

Working with ACSIL Arrays and Understanding Looping

- [Introduction](#)
 - [Overview of `sc.BaseData\[\]\[\]`, `sc.Subgraph\[\].Data\[\]`, `sc.Subgraph\[\].Arrays\[\]\[\]` Arrays](#)
 - [Array Indexing and Sizes](#)
 - [Automatic Array Bounds Correction](#)
 - [Automatic Looping/Iterating](#)
 - [Manual Looping/Iterating](#)
 - [When the Study Function Is Called](#)
 - [Array Types](#)
 - [Array Indexing in Trading DOM Windows](#)
 - [When Arrays are Cleared](#)
-

Introduction

This documentation page provides an explanation on how to work with the three most common arrays in the Sierra Chart Advanced Custom Study Interface.

It also discusses another important topic, the two methods of looping or iterating through all of the bars/columns in the chart. Looping through all of the bars/columns in the chart is necessary in order to fully calculate your study to produce a result that goes across the entire chart.

The two methods of looping are **Automatic Looping** and **Manual Looping**. These are exclusive of each other and you will never use both methods of looping in the same function. Otherwise, you would cause significant inefficiencies.

The preferred method is **Automatic** Looping.

Overview of `sc.BaseData[][]` `sc.Subgraph[].Data[]` `sc.Subgraph[].Arrays[][]`

The three most common arrays you will work with are:

- [sc.BaseData\[\]\[\]](#)
- [sc.Subgraph\[\].Data\[\]](#)
- [sc.Subgraph\[\].Arrays\[\]\[\]](#)

The **sc.BaseData[][]** arrays contain the data for the main price graph in the chart. The **sc.BaseData[][]** arrays are used as input data to your study.

The **sc.Subgraph[].Data[]** arrays, (shorthand notation: **sc.Subgraph[][]**), are for your studies displayable output and can be used to hold the results of background or intermediate calculations.

There are also the extra arrays, **sc.Subgraph[].Arrays[][]**, that can be used to hold the results of background or intermediate calculations for a **sc.Subgraph[].Data[]** array that is graphed on the chart.

The next two sections provide examples on how to work with these arrays.

A good way to understand how to organize your study calculations and use the **sc.Subgraph[]** arrays to hold the results of those calculations is to ask yourself how would I do this if I were using one of the Sierra Chart **Spreadsheet Studies**. If your study would require 2 Spreadsheet columns, one for a background calculation and another for the final result to be graphed and visible, then you would use for example **sc.Subgraph[0][sc.Index]** (shorthand notation) for the final result to be graphed. Make sure **sc.Subgraph[0].Name** is set for the Subgraph. For the background calculation you would use **sc.Subgraph[0].Arrays[0][sc.Index]**.

Array Indexing and Sizes

The [sc.BaseData\[\]\[\]](#) arrays have 2 indexing operators ([]). The second indexing operator is for accessing the individual values within the chart Base Data array specified by the first indexing operator.

For the second indexing operator, the first element starts at 0, and the last element is **sc.ArraySize - 1**. Examples: **sc.BaseData[SC_LAST][0]**, **sc.BaseData[SC_LAST][sc.ArraySize-1]**. An index value of 0 for the second indexing operator is the leftmost bar in the chart.

The size of the second array can also be determined by **sc.BaseData[0].GetArraySize()**. If this function returns 0, then the array is not allocated and is currently unused.

The **sc.BaseDateTimeIn[]** array has 1 indexing operator ([]). This operator is for accessing the individual [SCDateTime](#) variables within the **sc.BaseDateTimeIn[]** array. The first element starts at 0, and the last element is **sc.ArraySize - 1**. Examples: **sc.BaseDateTimeIn[0]** , **sc.BaseDateTimeIn[sc.ArraySize-1]**. The first element is the Date-Time of the leftmost bar in the chart. This array always contains the starting time of the chart bars.

The **sc.DateTimeOut[]** array has 1 indexing operator ([]). This operator is for accessing the individual [SCDateTime](#) variables within the **sc.DateTimeOut[]** array. The first element starts at 0, and the last element is **sc.OutArraySize - 1**. Examples: **sc.DateTimeOut[0]** , **sc.DateTimeOut[sc.OutArraySize-1]**. The first element is for the leftmost bar in the chart.

The **sc.Subgraph[][]** / **sc.Subgraph[].Data[]** arrays have 2 indexing operators ([]). **sc.Subgraph[][]** is a shorthand version of **sc.Subgraph[].Data[]**. The second indexing operator is for accessing the individual values within the Subgraph array specified by the first indexing operator. For the second indexing operator, the first element starts at 0, and the last element is **sc.ArraySize - 1**. Examples: Examples: **sc.Subgraph[0][0]** , **sc.Subgraph[0][sc.ArraySize-1]**. An index value of 0 for the second indexing operator is the leftmost bar in the chart.

If [sc.IsCustomChart](#) is set to 1, then the last element of `sc.Subgraph[].Data[]` is specified as **`sc.Subgraph[0].Data[sc.OutArraySize-1]`**.

The **`sc.Subgraph[].Arrays[][]`** arrays have 3 indexing operators (`[]`). The third indexing operator is for accessing the individual values within the specified Extra Array (second indexing operator) for the specified Subgraph (first indexing operator). For the third indexing operator, the first element starts at 0, and the last element is `sc.ArraySize - 1`. Examples: **`sc.Subgraph[0].Arrays[0][0]`** , **`sc.Subgraph[0].Arrays[0][sc.ArraySize-1]`** . An index value of 0 for the third indexing operator is the leftmost bar in the chart.

If [sc.IsCustomChart](#) is set to 1, then the last element of **`sc.Subgraph[].Arrays[][]`** is specified as **`sc.Subgraph[0].Arrays[0][sc.OutArraySize-1]`**.

With all of the above arrays and in the case of Automatic Looping, you would access the element at the current index (the index at which your study function needs to perform calculations at) by using **`sc.Index`** with the last indexing operator or with the single indexing operator if there is only one indexing operator supported with the array.

Example: **`sc.BaseData[SC_LAST][sc.Index]`**. This will be understood better when you review the [Automatic Looping/Iterating](#) section.

To get the size of an array, use the **`GetArraySize()`** member function. Example: **`sc.Subgraph[0].Data.GetArraySize()`**. This is particularly useful when you have accessed an array from another chart which has different array sizes compared to the chart where the arrays were gotten from.

Automatic Array Bounds Correction

With all Advanced Custom Study Interface and Language provided arrays, they are all safe with the use of invalid index values with the indexing operators.

When the specified index value is out of bounds, it will automatically be corrected to be just within the nearby bound. For example using a negative number with an indexing operator will result in the index being set to zero.

When using an index value which is greater than or equal to the array size, will result in the index being set to the array size minus 1. For example the following code will access the first element of the array and will not cause any type of error or exception: **`sc.Subgraph[].Data[-1]`**.

Automatic Looping/Iterating

When your study function uses automatic looping, then it is *automatically* called once for every bar or column in the chart when the study is initially calculated. If there are 100 bars in the chart, then it will be called 100 times when your study is initially calculated. After that, the study function is called as the latest bar is updated and new bars are added.

`sc.Index` is set to the index of the bar/column in the chart that your study function is being called for. This is a zero (0) based index. During normal chart updating, **`sc.Index`** will initially start at the **`sc.Index`** value of the last prior call to the study function. This will be the index of the last bar in the chart before

any new bars have been added.

For information about when the study function is called during normal chart updating after the initial calculation of the study, refer to [When the Study Function Is Called](#).

sc.CurrentIndex and **sc.Index** are the same. They are two different variables that are set to the same index value always. You can use either one. Normally the documentation will refer to **sc.Index**. **sc.Index** is equal to the elements in the [sc.BaseData\[\]](#) arrays that need to be processed and/or the elements in the [sc.Subgraph\[\]](#) arrays that need to be filled in. This will be more clear when you look at the code example below.

If you are creating a custom chart by setting [sc.IsCustomChart](#) to 1 (true), this is very unlikely, then **sc.Index** only refers to the elements in the [sc.BaseData\[\]](#) arrays to process, assuming your custom chart function uses the [sc.BaseData\[\]](#) arrays.

Automatic looping is activated by setting [sc.AutoLoop](#) = 1; in the code block at the top of your function for setting the defaults and configuration. When you create a new Advanced Custom Study file, the template function in that file, sets this variable to 1 (true). Therefore, by default, automatic looping is done for new functions. However, if **sc.AutoLoop** is not set, then its value will be zero and automatic looping will be off.

sc.Index initially starts at 0 and increments up to `sc.ArraySize - 1` when the study is fully recalculated. This happens when the chart is loaded or reloaded. Each time it increments, your study function is called again. There can be other cases for a full recalculation. For example, when you add or remove a study from a chart. Another case a full recalculation can occur is when you are using a custom chart such as the Renko Chart study or the Point and Figure Chart study and a new bar which was added by one of the studies is removed. In this case a full recalculation of the other studies is necessary. In this case **sc.Index** will start back at 0 and increment back up to the `sc.ArraySize-1`.

Here is an example of writing code that supports [automatic looping](#):

```

/*=====
This function demonstrates using sc.AutoLoop.
-----*/
SCSFExport scsf_AutoLoopExample(SCStudyInterfaceRef sc)
{
    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

        sc.Subgraph[0].Name = "Average";
        sc.Subgraph[1].Name = "Back Reference Example";

        sc.Subgraph[3].Name = "Current Low Price";
        return;
    }

    // Do data processing
    sc.SimpleMovAvg(sc.BaseData[SC_LAST], sc.Subgraph[0], sc.Index, 10);

    // The following line demonstrates referencing data one element back from
    // the current index.
    sc.Subgraph[1][sc.Index] = sc.BaseData[SC_LAST][sc.Index - 1];

    // The following line demonstrates referencing data at the current index.
    sc.Subgraph[3][sc.Index] = sc.BaseData[SC_LOW][sc.Index];
}

```

Manual Looping/Iterating

This section describes manual array element looping or iterating. Manual looping is going to be more efficient than automatic looping because the study function is called only once during a full recalculation.

However, [Automatic Looping](#) is easier to use unless your study function does not require automatic looping or it would instead work best with manual looping. For example, you would want to use [manual looping](#) if the custom study does not need to perform a calculation at each chart bar.

Manual looping needs to be properly implemented. If it is not implemented correctly by using [sc.UpdateStartIndex](#), then it becomes very inefficient.

If **sc.AutoLoop** is set to zero (0), which is the default if it is not specified in the [sc.SetDefaults](#) code block at the top of the study function for setting the defaults and configuration, then manual looping is specified and you need to use a **for** loop in the study function to iterate through all the **sc.BaseData[][]** and **sc.Subgraph[][]** data array elements.

The function must use the [sc.UpdateStartIndex](#) variable to determine what element Index to begin the **for** loop at. If you were to start the loop at position zero always, your study will be very inefficient. Instead you must use **sc.UpdateStartIndex**.

sc.Index is not used with manual looping.

During normal chart updating and after the initial study calculation, refer to [When the Study Function Is Called](#) to know when the study function will be called.

When there is an event which will cause the study function to be called, it will be called at the **Chart Update Interval** set in **Global Settings >> General Settings**, and not more often. So it does not happen immediately upon, for example new market data received. It will be soon as the **Chart Update Interval** has elapsed and there is new data which causes an update.

The following is more information on **sc.UpdateStartIndex** and example code for manual looping:

More information about sc.UpdateStartIndex

[sc.UpdateStartIndex](#) is set by Sierra Chart to the index where your primary **for** loop will start looping from. This is the index in the [sc.BaseData\[\]](#) arrays where updating has begun. This is the same index where updating should begin in the [sc.Subgraph\[\]](#) arrays.

If you are creating a custom chart, [sc.IsCustomChart](#) is set to true (this is very unlikely), then **sc.UpdateStartIndex** only refers to the [sc.BaseData\[\]](#) element to process.

Example

```
for (int Index = sc.UpdateStartIndex; Index < sc.ArraySize; ++Index)
{
    // fill in the first subgraph with the last values
    sc.Subgraph[0][Index] = sc.BaseData[SC_LAST][Index];
}
```

The above loop will always fill in and update the necessary elements in the one output (Subgraph) array we are using ([sc.Subgraph\[0\]](#)). If you are using manual looping, most studies will require a primary **for** loop to iterate through the elements in the arrays unless one is clearly not required based upon what the study function is doing.

This is a good example of the primary loop that you will need to use. You will begin at **sc.UpdateStartIndex**. You will loop from there up to, but not including, [sc.ArraySize](#). You do not include [sc.ArraySize](#) because the arrays are zero-based, meaning the last element in the array is at the index [sc.ArraySize - 1](#).

See below for a complete study function using manual looping.

Example Code

```

/*=====
This function demonstrates manual looping using a for loop.
-----*/
SCSFExport scsf_ManualLoopExample(SCStudyInterfaceRef sc)
{
    if (sc.SetDefaults)
    {
        // Set the configuration and defaults
        sc.GraphName = "Manual Loop Example";

        sc.StudyDescription = "This is an example of using manual looping.";
        sc.AutoLoop = 0; // 0 is the default: there is no auto-looping

        sc.Subgraph[0].Name = "High Low Difference";
        sc.Subgraph[1].Name = "High - Low Average";

        sc.Subgraph[2].Name = "Back Reference Example";
        sc.Subgraph[3].Name = "Forward Reference Example";

        return;
    }

    // Do data processing
    for (int Index = sc.UpdateStartIndex; Index < sc.ArraySize; Index++)
    {
        // Calculate the difference between the high and the low
        sc.Subgraph[0][Index] = sc.BaseData[SC_HIGH][Index] - sc.BaseData[SC_LOW][Index];

        // SimpleMovAvg will fill in the data element in sc.Subgraph[1] at index Index.
        sc.SimpleMovAvg(sc.Subgraph[0], sc.Subgraph[1], Index, 10);

        // Copy the previous last price (Index-1) to subgraph array number 3
        sc.Subgraph[2][Index] = sc.BaseData[SC_LAST][Index - 1];

        // Copy the next last price (Index+1) to subgraph array number 4
        sc.Subgraph[3][Index] = sc.BaseData[SC_LAST][Index + 1];
    }
}

```

When the Study Function is Called

This section describes the different conditions for when a study function is called by a chart it is contained within. A study function begins with **scsf_** and is contained in a CPP file. This file is compiled into a DLL file which is the executable file.

The study function will be called separately for each instance of the study.

Initial/First Call

The initial or first call into a study function occurs when a study instance is added to a chart, a Chartbook is opened and a chart in that Chartbook contains an instance of the study, or a Study Collection is applied to a chart and an instance of the study is part of that Study Collection.

In the case when a Chartbook is opened, a study function for study instance on that chart is only called when the chart data loading is complete for that chart. When that data loading is complete, all of the studies on that chart are fully recalculated.

There are at least two calls to the study function for the conditions described above. Once to set the defaults of the study. In this case [sc.SetDefaults](#) is set to 1. The second call is for the full calculation of the study.

During this second call into the study function there is a full calculation. Read the description below.

Full Calculation/Recalculations

A study will be fully calculated/recalculated when it is added to a chart, any time its Input settings are changed, another study is added or removed from a chart, when the Study Window is closed with OK or the settings are applied. Or under other conditions which can cause a full recalculation.

Other conditions include: When the chart the study is applied to is being referenced by another chart and the other chart has been fully recalculated, the chart data loaded, or a historical data download finishes. For more information, refer to [References to Other Charts and Tagging](#). The rebuilding of the internal Trades List in a chart, if it is being used, which can occur when changing the Trade Account on the chart or enabling or disabling Trade Simulation Mode.

In the case of [manual looping](#), the study function will be called 1 time with **sc.UpdateStartIndex** set to 0, and usually will be called again when the chart is immediately updated after. On this additional call, **sc.UpdateStartIndex** is set to **sc.ArraySize - 1**.

In the case of [automatic looping](#), the study function will be called once for each bar in the chart.

sc.Index starts at 0 and increments for each bar in the chart. Therefore, this is a series of calls which can be many thousands of times. Once for each chart bar. This group of function calls is considered the full calculation of the study.

After this full calculation is complete in the case of automatic looping, usually there will be an additional call into this study function when the chart is immediately updated after. During this additional call, **sc.Index** is set to **sc.ArraySize - 1**.

Update Study Function Calls

A study function will be called with either manual or automatic looping, with one or more of the conditions given below. None of the conditions described below, cause an immediate call. They flag that the study function needs to be called, and the call occurs at the **Chart Update Interval** set in **Global Settings >> General Settings**.

Each individual chart can also override this setting and use their own independent Chart Update Interval. We recommend charts be set to use their own independent [Chart Update Interval](#) if they need to use either a shorter or longer interval compared to the global settings.

The call into the study function will just be an update call with **sc.Index/sc.UpdateStartIndex** set to the [prior array size - 1](#). The following are the conditions for when this call happens.

- There is new trade market data received

- Historical downloaded data received
- Bid and ask data received
- Market depth data received
- Fundamental data received
- New orders or order updates for the symbol and trade account the chart is set to.
- Trade Position updates for the symbol and trade account the chart is set to.
- When records are read from the chart data file during a Replay of an Intraday chart
- When using **sc.UpdateAlways = 1**
- The **Trade >> Trade Simulation Mode On** state has been changed. This causes a call because of the order related data for the chart has changed even though there may not be any orders.

In these cases the study function will be called when the **Chart Update Interval** set in **Global Settings >> General Settings** elapses, and not more often. Therefore, it does not happen immediately upon, for example a new trade occurring for the symbol. It will be soon as the **Chart Update Interval** has elapsed and there is new data which causes an update for one of the given reasons.

During an accelerated Chart Replay, the **Chart Update Interval** is reduced internally to a shorter time.

To access individual trades and bid/ask updates in between calls into the study function, use the following functions:

- [sc.GetTimeAndSales](#)
- [sc.GetTimeAndSalesForSymbol](#)
- [sc.ReadIntradayFileRecordAtIndex](#)
- [sc.ReadIntradayFileRecordForBarIndexAndSubIndex](#)

If a study function takes longer to calculate than the Chart Update Interval, this will lead to skipping of update calculations. For example, if a study takes 200 ms to calculate, which is a very long time, and the Chart Update Interval is 100 ms, then there is going to be skipping of chart updates as needed based on the study calculation time.

Study Function Calling and Threads

Study calculations, the calling of a study function, the updating of a chart, and the drawing of the chart, all occurs on a single thread and this is the main thread of Sierra Chart.

Only one study function can run and be called at the same time. Each study is calculated one at a time and according to a [calculation order](#) determined by Sierra Chart.

Browsing of Custom Studies in DLL

When a user selects **Analysis >> Studies >> Add Custom Study**, all of the found study functions that begin with **scsf_** in the DLL file are called with [sc.SetDefaults](#) set to true/1. This is so the actual names of them can be discovered and listed to the user in the Add Custom Study

window.

Array Types

The following are descriptions of the different types of arrays used in the Advanced Custom Study Interface and Language for study and main price graph Subgraphs.

When getting one of these arrays with the various [ACSIL Functions](#) for getting an array from a study or a chart, a reference to the array is always made. Never in any case, is a copy made of the array.

- **SCFloatArray** : This is an array of 4 byte float variables. This is the type of array used with the [sc.BaseData\[\]](#) and [sc.Subgraph\[\].Data](#) arrays. This type of array is a reference to an array which is internally allocated and maintained within Sierra Chart. It does not contain a copy of the data. Only a reference to the data. The contents of this type of array can be modified by the study function.
- **SCFloatArrayRef**: This type is a reference to a **SCFloatArray** array. So it effectively just directly references another array of float variables.
- **SCDateTimeArray**: This is an array of [SCDateTime](#) variables. This is the type of array used with the [sc.BaseDateTimeln\[\]](#) array. This type of array is a reference to an array which is internally allocated and maintained within Sierra Chart. It does not contain a copy of the data. Only a reference to the data. The contents of this type of array can be modified by the study function.
- **SCDateTimeArrayRef**: This type is a reference to a **SCDateTimeArray** array. So it effectively just directly references another array of SCDateTime variables.

Array Indexing in Trading DOM Windows

A [Trading DOM](#) window is opened through **File >> Find Symbol >> Open Trading DOM**. It is a chart that consists of 1 chart bar which is not visible.

However, if it contains studies, then it contains more than one bar and the number of bars it contains is going to be based upon the **Chart >> Chart Settings**.

The bar indexing and bar spacing in a Trading DOM window works just like any other chart. Although the bars are just not visible. So therefore, all of the information on this page applies to a Trading DOM window as well.

When Arrays are Cleared

All of the main price graph and study arrays in a chart are cleared when the chart is reloaded. For example this happens when using **Chart >> Reload and Recalculate**.

The study arrays are cleared when making changes to studies through **Analysis >> Studies**.

*Last modified Monday, 17th July, 2023.