLOR_ORANGE;

        // now draw 8 lines with the different alignment/orientations
        Tool.BeginIndex = BarIndex - 80;
        int &LineNumber1 = sc.GetPersistentInt(1);
        Tool.LineNumber = LineNumber1 != 0 ? LineNumber1 : -1; // use line number once set, otherwise auto-set
        Tool.Text = "Vertical Text Top/Left Aligned";
        Tool.VerticalText = 1;
        Tool.TextAlignment = DT_TOP | DT_LEFT;

        sc.UseTool(Tool);
        LineNumber1 = Tool.LineNumber; //Remember line number which has been automatically set

        // line 2

```

```

Tool.BeginIndex = BarIndex - 70;
int &LineNumber2 = sc.GetPersistentInt(2);
Tool.LineNumber = LineNumber2 != 0 ? LineNumber2 : -1;
Tool.Text = "Vertical Text Top/Right Aligned";
Tool.VerticalText = 1;
Tool.TextAlignment = DT_TOP | DT_RIGHT;

sc.UseTool(Tool);
LineNumber2 = Tool.LineNumber; //Remember line number which has been automatically set

// line 3
Tool.BeginIndex = BarIndex - 60;
int &LineNumber3 = sc.GetPersistentInt(3);
Tool.LineNumber = LineNumber3 != 0 ? LineNumber3 : -1;
Tool.Text = "Vertical Text Bottom/Left Aligned";
Tool.VerticalText = 1;
Tool.TextAlignment = DT_BOTTOM | DT_LEFT;

sc.UseTool(Tool);
LineNumber3 = Tool.LineNumber; //Remember line number which has been automatically set

// line 4
Tool.BeginIndex = BarIndex - 50;
int &LineNumber4 = sc.GetPersistentInt(4);
Tool.LineNumber = LineNumber4 != 0 ? LineNumber4 : -1;
Tool.Text = "Vertical Text Bottom/Right Aligned";
Tool.VerticalText = 1;
Tool.TextAlignment = DT_BOTTOM | DT_RIGHT;

sc.UseTool(Tool);
LineNumber4 = Tool.LineNumber; //Remember line number which has been automatically set

// line 5
Tool.BeginIndex = BarIndex - 40;
int &LineNumber5 = sc.GetPersistentInt(5);
Tool.LineNumber = LineNumber5 != 0 ? LineNumber5 : -1;
Tool.Text = "Horiz Text Top/Left Aligned";
Tool.VerticalText = 0;
Tool.TextAlignment = DT_TOP | DT_LEFT;

sc.UseTool(Tool);
LineNumber5 = Tool.LineNumber; //Remember line number which has been automatically set

// line 6
Tool.BeginIndex = BarIndex - 30;
int &LineNumber6 = sc.GetPersistentInt(6);
Tool.LineNumber = LineNumber6 != 0 ? LineNumber6 : -1;
Tool.Text = "Horiz Text Bottom/Left Aligned";
Tool.VerticalText = 0;
Tool.TextAlignment = DT_BOTTOM | DT_LEFT;

sc.UseTool(Tool);
LineNumber6 = Tool.LineNumber; //Remember line number which has been automatically set

// line 7
Tool.BeginIndex = BarIndex - 20;
int &LineNumber7 = sc.GetPersistentInt(7);
Tool.LineNumber = LineNumber7 != 0 ? LineNumber7 : -1;
Tool.Text = "Horiz Text Top/Right Aligned";
Tool.VerticalText = 0;
Tool.TextAlignment = DT_TOP | DT_RIGHT;

sc.UseTool(Tool);
LineNumber7 = Tool.LineNumber; //Remember line number which has been automatically set

```

```

// line 8
Tool.BeginIndex = BarIndex - 10;
int &LineNumber8 = sc.GetPersistentInt(8);
Tool.LineNumber = LineNumber8 != 0 ? LineNumber8 : -1;
Tool.Text = "Horiz Text Bottom/Right Aligned";
Tool.VerticalText = 0;
Tool.TextAlignment = DT_BOTTOM | DT_RIGHT;

sc.UseTool(Tool);
LineNumber8 = Tool.LineNumber; //Remember line number which has been automatically set
}
}

/*=====*/
SCSFExport scsf_UseToolExampleGannGrid(SCStudyInterfaceRef sc)
{
    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: GannGrid";
        sc.StudyDescription = "";

        sc.GraphRegion = 0;

        sc.AutoLoop = 0; //No automatic looping

        return;
    }

    // Do data processing
    for (int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize - 1; BarIndex++)
    {
        if (BarIndex < 100)
            continue;

        s_UseTool Tool;

        Tool.ChartNumber = sc.ChartNumber;
        int &LineNumber = sc.GetPersistentInt(1);
        Tool.LineNumber = LineNumber != 0 ? LineNumber : -1; // use line number once set, otherwise auto-set
        Tool.Region = 0;
        Tool.AddMethod = UTAM_ADD_OR_ADJUST;
        Tool.Color = RGB(255, 255, 0);
        Tool.LineWidth = 2;

        Tool.DrawingType = DRAWING_GANNGRID;
        Tool.BeginDateTime = sc.BaseDateTimeIn[BarIndex-20];
        Tool.BeginValue = sc.Low[BarIndex-20];

        // second point sets Gann line
        Tool.EndDateTime = sc.BaseDateTimeIn[BarIndex-10];
        Tool.EndValue = sc.High[BarIndex-10];

        sc.UseTool(Tool); // Here we make the function call to add the marker
        LineNumber = Tool.LineNumber; //Remember line number which has been automatically set
    }
}

/*=====*/
SCSFExport scsf_UseToolExampleTriangle(SCStudyInterfaceRef sc)
{
    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Triangle";
    }
}

```

```

    sc.StudyDescription = "";

    sc.GraphRegion = 0;

    sc.AutoLoop = 0; //No automatic looping

    return;
}

// Do data processing
for (int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize - 1; BarIndex++)
{
    if (BarIndex < 100)
        continue;

    s_UseTool Tool;

    Tool.ChartNumber = sc.ChartNumber;
    int &LineNumber = sc.GetPersistentInt(1);

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

    Tool.Region = 0;
    Tool.AddMethod = UTAM_ADD_OR_ADJUST;
    Tool.Color = COLOR_DARKBLUE;
    Tool.SecondaryColor = COLOR_BLUE;
    Tool.TransparencyLevel = 70;
    Tool.LineWidth = 2;

    Tool.DrawingType = DRAWING_TRIANGLE;
    Tool.BeginDateTime = sc.BaseDateTimeIn[BarIndex-30];
    Tool.BeginValue = sc.Low[BarIndex-30];
    Tool.EndDateTime = sc.BaseDateTimeIn[BarIndex-20];
    Tool.EndValue = sc.High[BarIndex-20];
    Tool.ThirdDateTime = sc.BaseDateTimeIn[BarIndex-10];
    Tool.ThirdValue = sc.Close[BarIndex-10];

    sc.UseTool(Tool); // Here we make the function call to add the marker
    LineNumber = Tool.LineNumber; //Remember line number which has been automatically set
}
}

/*=====*/
SCSFExport scsf_UseToolExampleAddDrawingLineAtEveryUpdate(SCStudyInterfaceRef sc)
{
    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "UseTool Example: Add Line Drawing at Every Update";
        sc.GraphRegion = 0;
        sc.AutoLoop = 0; //No automatic looping

        return;
    }

    float& r_LineValue = sc.GetPersistentFloatFast(1);

    if (sc.IsFullRecalculation)
    {
        r_LineValue = sc.Close[sc.ArraySize - 1];
    }

    s_UseTool Tool;
    Tool.Clear();

```

```

Tool.ChartNumber = sc.ChartNumber;
int DrawingIndex = sc.ArraySize - 1;
Tool.DrawingType = DRAWING_LINE;
Tool.BeginDateTime = sc.BaseDateTimeIn[DrawingIndex - 20];
Tool.EndDateTime = sc.BaseDateTimeIn[DrawingIndex];
Tool.BeginValue = r_LineValue;
Tool.EndValue = r_LineValue;
Tool.AddAsUserDrawnDrawing = 1;
Tool.Region = 0;
Tool.Color = RGB(255, 255, 0);
Tool.LineWidth = 5;
Tool.AddMethod = UTAM_ADD_ALWAYS;
sc.UseTool(Tool);

r_LineValue += sc.TickSize;

}

/*=====*/
SCSFExport scsf_AdjustUserDrawnDrawingExample(SCStudyInterfaceRef sc)
{
    // Set configuration variables

    if (sc.SetDefaults)
    {
        sc.GraphName = "Adjust User Drawn Drawing Example";

        sc.GraphRegion = 0;

        sc.UpdateAlways = 1;

        // We have expressly set auto-looping to off because we are not filling
        // in any of the Subgraph[].Data[] arrays. Since auto-looping will
        // cause this function to be called for every bar in the chart and all
        // we are doing is using tools to do drawings, it would be inefficient
        // to use auto-looping.
        sc.AutoLoop = 0;

        sc.Input[0].Name = "Line Color";
        sc.Input[0].SetColor(RGB (255, 0, 255));

        return;
    }

    //The following code will get all of the Ray drawings on the chart and make them the 'Line Color ' Input color, change
    //the width to 10, and move the starting point back by one bar. Since this function is continuously called with
    //sc.UpdateAlways, this continuously occurs so you will see the Ray drawings continuously move back.
    int DrawingIndex = 0;
    s_UseTool DrawingObject;
    while( sc.GetUserDrawnChartDrawing(sc.ChartNumber, DRAWING_HORIZONTAL_RAY , DrawingObject,
    DrawingIndex) ) //DRAWING_RAY
    {
        DrawingObject.Color =sc.Input[0].GetColor();
        DrawingObject.LineWidth = 10;
        int BarIndex = sc.GetNearestMatchForSCDateTime(sc.ChartNumber,
        DrawingObject.BeginDateTime.GetAsDouble());
        DrawingObject.BeginDateTime = sc.BaseDateTimeIn[BarIndex - 1];
        DrawingObject.AddMethod = UTAM_ADD_OR_ADJUST;
        sc.UseTool(DrawingObject);
        DrawingIndex++;
    }
}

```



```

/*=====*/
SCSFExport scsf_DeleteACSCChartDrawingExample(SCStudyInterfaceRef sc)
{
    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "Delete ACS Chart Drawing Example";
        sc.StudyDescription = "";

        sc.GraphRegion = 0;

        sc.AutoLoop = 0; //No automatic looping

        return;
    }

    int& LineNumber = sc.GetPersistentInt(1);

    //Only do processing on the last bar
    if (sc.IsFullRecalculation)
    {
        //Set the previously remembered line number to 0 because on a full recalculation non-user drawn advanced custom
        study drawings are automatically deleted
        LineNumber = 0;
        return;
    }

    if (LineNumber != 0)
    {
        int Result = sc.DeleteACSCChartDrawing(sc.ChartNumber, TOOL_DELETE_CHARTDRAWING, LineNumber);
        LineNumber = 0;
    }

    s_UseTool Tool;

    Tool.ChartNumber = sc.ChartNumber;
    //Tool.LineNumber will be automatically set. No need to set this.
    Tool.DrawingType = DRAWING_MARKER;
    Tool.BeginIndex = sc.ArraySize - 1;

    Tool.MarkerType = MARKER_ARROWUP;
    Tool.BeginValue = sc.Low[sc.ArraySize - 1];

    Tool.Region = 0;
    Tool.Color = RGB(0, 255, 0);
    Tool.MarkerSize = 8;
    Tool.LineWidth = 4;
    Tool.AddMethod = UTAM_ADD_ALWAYS;

    sc.UseTool(Tool); // Here we make the function call to add the marker

    //Remember the line number so it can be deleted at the beginning of the function when set
    LineNumber = Tool.LineNumber;
}

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

```

SCString MessageText;

// Set configuration variables

```
if (sc.SetDefaults)
{
    sc.GraphName = "Control Bar Button, Menu and Pointer Interaction Example";

    sc.StudyDescription = "";

    sc.GraphRegion = 0;

    // We have expressly set Autolooping to off because we are not filling
    // in any of the Subgraph[].Data[] arrays. Since Autolooping will
    // cause this function to be called for every bar in the chart it
    // would be inefficient to use Autolooping.
    sc.AutoLoop = 0;

    // Indicate to receive the pointer events always
    sc.ReceivePointerEvents = ACS_RECEIVE_POINTER_EVENTS_WHEN_ACS_BUTTON_ENABLED;

    return;
}
```

```
int& r_MenuID      = sc.GetPersistentInt(1);
int& r_CheckedMenuID = sc.GetPersistentInt(2);
int& r_GreyedMenuID = sc.GetPersistentInt(3);
int& r_HideMenuID   = sc.GetPersistentInt(4);
int& r_CheckedState = sc.GetPersistentInt(5);
int& r_SeparatorMenuID = sc.GetPersistentInt(6);
```

```
if (sc.LastCallToFunction)
{
    // Set default hover text and button text.
    sc.SetCustomStudyControlBarButtonHoverText(1, "");
    sc.SetCustomStudyControlBarButtonText(1, "");

    // Remove menu items when study is removed
    sc.RemoveACSChartShortcutMenuItem(sc.ChartNumber, r_MenuID);
    sc.RemoveACSChartShortcutMenuItem(sc.ChartNumber, r_SeparatorMenuID);
    sc.RemoveACSChartShortcutMenuItem(sc.ChartNumber, r_CheckedMenuID);
    sc.RemoveACSChartShortcutMenuItem(sc.ChartNumber, r_GreyedMenuID);
    sc.RemoveACSChartShortcutMenuItem(sc.ChartNumber, r_HideMenuID);

    return;
}
```

```
if (sc.IsFullRecalculation != 0)
{
    // Set the ACS Control Bar button 1 hover text
    sc.SetCustomStudyControlBarButtonHoverText(1, "This is the custom hover text for ACS Control Bar button 1");

    // Set the ACS Control Bar button 1 text
    sc.SetCustomStudyControlBarButtonText(1, "ACS1");

    //sc.PlaceACSChartShortcutMenuItemsAtTopOfMenu = true;

    // Add chart shortcut menu item if not already added (persist var initialized to zero, negative previous fail)
    if (r_MenuID <= 0)
    {
        r_MenuID = sc.AddACSChartShortcutMenuItem(sc.ChartNumber, "Menu Item example 1");

        // add a menu separator
        r_SeparatorMenuID = sc.AddACSChartShortcutMenuSeparator(sc.ChartNumber);
    }
}
```

```

}

r_CheckedState = 1;

if (r_CheckedMenuID <= 0)
{
    r_CheckedMenuID = sc.AddACSChartShortcutMenuItem(sc.ChartNumber, "Checked Toggle Menu Item");
    sc.SetACSChartShortcutMenuItemChecked(sc.ChartNumber, r_CheckedMenuID, true);
}

if (r_GreyedMenuID <= 0)
    r_GreyedMenuID = sc.AddACSChartShortcutMenuItem(sc.ChartNumber, "Greyed When Not Checked");

if (r_HideMenuID <= 0)
    r_HideMenuID = sc.AddACSChartShortcutMenuItem(sc.ChartNumber, "Hidden When Not Checked");

if (r_MenuID < 0 || r_CheckedMenuID < 0 || r_HideMenuID < 0 || r_GreyedMenuID < 0)
{
    MessageText.Format("Add ACS Chart Shortcut menu item failed");
    sc.AddMessageToLog(MessageText, 1);
}
}

// wait for an event
if (sc.MenuEventID != 0)
{
    if (sc.PointerEventType == SC_POINTER_BUTTON_DOWN)
        MessageText.Append("Pointer Down Event, ");
    else if (sc.PointerEventType == SC_POINTER_BUTTON_UP)
        MessageText.Append("Pointer Up Event, ");
    else if (sc.PointerEventType == SC_POINTER_MOVE)
        MessageText.Append("Pointer Move Event, ");

    if (sc.MenuEventID >= ACS_BUTTON_1 && sc.MenuEventID <= ACS_BUTTON_150)
    {
        MessageText.AppendFormat("Got button event id %i, ", sc.MenuEventID);

        if (sc.PointerEventType == SC_ACS_BUTTON_ON)
        {
            MessageText.Append("ACS Button On Event, ");

            // reset button 1 to off
            if (sc.MenuEventID == ACS_BUTTON_1)
                sc.SetCustomStudyControlBarButtonEnable(ACS_BUTTON_1, 0);
        }
        else if (sc.PointerEventType == SC_ACS_BUTTON_OFF)
            MessageText.Append("ACS Button Off Event, ");

        sc.SetCustomStudyControlBarButtonEnable(sc.PriorSelectedCustomStudyControlBarButtonNumber, 0);
    }
    else if (sc.MenuEventID == r_MenuID)
    {
        MessageText.AppendFormat("Got menu event id %i, ", sc.MenuEventID);
    }
    else if (sc.MenuEventID == r_HideMenuID)
    {
        MessageText.AppendFormat("Got hidden menu event id %i, ", sc.MenuEventID);
    }
    else if (sc.MenuEventID == r_GreyedMenuID)
    {
        MessageText.AppendFormat("Got greyed menu event id %i, ", sc.MenuEventID);
    }
}

```

```

else if (sc.MenuEventID == r_CheckedMenuID)
{
    MessageText.AppendFormat("Got checked menu event id %i, CheckedState=%i, ", sc.MenuEventID,
r_CheckedState);

    // toggle states
    if (r_CheckedState == 0)
    {
        r_CheckedState = 1;
        sc.SetACSCChartShortcutMenuItemChecked(sc.ChartNumber, r_CheckedMenuID, true);
        sc.SetACSCChartShortcutMenuItemEnabled(sc.ChartNumber, r_GreyedMenuID, true);
        sc.SetACSCChartShortcutMenuItemDisplayed(sc.ChartNumber, r_HideMenuID, true);
    }
    else
    {
        r_CheckedState = 0;
        sc.SetACSCChartShortcutMenuItemChecked(sc.ChartNumber, r_CheckedMenuID, false);
        sc.SetACSCChartShortcutMenuItemEnabled(sc.ChartNumber, r_GreyedMenuID, false);
        sc.SetACSCChartShortcutMenuItemDisplayed(sc.ChartNumber, r_HideMenuID, false);
    }
}

MessageText.Append("Y Value=");
MessageText += sc.FormatGraphValue(sc.ActiveToolYValue, sc.GetValueFormat());

MessageText.Append(", Bar Index=");
MessageText += sc.FormatGraphValue(sc.ActiveToolIndex, 0);

sc.AddMessageToLog(MessageText, 0);
}
}

/*=====
-----*/
SCSFExport scsf_ReceiveKeyboardEventsExample(SCStudyInterfaceRef sc)
{
    SCString MessageText;

    // Set configuration variables
    if (sc.SetDefaults)
    {
        sc.GraphName = "Receive Keyboard Events Example";

        sc.StudyDescription = "";

        sc.GraphRegion = 0;

        // We have expressly set Autolooping to off because we are not filling
        // in any of the Subgraph[].Data[] arrays. Since Autolooping will
        // cause this function to be called for every bar in the chart it
        // would be inefficient to use Autolooping.
        sc.AutoLoop = 0;

        sc.ReceiveKeyboardKeyEvents = 1;
        sc.ReceiveCharacterEvents = 1;
        sc.SupportKeyboardModifierStates = 1;

        return;
    }

    // wait for an event
    if (sc.KeyboardKeyEventCode != 0)
    {
        MessageText.AppendFormat("Received keyboard key event. Windows virtual key code: %i, ",

```

```

sc.KeyboardKeyEventCode);
    sc.AddMessageToLog(MessageText, 0);
}

if(sc.CharacterEventCode != 0)
{
    MessageText.AppendFormat("Received character event. ASCII code: %i, ", sc.CharacterEventCode);
    sc.AddMessageToLog(MessageText, 0);
}

if (sc.IsKeyPressed_Alt)
{
    sc.AddMessageToLog("Alt key pressed", 0);
}

if (sc.IsKeyPressed_Shift)
{
    sc.AddMessageToLog("Shift key pressed", 0);
}

if (sc.IsKeyPressed_Control)
{
    sc.AddMessageToLog("Control key pressed", 0);
}
}

```

```

/*=====
This function gives examples of adding messages to the Message Log, playing
alert sounds and adding Alert Messages.
-----*/

```

```

SCSFExport scsf_LogAndAlertExample(SCStudyInterfaceRef sc)
{

```

```

    // Set configuration variables

```

```

    if (sc.SetDefaults)
    {
        sc.GraphName = "Log and Alert Example";

        sc.StudyDescription = "";

        sc.GraphRegion = 0;

        sc.AutoLoop = 0;//manual looping
    }

```

```

    return;
}

```

```

// Do data processing

```

```

if (sc.GetBarHasClosedStatus(sc.UpdateStartIndex) == BHCS_BAR_HAS_NOT_CLOSED)
    return;

```

```

// Show message to log

```

```

sc.AddMessageToLog("Popped up Message", 1);
sc.AddMessageToLog("Non-Popped up Message", 0);

```

```

// Alert sounds are configured by selecting "Global Settings >> General Settings" in Sierra Chart

```

```

sc.PlaySound(1); // Add alert sound 1 to be played
sc.PlaySound(50); // Add alert sound 50 to be played

```

```

// Add an Alert message to SierraChart Alert Log

```

```

sc.AddAlertLine("Condition is TRUE");

// Adding an alert line using an SCString. The Alert log will also be opened.
SCString AlertMessage("Alert Message");
sc.AddAlertLine(AlertMessage, 1);

// Playing a sound and adding an Alert Line at the same time.
sc.PlaySound(2, AlertMessage, 1); // Alert Log will open

sc.PlaySound(3, "Alert Message 2", 0); // Alert Log will not open

// Can also Play Sound from a text file, for example:
sc.PlaySound("C:\\Windows\\beep.wav", 2);

sc.AddAlertLineWithDateTime("Alert message with date-time", 0,
sc.BaseDateTimeIn[sc.UpdateStartIndex].GetAsDouble());
}

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

        sc.GraphName = "Select Scale Increment with Buttons";

        sc.StudyDescription = "This code serves as an example of how to set the base graph scale increment by using
control bar buttons.";

        sc.GraphRegion = 0;

        sc.AutoLoop = 0;

        sc.Input[0].Name = "Scale Increment 1";
        sc.Input[0].SetFloat(1.0f);

        sc.Input[1].Name = "Scale Increment 2";
        sc.Input[1].SetFloat(2.0f);

        sc.Input[2].Name = "Scale Increment 3";
        sc.Input[2].SetFloat(4.0f);

        sc.Input[3].Name = "Scale Increment 4";
        sc.Input[3].SetFloat(8.0f);

        return;
    }

    if (sc.IsFullRecalculation)
    {
        sc.SetCustomStudyControlBarButtonHoverText(ACS_BUTTON_1, "Change to Scale Increment 1 Input");
        sc.SetCustomStudyControlBarButtonText(ACS_BUTTON_1, "Scale1");
        sc.SetCustomStudyControlBarButtonHoverText(ACS_BUTTON_2, "Change to Scale Increment 2 Input");
        sc.SetCustomStudyControlBarButtonText(ACS_BUTTON_2, "Scale2");
        sc.SetCustomStudyControlBarButtonHoverText(ACS_BUTTON_3, "Change to Scale Increment 3 Input");
        sc.SetCustomStudyControlBarButtonText(ACS_BUTTON_3, "Scale3");
        sc.SetCustomStudyControlBarButtonHoverText(ACS_BUTTON_4, "Change to Scale Increment 4 Input");
        sc.SetCustomStudyControlBarButtonText(ACS_BUTTON_4, "Scale4");
    }
}

```

```

// wait for a button
if (sc.MenuEventID >= ACS_BUTTON_1 && sc.MenuEventID <= ACS_BUTTON_4)
{
    // reset button to off
    sc.SetCustomStudyControlBarButtonEnable(sc.MenuEventID, 0);
    sc.BaseGraphScaleIncrement = sc.Input[sc.MenuEventID - 1].GetFloat();
    sc.BaseGraphHorizontalGridLineIncrement = sc.BaseGraphScaleIncrement;
}
}

/*=====
This study displays the simple, exponential, and adaptive moving
averages.
-----*/
SCSFExport scsf_MovingAverageExample1(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_SimpleMA = sc.Subgraph[0];
    SCSubgraphRef Subgraph_ExponentialMA = sc.Subgraph[1];
    SCSubgraphRef Subgraph_AdaptiveMA = sc.Subgraph[2];

    // Set configuration variables

    if (sc.SetDefaults)
    {
        // Set defaults

        sc.GraphName = "Moving Average Example 1";

        sc.StudyDescription = "";

        sc.GraphRegion = 0;

        Subgraph_SimpleMA.Name = "Simple Moving Average";
        Subgraph_SimpleMA.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_SimpleMA.PrimaryColor = RGB(0,255,0);

        Subgraph_ExponentialMA.Name = "Exponential Moving Average";
        Subgraph_ExponentialMA.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_ExponentialMA.PrimaryColor = RGB(255,0,255);

        Subgraph_AdaptiveMA.Name = "Adaptive Moving Average";
        Subgraph_AdaptiveMA.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_AdaptiveMA.PrimaryColor = RGB(255,255,0);

        sc.AutoLoop = 1;

        return;
    }

    // Do data processing

    // Simple moving average in the first subgraph
    sc.SimpleMovAvg(sc.Close, Subgraph_SimpleMA, 10);

    // Exponential moving average in the second subgraph
    sc.ExponentialMovAvg(sc.Close, Subgraph_ExponentialMA, 10);

    // Adaptive moving average in the third subgraph
    sc.AdaptiveMovAvg(sc.Close, Subgraph_AdaptiveMA, 10, 2.0f, 30.0f);
}

```

```

/*=====
This study displays the linear regression indicator, weighted moving
average, and wilders moving average.
-----*/
SCSFExport scsf_MovingAverageExample2(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_LinearRegressionIndicator = sc.Subgraph[0];
    SCSubgraphRef Subgraph_WeightedMA = sc.Subgraph[1];
    SCSubgraphRef Subgraph_WildersMA = sc.Subgraph[2];

    // Set configuration variables

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

        sc.GraphName = "Moving Average Example 2";

        sc.StudyDescription = "";

        sc.GraphRegion = 0;

        Subgraph_LinearRegressionIndicator.Name = "Linear Regression Indicator";
        Subgraph_LinearRegressionIndicator.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_LinearRegressionIndicator.PrimaryColor = RGB(0,255,0);

        Subgraph_WeightedMA.Name = "Weighted Moving Average";
        Subgraph_WeightedMA.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_WeightedMA.PrimaryColor = RGB(255,0,255);

        Subgraph_WildersMA.Name = "Wilders Moving Average";
        Subgraph_WildersMA.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_WildersMA.PrimaryColor = RGB(255,255,0);

        sc.AutoLoop = 1;

        return;
    }

    // Do data processing

    // Linear regression indicator in the first subgraph
    sc.LinearRegressionIndicator(sc.Close, Subgraph_LinearRegressionIndicator, 10);

    // Weighted moving average in the second subgraph
    sc.WeightedMovingAverage(sc.Close, Subgraph_WeightedMA, 10);

    // Wilders moving average in the third subgraph
    sc.WildersMovingAverage(sc.Close, Subgraph_WildersMA, 10);
}

/*=====
This study function gives an example of using the sc.GetChartDrawing function.
-----*/
SCSFExport scsf_GetChartDrawingExample(SCStudyInterfaceRef sc)
{
    // Set configuration variables

    if (sc.SetDefaults)
    {
        sc.GraphName = "GetChartDrawing Example";
    }
}

```



```

sc.StudyDescription = "This function demonstrates using the sc.GetUserDrawnChartDrawing function.";

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

return;
}

int& r_MarkerLineNumber = sc.GetPersistentInt(1);

if (sc.LastCallToFunction && r_MarkerLineNumber != 0)
{
    // Be sure to remove the drawing added as a user drawn drawing
    sc.DeleteUserDrawnACSDrawing(sc.ChartNumber, r_MarkerLineNumber);
    return; //No further processing needed in this case.
}

// Do data processing
int& MarkerIndex = sc.GetPersistentInt(2);

if (sc.UpdateStartIndex == 0)
{
    MarkerIndex = -1;
}

if (MarkerIndex == -1 )
{
    // place a user drawn marker on the chart
    s_UseTool Tool;
    Tool.Clear();
    Tool.ChartNumber = sc.ChartNumber;
    Tool.Region = sc.GraphRegion;

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

    // Must be user drawn if we are going to be able to use sc.GetUserDrawnDrawingByLineNumber().
    Tool.AddAsUserDrawnDrawing = 1;

    Tool.DrawingType = DRAWING_MARKER;
    int BarIndex = max(0, sc.ArraySize - 5);
    Tool.BeginIndex = BarIndex;
    Tool.BeginValue = sc.High[BarIndex];
    Tool.Color = RGB(0,200,200);
    Tool.AddMethod = UTAM_ADD_OR_ADJUST;
    Tool.MarkerType = MARKER_X;
    Tool.MarkerSize = 8;
    Tool.LineWidth = 5;

    sc.UseTool(Tool);

    r_MarkerLineNumber = Tool.LineNumber;

    MarkerIndex = BarIndex;
}

s_UseTool ChartDrawing;

if (sc.GetUserDrawnChartDrawing(sc.ChartNumber, DRAWING_UNKNOWN, ChartDrawing, -1) > 0)

```

```

{
    SCString BeginDateTimeString = sc.FormatDateTime(ChartDrawing.BeginDateTime);
    SCString EndDateTimeString = sc.FormatDateTime(ChartDrawing.EndDateTime);
    SCString Msg;
    Msg.Format("Details of last Chart Drawing on chart: BeginDateTime=%s, EndDateTime=%s, BeginValue=%f,
EndValue=%f, LineNumber = %i.", BeginDateTimeString.GetChars(), EndDateTimeString.GetChars(),
ChartDrawing.BeginValue, ChartDrawing.EndValue, ChartDrawing.LineNumber);

    sc.AddMessageToLog(Msg, 1);
}

// Gets the user drawn marker and writes log if it moved
if (sc.GetUserDrawnDrawingByLineNumber(0, r_MarkerLineNumber, ChartDrawing))
{
    if (ChartDrawing.BeginIndex != MarkerIndex)
    {
        SCString Msg;
        Msg.Format("Marker was found on current chart, and it moved to index %i", ChartDrawing.BeginIndex);
        sc.AddMessageToLog(Msg, 1);
        MarkerIndex = ChartDrawing.BeginIndex;
    }
}
}

/*=====*/
SCSFExport scsf_ChangeSettingsExample(SCStudyInterfaceRef sc)
{
    if (sc.SetDefaults)
    {
        sc.GraphName = "Change Settings Example";

        sc.StudyDescription = "Changes bar period to five minutes.";

        return;
    }

    // Set the bar period to 5 minutes. This change will occur after this function returns, during the next chart update.
    sc.SecondsPerBar = 5*SECONDS_PER_MINUTE; // 5*SECONDS_PER_MINUTE is the same as 5*60 or 300

    if (sc.LastCallToFunction == 1)
        sc.SecondsPerBar = 1*SECONDS_PER_MINUTE; // Set back to 1 minute
}

/*=====*/
This is an example of how to use the GetStudyArraysFromChart function.
-----*/
SCSFExport scsf_GetStudyArraysFromChartExample(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Line = sc.Subgraph[0];

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

        sc.GraphName = "GetStudyArraysFromChart Example";

        sc.StudyDescription = "This is an example of how to use the GetStudyArraysFromChart function.";

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

```

```

    return;
}

// Do data processing

int ChartNumber = 1;

// Get the study arrays from the first study in the specified chart
SCGraphData StudyData;
sc.GetStudyArraysFromChart(ChartNumber, 1, StudyData);

// Get the first subgraph array from the study data
if(StudyData.GetArraySize() == 0)
    return;

SCFloatArrayRef StudyArray = StudyData[0];

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

// Copy the array from back to front into our first subgraph
int RefIndex = StudyArray.GetArraySize();
for (int Index = sc.ArraySize - 1; Index >= 0; Index--)
{
    Subgraph_Line[Index] = StudyArray[RefIndex];

    --RefIndex;

    if (RefIndex < 0)
        break;
}
}

/*=====
This example code calculates a 30 period simple moving average and colors
the data based on the slope.
-----*/
SCSFExport scsf_SimpMovAvgColored(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Average = sc.Subgraph[0];

    SCInputRef Input_Length = sc.Input[0];

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

        sc.GraphName = "Simple Moving Average (Colored)";

        sc.StudyDescription = "";

        Subgraph_Average.Name = "Average";
        Subgraph_Average.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Average.PrimaryColor = RGB(0,255,0);
        Subgraph_Average.SecondaryColorUsed = 1; // true
        Subgraph_Average.SecondaryColor = RGB(0,127,0);

        Input_Length.Name = "Length";
        Input_Length.SetInt(30);

```

```

    Input_Length.SetIntLimits(1,MAX_STUDY_LENGTH);

    sc.AutoLoop = 1;

    sc.GraphRegion = 0;

    return;
}

// Do data processing

// Set the index of the first array element to begin drawing at
sc.DataStartIndex = Input_Length.GetInt() - 1;

// Calculate a simple moving average for the input array
sc.SimpleMovAvg(sc.Close,Subgraph_Average,Input_Length.GetInt());

// Color the data according to the slope
if (Subgraph_Average[sc.Index] > Subgraph_Average[sc.Index-1])
{
    // Rising
    // Use the primary color
    Subgraph_Average.DataColor[sc.Index] = Subgraph_Average.PrimaryColor;
}
else
{
    // Falling
    // Use the secondary color
    Subgraph_Average.DataColor[sc.Index] = Subgraph_Average.SecondaryColor;
}

}

/*=====
This is an example of how to use colors, secondary colors, and coloring
individual bars in a price graph.
-----*/
SCSFExport scsf_ColoredPriceGraph(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Open = sc.Subgraph[0];
    SCSubgraphRef Subgraph_High = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Low = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Close = sc.Subgraph[3];

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

        sc.GraphName = "Colored Price Graph";

        sc.StudyDescription = "This is an example of how to use colors, secondary colors, and coloring individual bars in a
price graph.";

        sc.GraphDrawType = GDT_OHLCBAR;

        // Name the subgraphs

```

```

// Flag each of the four subgraphs as being able to use secondary colors.

Subgraph_Open.Name = "Open";
Subgraph_Open.DrawStyle = DRAWSTYLE_LINE;
Subgraph_Open.PrimaryColor = RGB(0,255,0);
Subgraph_Open.SecondaryColorUsed = 1;
Subgraph_Open.DrawZeros = false;
Subgraph_Open.SecondaryColor = RGB(0,127,0);

Subgraph_High.Name = "High";
Subgraph_High.DrawStyle = DRAWSTYLE_LINE;
Subgraph_High.PrimaryColor = RGB(255,0,255);
Subgraph_High.SecondaryColorUsed = 1;
Subgraph_High.DrawZeros = false;
Subgraph_High.SecondaryColor = RGB(127,0,127);

Subgraph_Low.Name = "Low";
Subgraph_Low.DrawStyle = DRAWSTYLE_LINE;
Subgraph_Low.PrimaryColor = RGB(255,255,0);
Subgraph_Low.SecondaryColorUsed = 1;
Subgraph_Low.DrawZeros = false;
Subgraph_Low.SecondaryColor = RGB(127,127,0);

Subgraph_Close.Name = "Close";
Subgraph_Close.DrawStyle = DRAWSTYLE_LINE;
Subgraph_Close.PrimaryColor = RGB(255,127,0);
Subgraph_Close.SecondaryColorUsed = 1;
Subgraph_Close.DrawZeros = false;
Subgraph_Close.SecondaryColor = RGB(127,63,0);


sc.AutoLoop = 1;


return;
}


// Do data processing

// Loop through the four subgraphs
for (int SGIndex = 0; SGIndex < 4; ++SGIndex)
{
    // Copy the data
    float Value = sc.BaseDataIn[SGIndex][sc.Index];
    sc.Subgraph[SGIndex][sc.Index] = Value;

    // Color with the primary color if there was an increase,
    // otherwise use the secondary color.
    if (sc.BaseDataIn[SGIndex][sc.Index] > sc.BaseDataIn[SGIndex][sc.Index - 1])
        sc.Subgraph[SGIndex].DataColor[sc.Index] = sc.Subgraph[SGIndex].PrimaryColor;
    else
        sc.Subgraph[SGIndex].DataColor[sc.Index] = sc.Subgraph[SGIndex].SecondaryColor;
}
}

/*=====
This function is an example of creating blank space on the right side of
the chart that is preserved during updating.
-----*/
SCSFExport scsf_FillSpaceExample(SCStudyInterfaceRef sc)

```

```

{
    SCInputRef Input_FillSpace = sc.Input[10];

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

        sc.GraphName = "Fill Space Example";

        sc.StudyDescription = "This function is an example of creating blank space on the right side of the chart that is
preserved during updating.";

        Input_FillSpace.Name = "FillSpace";
        Input_FillSpace.SetYesNo(0); // No

        return;
    }

    // Do data processing

    if (Input_FillSpace.GetYesNo())
    {
        sc.PreserveFillSpace = 1; // true

        if (sc.NumFillSpaceBars < 2)
            sc.NumFillSpaceBars = 10;
    }
}

/*=====
   This function is an example of using the storage block.
   -----*/
SCSFExport scsf_StorageBlockExample(SCStudyInterfaceRef sc)
{
    if (sc.SetDefaults)
    {
        // Set the configuration and defaults

        sc.GraphName = "Storage Block Example";

        sc.StudyDescription = "This function is an example of using the storage block.";

        return;
    }

    // Do data processing

    // This is the structure of our data that is to be stored in the storage block
    struct s_PermData
    {
        int Number;
        char Text[32];
    };

    // Here we make a pointer to the storage block as if it was a pointer to our structure

```

```

s_PermData* PermData;
PermData = (s_PermData*)sc.StorageBlock;

// Here we set data using the members of our structure
// This uses the memory of the storage block
PermData->Number = 10;
strcpy(PermData->Text, "Sample Text");
}

/*=====
This function gives an example of making a subgraph name change
dynamically.
-----*/
SCSFExport scsf_DynamicNameExample(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Value = sc.Subgraph[0];

    SCInputRef Input_Value = sc.Input[0];

    // Set configuration variables

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

        sc.GraphName = "Dynamic Name Example";

        sc.StudyDescription = "This function demonstrates how to set a dynamic subgraph name.";

        // Set the initial subgraph name so that it shows up in the
        // Subgraph Settings without having to apply the study.
        Subgraph_Value.Name = "Subgraph";
        Subgraph_Value.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Value.PrimaryColor = RGB(0,255,0);

        Input_Value.Name = "Value";
        Input_Value.SetFloat(1.5f);

        return;
    }

    // Set the subgraph name to include the value of the input.
    // This must be outside the above if (sc.SetDefaults) code block so that
    // it gets executed every time.
    static SCString SubgraphName; // This must be static so that it doesn't fall out of scope
    SubgraphName.Format("Value %f Subgraph", Input_Value.GetFloat());
    Subgraph_Value.Name = SubgraphName;

    // Do data processing
}

/*=====
This shows how to do a vertical line using the draw style set to Bar.
-----*/
SCSFExport scsf_VerticalLine(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_VerticalLines = sc.Subgraph[0];

    // Set the configuration variables

    // Force these settings -- the user will not be able to change them

```

```

sc.ScaleRangeType = SCALE_USERDEFINED;
sc.ScaleRangeTop = 2.0f;
sc.ScaleRangeBottom = 1.0f;
sc.Subgraph[0].DrawStyle = DRAWSTYLE_BAR;

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

    sc.GraphName = "Vertical Line";

    sc.StudyDescription = "Makes a single vertical line at 7 bars back.";

    sc.GraphRegion = 0;

    Subgraph_VerticalLines.Name = "Vertical Lines";
    Subgraph_VerticalLines.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_VerticalLines.PrimaryColor = RGB(255,255,0); // Yellow
    Subgraph_VerticalLines.DrawZeros = false;

    return;
}

// Do data processing

// Get the index that the vertical line should be placed at
int BarIndex = sc.ArraySize - 7; // 7 bars back

// Figure out the number of bars that have been added with this update
int BarsAdded = sc.ArraySize - sc.UpdateStartIndex - 1;

// Clear the old vertical line in case this is an update
if (BarIndex - BarsAdded >= 0)
    Subgraph_VerticalLines[BarIndex - BarsAdded] = 0.0f;

// Add the vertical line
if (BarIndex >= 0)
    Subgraph_VerticalLines[BarIndex] = 3.0f;
}

/*=====
This shows how to highlight a bar using the Draw Style set to Fill Top
and Fill Bottom.
-----*/
SCSFExport scsf_BarHighlight(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_FillTop = sc.Subgraph[0];
    SCSubgraphRef Subgraph_FillBottom = sc.Subgraph[1];

    // Set the configuration variables

    // Force these settings -- the user will not be able to change them
    sc.ScaleRangeType = SCALE_USERDEFINED;
    sc.ScaleRangeTop = 2.0f;
    sc.ScaleRangeBottom = 1.0f;
    sc.Subgraph[0].DrawStyle = DRAWSTYLE_FILL_RECTANGLE_TOP;
    sc.Subgraph[1].DrawStyle = DRAWSTYLE_FILL_RECTANGLE_BOTTOM;

    if (sc.SetDefaults)
    {

```



```

// Set the configuration and defaults

sc.GraphName = "Bar Highlight";

sc.StudyDescription = "Highlights a single bar at 10 bars back.";

sc.GraphRegion = 0;

Subgraph_FillTop.Name = "Fill Top";
Subgraph_FillTop.DrawStyle = DRAWSTYLE_LINE;
Subgraph_FillTop.PrimaryColor = RGB(127,0,0); // Dark red
Subgraph_FillTop.DrawZeros = false;

Subgraph_FillBottom.Name = "Fill Bottom";
Subgraph_FillBottom.DrawStyle = DRAWSTYLE_LINE;
Subgraph_FillBottom.PrimaryColor = RGB(127,0,0); // Dark red -- it's not really necessary to set this since the color
from Fill Top is always used
Subgraph_FillBottom.DrawZeros = false;

return;
}

// Do data processing

// Get the index that the vertical line should be placed at
int BarIndex = sc.ArraySize - 10; // 10 bars back

// Figure out the number of bars that have been added with this update
int BarsAdded = sc.ArraySize - sc.UpdateStartIndex - 1;

// Clear the old vertical line in case this is an update
if (BarIndex - BarsAdded >= 0)
{
    Subgraph_FillTop[BarIndex- BarsAdded] = 0.0f;
    Subgraph_FillBottom[BarIndex- BarsAdded] = 0.0f;
}

// Add the vertical line
if (BarIndex >= 0)
{
    Subgraph_FillTop[BarIndex] = 2.0f; // Top
    Subgraph_FillBottom[BarIndex] = 1.0f; // Bottom
}
}

/*=====
This shows some examples of how to use Persistent variables.
-----*/
SCSFExport scsf_PersistentVariablesExample(SCStudyInterfaceRef sc)
{
    // Set the configuration variables

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

        sc.GraphName = "Persistent Variables Example";

        sc.AutoLoop = 0;

```

//This is set to true to cause the persistent variables to continuously change as a test for when other charts are referencing them and verifying those other charts are getting updated automatically.

```
sc.UpdateAlways = 1;
return;
}

// You can use references to give the user variables more descriptive
// names that you can use below.
int& r_PersistentInteger = sc.GetPersistentInt(1);
float& r_PersistentFloat = sc.GetPersistentFloat(1);

if (sc.UpdateStartIndex == 0)
{
    // When there is a full recalculation of the study,
    // reset the persistent variables we are using
    r_PersistentInteger = 0;
    r_PersistentFloat = 0.0f;
}

r_PersistentInteger += 12;
r_PersistentFloat += 25.5f;

// Can also use the set function instead of r_PersistentFloat.
sc.SetPersistentFloat(2, 2.5f);
}

/*=====
Standard Error study
-----*/
SCSFExport scsf_StdError(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_StdErr = sc.Subgraph[0];

    SCInputRef Input_Length = sc.Input[0];

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

        sc.GraphName = "Standard Error";

        sc.StudyDescription = "";

        // Set the region to draw the graph in. Region zero is the main price graph region.
        sc.GraphRegion = 1;

        Subgraph_StdErr.Name = "Std Err";
        // Set the color and style of the graph line. If these are not set the default will be used.
        Subgraph_StdErr.PrimaryColor = RGB(255,0,0); // Red
        Subgraph_StdErr.DrawStyle = DRAWSTYLE_LINE;

        Input_Length.Name = "Length";
        Input_Length.SetInt(30);
        Input_Length.SetIntLimits(1,MAX_STUDY_LENGTH);

        sc.AutoLoop = 1;

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

```

}

// Do data processing

// Use the closing price as the input and the first subgraph for the output
sc.StdError(sc.Close, Subgraph_StdErr.Data, Input_Length.GetInt());
}

/*=====*/
SCSFExport scsf_T3(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_T3 = sc.Subgraph[0];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_Length = sc.Input[1];
    SCInputRef Input_Multiplier = sc.Input[2];

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

        sc.GraphName = "T3";
        sc.GraphRegion = 0;

        sc.StudyDescription = "T3 indicator with adjustable multiplier.";

        // Settings for the first subgraph
        Subgraph_T3.Name = "T3";
        Subgraph_T3.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_T3.PrimaryColor = RGB(0,255,0);
        Subgraph_T3.DrawZeros = false;

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

        Input_Length.Name = "Length";
        Input_Length.SetInt(50);
        Input_Length.SetIntLimits(1,MAX_STUDY_LENGTH);

        Input_Multiplier.Name = "Multiplier";
        Input_Multiplier.SetFloat(0.84f);

        sc.AutoLoop = 1;

        return;
    }

    if (sc.Index == 0)
    {
        for (int SubgraphIndex = 1; SubgraphIndex <= 6; SubgraphIndex++)
        {
            sc.Subgraph[SubgraphIndex].Name.Format("%d", SubgraphIndex);
            sc.Subgraph[SubgraphIndex].DrawStyle = DRAWSTYLE_IGNORE;
            sc.Subgraph[SubgraphIndex].DisplayNameValueInWindowsFlags = 0;
        }
    }

    // Do data processing
    sc.DataStartIndex = Input_Length.GetInt() - 1;

    // The T3 is calculated with a series of exponential moving averages. We

```

```

// use the internal extra arrays of sc.Subgraph 0 to hold the intermediate data
// that will be used to calculate the final result.

sc.T3MovingAverage(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_T3, Input_Multiplier.GetFloat(),
Input_Length.GetInt());

sc.Subgraph[1][sc.Index] = Subgraph_T3.Arrays[0][sc.Index];
sc.Subgraph[2][sc.Index] = Subgraph_T3.Arrays[1][sc.Index];
sc.Subgraph[3][sc.Index] = Subgraph_T3.Arrays[2][sc.Index];
sc.Subgraph[4][sc.Index] = Subgraph_T3.Arrays[3][sc.Index];
sc.Subgraph[5][sc.Index] = Subgraph_T3.Arrays[4][sc.Index];
sc.Subgraph[6][sc.Index] = Subgraph_T3.Arrays[5][sc.Index];
}

/*=====
This function simply shows some examples of working with date and time
values.
-----*/
SCSFExport scsf_DateAndTimeExample(SCStudyInterfaceRef sc)
{
    if (sc.SetDefaults)
    {
        // Set the configuration and defaults

        sc.GraphName = "Date and Time Example";

        sc.StudyDescription = "Example code for date and time related processing.";

        return;
    }

    // Do data processing

    // Make sure there's at least one bar
    if (sc.ArraySize < 1)
        return;

    SCString MessageString;

    // Get the index of the last bar
    int LastIndex = sc.ArraySize - 1;

    // Get the last date and time
    SCDateTime LastDateTime = sc.BaseDateTimeIn[LastIndex];
    int LastDate = LastDateTime.GetDate();
    int LastTime = LastDateTime.GetTimeInSeconds();
    // Write to the message log
    MessageString.Format("LastDate: %d (days since 1899-12-30), LastTime: %d (seconds since 0:00)", LastDate,
LastTime);
    sc.AddMessageToLog(MessageString, 1);

    // Get the date and time using the DateAt and TimeAt functions on the date-time array
    int Date = sc.BaseDateTimeIn.DateAt(LastIndex);
    int Time = sc.BaseDateTimeIn.TimeAt(LastIndex);
    // Write to the message log
    MessageString.Format("Date: %d, Time: %d", Date, Time);
    sc.AddMessageToLog(MessageString, 1);

    // Get the hour, minute, second for the time
    int Hour, Minute, Second;

```

```

TIME_TO_HMS(Time, Hour, Minute, Second);
// Write to the message log
MessageString.Format("Hour: %d, Minute: %d, Second: %d", Hour, Minute, Second);
sc.AddMessageToLog(MessageString, 1);

// Get the day of the week for the date
int DayOfWeek = DAY_OF_WEEK(Date);
// Write to the message log
MessageString.Format("Day of Week: %d (0 = Sunday, 1 = Monday)", DayOfWeek);
sc.AddMessageToLog(MessageString, 1);

// Construct a time value
int Time830 = HMS_TIME(8,30,0); // 8:30:00 AM
// Write to the message log
MessageString.Format("Time value for 08:30:00: %d", Time830);
sc.AddMessageToLog(MessageString, 1);

// Construct a DateTime value
int Yesterday = LastDate - 1;
int Evening = HMS_TIME(17,0,0); // 5:00:00 PM
SCDateTime YesterdayEvening = SCDateTime(Yesterday, Evening);
// Write to the message log
MessageString.Format("DateTime value for yesterday evening: %s",
sc.FormatDateTime(YesterdayEvening).GetChars());
sc.AddMessageToLog(MessageString, 1);

// Get the date and time back out of the SCDateTime value by using GetDate and GetTime
Date = YesterdayEvening.GetDate();
Time = YesterdayEvening.GetTimeInSeconds();
// Write to the message log
MessageString.Format("Date: %d, Time: %d", Date, Time);
sc.AddMessageToLog(MessageString, 1);

// Get the year/month/day of yesterday
int Year, Month, Day;
YesterdayEvening.GetDateYMD(Year, Month, Day);
// Write to the message log
MessageString.Format("Yesterday: %d-%02d-%02d", Year, Month, Day);
sc.AddMessageToLog(MessageString, 1);

// Construct the full date-time for Jan 1, 2000 at noon
SCDateTime DateTime(2000, 1, 1, 12, 0, 0);
// Write to the message log
MessageString.Format("Jan 1, 2000 at noon: %s",sc.FormatDateTime(DateTime).GetChars());
sc.AddMessageToLog(MessageString, 1);

// Print the year/month/day/hour/minute/second from the date-time
// constructed above. This should print 2000-01-01 12:00:00.
DateTime.GetDateTimeYMDHMS(Year, Month, Day, Hour, Minute, Second);
// Write to the message log
MessageString.Format("Jan 1, 2000 at noon from DateTime: %d-%02d-%02d %02d:%02d:%02d", Year, Month, Day,
Hour, Minute, Second);
sc.AddMessageToLog(MessageString, 1);

// Go back 38 hours and print that date-time
DateTime.SubtractHours(38);
DateTime.GetDateTimeYMDHMS(Year, Month, Day, Hour, Minute, Second);
// Write to the message log
MessageString.Format("38 hours back: %d-%02d-%02d %02d:%02d:%02d", Year, Month, Day, Hour, Minute,
Second);
sc.AddMessageToLog(MessageString, 1);

// Convert the DateTime from New York time to the time zone used for charts
DateTime = sc.ConvertDateTimeToChartTimeZone(DateTime, TIMEZONE_NEW_YORK);
DateTime.GetDateTimeYMDHMS(Year, Month, Day, Hour, Minute, Second);

```

```

// Write to the message log
MessageString.Format("Converted from New York time: %d-%02d-%02d %02d:%02d:%02d", Year, Month, Day, Hour,
Minute, Second);
sc.AddMessageToLog(MessageString, 1);
}

/*=====
   Graphs the highest and lowest value over the last Length bars.
-----*/
SCSFExport scsf_HighestAndLowest(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_HighestHigh = sc.Subgraph[0];
    SCSubgraphRef Subgraph_LowestLow = sc.Subgraph[1];

    SCInputRef Input_HighInputData = sc.Input[0];
    SCInputRef Input_LowInputData = sc.Input[1];
    SCInputRef Input_Length = sc.Input[2];

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

        sc.GraphName = "Highest High and Lowest Low";

        sc.StudyDescription = "This graphs the highest high and lowest low over a specified length.";

        sc.GraphRegion = 0;

        Subgraph_HighestHigh.Name = "Highest High";
        Subgraph_HighestHigh.DrawStyle = DRAWSTYLE_DASH;
        Subgraph_HighestHigh.PrimaryColor = RGB(0,255,0);

        Subgraph_LowestLow.Name = "Lowest Low";
        Subgraph_LowestLow.DrawStyle = DRAWSTYLE_DASH;
        Subgraph_LowestLow.PrimaryColor = RGB(255,0,255);

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

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

        Input_Length.Name = "Length";
        Input_Length.SetInt(30);
        Input_Length.SetIntLimits(1,MAX_STUDY_LENGTH);

        sc.AutoLoop = 1;

        return;
    }

    // Do data processing

    // Fill in the highest high and lowest low
    sc.Highest(sc.BaseDataIn[Input_HighInputData.GetInputDataIndex()], Subgraph_HighestHigh, Input_Length.GetInt());
    sc.Lowest(sc.BaseDataIn[Input_LowInputData.GetInputDataIndex()], Subgraph_LowestLow, Input_Length.GetInt());
}

/*=====
   Calculates the advance decline line.
-----*/

```

```

SCSFExport scsf_AdvanceDeclineLine(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_ADL = sc.Subgraph[0];
    SCInputRef Input_ResetAtSessionStart = sc.Input[1];

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

        sc.GraphName = "Advance Decline Line";

        sc.StudyDescription = "This calculates the advance decline line from a symbol that indicates advancing issues
        minus declining issues. It normally is added to a chart with the symbol NISS-NYSE.";

        sc.AutoLoop = 0;

        Subgraph_ADL.Name = "ADL";
        Subgraph_ADL.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_ADL.PrimaryColor = RGB(0,255,0);

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

    }

    return;
}

// Do data processing

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

if (sc.UpdateStartIndex == 0)
{
    Subgraph_ADL[0] = sc.Close[0];
}

for (int BarIndex = max(sc.UpdateStartIndex, 1); BarIndex < sc.ArraySize; BarIndex++)
{
    Subgraph_ADL[BarIndex] = Subgraph_ADL[BarIndex - 1] + sc.Close[BarIndex];

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

        if (IndexDateTime >= NextSessionStart)
        {
            Subgraph_ADL[BarIndex] = sc.Close[BarIndex];

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

}

/*****
SCSFExport scsf_AdvanceDeclineLine2Chart(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_ADL = sc.Subgraph[0];

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

```

```

sc.GraphName = "Advance Decline Line - 2 Chart";

sc.StudyDescription = "This calculates the advance decline line from two symbols that indicate advancing issues and declining issues.";


sc.AutoLoop= 1;

Subgraph_ADL.Name = "ADL";
Subgraph_ADL.DrawStyle = DRAWSTYLE_LINE;
Subgraph_ADL.PrimaryColor = RGB(0,255,0);

return;
}

// Do data processing

if (sc.Index < 1)
return;

// sc.GetCharArray()

Subgraph_ADL[sc.Index] = Subgraph_ADL[sc.Index - 1] + sc.Close[sc.Index];

}
/*****
=====
This uses two subgraphs designed to control the high and low of the
chart's scale so that the last bar fills a certain percentage of the
height of the chart.
-----*/
SCSFExport scsf_CustomScaling(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Top = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Bottom = sc.Subgraph[1];

    SCInputRef Input_PercentageInput = sc.Input[0];
    SCInputRef Input_DefaultRange = sc.Input[1];

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

        sc.GraphName = "Custom Scale Control";

        sc.StudyDescription = "Apply this study to a chart. Go to Chart >> Chart Settings, press Scale, and set the Scale Range to Same As Region. All the studies in chart region 1 also need to be set to Same As Region. The \"Percentage\" input is the percentage of the height of the chart that the last bar will fill from high to low. The \"Default Range\" input is the range that is used when the last bar has no range (i.e. high and low are the same).";

        sc.UpdateAlways = 1; // true
        sc.GraphRegion = 0;
        sc.AutoLoop = 0;

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

        Subgraph_Bottom.Name = "Bottom";
        Subgraph_Bottom.DrawStyle = DRAWSTYLE_HIDDEN;

```



```

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

Input_PercentageInput.Name = "Percentage";
Input_PercentageInput.SetFloat(30.0f);
Input_PercentageInput.SetFloatLimits(1.0f, 100.0f);

Input_DefaultRange.Name = "Default Range";
Input_DefaultRange.SetFloat(0.1f);
Input_DefaultRange.SetFloatLimits(0.00001f, FLT_MAX);

    return;
}

// Do data processing

// Persistent variables
int& LastScaleBar = sc.GetPersistentInt(1);

// Clear old scaling
Subgraph_Top[LastScaleBar] = 0.0f;
Subgraph_Bottom[LastScaleBar] = 0.0f;

// Get the high and low of the last bar
float LastBarHigh = sc.High[sc.IndexOfLastVisibleBar];
float LastBarLow = sc.Low[sc.IndexOfLastVisibleBar];

// Figure out the range and center of the last bar
float LastBarRange = LastBarHigh - LastBarLow;
float LastBarCenter = LastBarLow + LastBarRange / 2;

// Make sure we have a range other than zero
if (LastBarRange == 0)
    LastBarRange = Input_DefaultRange.GetFloat();

// Get the actual percentage
float Percentage = Input_PercentageInput.GetFloat() * 0.01f;
if (Percentage == 0.0f) // Prevent a divide-by-zero, though the input limits should already prevent this
    Percentage = 0.01f;

// Calculate the range for the scale
float ScaleRange = LastBarRange / Percentage;
float ScaleRangeTop = LastBarCenter + ScaleRange / 2;
float ScaleRangeBottom = LastBarCenter - ScaleRange / 2;

// Pick a bar in the middle to put the scale top and bottom on
int ScaleBarIndex = sc.IndexOfFirstVisibleBar + (sc.IndexOfLastVisibleBar - sc.IndexOfFirstVisibleBar) / 2;

Subgraph_Top[ScaleBarIndex] = ScaleRangeTop;
Subgraph_Bottom[ScaleBarIndex] = ScaleRangeBottom;

// Remember the bar we put the scaling on
LastScaleBar = ScaleBarIndex;

SCString NumberText;
NumberText.Format("%d", sc.Index);
sc.AddMessageToLog(NumberText, false);
}

```

```

/*=====
Example function for using sc.GetBarHasClosedStatus().
-----*/
SCSFExport scsf_BarHasClosedExample(SCStudyInterfaceRef sc)
{
    if (sc.SetDefaults)
    {
        // Set the configuration and defaults

        sc.GraphName = "Bar Has Closed";

        sc.StudyDescription = "This study is an example of how to use the BarHasClosedStatus member function.";

        return;
    }

    // Do data processing
    int status = sc.GetBarHasClosedStatus(sc.UpdateStartIndex);

    if (status == BHCS_BAR_HAS_CLOSED)
        sc.AddMessageToLog("Bar Has Closed", true);

    if (status == BHCS_BAR_HAS_NOT_CLOSED)
        sc.AddMessageToLog("Bar Has Not Closed", true);

    return;
}

/*=====
This study function colors bars according to if they are above or below a
given study subgraph.
-----*/
SCSFExport scsf_ColorBarAboveBelow(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_ColorBar = sc.Subgraph[0];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_StudyReference = sc.Input[1];
    SCInputRef Input_Mode = sc.Input[3];
    SCInputRef Input_OutputType = sc.Input[4];
    SCInputRef Input_Displacement = sc.Input[5];

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

        sc.GraphName = "Color Bar Based On Above/Below Study";

        sc.StudyDescription = "Colors the main price graph bars depending on if the price is above or below a given study
subgraph.";

        sc.GraphRegion = 0;
        sc.CalculationPrecedence = LOW_PREC_LEVEL;

        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 = false;

        Input_InputData.Name = "Input Data";

```

```

Input_InputData.SetInputDataIndex(SC_LAST);

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


Input_Mode.Name = "Mode";
Input_Mode.SetCustomInputStrings("Color Above/Below;Color Below;Color Above");
Input_Mode.SetCustomInputIndex(0);

Input_OutputType.Name = "Output Input Data Values Instead of 1.0";
Input_OutputType.SetYesNo(false);

Input_Displacement.Name = "Study Reference Displacement";
Input_Displacement.SetInt(0);


    sc.AutoLoop = 1;

    return;
}


// Do data processing

// Get the study subgraph
SCFloatArray SubgraphArray;
sc.GetStudyArrayUsingID(Input_StudyReference.GetStudyID(), Input_StudyReference.GetSubgraphIndex(),
SubgraphArray);
if (SubgraphArray.GetArraySize() == 0)
    return; // No subgraph data

// Get the values from the main price graph and the study subgraph
float BaseValue = sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index];
float SubgraphValue = SubgraphArray[sc.Index-Input_Displacement.GetInt()];

// Color above values using the primary color, and below values using the secondary color. The values in the ColorBar
Subgraph need to be non-zero for the main price graph bars to be colored.

float OutputValue = 1;

if (Input_OutputType.GetYesNo())
    OutputValue = BaseValue;

if (BaseValue > SubgraphValue && (Input_Mode.GetInt() == 0 || Input_Mode.GetInt() == 2))
{
    Subgraph_ColorBar[sc.Index] = OutputValue;// 1;
    Subgraph_ColorBar.DataColor[sc.Index] = Subgraph_ColorBar.PrimaryColor;
}
else if (BaseValue < SubgraphValue && (Input_Mode.GetInt() == 0 || Input_Mode.GetInt() == 1))
{
    Subgraph_ColorBar[sc.Index] = OutputValue;//1;
    Subgraph_ColorBar.DataColor[sc.Index] = Subgraph_ColorBar.SecondaryColor;
}
else
{
    Subgraph_ColorBar[sc.Index] = 0;
}
}

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

```

```

SCSubgraphRef Subgraph_VolatilityTrend = sc.Subgraph[0];
SCSubgraphRef Subgraph_TrueRange = sc.Subgraph[1];
SCSubgraphRef Subgraph_AverageTrueRange = sc.Subgraph[2];
SCSubgraphRef Subgraph_Direction = sc.Subgraph[3];
SCSubgraphRef Subgraph_DynamicPeriodLength = sc.Subgraph[4];

SCInputRef Input_ATRLength = sc.Input[0];
SCInputRef Input_ATRType = sc.Input[1];
SCInputRef Input_ATRMultiplier = sc.Input[2];
SCInputRef Input_MaxDynamicPeriod = sc.Input[3];
SCInputRef Input_DynamicPeriodDataInput = sc.Input[4];

// Configuration
if (sc.SetDefaults)
{
    sc.GraphName = "Volatility Trend Indicator";
    sc.StudyDescription = "Volatility Trend Indicator";
    sc.GraphRegion = 0; // Zero based chart region number

    Input_ATRLength.Name = "Average True Range Length";
    Input_ATRLength.SetInt(21);
    Input_ATRLength.SetIntLimits(1, MAX_STUDY_LENGTH);

    Input_ATRType.Name = "ATR Moving Average Type";
    Input_ATRType.SetMovAvgType(MOAVGTYPE_WEIGHTED);

    Input_ATRMultiplier.Name = "ATR Multiplier";
    Input_ATRMultiplier.SetFloat(3);

    Input_MaxDynamicPeriod.Name = "Max. Dynamic Period for Trend Calculation";
    Input_MaxDynamicPeriod.SetInt(50);
    Input_MaxDynamicPeriod.SetIntLimits(1, MAX_STUDY_LENGTH);

    Input_DynamicPeriodDataInput.Name = "Dynamic Period Input Data";
    Input_DynamicPeriodDataInput.SetInputDataIndex(SC_LAST);

    Subgraph_VolatilityTrend.Name = "Volatility Trend";
    Subgraph_VolatilityTrend.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_VolatilityTrend.PrimaryColor = RGB(0, 128, 0); // green
    Subgraph_VolatilityTrend.SecondaryColor = RGB(128, 0, 0); // red
    Subgraph_VolatilityTrend.SecondaryColorUsed = 1;
    Subgraph_VolatilityTrend.DrawZeros = false;

    Subgraph_DynamicPeriodLength.Name = "Dynamic Period Length";
    Subgraph_DynamicPeriodLength.DrawStyle = DRAWSTYLE_IGNORE;
    Subgraph_DynamicPeriodLength.PrimaryColor = RGB(255, 0, 0);
    Subgraph_DynamicPeriodLength.DrawZeros = false;

    sc.AutoLoop = 1;

    return;
}

uint32_t UpTrendColor = Subgraph_VolatilityTrend.PrimaryColor;
uint32_t DownTrendColor = Subgraph_VolatilityTrend.SecondaryColor;

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

// Data processing
if (sc.Index < 1)
    return;

```

```

    sc.ATR(sc.BaseDataIn, Subgraph_TrueRange, Subgraph_AverageTrueRange, Input_ATRLength.GetInt(),
    Input_ATRType.GetMovAvgType());

```

//trend is considered up when price bar value is greater than the volatility trend of the prior bar. Otherwise considered down.

```

    if (sc.BaseDataIn[Input_DynamicPeriodDataInput.GetInputDataIndex()][sc.Index] >
    Subgraph_VolatilityTrend[sc.Index-1])
        Subgraph_Direction[sc.Index] = 1; // uptrend
    else
        Subgraph_Direction[sc.Index] = -1; // downtrend

    int Period = static_cast<int>(Subgraph_DynamicPeriodLength[sc.Index-1]+0.5f);

    if (Subgraph_Direction[sc.Index] != Subgraph_Direction[sc.Index-1]) // Different trend than previous
        Period = 0;

    if (Period < Input_MaxDynamicPeriod.GetInt())
        Period++;

    Subgraph_DynamicPeriodLength[sc.Index] = static_cast<float>(Period);

    if (Subgraph_Direction[sc.Index] == 1) // uptrend
    {
        float PeriodHigh = sc.GetHighest(sc.BaseDataIn[Input_DynamicPeriodDataInput.GetInputDataIndex()], Period);

        Subgraph_VolatilityTrend[sc.Index] = PeriodHigh - Input_ATRMultiplier.GetFloat() *
    Subgraph_AverageTrueRange[sc.Index];
        Subgraph_VolatilityTrend.DataColor[sc.Index] = UpTrendColor;
    }
    else // downtrend
    {
        float PeriodLow = sc.GetLowest(sc.BaseDataIn[Input_DynamicPeriodDataInput.GetInputDataIndex()], Period);

        Subgraph_VolatilityTrend[sc.Index] = PeriodLow + Input_ATRMultiplier.GetFloat() *
    Subgraph_AverageTrueRange[sc.Index];
        Subgraph_VolatilityTrend.DataColor[sc.Index] = DownTrendColor;
    }
}

```

/\*=====\*/

```

SCSFExport scsf_AverageTrueRangeStop(SCStudyInterfaceRef sc)

```

```

{
    SCSubgraphRef Subgraph_StopL = sc.Subgraph[0];
    SCSubgraphRef Subgraph_StopS = sc.Subgraph[1];

    // Configuration
    if (sc.SetDefaults)
    {
        sc.GraphName = "Stop";
        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);

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

        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)

// ATR(10, Wilder's MA): subgraph #11. Intermediate TR: subgraph #10 (can't be removed)
sc.ATR(sc.BaseDataIn, sc.Subgraph[2], sc.Subgraph[3], 10, MOVAVGTYPE_WILDERS);

// LLV(L,20): subgraph #6
sc.Lowest(sc.Low, sc.Subgraph[6], 20);

// HHV(H,20): subgraph#7
sc.Highest(sc.High, sc.Subgraph[7], 20);

// LLV(L,p) + 2*ATR(p,m): subgraph #8
sc.Subgraph[8][sc.Index] = sc.Subgraph[6][sc.Index] + 2 * sc.Subgraph[3][sc.Index];

// HHV(H,p) - 2*ATR(p,m): subgraph #9
sc.Subgraph[9][sc.Index] = sc.Subgraph[7][sc.Index] - 2 * sc.Subgraph[3][sc.Index];

// Close - 2*ATR(p,m): subgraph #12
sc.Subgraph[12][sc.Index] = sc.Close[sc.Index] - 2 * sc.Subgraph[3][sc.Index];

// Close + 2*ATR(p,m): subgraph #13
sc.Subgraph[13][sc.Index] = sc.Close[sc.Index] + 2 * sc.Subgraph[3][sc.Index];

// Continuous LongStopValue: subgraph #14
sc.Highest(sc.Subgraph[12], sc.Subgraph[14], 15);

// Continuous ShortStopValue: subgraph #15
sc.Lowest(sc.Subgraph[13], sc.Subgraph[15], 15);

// If LongSignal=true, plot LongStopValue in Subgraph #0, otherwise plot null
if (sc.Close[sc.Index] > sc.Subgraph[8][sc.Index])
    Subgraph_StopL[sc.Index] = sc.Subgraph[14][sc.Index];
else
    Subgraph_StopL[sc.Index] = 0;

// If ShortSignal=true, plot ShortStopValue in Subgraph #1, otherwise plot null
if (sc.Close[sc.Index] < sc.Subgraph[9][sc.Index])
    Subgraph_StopS[sc.Index] = sc.Subgraph[15][sc.Index];
else
    Subgraph_StopS[sc.Index] = 0;
}

/*=====*/
SCSFExport scsf_DepthOfMarketData(SCStudyInterfaceRef sc) //scsf_DOMAccess
{
    SCSubgraphRef Subgraph_BidSize = sc.Subgraph[0];
    SCSubgraphRef Subgraph_BidValue = sc.Subgraph[1];
    SCSubgraphRef Subgraph_AskSize = sc.Subgraph[2];
    SCSubgraphRef Subgraph_AskValue = sc.Subgraph[3];
    SCSubgraphRef Subgraph_BidStackPullValue = sc.Subgraph[4];
    SCSubgraphRef Subgraph_AskStackPullValue = sc.Subgraph[5];

    if (sc.SetDefaults)
    {

```

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

```
sc.GraphName = "Depth of Market Data";
```

```
sc.StudyDescription = "This is a study to display current market depth data. When adding this study, also add the Spreadsheet Study to your chart to view the market depth data in a table form on the spreadsheet.";
```

```
sc.UsesMarketDepthData = 1;
```

```
Subgraph_BidSize.Name = "Bid Size";  
Subgraph_BidSize.DrawStyle = DRAWSTYLE_DASH;  
Subgraph_BidSize.PrimaryColor = RGB(0,255,0);  
Subgraph_BidSize.DrawZeros = false;
```

```
Subgraph_BidValue.Name = "Bid Value";  
Subgraph_BidValue.DrawStyle = DRAWSTYLE_DASH;  
Subgraph_BidValue.PrimaryColor = RGB(255,0,255);  
Subgraph_BidValue.DrawZeros = false;
```

```
Subgraph_AskSize.Name = "Ask Size";  
Subgraph_AskSize.DrawStyle = DRAWSTYLE_DASH;  
Subgraph_AskSize.PrimaryColor = RGB(255,255,0);  
Subgraph_AskSize.DrawZeros = false;
```

```
Subgraph_AskValue.Name = "Ask Value";  
Subgraph_AskValue.DrawStyle = DRAWSTYLE_DASH;  
Subgraph_AskValue.PrimaryColor = RGB(255,127,0);  
Subgraph_AskValue.DrawZeros = false;
```

```
Subgraph_AskStackPullValue.Name = "Ask Stack/Pull Value";  
Subgraph_AskStackPullValue.DrawStyle = DRAWSTYLE_IGNORE;  
Subgraph_AskStackPullValue.PrimaryColor = RGB(128, 128, 128);  
Subgraph_AskStackPullValue.DrawZeros = false;
```

```
Subgraph_BidStackPullValue.Name = "Bid Stack/Pull Value";  
Subgraph_BidStackPullValue.DrawStyle = DRAWSTYLE_IGNORE;  
Subgraph_BidStackPullValue.PrimaryColor = RGB(128, 128, 128);  
Subgraph_BidStackPullValue.DrawZeros = false;
```

```
return;  
}
```

```
// Do data processing
```

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

```
int MaximumMarketDepthLevels = sc.GetMaximumMarketDepthLevels();
```

```
if (MaximumMarketDepthLevels > sc.ArraySize)  
    MaximumMarketDepthLevels = sc.ArraySize;
```

```
int BidArrayLevels = min(MaximumMarketDepthLevels, sc.GetBidMarketDepthNumberOfLevels());
```

```
for (int Level = 0; Level < BidArrayLevels; Level++)  
{  
    int OutputIndex = sc.ArraySize - 1 - Level;
```

```
s_MarketDepthEntry DepthEntry;  
sc.GetBidMarketDepthEntryAtLevel(DepthEntry, Level);
```

```
Subgraph_BidSize[OutputIndex] = static_cast<float>(DepthEntry.Quantity);
```

```
Subgraph_BidValue[OutputIndex] = DepthEntry.AdjustedPrice;
```

```
float StackPullValue = static_cast<float>(sc.GetBidMarketDepthStackPullValueAtPrice(DepthEntry.AdjustedPrice));
```

```

    Subgraph_BidStackPullValue[OutputIndex] = StackPullValue;
}

int AskArrayLevels = min(MaximumMarketDepthLevels, sc.GetAskMarketDepthNumberOfLevels());

for (int Level = 0; Level < AskArrayLevels; Level++)
{
    int OutputIndex = sc.ArraySize - 1 - Level;

    s_MarketDepthEntry DepthEntry;
    sc.GetAskMarketDepthEntryAtLevel(DepthEntry, Level);

    Subgraph_AskSize[OutputIndex] = static_cast<float>(DepthEntry.Quantity);

    Subgraph_AskValue[OutputIndex] = DepthEntry.AdjustedPrice;

    float StackPullValue = static_cast<float>(sc.GetAskMarketDepthStackPullValueAtPrice(DepthEntry.AdjustedPrice));
    Subgraph_AskStackPullValue[OutputIndex] = StackPullValue;
}

int LastIndex = sc.ArraySize - MaximumMarketDepthLevels;

sc.EarliestUpdateSubgraphDataArrayIndex = LastIndex;

if (PriorIndex < LastIndex)
{
    for (int ClearIndex = LastIndex - 1; ClearIndex >= PriorIndex; ClearIndex--)
    {
        Subgraph_BidSize[ClearIndex] = 0;
        Subgraph_BidValue[ClearIndex] = 0;
        Subgraph_AskSize[ClearIndex] = 0;
        Subgraph_AskValue[ClearIndex] = 0;
        Subgraph_BidStackPullValue[ClearIndex] = 0;
        Subgraph_AskStackPullValue[ClearIndex] = 0;
    }
}

PriorIndex = (LastIndex >= 0) ? LastIndex : 0;
}

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

        sc.GraphName = "AutoLoop / Shorthand SubGraph Example";

        sc.StudyDescription = "This function demonstrates a shorthand way to access Subgraph Arrays and use Auto-Looping for simpler programming.";

        sc.AutoLoop = 1;

        return;
    }

    // Do data processing

```



```

    sc.Subgraph[0][sc.Index] = sc.Close[sc.Index];
}

/*=====*/
SCSFExport scsf_MutualFundBars(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Open = sc.Subgraph[0];
    SCSubgraphRef Subgraph_High = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Low = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Close = sc.Subgraph[3];

    // Configuration
    if (sc.SetDefaults)
    {
        sc.GraphName = "Mutual Fund Bars";
        sc.StudyDescription = "This study is used with Mutual Fund symbols and creates OHLC bars from the data.";
        sc.GraphRegion = 0; // zero based region number
        sc.AutoLoop= 1;
        sc.GraphDrawType = GDT_CANDLESTICK;

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

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

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

        Subgraph_Close.Name = "Close";
        Subgraph_Close.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Close.PrimaryColor = RGB(255,127,0);

        sc.StandardChartHeader = 1;
        sc.DisplayAsMainPriceGraph = 1;

        return;
    }

    // Data processing

    Subgraph_Open[sc.Index] = sc.Close[sc.Index - 1];
    Subgraph_High[sc.Index] = max(sc.Close[sc.Index], Subgraph_Open[sc.Index]);
    Subgraph_Low[sc.Index] = min(sc.Close[sc.Index], Subgraph_Open[sc.Index]);
    Subgraph_Close[sc.Index] = sc.Close[sc.Index];
}

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

        sc.GraphName = "String Examples";
    }
}

```

```

    sc.StudyDescription = "Working with Strings Examples.";

    return;
}

// Do data processing

// Comparison Example
SCString Message;
int ReturnValue;
Message.Format("This here is a Test");
ReturnValue = Message.CompareNoCase("This is a test");

// Direct String Access Example
const char* p_Symbol;
p_Symbol = sc.Symbol.GetChars();

SCString Left = Message.Left(4);
SCString Right = Message.Right(4);

Message = "Left: ";
Message += Left;
Message += ", Right: ";
Message += Right;
sc.AddMessageToLog(Message, 1);

Message.AppendFormat(", Length: %d", Message.GetLength());
sc.AddMessageToLog(Message, 0);

Message += ", DateTimeMS: ";
Message += sc.FormatDateTimeMS(sc.CurrentSystemDateTimeMS);
sc.AddMessageToLog(Message, 0);
}

/*=====
   This function demonstrates using sc.AutoLoop.
   =====*/
SCSFExport scsf_AutoLoopExample(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Average = sc.Subgraph[0];
    SCSubgraphRef Subgraph_BackReference = sc.Subgraph[1];
    SCSubgraphRef Subgraph_ForwardReference = sc.Subgraph[2];

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

        sc.GraphName = "Auto Loop Example";

        sc.StudyDescription = "This is an example of the new auto loop method for Advanced Custom Studies.";

        // Setting sc.AutoLoop to 1 (true) means looping is performed
        // automatically. This means that if there are 100 bars in your
        // chart, this function is called 100 times initially.
        sc.AutoLoop = 1; // true

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

```

```

Subgraph_BackReference.Name = "Back Reference Example";
Subgraph_BackReference.DrawStyle = DRAWSTYLE_LINE;
Subgraph_BackReference.PrimaryColor = RGB(255,0,255);

Subgraph_ForwardReference.Name = "Forward Reference Example";
Subgraph_ForwardReference.DrawStyle = DRAWSTYLE_LINE;
Subgraph_ForwardReference.PrimaryColor = RGB(255,255,0);

return;
}

// Do data processing

sc.SimpleMovAvg(sc.Close, Subgraph_Average, 10);

// The following line demonstrates referencing data one element back from
// the current index.
Subgraph_BackReference[sc.Index] = sc.Close[sc.Index - 1];

// The following line demonstrates referencing data one element forward
// from the current index.
Subgraph_ForwardReference[sc.Index] = sc.High[sc.Index + 1];
}

/*=====
This function demonstrates manual looping using a for loop.
-----*/
SCSFExport scsf_ManualLoopExample(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_HighLowDifference = sc.Subgraph[0];
    SCSubgraphRef Subgraph_HighLowAverage = sc.Subgraph[1];
    SCSubgraphRef Subgraph_BackdReference = sc.Subgraph[2];
    SCSubgraphRef Subgraph_ForwardReference = sc.Subgraph[3];

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

        sc.GraphName = "Manual Loop Example";

        sc.StudyDescription = "This is an example of using manual looping. It also demonstrates the new method to use
analysis functions within a for loop.";

        sc.AutoLoop = 0; // 0 is the default: there is no auto-looping

        Subgraph_HighLowDifference.Name = "High Low Difference";
        Subgraph_HighLowDifference.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_HighLowDifference.PrimaryColor = RGB(0,255,0);

        Subgraph_HighLowAverage.Name = "High - Low Average";
        Subgraph_HighLowAverage.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_HighLowAverage.PrimaryColor = RGB(255,0,255);

        Subgraph_BackdReference.Name = "Back Reference Example";
        Subgraph_BackdReference.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_BackdReference.PrimaryColor = RGB(255,255,0);

        Subgraph_ForwardReference.Name = "Forward Reference Example";
        Subgraph_ForwardReference.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_ForwardReference.PrimaryColor = RGB(255,127,0);

```

```

    return;
}

// Do data processing

for (int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize; BarIndex++)
{
    // Calculate the difference between the high and the low
    Subgraph_HighLowDifference[BarIndex] = sc.High[BarIndex] - sc.Low[BarIndex];

    // SimpleMovAvg will fill in the BarIndex data element in HighLowAverage
    // and not the entire array.
    sc.SimpleMovAvg(Subgraph_HighLowDifference, Subgraph_HighLowAverage, BarIndex, 10);

    // Copy the previous last price (BarIndex-1) to subgraph array number 3
    Subgraph_BackdReference[BarIndex] = sc.Close[BarIndex - 1];

    // Copy the next last price (BarIndex+1) to subgraph array number 4
    Subgraph_ForwardReference[BarIndex] = sc.Close[BarIndex + 1];
}
}

/*=====
-----*/
SCSFExport scsf_ColoredCandlesticks(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Open = sc.Subgraph[0];
    SCSubgraphRef Subgraph_High = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Low = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Close = sc.Subgraph[3];

    // Configuration
    if (sc.SetDefaults)
    {
        sc.GraphName = "Colored Candlesticks"; // study name
        sc.StudyDescription = "Colored candlesticks";
        sc.GraphRegion = 0; // zero based region number
        sc.GraphDrawType = GDT_CANDLESTICK;

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

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

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

        Subgraph_Close.Name = "Close";
        Subgraph_Close.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Close.PrimaryColor = RGB(255,127,0);

        sc.AutoLoop = 1;

        return;
    }
}

```

```

// Data processing.

sc.DataStartIndex = 1; // start drawing subgraphs at bar #n (zero based)

Subgraph_Open[sc.Index] = sc.Open[sc.Index];
Subgraph_High[sc.Index] = sc.High[sc.Index];
Subgraph_Low[sc.Index] = sc.Low[sc.Index];
Subgraph_Close[sc.Index] = sc.Close[sc.Index];

if (sc.Close[sc.Index] > sc.Close[sc.Index-1])
{
    Subgraph_Open.DataColor[sc.Index] = RGB(0,185,47); // green
    Subgraph_High.DataColor[sc.Index] = RGB(0,185,47);
    Subgraph_Low.DataColor[sc.Index] = RGB(0,185,47);
    Subgraph_Close.DataColor[sc.Index] = RGB(0,185,47);
}
else if (sc.Close[sc.Index] < sc.Close[sc.Index-1])
{
    Subgraph_Open.DataColor[sc.Index] = RGB(255,0,128); // red
    Subgraph_High.DataColor[sc.Index] = RGB(255,0,128);
    Subgraph_Low.DataColor[sc.Index] = RGB(255,0,128);
    Subgraph_Close.DataColor[sc.Index] = RGB(255,0,128);
}
else
{
    Subgraph_Open.DataColor[sc.Index] = RGB(130,205,251); // light blue
    Subgraph_High.DataColor[sc.Index] = RGB(130,205,251);
    Subgraph_Low.DataColor[sc.Index] = RGB(130,205,251);
    Subgraph_Close.DataColor[sc.Index] = RGB(130,205,251);
}
}

/*=====
This study function colors bars based on the comparison between open and
close.
-----*/
SCSFExport scsf_ColorBarOpenClose(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_ColorBar = sc.Subgraph[0];

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

        sc.GraphName = "Color Bar Open/Close";

        sc.StudyDescription = "This study function colors bars based on the comparison between open and close.";

        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);
        sc.AutoLoop = 1;

        return;
    }

    // Do data processing

```

```

// Get the values for open and close
float OpenValue = sc.Open[sc.Index];
float CloseValue = sc.Close[sc.Index];

if (CloseValue > OpenValue)
{
    Subgraph_ColorBar[sc.Index] = 1;
    Subgraph_ColorBar.DataColor[sc.Index] = Subgraph_ColorBar.PrimaryColor;
}
else if (CloseValue < OpenValue)
{
    Subgraph_ColorBar[sc.Index] = 1;
    Subgraph_ColorBar.DataColor[sc.Index] = Subgraph_ColorBar.SecondaryColor;
}
else
{
    float OpenValue = sc.Open[sc.Index-1];
    float CloseValue = sc.Close[sc.Index-1];

    if (CloseValue >= OpenValue)
    {
        Subgraph_ColorBar[sc.Index] = 1;
        Subgraph_ColorBar.DataColor[sc.Index] = Subgraph_ColorBar.PrimaryColor;
    }
    else if (CloseValue < OpenValue)
    {
        Subgraph_ColorBar[sc.Index] = 1;
        Subgraph_ColorBar.DataColor[sc.Index] = Subgraph_ColorBar.SecondaryColor;
    }
    /*else
    {
        ColorBar.DataColor[sc.Index] = 0;
        ColorBar[sc.Index] = 0;
    }*/
}
}

/*=====
Heikin-ashi

```

Formula Used:

Heikin Ashi close:  $haClose = (O + H + L + C) / 4$

Heikin Ashi open:  $haOpen = (\text{yesterday's } haOpen + \text{yesterday's } haClose) / 2$

Heikin Ashi high:  $haHigh = \text{the higher of High and today's } haOpen$

Heikin Ashi low:  $haLow = \text{the lower of Low and today's } haOpen$

-----\*/

SCSFExport scsf\_HeikinAshi(SCStudyInterfaceRef sc)

```

{
    SCSubgraphRef Subgraph_Open = sc.Subgraph[0];
    SCSubgraphRef Subgraph_High = sc.Subgraph[1];
    SCSubgraphRef Subgraph_Low = sc.Subgraph[2];
    SCSubgraphRef Subgraph_Last = sc.Subgraph[3];
    SCSubgraphRef Subgraph_Volume = sc.Subgraph[4];
    SCSubgraphRef Subgraph_NumTrades = sc.Subgraph[5];
    SCSubgraphRef Subgraph_OHLCAvg = sc.Subgraph[6];
    SCSubgraphRef Subgraph_HLCAvg = sc.Subgraph[7];
    SCSubgraphRef Subgraph_HLAvg = sc.Subgraph[8];
    SCSubgraphRef Subgraph_BidVol = sc.Subgraph[9];
    SCSubgraphRef Subgraph_AskVol = sc.Subgraph[10];
}

```

```

SCInputRef Input_SetCloseCurrentPrice = sc.Input[0];

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

    sc.GraphName = "Heikin-Ashi";

    sc.StudyDescription = "Heikin-Ashi.";

    sc.GraphDrawType = GDT_CANDLESTICK;
    sc.StandardChartHeader = 1;
    sc.GraphRegion = 1;
    sc.ValueFormat = VALUEFORMAT_INHERITED;

    //sc.DisplayAsMainPriceGraph = 1;

    Subgraph_Open.Name = "Open";
    Subgraph_Open.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_Open.PrimaryColor = RGB(0,255,0);
    Subgraph_Open.SecondaryColorUsed = true;
    Subgraph_Open.SecondaryColor = RGB(0,255,0);

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

    Subgraph_Low.Name = "Low";
    Subgraph_Low.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_Low.PrimaryColor = RGB(255,0,0);
    Subgraph_Low.SecondaryColorUsed = true;
    Subgraph_Low.SecondaryColor = RGB(255,0,0);

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

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

    Subgraph_NumTrades.Name = "NumTrades";
    Subgraph_NumTrades.DrawStyle = DRAWSTYLE_IGNORE;
    Subgraph_NumTrades.PrimaryColor = RGB(0,0,255);

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

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

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

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

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

```

```

Input_SetCloseCurrentPrice.Name = "Set Close to Current Price for Last Bar";
Input_SetCloseCurrentPrice.SetYesNo(true);

sc.AutoLoop = 1;

return;
}

//create a new name that includes the Base Graph name.
if (sc.Index == 0)
{
    sc.GraphName.Format("%s Heikin-Ashi", sc.GetStudyName(0).GetChars());
}

// Do data processing

const float OpenVal = sc.Open[sc.Index];
const float HighVal = sc.High[sc.Index];
const float LowVal = sc.Low[sc.Index];
const float CloseVal = sc.Close[sc.Index];

if (sc.Index == sc.ArraySize - 1 && Input_SetCloseCurrentPrice.GetYesNo())
    Subgraph_Last[sc.Index] = CloseVal;
else
    Subgraph_Last[sc.Index] = (OpenVal + HighVal + LowVal + CloseVal) / 4.0f;

if (sc.Index == 0)
    Subgraph_Open[sc.Index] = OpenVal;
else
{
    Subgraph_Open[sc.Index]
        = (Subgraph_Open[sc.Index - 1] + Subgraph_Last[sc.Index - 1]) / 2.0f;
}

Subgraph_High[sc.Index] = max(HighVal, Subgraph_Open[sc.Index]);
Subgraph_Low[sc.Index] = min(LowVal, Subgraph_Open[sc.Index]);

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

for(int SubgraphIndex = SC_BIDVOL; SubgraphIndex <= SC_ASK_PRICE; SubgraphIndex++)
{
    if(sc.BaseData[SubgraphIndex].GetArraySize() > 0)
        sc.Subgraph[SubgraphIndex][sc.Index] = sc.BaseData[SubgraphIndex][sc.Index];
}

sc.CalculateOHLCAverages();
}

/*=====
Heikin-Ashi Smoothed
-----*/
SCSFExport scsf_HeikinAshiSmoothed(SCStudyInterfaceRef sc)
{
    SCInputRef Input_MAType1      = sc.Input[1];
    SCInputRef Input_MAPeriod1    = sc.Input[2];
    SCInputRef Input_MAType2      = sc.Input[3];
    SCInputRef Input_MAPeriod2    = sc.Input[4];
    SCInputRef Input_SetCloseCurrentPrice = sc.Input[5];

```



```

SCSubgraphRef Subgraph_HASOpen = sc.Subgraph[0];
SCSubgraphRef Subgraph_HASHigh = sc.Subgraph[1];
SCSubgraphRef Subgraph_HASLow = sc.Subgraph[2];
SCSubgraphRef Subgraph_HASClose = sc.Subgraph[3];
SCSubgraphRef Subgraph_Volume = sc.Subgraph[4];

SCFloatArrayRef Array_OpenMA1 = Subgraph_HASOpen.Arrays[0];
SCFloatArrayRef Array_HighMA1 = Subgraph_HASHigh.Arrays[0];
SCFloatArrayRef Array_LowMA1 = Subgraph_HASLow.Arrays[0];
SCFloatArrayRef Array_CloseMA1 = Subgraph_HASClose.Arrays[0];

SCFloatArrayRef Array_HAOpen = Subgraph_HASOpen.Arrays[1];
SCFloatArrayRef Array_HAHigh = Subgraph_HASHigh.Arrays[1];
SCFloatArrayRef Array_HALow = Subgraph_HASLow.Arrays[1];
SCFloatArrayRef Array_HAClose = Subgraph_HASClose.Arrays[1];

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

    sc.GraphName = "Heikin-Ashi Smoothed";

    sc.GraphDrawType = GDT_CANDLESTICK;
    sc.StandardChartHeader = 1;
    sc.GraphRegion = 1;
    sc.ValueFormat = VALUEFORMAT_INHERITED;

    Input_MAType1.Name = "Moving Average Type 1";
    Input_MAType1.SetMovAvgType(MOAVGTYPE_SMOOTHED);

    Input_MAPeriod1.Name = "Moving Average Period 1";
    Input_MAPeriod1.SetInt(6);

    Input_MAType2.Name = "Moving Average Type 2";
    Input_MAType2.SetMovAvgType(MOAVGTYPE_WEIGHTED);

    Input_MAPeriod2.Name = "Moving Average Period 2";
    Input_MAPeriod2.SetInt(2);

    Input_SetCloseCurrentPrice.Name = "Set Close to Current Price for Last Bar";
    Input_SetCloseCurrentPrice.SetYesNo(0);

    Subgraph_HASOpen.Name = "HA Open";
    Subgraph_HASOpen.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_HASOpen.PrimaryColor = RGB(0,255,0);
    Subgraph_HASOpen.SecondaryColorUsed = true;
    Subgraph_HASOpen.SecondaryColor = RGB(0,255,0);

    Subgraph_HASHigh.Name = "HA High";
    Subgraph_HASHigh.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_HASHigh.PrimaryColor = RGB(128,255,128);

    Subgraph_HASLow.Name = "HA Low";
    Subgraph_HASLow.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_HASLow.PrimaryColor = RGB(255,0,0);
    Subgraph_HASLow.SecondaryColorUsed = true;
    Subgraph_HASLow.SecondaryColor = RGB(255,0,0);

    Subgraph_HASClose.Name = "HA Close";
    Subgraph_HASClose.DrawStyle = DRAWSTYLE_LINE;
    Subgraph_HASClose.PrimaryColor = RGB(255,128,128);

```

```

    sc.AutoLoop = 1;

    return;
}

// first smoothing
sc.MovingAverage(sc.Open, Array_OpenMA1, Input_MAType1.GetMovAvgType(), Input_MAPeriod1.GetInt());
sc.MovingAverage(sc.High, Array_HighMA1, Input_MAType1.GetMovAvgType(), Input_MAPeriod1.GetInt());
sc.MovingAverage(sc.Low, Array_LowMA1, Input_MAType1.GetMovAvgType(), Input_MAPeriod1.GetInt());
sc.MovingAverage(sc.Close, Array_CloseMA1, Input_MAType1.GetMovAvgType(), Input_MAPeriod1.GetInt());

// initial HA
Array_HAClose[sc.Index] = (Array_OpenMA1[sc.Index] + Array_HighMA1[sc.Index] + Array_LowMA1[sc.Index] +
Array_CloseMA1[sc.Index]) / 4.0f;

if (sc.Index == 0)
    Array_HAOpen[sc.Index] = Array_OpenMA1[sc.Index];
else
    Array_HAOpen[sc.Index] = (Array_HAOpen[sc.Index - 1] + Array_HAClose[sc.Index - 1]) / 2.0f;

Array_HAHigh[sc.Index] = max(Array_HighMA1[sc.Index], Array_HAOpen[sc.Index]);
Array_HALow[sc.Index] = min(Array_LowMA1[sc.Index], Array_HAOpen[sc.Index]);

// final smoothing
sc.MovingAverage(Array_HAOpen, Subgraph_HASOpen, Input_MAType2.GetMovAvgType(),
Input_MAPeriod2.GetInt());
sc.MovingAverage(Array_HAHigh, Subgraph_HASHigh, Input_MAType2.GetMovAvgType(),
Input_MAPeriod2.GetInt());
sc.MovingAverage(Array_HALow, Subgraph_HASLow, Input_MAType2.GetMovAvgType(),
Input_MAPeriod2.GetInt());
sc.MovingAverage(Array_HAClose, Subgraph_HASClose, Input_MAType2.GetMovAvgType(),
Input_MAPeriod2.GetInt());

if (sc.Index == sc.ArraySize - 1 && Input_SetCloseCurrentPrice.GetYesNo())
    Subgraph_HASClose[sc.Index] = sc.Close[sc.Index];

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

for(int SubgraphIndex = SC_BIDVOL; SubgraphIndex <= SC_ASK_PRICE; SubgraphIndex++)
{
    if(sc.BaseData[SubgraphIndex].GetArraySize() > 0)
        sc.Subgraph[SubgraphIndex][sc.Index] = sc.BaseData[SubgraphIndex][sc.Index];
}

sc.CalculateOHLCAverages();
}

/*=====
This function demonstrates using the extra arrays in a subgraph.
-----*/
SCSFExport scsf_ExtraArraysExample(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_CCI = sc.Subgraph[0];
    SCSubgraphRef Subgraph_CCIAverage = sc.Subgraph[1];

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

        sc.GraphName = "Extra Arrays Example";

```

```

sc.StudyDescription = "This function uses extra arrays in the subgraphs.";

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

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

// sc.Subgraph[2].Name = "Test";

sc.AutoLoop = 1; // true

return;
}

// Do data processing

sc.CCI(sc.Close, Subgraph_CCI, 10, 0.015f);

sc.SimpleMovAvg(Subgraph_CCI, Subgraph_CCIAverage.Arrays[0], sc.Index, 10);

Subgraph_CCIAverage[sc.Index] = Subgraph_CCIAverage.Arrays[0][sc.Index] + 300;

// sc.Subgraph[10].Arrays[0][sc.Index] = 0;
// sc.Subgraph[2][sc.Index] = sc.Subgraph[10].Arrays[0].GetArraySize();
}

/*=====
An Example on how to use the sc.GetOHLCForDate() function. This function
is most useful for intraday charts.
-----*/
SCSFExport scsf_GetOHLCTest(SCStudyInterfaceRef sc)
{
    if (sc.SetDefaults)
    {
        // Set the configuration and defaults

        sc.GraphName = "Get OHLC Test";

        sc.StudyDescription = "An example on how to use the sc.GetOHLCForDate() function. This function is most useful
for intraday charts.";

        sc.UpdateAlways = 1;

return;
    }

// Do data processing

float Open;
float High;
float Low;
float Close;

sc.GetOHLCForDate(sc.BaseDateTimeIn[sc.ArraySize-1], Open, High, Low, Close);
SCString message;

```

```

message.Format("O: %f, H: %f, L: %f, C: %f", Open, High, Low, Close);
sc.AddMessageToLog(message, 1);
}

/*=====
This is like a simple moving average, except it is calculating the median
instead of the average.
-----*/
SCSFExport scsf_MovingMedian(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Median = sc.Subgraph[0];

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

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

        sc.GraphName = "Moving Median";

        sc.StudyDescription = "This is like a simple moving average, except it is calculating the median instead of the average.";
        sc.GraphRegion = 0;
        sc.AutoLoop = 1;

        Subgraph_Median.Name = "Median";
        Subgraph_Median.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Median.PrimaryColor = RGB(0, 255, 0);

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

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

        return;
    }

    SCFloatArrayRef InData = sc.BaseDataIn[Input_InputData.GetInputDataIndex()];

    // Do data processing
    sc.MovingMedian(InData, Subgraph_Median, Input_Length.GetInt());
}

/*=====
Local function - used by scsf_PassingExtraArray
-----*/
float SumFunction(SCStudyInterfaceRef sc, SCFloatArrayRef array1)
{
    array1[sc.Index] = sc.Close[sc.Index] + array1[sc.Index - 1];
    return array1[sc.Index];
}

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

```

```

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

    sc.GraphName = "Passing Extra Array";

    sc.AutoLoop = 1;


    Subgraph_Output.Name = "Output";
    Subgraph_Output.DrawStyle = DRAWSTYLE_LINE;


    return;
}


// Do data processing
SCFloatArrayRef myArray = Subgraph_Output.Arrays[0] ;
Subgraph_Output[sc.Index] = SumFunction(sc,myArray);
}


/*=====
This function is an example of using the sc.IsUserAllowedForSCDLLName variable
-----*/
SCSFExport scsf_IsUserAllowedForSCDLLNameExample(SCStudyInterfaceRef sc)
{

    int UserAllowed1 = sc.IsUserAllowedForSCDLLName;
    int64_t ServiceLevel1 = sc.DLLNameUserServiceLevel;

    int UserAllowed2 = UserAllowed1;
    int64_t ServiceLevel2 = ServiceLevel1;

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

        sc.GraphName = "IsUserAllowedForSCDLLName";

        sc.StudyDescription = "This function is an example of using the sc.IsUserAllowedForSCDLLName variable to protect a study.";

        sc.AutoLoop = 1;

        return;
    }


    // Do data processing

    if(sc.IsUserAllowedForSCDLLName == false)
    {
        if(sc.Index == 0)
        {
            sc.AddMessageToLog("You are not allowed to use this study",1);
        }
        return;
    }

    //sc.DLLNameUserServiceLevel is set through the "Service Level (optional)" field on the Manage Custom Studies

```

Users control panel page.

//Additional authorization check assuming the service level is not used for other purposes. Check that the two upper bits (-4611686018427387904) are set. Can use any type of check.

```
if (sc.DLLNameUserServiceLevel >> 62 != 3)
{
    return;
}

if (sc.Index == 0)
{
    //sc.DLLNameUserServiceLevel is set through the "Service Level (optional)" field on the Manage Custom Studies
    Users control panel page.
    SCString Message;
    Message.Format("SCDLLName Service Level: %d", static_cast<int>(sc.DLLNameUserServiceLevel));
    sc.AddMessageToLog(Message, 1);
}
```

```
//if (ServiceLevel1 >> 62 != 3)
// return;
```

```
}
```

```
/*=====
This study function shows how to use the custom strings input type.
-----*/
```

```
SCSFExport scsf_CustomStringsInputExample(SCStudyInterfaceRef sc)
```

```
{
    SCInputRef Input_IceCreamFlavor = sc.Input[0];
    SCInputRef Input_Topping = sc.Input[1];

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

        sc.GraphName = "Custom Strings Input Example";

        sc.StudyDescription = "This study demonstrates using the custom string input type. This study does not calculate anything.";

        sc.AutoLoop = 0;

        Input_IceCreamFlavor.Name = "Ice Cream Flavor";
        Input_IceCreamFlavor.SetCustomInputStrings("Vanilla;Chocolate;Strawberry");
        Input_IceCreamFlavor.SetCustomInputIndex(1); // Default is Chocolate

        Input_Topping.Name = "Topping";
        Input_Topping.SetCustomInputStrings("None;Cherry;Sprinkles;Hot Fudge Sauce"); // Sorry, you can only select one
option with this input type
        Input_Topping.SetCustomInputIndex(0); // Default is None

        return;
    }

    // Do data processing

    switch (Input_IceCreamFlavor.GetIndex())
    {
        case 0:
            // Vanilla
            break;
```

```

    case 1:
        // Chocolate
    break;

    case 2:
        // Strawberry
    break;
}

if (Input_Topping.GetIndex() == 0)
{
    // No topping was selected
}

SCString Message;
Message.Format("You selected %s with %s.", Input_IceCreamFlavor.GetSelectedCustomString().GetChars(),
Input_Topping.GetSelectedCustomString().GetChars());
sc.AddMessageToLog(Message, 0);
}

/*=====
Example study function for the OpenChartOrGetChartReference function.
-----*/
SCSFExport scsf_OpenChartOrGetChartReferenceExample(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Output = sc.Subgraph[0];

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

        sc.GraphName = "OpenChartOrGetChartReference Example";

        sc.StudyDescription = "This study demonstrates using the OpenChartOrGetChartReference function. This study
does not calculate anything.";

        sc.AutoLoop = 0;//Manual looping

        Subgraph_Output.Name = "Output";
        Subgraph_Output.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_Output.PrimaryColor = RGB(0,255,0);

        return;
    }

    // Do data processing

    // Remember the chart number in a persistent variable to make the chart
    // lookup more efficient.
    int& r_ChartNumber = sc.GetPersistentIntFast(0);

    //Only do this on a full recalculation. Could also use sc.UpdateStartIndex == 0 in the case of manual looping which we
are using.
    if (sc.IsFullRecalculation)
    {
        s_ACSOpenChartParameters OpenChartParameters;
        OpenChartParameters.PriorChartNumber = r_ChartNumber;
        OpenChartParameters.ChartDataType = INTRADAY_DATA; //This can also be set to: DAILY_DATA
        OpenChartParameters.Symbol = sc.GetRealTimeSymbol();//When want to use the symbol of the chart the study
function is on, use sc.GetRealTimeSymbol()
        OpenChartParameters.IntradayBarPeriodType = IBPT_DAYS_MINS_SECS;
        OpenChartParameters.IntradayBarPeriodLength = 30 * SECONDS_PER_MINUTE; // 30 minutes
        OpenChartParameters.DaysToLoad = 0;//same as calling chart

```

```

// These are optional
OpenChartParameters.SessionStartTime.SetTimeHMS(12, 0, 0);
OpenChartParameters.SessionEndTime.SetTimeHMS(23,59,59);
//OpenChartParameters.EveningSessionStartTime.SetTimeHMS(0,0,0);
//OpenChartParameters.EveningSessionEndTime.SetTimeHMS(23,59,59);
OpenChartParameters.LoadWeekendData = 0;

OpenChartParameters.UseEveningSession = 1;
OpenChartParameters.HideNewChart = 1;
OpenChartParameters.UpdatePriorChartNumberParametersToMatch = 1;

r_ChartNumber = sc.OpenChartOrGetChartReference(OpenChartParameters);

}

if (r_ChartNumber != 0)
{
    SCGraphData ReferenceChartData;

    // Get the arrays from the reference chart. This will setup a reference to r_ChartNumber which causes this study to
    be fully recalculated under certain conditions like the completion of historical data downloading and when the chart is
    reloaded.
    sc.GetChartData(r_ChartNumber, ReferenceChartData);

    if (ReferenceChartData[SC_LAST].GetArraySize() > 0)
    {
        // Copy the reference chart array Last values to Subgraph 0.
        // Most likely the array from the reference chart is not
        // the same size as the array this study function is applied to.
        // Therefore, there is not going to be a correct column to column
        // correspondence. However, this is just a simple example.
        for (int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize; BarIndex++)
            Subgraph_Output[BarIndex] = ReferenceChartData[SC_LAST][BarIndex];
    }
}

//This is an example of opening a Historical Chart with a Weekly bar period
int& WeeklyChartNumber = sc.GetPersistentInt(2);

if (sc.IsFullRecalculation)
{
    s_ACSOpenChartParameters OpenChartParameters;
    //OpenChartParameters.Reset();
    OpenChartParameters.PriorChartNumber = WeeklyChartNumber;
    OpenChartParameters.ChartDataType = DAILY_DATA;
    OpenChartParameters.HistoricalChartBarPeriod = HISTORICAL_CHART_PERIOD_WEEKLY;
    OpenChartParameters.Symbol = sc.GetRealTimeSymbol();
    OpenChartParameters.DaysToLoad = 0;

    WeeklyChartNumber = sc.OpenChartOrGetChartReference(OpenChartParameters);
}
}

/*=====
This study displays a colored bar at the bottom of the graph to indicate
which session each bar is in.
-----*/
SCSFExport scsf_SessionIndicator(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Output = sc.Subgraph[0];

    // Force these settings. The user will not be able to change them.
    sc.ScaleRangeType = SCALE_USERDEFINED;

```



```

sc.ScaleRangeTop = 1000.0f;
sc.ScaleRangeBottom = 0.0f;

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

    sc.GraphName = "Session Indicator";

    sc.StudyDescription = "This study displays a colored bar at the bottom of the graph to indicate which session each
bar is in.";

    sc.AutoLoop = 1;


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

    sc.GlobalDisplayStudySubgraphsNameAndValue = 0; // false

    Subgraph_Output.Name = "Session Indicator";
    Subgraph_Output.DrawStyle = DRAWSTYLE_DASH;
    Subgraph_Output.LineWidth = 5;
    Subgraph_Output.SecondaryColorUsed = 1; // true
    Subgraph_Output.PrimaryColor = RGB(255,255,0);
    Subgraph_Output.SecondaryColor = RGB(0,127,255);
    Subgraph_Output.DrawZeros = 1;

    return;
}
// Do data processing

Subgraph_Output[sc.Index] = 1;

if (sc.IsDateTimeInDaySession(sc.BaseDateTimeIn[sc.Index]))
    Subgraph_Output.DataColor[sc.Index] = Subgraph_Output.PrimaryColor;
else
    Subgraph_Output.DataColor[sc.Index] = Subgraph_Output.SecondaryColor;
}

/*=====*/
SCSFExport scsf_DepthBarsMaxDepthQuantity(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_SubgraphMaxBidDepthQuantity = sc.Subgraph[0];
    SCSubgraphRef Subgraph_SubgraphMaxAskDepthQuantity = sc.Subgraph[1];

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

        sc.GraphName = "Depth Bars - Max Depth Quantity";

        sc.StudyDescription = "Graphs the maximum depth quantity found in each historical depth bar.";

        sc.AutoLoop = 1;

        Subgraph_SubgraphMaxBidDepthQuantity.Name = "Max Bid Depth Quantity";

        Subgraph_SubgraphMaxAskDepthQuantity.Name = "Max Ask Depth Quantity";

        return;
    }
}

```

```

// Do data processing.

// Get access to the depth bars in the current chart.
c_ACSILDepthBars* p_DepthBars = sc.GetMarketDepthBars();
if (p_DepthBars == NULL)
    return;

// Do nothing if the bar at the current index has no data.
if (!p_DepthBars->DepthDataExistsAt(sc.Index))
    return;

int MaxBidQuantityForBar = 0;
int MaxAskQuantityForBar = 0;

// Iterate through each price index for the bar at the current index.
int PriceTickIndex = p_DepthBars->GetBarLowestPriceTickIndex(sc.Index);
do
{
    const int MaxBidQuantityAtPriceTick
        = p_DepthBars->GetMaxBidQuantity(sc.Index, PriceTickIndex);

    if (MaxBidQuantityForBar < MaxBidQuantityAtPriceTick)
        MaxBidQuantityForBar = MaxBidQuantityAtPriceTick;

    const int MaxAskQuantityAtPriceTick
        = p_DepthBars->GetMaxAskQuantity(sc.Index, PriceTickIndex);

    if (MaxAskQuantityForBar < MaxAskQuantityAtPriceTick)
        MaxAskQuantityForBar = MaxAskQuantityAtPriceTick;
}
while (p_DepthBars->GetNextHigherPriceTickIndex(sc.Index, PriceTickIndex));

Subgraph_SubgraphMaxBidDepthQuantity[sc.Index] = static_cast<float>(MaxBidQuantityForBar);
Subgraph_SubgraphMaxAskDepthQuantity[sc.Index] = static_cast<float>(MaxAskQuantityForBar);
}

/*=====*/
SCSFExport scsf_LinearEstimation(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_Slope = sc.Subgraph[0];
    SCSubgraphRef Subgraph_Intercept = sc.Subgraph[1];

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

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

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

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

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

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

```

```

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

    return;
}

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

sc.LinearRegressionSlope(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_Slope,
Input_Length.GetInt());

sc.LinearRegressionIntercept(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Subgraph_Intercept,
Input_Length.GetInt());
}

/*=====*/
SCSFExport scsf_WaveTrendOscillator(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Subgraph_WT1 = sc.Subgraph[0];
    SCSubgraphRef Subgraph_WT2 = sc.Subgraph[1];
    SCSubgraphRef Subgraph_WTDiff = sc.Subgraph[2];
    SCSubgraphRef Subgraph_OB1 = sc.Subgraph[3];
    SCSubgraphRef Subgraph_OB2 = sc.Subgraph[4];
    SCSubgraphRef Subgraph_OS1 = sc.Subgraph[5];
    SCSubgraphRef Subgraph_OS2 = sc.Subgraph[6];

    SCFloatArrayRef Array_EMAPrice = sc.Subgraph[0].Arrays[0];
    SCFloatArrayRef Array_PriceDiff = sc.Subgraph[0].Arrays[1];
    SCFloatArrayRef Array_EMAPriceDiff = sc.Subgraph[0].Arrays[2];
    SCFloatArrayRef Array_CI = sc.Subgraph[0].Arrays[3];
    SCFloatArrayRef Array_TCI = sc.Subgraph[0].Arrays[4];

    SCInputRef Input_InputData = sc.Input[0];
    SCInputRef Input_MAType1 = sc.Input[1];
    SCInputRef Input_MAType2 = sc.Input[2];
    SCInputRef Input_ChannelLength = sc.Input[3];
    SCInputRef Input_AvgLength = sc.Input[4];
    SCInputRef Input_OB1 = sc.Input[5];
    SCInputRef Input_OB2 = sc.Input[6];
    SCInputRef Input_OS1 = sc.Input[7];
    SCInputRef Input_OS2 = sc.Input[8];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Wave Trend Oscillator";

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

        Subgraph_WT1.Name = "Wave Trend Oscillator";
        Subgraph_WT1.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_WT1.PrimaryColor = RGB(0, 255, 0);

        Subgraph_WT2.Name = "Smoothed Wave Trend Oscillator";
        Subgraph_WT2.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_WT2.PrimaryColor = RGB(255, 0, 0);

        Subgraph_WTDiff.Name = "Wave Trend Oscillator Difference";
        Subgraph_WTDiff.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_WTDiff.PrimaryColor = RGB(0, 0, 255);

        Subgraph_OB1.Name = "Overbought Level 1";
        Subgraph_OB1.DrawStyle = DRAWSTYLE_LINE;

```

```

Subgraph_OB1.PrimaryColor = RGB(255, 0, 0);

Subgraph_OB2.Name = "Overbought Level 2";
Subgraph_OB2.DrawStyle = DRAWSTYLE_POINT;
Subgraph_OB2.PrimaryColor = RGB(255, 0, 0);

Subgraph_OS1.Name = "Oversold Level 1";
Subgraph_OS1.DrawStyle = DRAWSTYLE_LINE;
Subgraph_OS1.PrimaryColor = RGB(0, 0, 255);

Subgraph_OS2.Name = "Oversold Level 2";
Subgraph_OS2.DrawStyle = DRAWSTYLE_POINT;
Subgraph_OS2.PrimaryColor = RGB(0, 0, 255);

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

Input_MAType1.Name = "Moving Average Type 1";
Input_MAType1.SetMovAvgType(MOVAVGTYPE_EXPONENTIAL);

Input_MAType2.Name = "Moving Average Type 2";
Input_MAType2.SetMovAvgType(MOVAVGTYPE_SIMPLE);

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

Input_AvgLength.Name = "Average Length";
Input_AvgLength.SetInt(21);
Input_AvgLength.SetIntLimits(1, INT_MAX);

Input_OB1.Name = "Overbought Level 1";
Input_OB1.SetFloat(60.0f);
Input_OB1.SetFloatLimits(0.0f, 100.0f);

Input_OB2.Name = "Overbought Level 2";
Input_OB2.SetFloat(53.0f);
Input_OB2.SetFloatLimits(0.0f, 100.0f);

Input_OS1.Name = "Oversold Level 1";
Input_OS1.SetFloat(-60.0f);
Input_OS1.SetFloatLimits(-100.0f, 0.0f);

Input_OS2.Name = "Oversold Level 2";
Input_OS2.SetFloat(-53.0f);
Input_OS2.SetFloatLimits(-100.0f, 0.0f);

return;
}

sc.MovingAverage(sc.BaseDataIn[Input_InputData.GetInputDataIndex()], Array_EMAPrice,
Input_MAType1.GetMovAvgType(), Input_ChannelLength.GetInt());

Array_PriceDiff[sc.Index] = fabs(sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index] -
Array_EMAPrice[sc.Index]);

sc.MovingAverage(Array_PriceDiff, Array_EMAPriceDiff, Input_MAType1.GetMovAvgType(),
Input_ChannelLength.GetInt());

if (Array_EMAPriceDiff[sc.Index] != 0.0f)
{
    Array_CI[sc.Index] =
        (sc.BaseDataIn[Input_InputData.GetInputDataIndex()][sc.Index] -
        Array_EMAPrice[sc.Index]) / (0.015f * Array_PriceDiff[sc.Index]);
}

```

```

else
    Array_CI[sc.Index] = 0.0f;

sc.MovingAverage(Array_CI, Array_TCI, Input_MAType1.GetMovAvgType(), Input_AvgLength.GetInt());

Subgraph_WT1[sc.Index] = Array_TCI[sc.Index];

sc.MovingAverage(Subgraph_WT1, Subgraph_WT2, Input_MAType2.GetMovAvgType(), 4);

Subgraph_WTDiff[sc.Index] = Subgraph_WT1[sc.Index] - Subgraph_WT2[sc.Index];

Subgraph_OB1[sc.Index] = Input_OB1.GetFloat();

Subgraph_OB2[sc.Index] = Input_OB2.GetFloat();

Subgraph_OS1[sc.Index] = Input_OS1.GetFloat();

Subgraph_OS2[sc.Index] = Input_OS2.GetFloat();
}

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

    // Set configuration variables

    if (sc.SetDefaults)
    {
        // Set defaults

        sc.GraphName = "Chart Graphics Settings Example";

        sc.StudyDescription = "This example will change the color of the chart background, when the price is above a simple moving average. Add this study during a chart replay for a demonstration.";

        sc.GraphRegion = 0;

        Subgraph_SimpleMA.Name = "Simple Moving Average";
        Subgraph_SimpleMA.DrawStyle = DRAWSTYLE_LINE;
        Subgraph_SimpleMA.PrimaryColor = RGB(0, 255, 0);

        sc.AutoLoop = 0;

        return;
    }

    // Do data processing
    if (sc.IsFullRecalculation)
    {
        //Specify to use the chart graphic settings.
        sc.SetUseGlobalGraphicsSettings(sc.ChartNumber, false);
    }

    uint32_t Color = 0;
    uint32_t LineWidth = 0;
    SubgraphLineStyle LineStyle = LINESTYLE_UNSET;

    for (int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize; BarIndex++)
    {
        // Simple moving average in the first subgraph
        sc.SimpleMovAvg(sc.Close, Subgraph_SimpleMA, BarIndex, 10);

        sc.GetGraphicsSetting(sc.ChartNumber, n_ACSIL::GRAPHICS_SETTING_CHART_BACKGROUND, Color,

```

LineWidth, LineStyle);

```
    if (Subgraph_SimpleMA.Data[BarIndex] < sc.Close[BarIndex])
    {
        if (Color != RGB(255, 0, 0))
        {
            Color = RGB(255, 0, 0);
            sc.SetGraphicsSetting(sc.ChartNumber, n_ACSIL::GRAPHICS_SETTING_CHART_BACKGROUND, Color);
        }

    }
    else
    {
        if (Color != RGB(0, 0, 0))
        {
            Color = RGB(0, 0, 0);
            sc.SetGraphicsSetting(sc.ChartNumber, n_ACSIL::GRAPHICS_SETTING_CHART_BACKGROUND, Color);
        }
    }
}
```

}

/\*=====\*/

// Note: Avoid putting anything below the template function so that the template  
// function is easy to find.

/\*=====

Add function description here.

-----\*/

SCSFExport scsf\_TemplateFunction(SCStudyInterfaceRef sc)

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

        sc.GraphName = "New Study";

        sc.StudyDescription = "Insert description here.";

        sc.AutoLoop = 1;

        return;
    }
}
```

// Do data processing

}

// Note: Avoid putting anything below the template function so that the template  
// function is easy to find.