

// The top of every source code file must include this line

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

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

This file contains Sierra Chart custom study template functions and example functions.

For reference, refer to the Advanced Custom Study Interface and Language documentation on the Sierra Chart website:

<http://www.sierrachart.com/index.php?page=doc/AdvancedCustomStudyInterfaceAndLanguage.php>

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

// This line is required. Change the text within the quote

// marks to what you want to name your group of custom studies.

```
SCDLLName("SC Custom Studies")
```

//This is the basic framework of a study function.

```
SCSFExport scsf_SkeletonFunction(SCStudyGraphRef sc)
```

```
{
```

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

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

```
    {
```

```
        sc.GraphName = "Skeleton Function";
```

```
        sc.AutoLoop = 1; // Automatic looping is enabled.
```

// During development set this flag to 1, so the DLL can be rebuilt without restarting Sierra Chart. When development is completed, set it to 0 to improve performance.

```
        sc.FreeDLL = 1;
```

```
        return;
```

```
    }
```

```
    // Data processing
```

```
}
```

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

This is another template function you can copy and modify to start a new study function.

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

```
SCSFExport scsf_TemplateFunction1(SCStudyGraphRef sc)
```

```
{
```

```
    // Set the configuration variables
```

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

```
    {
```

```
        sc.GraphName = "Template Function 1";
```

```
        sc.StudyDescription = "Insert description here.";
```

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

// During development set this flag to 1, so the DLL can be rebuilt without restarting Sierra Chart. When development is completed, set it to 0 to improve performance.

```
        sc.FreeDLL = 1;
```

```

    sc.Subgraph[0].Name = "Subgraph 1 Name";

    return;
}

// Data processing

// Fill in the first subgraph output data array. Modify this line to do what you want. This line simply will make this study
draw a line at the value of 1 .
    sc.Subgraph[0][sc.Index] = 1;
}

/*=====
This is another template function you can copy and modify to start a new study
function.
-----*/
SCSFExport scsf_TemplateFunction2(SCStudyGraphRef sc)
{
    // Section 1 - Set the configuration variables

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

        sc.GraphName = "New Study";

        sc.StudyDescription = "Insert description here.";

        sc.AutoLoop = 1; // true

        // During development set this flag to 1, so the DLL can be rebuilt without restarting Sierra Chart. When development
        is completed, set it to 0 to improve performance.
        sc.FreeDLL = 1;

        sc.Subgraph[0].Name = "Subgraph 1 Name";
        sc.Subgraph[0].PrimaryColor = RGB(255,0,0); // Red
        sc.Subgraph[0].DrawStyle = DRAWSTYLE_LINE; // Look in scconstants.h for other draw styles

        sc.Input[0].Name = "Input 1 Name";
        sc.Input[0].SetInt(20); // Set the default value to 20
        sc.Input[0].SetIntLimits(1, 1000); //Optional: Limit the range of this input to 1-1000

        return;
    }

    // Section 2 - Data processing

    // Fill in the output array. Modify this line to do what you want.
    sc.Subgraph[0][sc.Index] = (float)sc.Input[0].GetInt();
}

/*=====
'scsf_MovingAverageExample' shows some ways to use the Sierra Chart
intermediate study calculation functions available through the custom study
interface.
-----*/
SCSFExport scsf_MovingAverageExample(SCStudyGraphRef sc)
{
    // Section 1 - Set the configuration variables
    // This section is only run once. Refer to documentation for further details.
    if (sc.SetDefaults)

```

```

{
    // Set the configuration and defaults

    sc.GraphName = "Moving Average Example 1";

    sc.StudyDescription = "Insert study description here.";

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

    sc.GraphRegion = 0;

    sc.Subgraph[0].Name = "Simple Moving Average";
    sc.Subgraph[0].PrimaryColor = RGB(102,255,102); // Optionally, you may predefine your graph's colors
    sc.Subgraph[0].LineStyle = LINESTYLE_DOT; // Optionally, you may predefine a line style

    sc.Subgraph[1].Name = "Exponential Moving Average";
    sc.Subgraph[1].PrimaryColor = RGB(0,255,0); // Green
    sc.Subgraph[1].DrawStyle = DRAWSTYLE_POINT; // Here we set our default 'DrawStyle' to Point

    sc.Input[0].Name = "Moving Average Length"; // Your Input variables
    sc.Input[0].SetFloat(10.0f); // Here we set its default value to 10

    sc.Input[1].Name = "Plot Simple Moving Average";
    sc.Input[1].SetYesNo(1); // Yes

    sc.Input[2].Name = "Plot Exponential Moving Average";
    sc.Input[2].SetYesNo(0); // No

    sc.Input[3].Name = "Input Data";
    sc.Input[3].SetInputDataIndex(SC_LAST);

    return;
}

// Section 2 - Data processing

// Get the input values
int InMALength = sc.Input[0].GetInt(); // Input 0 - "Moving Average Length"
bool InShowSMA = sc.Input[1].GetYesNo(); // Input 1 - "Plot Simple Moving Average"
bool InShowEMA = sc.Input[2].GetYesNo(); // Input 2 - "Plot Exponential Moving Average"
int InputDataIndex = sc.Input[3].GetInputDataIndex();

// Set the index of the first array element to begin drawing at
sc.DataStartIndex = InMALength - 1;

if (InShowSMA)
{
    // We calculate the simple moving average of the last trade prices of the chart bars ('SC_LAST')
    // in the chart. Subgraph 0 is used to store and display the simple moving average.
    sc.SimpleMovAvg(
        sc.BaseDataIn[InputDataIndex], // This is the input array or source data. This is using the Last / Close from each
bar.
        sc.Subgraph[0].Data, // This designates your output array (subgraph/plot). This is using Subgraph 0.
        InMALength); // This is the length for the simple moving average, which is from Input 0.
    }

if (InShowEMA)
{
    // We calculate the exponential moving average of the last trade prices of the chart bars ('SC_LAST' by default) in
the chart.
    // Subgraph 1 is used to store and display the exponential moving average.

```

```

    sc.ExponentialMovAvg(
        sc.BaseDataIn[InputDataIndex], // This is the input array or source data.
        sc.Subgraph[1].Data, // This designates your output array (subgraph/plot). This is using Subgraph 1.
        InMALength); // This is the length for the exponential moving average, which is from Input 0.
    }
}

```

```

/*=====

```

This example Study Function calculates a Simple Moving Average.

'scsf\_SimpMovAvg' demonstrates the creation of a custom study.  
The chart data foundation of a chart are the 'sc.BaseDataIn[InputData]' arrays, where 'InputData' is one of the common chart elements such as 'SC\_OPEN' or 'SC\_VOLUME'.

The study function is called whenever there is new data and under other conditions.

Since auto looping is enabled, this function only needs to fill one index of the output arrays per call.

Note: All of Sierra Chart's arrays are zero based. That is, 1000 elements are designated from 0-999 (For example, Data[0] to Data[999]).

```

-----*/

```

```

SCSFExport scsf_SimpMovAvg(SCStudyGraphRef sc)

```

```

{
    // Section 1 - Set the configuration variables and defaults

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

        sc.StudyDescription = "Example function for calculating a simple moving average from scratch.";

        sc.AutoLoop = 1; // true
        sc.FreeDLL = 0;

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

        // Set the name of the first subgraph
        sc.Subgraph[0].Name = "Average";

        // Set the color and style of the subgraph line.
        sc.Subgraph[0].PrimaryColor = RGB(255,0,0); // Red
        sc.Subgraph[0].DrawStyle = DRAWSTYLE_LINE;

        sc.Input[0].Name = "Length";
        sc.Input[0].SetInt(30);
        sc.Input[0].SetIntLimits(1, 1000);

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

```

```

// Data processing

```

// Everything below here is run for each bar in the chart initially or when the chart is reloaded, and on every chart update.

```

// Get the inputs
int InMALength = sc.Input[0].GetInt(); // Input 0 - "Length"

```

```

// Set the index of the first array element to begin drawing at.
sc.DataStartIndex = InMALength - 1;

// Calculate a simple moving average from the data in the input arrays
// InMALength is 30 by default. Therefore, the 1st data element plotted begins with element
// 29 (InMALength - 1). If there are 1000 bars in the chart, then the last data element
// plotted begins with element 970 (999 - (InMALength - 1)) and goes to element 999.

// Add together the closing prices over the length for the prior bars
float Sum = 0;
for (int InIndex = sc.Index - (InMALength - 1); InIndex <= sc.Index; ++InIndex)
{
    // 'BaseDataIn[SC_LAST]' is the closing prices array for the chart
    Sum = Sum + sc.BaseDataIn[SC_LAST][InIndex];
}

// 'sc.Subgraph[0].Data' is the output array for the moving average to be graphed.
// The Sum is divided by the Length to get the average
sc.Subgraph[0].Data[sc.Index] = Sum / InMALength;
}

/*=====*/
SCSFExport scsf_HorizontalLine(SCStudyGraphRef sc)
{
    // Configuration
    if (sc.SetDefaults)
    {
        sc.GraphName = "Horizontal Line"; // study name shown in Chart Studies window

        sc.StudyDescription = "Test of horizontal line";

        sc.AutoLoop = 1; // true

        sc.GraphRegion = 1; // zero based chart region number

        // During development set this flag to 1, so the DLL can be rebuilt without restarting Sierra Chart. When development
        // is completed, set it to 0 to improve performance.
        sc.FreeDLL = 1;

        sc.Subgraph[0].Name = "Line";
        sc.Subgraph[0].DrawStyle = DRAWSTYLE_LINE;
        sc.Subgraph[0].PrimaryColor = RGB(0,255,0); // green

        sc.Input[0].Name = "Line Value";
        sc.Input[0].SetFloat(50); // default value
        sc.Input[0].SetFloatLimits(1, 20000); // min-max value

        return;
    }

    // Data processing

    sc.DataStartIndex = 0;

    float LineVal = sc.Input[0].GetFloat(); // get integer value from Input 0

    sc.Subgraph[0][sc.Index] = LineVal;
}

/*=====*/
SCSFExport scsf_SimpleArithmeticExample(SCStudyGraphRef sc)

```

```

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

        sc.GraphName = "Simple Arithmetic Example";

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

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

        sc.Subgraph[0].Name = "Line";
        sc.Subgraph[0].PrimaryColor = RGB(0,255,0); // Green

        return;
    }

    // Data processing

    sc.DataStartIndex = 0;

    sc.Subgraph[0][sc.Index] = (sc.BaseDataIn[SC_LAST][sc.Index] - sc.BaseDataIn[SC_LAST][sc.Index - 1]) * 10.0f;
}

```