

Referencing Other Time Frames and Symbols When Using the ACSIL

- [Introduction](#)
 - [Referencing Data from Other Time Frames by Using the Overlay Study](#)
 - [Referencing Data from Other Time Frames By Direct Referencing of the Other Timeframe](#)
 - [Example](#)
 - [Referencing Data from Different Symbols](#)
 - [Programmatically Opening Charts](#)
 - [Accessing Correct Array Indexes in Other Chart Arrays](#)
 - [Controlled Order Chart Updating](#)
-

Introduction

When using the [Advanced Custom Study Interface and Language \(ACSIL\)](#) and you want to reference bar data, custom or built-in studies data from charts with a different timeframe per bar, different Session Times, or a different Symbol, then this is possible using several different methods.

For example, you may want to access chart data from a chart with a timeframe of 5 minutes per bar or study data from the same chart and use that within a 1 minute per bar chart.

All of this information applies whether you are creating a study that displays values through line graphs or a trading system study that generates order signals and displays Buy/Sell arrow signals on the chart.

First, it must be understood that it is necessary in Sierra Chart to have the charts open for each other chart you want to work with when you want to reference data in a different timeframe per bar as compared to the chart you are referencing the data from.

For example, if you have a 1 minute per bar chart and you want to access the price bar data or study data from a 5 minute per bar chart, then you need to have the 1 minute chart open and the 5 minute chart open.

The advantage of this, is that you clearly see the data loaded, can verify that it is correct and can actually see the data you are working with and have a better understanding of it. All of these charts can be contained in the same chartbook and saved, so the setup can be immediately opened from the **File** menu with one step.

To simplify the process of accessing data from other charts, you are able to [Programmatically Opening Charts](#).

It is supported to reference data in a study function between different chart types. For example, if your study is applied to an Intraday chart, you can easily reference data from a Historical Daily chart. Or the

other way around. There are no restrictions.

Another use of the methods described here is to access data from different symbols rather than different time frames per bar. So the methods documented on this page also apply to accessing data from charts with different symbols. However, if you want to just access data for a different symbol but for the same timeframe per bar, refer to [Referencing Data from Different Symbols](#).

When using the **Study/Price Overlay** study or the ACSIL functions for referencing data from other charts in a Chartbook as documented on this page, this establishes an internal reference between these charts.

With this internal reference, when data changes in the source chart, the destination chart where the data is copied to, is notified of this change and the destination chart is updated and has access to the changes. This all happens automatically.

The different methods to accomplish all of the above are explained in the sections below, along with the different ways that each method can be used.

Referencing Data from Other Time Frames by Using the Overlay Study

The simplest and easiest method to reference data from other time frames per bar, whether this is main price graph bar data or study data, is to overlay that data on the destination chart by using the **Study/Price Overlay** study.

The destination chart will be the chart where your custom study is applied to. You can then directly access the data from the **Study/Price Overlay** study from within the code in your custom study.

For complete documentation for the **Study/Price Overlay** study, refer to [Study/Price Overlay Study](#).

Use the **Study/Price Overlay** study on the destination chart to reference the source chart data. Once you get the study configured as you require, then the data can be referenced from that study as explained below. If you cannot accomplish what you want with the overlay study, then what you require is not supported.

Once you have overlaid the studies on the destination chart, they can be hidden by enabling the **Hide Study** setting in the **Study Settings** window for the study. In this way they will not interfere with the view if you do not need to see them.

You are able to add multiple instances of the **Study/Price Overlay** study in order to overlay multiple price graphs and studies from other charts.

Once you have all of the chart bars and/or studies overlayed on the destination chart, then the next step is to reference the data in your custom study. Below is a code example which demonstrates this. The primary function that is used is [sc.GetStudyArrayUsingID](#). This code below can be found in **studies7.cpp** in the ACS_Source folder in the Sierra Chart program folder.

Example

```

SCSFExport scsf_ReferenceStudyData(SCStudyInterfaceRef sc)
{
    SCSubgraphRef Average = sc.Subgraph[0];
    SCInputRef Study1 = sc.Input[0];
    SCInputRef Study1Subgraph = sc.Input[1];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Reference Study Data";
        sc.StudyDescription = "This study function is an example of referencing data from other studies on the chart.";
        sc.AutoLoop = 1;

        // We must use a low precedence level to ensure
        // the other studies are already calculated first.
        sc.CalculationPrecedence = LOW_PREC_LEVEL;

        Average.Name = "Average";
        Average.DrawStyle = DRAWSTYLE_LINE;
        Study1.Name = "Input Study 1";
        Study1.SetStudyID(0);
        Study1Subgraph.Name = "Study 1 Subgraph";
        Study1Subgraph.SetSubgraphIndex(0);

        return;
    }

    // Get the Subgraph specified with the Study 1
    // Subgraph input from the study specified with
    // the Input Study 1 input.
    SCFloatArray Study1Array;

    sc.GetStudyArrayUsingID(Study1.GetStudyID(),Study1Subgraph.GetSubgraphIndex(),Study1Array);

    // We are getting the value of the study Subgraph
    // at sc.Index. For example, this could be
    // a moving average value if the study we got in
    // the prior step is a moving average.
    float RefStudyCurrentValue = Study1Array[sc.Index];

    // Here we will add 10 to this value and compute
    // an average of it. Since the moving average
    // function we will be calling requires an input
    // array, we will use one of the internal arrays
    // on a Subgraph to hold this intermediate
    // calculation. This internal array could be
    // thought of as a Spreadsheet column where you
    // are performing intermediate calculations.

    sc.Subgraph[0].Arrays[9][sc.Index] = RefStudyCurrentValue + 10;

    sc.SimpleMovAvg(sc.Subgraph[0].Arrays[9],sc.Subgraph[0],15);
}

```

Referencing Data from Other Time Frames By Direct Referencing of the Other Timeframe

If you need to reference bar data (Base Data) or study data from another chart with a different timeframe per bar, different Session Times, or a different Symbol, then this is possible by directly accessing the

sc.Subgraph[].Data arrays of the other chart.

A Subgraph is an array of data for an individual line within a study or an array that contains the Open, High, Low, or Close values of the main price graph bars.

For example, if you have a trading system study on chart #1 and you want to access a particular study with results needed for your trading system from chart #2, which has different Chart Settings, then you would make a function call to get the **sc.Subgraph[].Data** arrays for the study from chart #2.

The two functions usually used for this purpose are [sc.GetChartData](#) and [sc.GetStudyArraysFromChartUsingID](#).

For example, if you need to get the results of a Moving Average on a different timeframe, then that Moving Average must be on the other timeframe chart and you will need to make a function call to get the **sc.Subgraph[].Data** array that contains the Moving Average data.

This study on the other timeframe does not have to be visible. In the **Study Settings** window for the study, the **Hide Study** option can be enabled. Or, if your study function is referencing data from a custom study that you created, then that study that you created can use **sc.Subgraph[].Data** arrays that have no **sc.Subgraph[].Name** set. In which case those particular study Subgraph arrays will not even be visible on the source chart.

Below is a code example which demonstrates getting both the Base Data arrays and the study arrays from another chart. In the case of a study, this can be any study on another chart.

The code also demonstrates different methods of finding the corresponding index between the arrays for the two charts. For additional information about this, refer to [Accessing Correct Array Indexes in Other Chart Arrays](#).

Finding the corresponding array index is an essential step when working with arrays from another chart.

The code uses a [Study Input](#) to specify the **Chart Number** and **Study ID**.

The code below can be found in the **/ACS_Source/studies8.cpp** file in the Sierra Chart program folder.

It must also be made clear that it is not possible to get the the chart Base Data arrays from another chart, use intermediate study calculation functions like **sc.SimpleMovAvg** in the destination study function and pass one of those Base Data arrays from another chart, and calculate a moving average.

The general reason this does not work reliably is because the source and destination charts have different array sizes. In other words, they contain a different number of chart bars. The indexing in the arrays do not correspond directly to each other.

Example

```
SCSFExport scsf_ReferenceDataFromAnotherChart(SCStudyInterfaceRef sc)
{
    SCInputRef ChartStudy = sc.Input[0];

    if (sc.SetDefaults)
    {
        sc.GraphName = "Reference Data";
    }
}
```

```

sc.StudyDescription = "This is an example of referencing data from another chart.";
sc.AutoLoop = 1;

ChartStudy.Name = "Study Reference";
ChartStudy.SetChartStudyValues(1, 1);

return;
}

// The following code is for getting the High array
// and corresponding index from another chart.

// Define a graph data object to get all of the base graph data from the specified chart
SCGraphData BaseGraphData;

// Get the base graph data from the specified chart
sc.GetChartBaseData(ChartStudy.GetChartNumber(), BaseGraphData);

// Define a reference to the High array
SCFloatArrayRef HighArray = BaseGraphData[SC_HIGH];

// Array is empty. Nothing to do.
if(HighArray.GetArraySize() == 0)
    return;

// Get the index in the specified chart that is
// nearest to current index.
int RefChartIndex = sc.GetNearestMatchForDateTimeIndex(ChartStudy.GetChartNumber(), sc.Index);

float NearestRefChartHigh = HighArray[RefChartIndex];

// Get the index in the specified chart that contains
// the DateTime of the bar at the current index.
RefChartIndex = sc.GetContainingIndexForDateTimeIndex(ChartStudy.GetChartNumber(), sc.Index);

float ContainingRefChartHigh = HighArray[RefChartIndex];

// Get the index in the specified chart that exactly
// matches the DateTime of the current index.
RefChartIndex = sc.GetExactMatchForSCDateTime(ChartStudy.GetChartNumber(), sc.BaseDateTimeIn[sc.Index]);

if(RefChartIndex != -1)
{
    float ExactMatchRefChartHigh = HighArray[RefChartIndex];
}

// The following code is for getting a study Subgraph array
// and corresponding index from another chart.
// For example, this could be a moving average study Subgraph.

// Define a graph data object to get all of the study data
SCGraphData StudyData;

// Get the study data from the specified chart
sc.GetStudyArraysFromChartUsingID(ChartStudy.GetChartNumber(), ChartStudy.GetStudyID(), StudyData);

// Define a reference to the first Subgraph array
SCFloatArrayRef SubgraphArray = StudyData[0];

// Array is empty. Nothing to do.
if(SubgraphArray.GetArraySize() == 0)
    return;

// Get the index in the specified chart that is nearest
// to current index.
RefChartIndex = sc.GetNearestMatchForDateTimeIndex(ChartStudy.GetChartNumber(), sc.Index);

```

```

float NearestSubgraphValue = SubgraphArray[RefChartIndex];

// Get the index in the specified chart that contains
// the DateTime of the bar at the current index.
RefChartIndex = sc.GetContainingIndexForDateTimeIndex(ChartStudy.GetChartNumber(), sc.Index);

float ContainingSubgraphValue = SubgraphArray[RefChartIndex];

// Get the index in the specified chart that exactly
// matches the DateTime of the current index.
RefChartIndex = sc.GetExactMatchForSCDateTime(ChartStudy.GetChartNumber(), sc.BaseDateTimeIn[sc.Index]);

if(RefChartIndex != -1)//-1 means that there was not an exact match and therefore we do not have a valid index to
{
    float ExactMatchSubgraphValue = SubgraphArray[RefChartIndex];
}
}

```

Referencing Data from Different Symbols

When needing to reference data for a different symbol than the chart which needs to reference the different symbol, it is recommended to use the [Add Additional Symbol](#) study instead of the other methods documented on this page.

This method should only be used when you need to get data for a different symbol which has the same chart bar timeframe as the same chart the **Add Additional Symbol** study is applied to.

To access the data from this study, refer to [Using or Referencing Study/Indicator Data in an ACSIL Function](#).

Programmatically Opening Charts

When you are making references to other charts in your ACSIL study, you may want to automatically open those charts, so there is no need to manually open them.

This allows your study to be fully self-contained and does all that it needs to do, to reference the necessary data. You are able to open charts programmatically by using the [sc.OpenChartOrGetChartReference](#) function.

When a chart is programmatically opened it can also be hidden. This minimizes system resources. This can be done either programmatically or manually. In the case of manually, refer to [Window >> Hide Window](#).

Accessing Correct Array Indexes in Other Chart Arrays

When accessing main price graph and study arrays from other charts, it must be understood that the sizes of those arrays (the number of elements in them) are going to be different than the size of the arrays in the destination chart which contains the study function where you are calling the functions requesting these arrays.

The two functions to get data from another chart usually you should be using are:

- [sc.GetChartData](#)
- [sc.GetStudyArraysFromChartUsingID](#)

Use the **GetArraySize()** member function on the SCFloatArray class to get the size of an array.

For example, if you want to get the last element of an array from another chart use the following code:

Example

```
SCGraphData BaseData;  
sc.GetChartData(1, BaseData);  
int LastElementIndex = BaseData[SC_HIGH].GetArraySize() - 1;  
float HighValue = BaseData[SC_HIGH][LastElementIndex];
```

Use the following functions to find the corresponding array index in an array from another chart which corresponds to the [sc.BaseData\[\]\[\]](#) and [sc.Subgraph\[\].Data\[\]](#) array indexes in the chart you are using the array from another chart in:

- [sc.GetContainingIndexForDateTimeIndex](#)
- [sc.GetContainingIndexForSCDateTime](#)
- [sc.GetExactMatchForSCDateTime](#)
- [sc.GetNearestMatchForDateTimeIndex](#)
- [sc.GetNearestMatchForSCDateTime](#)

Controlled Order Chart Updating

When referencing data from other charts in an ACSIL study function, you may want to have the charts updated in an order according to dependency if you have a requirement for this. In other words, the charts being referenced will be calculated before the chart referencing them. This should only be done if you know that you require it. Otherwise, do not do this because it can make Sierra Chart less responsive.

For further information, refer to [Use Controlled Order Chart Updating](#).

*Last modified Wednesday, 22nd February, 2023.