

ACSIL Interface Members - Variables and Arrays

Related Documentation

- [ACSIL Interface Members - Introduction](#)
 - **ACSIL Interface Members - Variables and Arrays**
 - [ACSIL Interface Members - sc.Input Array](#)
 - [ACSIL Interface Members - sc.Subgraph Array](#)
 - [ACSIL Interface Members - Functions](#)
-

On This Page

- [sc.ACSVersion](#)
- [sc.ActiveToolIndex](#)
- [sc.ActiveToolYPosition](#)
- [sc.ActiveToolYValue](#)
- [sc.AlertConditionEnabled](#)
- [sc.AlertConditionFlags](#)
- [sc.AlertOnlyOncePerBar](#)
- [sc.AllocateAndNameRenkoChartBarArrays](#)
- [sc.ArraySize](#)
- [sc.Ask](#)
- [sc.AskSize](#)
- [sc.AutoLoop](#)
- [sc.AutoScalePaddingPercentage](#)
- [sc.BaseDataEndDateTime\[\]](#)
- [sc.BaseDataIn\[\]\[\] / sc.BaseData\[\]\[\]](#)
- [sc.BaseDateTimeIn\[\]](#)
- [sc.BasedOnGraphValueFormat](#)
- [sc.BaseGraphAutoScalePaddingPercentage](#)
- [sc.BaseGraphConstantRangeScaleMode](#)
- [sc.BaseGraphGraphDrawType](#)
- [sc.BaseGraphHorizontalGridLineIncrement](#)
- [sc.BaseGraphScaleConstRange](#)
- [sc.BaseGraphScaleIncrement](#)
- [sc.BaseGraphScaleValueOffset](#)
- [sc.BaseGraphScaleRangeBottom](#)
- [sc.BaseGraphScaleRangeTop](#)

- [sc.BaseGraphScaleRangeType](#)
- [sc.BaseGraphValueFormat](#)
- [sc.Bid](#)
- [sc.BidSize](#)
- [sc.BlockChartDrawingSelection](#)
- [sc.CalculationPrecedence](#)
- [sc.CancelAllOrdersOnEntries](#)
- [sc.CancelAllOrdersOnReversals](#)
- [sc.CharacterEventCode](#)
- [sc.ChartBackgroundColor](#)
- [sc.ChartBarSpacing](#)
- [sc.ChartbookName](#)
- [sc.ChartDataEndDate](#)
- [sc.ChartDataStartDate](#)
- [sc.ChartDataType](#)
- [sc.ChartNumber](#)
- [sc.ChartRegion1BottomCoordinate](#)
- [sc.ChartRegion1LeftCoordinate](#)
- [sc.ChartRegion1RightCoordinate](#)
- [sc.ChartRegion1TopCoordinate](#)
- [sc.ChartTextFont](#)
- [sc.ChartTradeModeEnabled](#)
- [sc.ChartTradingOrderPrice](#)
- [sc.ChartWindowHandle](#)
- [sc.ChartWindowsActive](#)
- [sc.ConnectToExternalServiceServer](#)
- [sc.ConstantRangeScaleModeTicksFromCenterOrEdge](#)
- [sc.ContinuousFuturesContractLoading](#)
- [sc.ContinuousFuturesContractOption](#)
- [sc.ContractRolloverDate](#)
- [sc.CurrencyValuePerTick](#)
- [sc.CurrentSystemDateTime](#)
- [sc.CurrentDateTimeForReplay](#)
- [sc.CurrentSystemDateTimeMS](#)
- [sc.CurrentlySelectedDrawingTool](#)
- [sc.CurrentlySelectedDrawingToolState](#)
- [sc.CustomAffiliateCode](#)
- [sc.CustomChartTitleBarName](#)
- [sc.DailyHigh](#)
- [sc.DailyLow](#)
- [sc.DailyStatsResetTime](#)
- [sc.DailyVolume](#)
- [sc.DataFeedActivityCounter](#)
- [sc.DataFile](#) (This can be used to change the symbol of the chart)
- [sc.DataFilesFolder](#)
- [sc.DataStartIndex](#)

- [sc.DateTimeOfLastFileRecord](#)
- [sc.DateTimeOut\[\]](#)
- [sc.DaysToLoadInChart](#)
- [sc.DeltaVolumePerBar](#)
- [sc.DisconnectFromExternalServiceServer](#)
- [sc.DisplayAsMainPriceGraph](#)
- [sc.DisplayStudyInputValues](#)
- [sc.DisplayStudyName](#)
- [sc.DLLNameUserServiceLevel](#)
- [sc.DocumentationImageURL](#)
- [sc.DoNotRedrawChartAfterStudyReturns](#)
- [sc.DownloadingHistoricalData](#)
- [sc.DrawACSDrawingsAboveOtherDrawings](#)
- [sc.DrawBaseGraphOverStudies](#)
- [sc.DrawStudyUnderneathMainPriceGraph](#)
- [sc.EarliestUpdateSubgraphDataArrayIndex](#)
- [sc.EndTime1](#)
- [sc.EndTime2](#)
- [sc.ExternalServiceUsername](#)
- [sc.FileRecordIndexOfLastDataRecordInChart\(\)](#)
- [sc.FilterChartVolumeGreaterThanOrEqualTo](#)
- [sc.FilterChartVolumeLessThanOrEqualTo](#)
- [sc.FilterChartVolumeTradeCompletely](#)
- [sc.FlagFullRecalculate](#)
- [sc.FlagToReloadChartData](#)
- [sc.FreeDLL](#)
- [sc.GlobalDisplayStudySubgraphsNameAndValue](#)
- [sc.GlobalTradeSimulationIsOn](#)
- [sc.GraphDrawType](#)
- [sc.GraphName](#)
- [sc.GraphRegion](#)
- [sc.GraphShortName](#)
- [sc.GraphUsesChartColors](#)
- [sc.HideDLLAndFunctionNames](#)
- [sc.HideStudy](#)
- [sc.HistoricalHighPullbackVolumeAtPriceForBars](#)
- [sc.HistoricalLowPullbackVolumeAtPriceForBars](#)
- [sc.HistoricalPriceMultiplier](#)
- [sc.HTTPBinaryResponse](#)
- [sc.HTTPRequestID](#)
- [sc.IncludeInSpreadsheet](#)
- [sc.IncludeInStudySummary](#)
- [sc.CurrentIndex / sc.Index](#) (Automatic Looping)
- [sc.IndexOfFirstVisibleBar](#)
- [sc.IndexOfLastVisibleBar](#)
- [sc.IntradayDataStorageTimeUnit](#)

- [sc.IntradayChartRecordingState](#)
- [sc.IsAutoTradingEnabled](#)
- [sc.IsAutoTradingOptionEnabledForChart](#)
- [sc.IsChartbookBeingSaved](#)
- [sc.IsChartTradeModeOn](#)
- [sc.IsCustomChart](#)
- [sc.IsFullRecalculation](#)
- [sc.IsKeyPressed_Alt](#)
- [sc.IsKeyPressed_Control](#)
- [sc.IsKeyPressed_Shift](#)
- [sc.IsUserAllowedForSCDLLName](#)
- [sc.KeyboardKeyEventCode](#)
- [sc.LastCallToFunction](#)
- [sc.LastFullCalculationTimeInMicroseconds](#)
- [sc.LastSize](#)
- [sc.LastTradePrice](#)
- [sc.LatestDateTimeForLastBar](#)
- [sc.LoadChartDataByDateRange](#)
- [sc.MaintainAdditionalChartDataArrays](#)
- [sc.MaintainHistoricalMarketDepthData](#)
- [sc.MaintainReferenceToOtherChartsForPersistentVariables](#)
- [sc.MaintainTradeStatisticsAndTradesData](#)
- [sc.MaintainVolumeAtPriceData](#)
- [sc.NewBarAtSessionStart](#)
- [sc.NumberOfArrays](#)
- [sc.NumberOfForwardColumns](#)
- [sc.NumberOfTradesPerBar](#)
- [sc.NumFillSpaceBars](#)
- [sc.OutArraySize](#)
- [sc.p_VolumeLevelAtPriceForBars](#)
- [sc.PersistVars](#)
- [sc.PlaceACSCChartShortcutMenuItemsAtTopOfMenu](#)
- [sc.PointAndFigureBoxSizeInTicks](#)
- [sc.PointAndFigureReversalSizeNumBoxes](#)
- [sc.PointAndFigureXOGraphDrawTypeBoxSize](#)
- [sc.PointerHorzWindowCoord](#)
- [sc.PointerVertWindowCoord](#)
- [sc.PreserveFillSpace](#)
- [sc.PreviousClose](#)
- [sc.PriceChangesPerBar](#)
- [sc.ProcessIdentifier](#)
- [sc.ProtectStudy](#)
- [sc.PullbackVolumeAtPrice](#)
- [sc.RangeBarType](#)
- [sc.RangeBarValue](#)
- [sc.RealTimePriceMultiplier](#)

- [sc.ReceiveCharacterEvents](#)
- [sc.ReceiveKeyboardKeyEvents](#)
- [sc.ReceivePointerEvents](#)
- [sc.ReconnectToExternalServiceServer](#)
- [sc.RenkoNewBarWhenExceeded](#)
- [sc.RenkoReversalOpenOffsetInTicks](#)
- [sc.RenkoTicksPerBar](#)
- [sc.RenkoTrendOpenOffsetInTicks](#)
- [sc.ReplayStatus](#)
- [sc.ResetAlertOnNewBar](#)
- [sc.ResetAllScales](#)
- [sc.ReversalTicksPerBar](#)
- [sc.RightValuesScaleLeftCoordinate](#)
- [sc.RightValuesScaleRightCoordinate](#)
- [sc.RoundTurnCommission](#)
- [sc.SaveChartImageToFile](#)
- [sc.ScaleBorderColor](#)
- [sc.ScaleConstRange](#)
- [sc.ScaleIncrement](#)
- [sc.ScaleRangeBottom](#)
- [sc.ScaleRangeTop](#)
- [sc.ScaleRangeType](#)
- [sc.ScaleType](#)
- [sc.ScaleValueOffset](#)
- [sc.SCDataFeedSymbol](#)
- [sc.ScrollToDateTime](#)
- [sc.SecondsPerBar](#)
- [sc.SelectedAlertSound](#)
- [sc.SelectedTradeAccount](#)
- [sc.ServerConnectionState](#)
- [sc.ServiceCodeForSelectedDataTradingService](#)
- [sc.SetDefaults](#)
- [sc.StandardChartHeader](#)
- [sc.StartTime1](#)
- [sc.StartTime2](#)
- [sc.StartTimeOfDay](#)
- [sc.StorageBlock](#)
- [sc.StudyDescription](#)
- [sc.StudyGraphInstanceId](#)
- [sc.StudyRegionBottomCoordinate](#)
- [sc.StudyRegionLeftCoordinate](#)
- [sc.StudyRegionRightCoordinate](#)
- [sc.StudyRegionTopCoordinate](#)
- [sc.StudyVersion](#)
- [SC_SUBGRAPHS_AVAILABLE](#)
- [SupportAttachedOrdersForTrading](#)

- [sc.SupportKeyboardModifierStates](#)
- [sc.SupportTradingScaleIn](#)
- [sc.SupportTradingScaleOut](#)
- [sc.Symbol](#)
- [sc.SymbolData](#)
- [sc.TextInput](#)
- [sc.TextInputName](#)
- [sc.TickSize](#)
- [sc.TimeScaleAdjustment](#)
- [sc.TradeAndCurrentQuoteSymbol](#)
- [sc.TradeServiceAccountBalance](#)
- [sc.TradeServiceAvailableFundsForNewPositions](#)
- [sc.TradeWindowConfigFileName](#)
- [sc.TradeWindowOrderQuantity](#)
- [sc.TradingIsLocked](#)
- [sc.TransparencyLevel](#)
- [sc.UpdateAlways](#)
- [sc.UpdateStartIndex](#) (Manual Looping)
- [sc.UseGlobalChartColors](#)
- [sc.UseGUIAttachedOrderSetting](#)
- [sc.UseHighResolutionWindowRelativeCoordinatesForChartDrawings](#)
- [sc.UserName](#)
- [sc.UseSecondStartEndTimes](#)
- [sc.UsesMarketDepthData](#)
- [sc.ValueFormat](#)
- [sc.ValueIncrementPerBar](#)
- [sc.VersionNumber](#)
- [sc.VolumeAtPriceForBars](#)
- [sc.VolumeAtPriceForStudy](#)
- [sc.VolumeAtPriceMultiplier](#)
- [sc.VolumePerBar](#)
- [sc.VolumeValueFormat](#)

Variables and Arrays

sc.ACSVersion

Type: Read-only integer variable.

The **sc.ACSVersion** variable is set to the version number in the SierraChart.h header file that the custom study was compiled with. This version number corresponds to a particular Sierra Chart version number.

For the study to function in Sierra Chart it requires a Sierra Chart version equal to this version or higher.

sc.ActiveToolIndex

Type: Read-only integer variable.

sc.ActiveToolIndex is the array index corresponding to the second index used with the arrays **sc.BaseData[][]** and **sc.Subgraph[][]**, that the current [Drawing Tool](#) is pointing to on the chart that the study function is applied to.

This variable will not be updated when the **Pointer** tool is selected or when the **Chart Values** tool is selected, and not active.

This variable is always guaranteed to be up-to-date when **sc.ReceivePointerEvents** is set to a value which causes pointer events to be received by the study function, and the **Pointer**, **Chart Values** or **Hand** tool is selected, whether they are active or not.

When you use this variable it may be a good idea to set **sc.UpdateAlways** to 1, so that your study function is continuously called so it can be aware of the new tool position more often.

Alternatively you may want to use the [Advanced Custom Study Interaction With Menus, Control Bars, Pointer Events](#) functionality.

sc.ActiveToolIndex can also be at an index value which is in the fill space on the right side of the chart after the last bar.

This index can be converted into a pixel coordinate with the [sc.BarIndexToXPixelCoordinate](#) function.

Example

```
// Get the Open value of the bar that the tool is over
float OpenValueAtTool = sc.BaseData[SC_OPEN][sc.ActiveToolIndex];

//Get the Date-Time of the bar that the current tool is over.
SCDateTime BarDateTime = sc.BaseDateTimeIn[sc.ActiveToolIndex];
```

sc.ActiveToolYPosition

Type: Read-only integer variable.

sc.ActiveToolYPosition is the Y-axis coordinate in pixels of the current drawing tool's position over the chart your study function is applied to. This variable will not be updated when the Pointer tool is selected or when the Chart Values tool is selected, however not active.

When you use this variable it may be a good idea to set **sc.UpdateAlways** to 1, so that your study function is continuously called so it can be aware of the new tool position more often.

sc.ActiveToolYValue

Type: Read-only float variable.

The `sc.ActiveToolYValue` variable is the Y-axis value in the chart region that the mouse pointer is at when using one of the chart tools including the Chart Values tool. This is not the pixel position. It is the actual chart region value derived from the displayed graphs.

Example

```
float Value = sc.ActiveToolYValue;
```

sc.AlertConditionEnabled

Type: Read/Write integer variable.

A value of 0 is used to disable the Alert Condition Formula for the study. A value of 1 is used to enable the Alert Condition Formula for the study.

sc.AlertConditionFlags

Type: Read-only integer variable.

sc.AlertConditionFlags will be a nonzero value when the Simple Alert Condition Formula on the the study is TRUE

Example

```
int AlertState = sc.AlertConditionFlags;
```

sc.AlertOnlyOncePerBar

Type: Read/Write integer variable.

sc.AlertOnlyOncePerBar can be set to 1 or 0 (TRUE/FALSE) to cause an Alert to be given only once per bar in the chart. This variable works with the [sc.SetAlert\(\)](#) function.

The **sc.AlertOnlyOncePerBar** variable can also be changed through the [Alert Options](#) on the Study Settings window.

sc.AllocateAndNameRenkoChartBarArrays

Type: Integer variable.

The **sc.AllocateAndNameRenkoChartBarArrays** variable only applies when **sc.UsesCustomChartBarFunction** is set to 1 and the study is creating custom chart bars.

When **sc.AllocateAndNameRenkoChartBarArrays** is set to 1, then the [sc.BaseData\[SC_RENKO_OPEN\]\[i\]](#) and the [sc.BaseData\[SC_RENKO_CLOSE\]\[i\]](#) arrays have array elements added to them to match the standard arrays like `sc.BaseData[SC_OPEN][i]`, so

that the elements of them which correspond with chart bars can then be set.

These arrays will also be named "Renko Open" and "Renko Close" respectively.

sc.ArraySize

Type: Read-only integer variable.

sc.ArraySize is set to the number of array elements (equivalent to chart bars), that are in the arrays of the chart the study instance is applied to.

sc.ArraySize applies to the [sc.BaseDateTimeIn\[\]](#), [sc.BaseDataIn\[\]\[\]](#), and [sc.Subgraph\[\]\[\]](#) arrays.

If you set [sc.IsCustomChart](#) to 1 (TRUE), **sc.ArraySize** does not apply to the [sc.Subgraph\[\]\[\]](#) arrays. Refer to [sc.OutArraySize](#), for more information.

Setting [sc.IsCustomChart](#) is uncommon, so in most cases this is not applicable.

Example

```
// Copy all the Close elements from the BaseDataIn array
// to the Data array of the first Subgraph.
// This example assumes that sc.AutoLoop = 0 (manual looping).

for (int BarIndex = sc.UpdateStartIndex; BarIndex < sc.ArraySize; ++BarIndex)
{
    // SC_LAST is for the Close values in OHLC bars
    sc.Subgraph[0].Data[BarIndex] = sc.BaseDataIn[SC_LAST][BarIndex];
}
```

sc.Ask

Type: Read-only float variable.

sc.Ask is the current Ask price value for the symbol if Bid and Ask data is available for the symbol.

If the chart is not replaying, then this will only be set and up-to-date when Sierra Chart is connected to the data feed.

During a replay, this value is set. For additional information, refer to [Trade Simulation Accuracy and Bid/Ask Prices During Replays](#).

sc.AskSize

Type: Read-only integer variable.

sc.AskSize is the current Ask size/quantity value for the symbol if Bid and Ask data is available for the symbol.

If the chart is not replaying, then this will only be set and up-to-date when Sierra Chart is

connected to the data feed.

During a replay, this value is set. For additional information, refer to [Trade Simulation Accuracy and Bid/Ask Prices During Replays](#).

sc.AutoLoop

Type: Read/Write integer variable.

Initial value: 0 (FALSE)

When **sc.AutoLoop** is set to 1 (TRUE), then **sc.BaseData** and **sc.Subgraph[].Data[]** array element looping is automatically performed.

Otherwise, array element looping is manual and you will need to create your own internal loop to iterate through these arrays.

It is preferred that you set **sc.AutoLoop** to 1 (TRUE) unless you do not require it. For a complete discussion on this, refer to [Automatic Looping/Iterating](#).

sc.AutoLoop must only be set within the **sc.SetDefaults** code block at the top of the study function.

Example

```
if (sc.SetDefaults)
{
    sc.AutoLoop = 1; // Enable auto-looping
}
```

sc.AutoScalePaddingPercentage

Type: Read/Write float variable.

Initial value: .02

sc.AutoScalePaddingPercentage is a percentage value within the range of -1 to 1 where positive values mean more vertical scale padding, and negative values mean inverse vertical scale padding.

A value of 0.4 means that padding takes up 40% of the chart, and the graph uses the rest of the 60%.

This is the value that gets changed when you use the [Interactive Scale Range](#) feature when you left click and drag the scale on the right side of the chart.

Example

```
sc.AutoScalePaddingPercentage = 0.5f;
```

sc.BaseDataEndTime[]

Type: Read-only [SCDateTime](#) Array.

The **sc.BaseDataEndTime[]** array contains the actual ending date time of a chart bar at the specified array index. The values in this array are only going to be most accurate when the [Intraday Data Storage Time Unit](#) is set to **1 Tick** or **1 Second**.

For information about array indexing and array sizes, refer to [Array Indexing and Sizes](#).

For this array to be maintained and filled in, it is necessary to set [sc.MaintainAdditionalChartDataArrays](#) to 1 in the [sc.SetDefaults](#) code block.

The Date-Time at each element for a chart bar in this array, contains the last trade Date-Time within that chart bar. It is not necessarily at the last second or millisecond for the chart bar since trading does not necessarily end exactly at that time for a chart bar.

Example

```
// Get the End DateTime at the current index.  
SCDateTime BarEndTime = sc.BaseDataEndTime[sc.Index];
```

sc.BaseDataIn[][] / sc.BaseData[][]

Type: Read-only array of float arrays.

sc.BaseDataIn[][] / sc.BaseData[][] is an array of float arrays that contain the data for the main graph in the chart.

The first use of square brackets on this object will get the specified array of the main graph on the chart. For example: **sc.BaseDataIn[SC_OPEN]** will get the array of opening prices for each bar on the chart. The second use of square brackets will get an element inside the array that you received with the first set of square brackets.

For example: **sc.BaseDataIn[SC_OPEN][sc.Index]** will get the opening price for the bar at the array index your study function should do processing at when using Auto-Looping (the standard method of looping). The main graph is not necessarily the underlying data in the chart. For example, if you are using the **Point and Figure** study, then **sc.BaseData[][]** will contain the Point and Figure bar data.

For complete information about indexing and array sizes for the **sc.BaseData[][]** arrays, refer to the [Array Indexing and Sizes](#) section.

sc.BaseData[] and **sc.BaseDataIn[]** are both the same. They just have different names referring to the same array of arrays.

sc.BaseData[] / **sc.BaseDataIn[]** is meant to be read only, but there is no compiler enforcement of that implemented to avoid potential issues with defining these arrays as constant. A custom study can modify both of these arrays. But it should never do so.

The following lists all of the supported **sc.BaseData[]** / **sc.BaseDataIn[]** arrays and the corresponding constants that can be used. The descriptions for each are provided. These arrays provide access to the main price graph. The direct referencing arrays are also given, such as **sc.Open[]**.

- **sc.BaseData[SC_OPEN]** or **sc.Open[]**: The array of opening prices for each bar.
- **sc.BaseData[SC_HIGH]** or **sc.High[]**: The array of high prices for each bar.
- **sc.BaseData[SC_LOW]** or **sc.Low[]**: The array of low prices for each bar.
- **sc.BaseData[SC_LAST]** or **sc.Close[]**: The array of closing/last prices for each bar.
- **sc.BaseData[SC_VOLUME]** or **sc.Volume[]**: The array of trade volumes for each bar.
- **sc.BaseData[SC_NUM_TRADES]** or **sc.NumberOfTrades[]**: The array of the number of trades for each bar for Intraday charts.
- **sc.BaseData[SC_OPEN_INTEREST]** or **sc.OpenInterest[]**: The array of the open interest data for each bar for Historical Daily or higher timeframe charts. This is not valid for Intraday charts. It will return the number of trades for each bar for Intraday charts.
- **sc.BaseData[SC_OHLC]** or **sc.OHLCAvg[]**: The array of the average prices of the open, high, low, and close prices for each bar.
- **sc.BaseData[SC_HLC]** or **sc.HLCAvg[]**: The array of the average prices of the high, low, and close prices for each bar.
- **sc.BaseData[SC_HL]** or **sc.HLAvg[]**: The array of the average prices of the high and low prices for each bar.
- **sc.BaseData[SC_BIDVOL]** or **sc.BidVolume[]**: The array of Bid Volumes for each bar. This represents the volumes of the trades that occurred at the Bid price or lower. A trade that occurs between the Bid or Ask and is considered a downtick, will have the volume of that trade added to Bid Volume.
- **sc.BaseData[SC_ASKVOL]** or **sc.AskVolume[]**: The array of Ask Volumes for each bar. This represents the volumes of the trades that occurred at the Ask price or higher. A trade that occurs between the Bid or Ask and is considered an uptick, will have the volume of the trade added to the Ask Volume.
- **sc.BaseData[SC_UPVOL]** or **sc.UpTickVolume**: This array contains the total volume of trades for the bar where the trades occurred at a higher price than the trade before or the symbol traded at the same price as before and previously it traded higher.

For this to work properly, the **Intraday Data Storage Time Unit** setting in **Global Settings >> Data/Trade Service Settings** needs to be **1 Tick**. If it is not, then when the chart is reloaded, the historical up volume can change compared to real-time updating. In the case of **Number of Trades, Volume, Range, Reversal, Renko, Delta Volume** chart bars, the data in this array may

not be correct if **Chart >>Chart Settings >> Split Data Records** is enabled, unless there is 1 Tick data being used and the chart bars are based on **Number of Trades** or a **Range**.

If you are using this array in your study function, you must set

sc.MaintainAdditionalChartDataArrays to 1 in the sc.SetDefaults code block.

- **sc.BaseData[SC_DOWNVOL] or sc.DownTickVolume:** This array contains the total volume of trades for the bar where the trades occurred at a lower price than the trade before or the symbol traded at the same price as before and previously it traded down.

For this to work properly, the **Intraday Data Storage Time Unit** setting in

Global Settings >> Data/Trade Service Settings needs to be **1 Tick**. If it is

not, then when the chart is reloaded, the historical down volume can change

compared to real-time updating. In the case of **Number of Trades, Volume,**

Range, Reversal, Renko, Delta Volume chart bars, the data in this array may

not be correct if **Chart >>Chart Settings >> Split Data Records** is enabled,

unless there is 1 Tick data being used and the chart bars are based on **Number**

of Trades or a **Range**.

If you are using this array in your study function, you must set

sc.MaintainAdditionalChartDataArrays to 1 in the sc.SetDefaults code block.

- **sc.BaseData[SC_BIDNT] or sc.NumberOfBidTrades:** The array containing the total number of trades at the bid price or lower. For this to work properly, the **Intraday Data Storage Time Unit** setting in

Global Settings >> Data/Trade Service Settings needs to be **1 Tick**. This will also not work on historical data that does not have bid volume. In the case of

Tick, Volume, and Range charts, the data in this array may not be correct if

Chart >>Chart Settings >> Split Data Records is enabled, unless you are using tick data and the chart is based on Ticks or a Range.

If you are using this array in your study function, you must set

sc.MaintainAdditionalChartDataArrays to 1 in the sc.SetDefaults code block.

- **sc.BaseData[SC_ASKNT] or sc.NumberOfAskTrades:** The array containing the total number of trades at the ask price or higher. For this to work properly, the **Intraday Data Storage Time Unit** setting in

Global Settings >> Data/Trade Service Settings needs to be 1 Tick. This will

also not work on historical data that does not have ask volume. In the case of

Tick, Volume, and Range charts, the data in this array may not be correct if

Chart >>Chart Settings >> Split Data Records is enabled, unless you are using tick data and the chart is based on Ticks or a Range.

If you are using this array in your study function, you must set

sc.MaintainAdditionalChartDataArrays to 1 in the sc.SetDefaults code block.

- **sc.BaseData[SC_ASKBID_DIFF_HIGH]:** The array containing the maximum difference between the Ask volume and the Bid volume for the bar at the specified Index. This is calculated at every tick during the creation of the bar.

For the data in this array to be most accurate, the **Intraday Data Storage Time**

Unit setting in **Global Settings >> Data/Trade Service Settings** needs to be 1

Tick. The data in this array will not be available for historical data that does not have Ask Volume and Bid Volume.

For the chart to maintain the data in this array, you need to set **sc.MaintainAdditionalChartDataArrays** to TRUE in the **sc.SetDefaults** code block in your study function.

- **sc.BaseData[SC_ASKBID_DIFF_LOW]**: The array containing the minimum difference between the Ask volume and the Bid volume for the bar at the specified Index. This is calculated at every tick during the creation of the bar. For the data in this array to be most accurate, the **Intraday Data Storage Time Unit** setting in **Global Settings >> Data/Trade Service Settings** needs to be 1 Tick. The data in this array will not be available for historical data that does not have Ask volume and Bid Volume.

For the chart to maintain the data in this array, you need to set **sc.MaintainAdditionalChartDataArrays** to TRUE in the **sc.SetDefaults** code block in your study function.

- **sc.BaseData[SC_ASKBID_NUM_TRADES_DIFF_HIGH]**: The array containing the maximum difference between the number of trades at the Ask price or higher and the number of trades at the Bid price or lower, for the bar at the specified Index. This is calculated at every tick during the creation of the bar. For the data in this array to be accurate, the **Intraday Data Storage Time Unit** setting in **Global Settings >> Data/Trade Service Settings** must be 1 Tick. The data in this array will not be available for historical data that does not have Ask and Bid volume.

If you are using this array in your study function, you must set **sc.MaintainAdditionalChartDataArrays** to 1 in the **sc.SetDefaults** code block.

- **sc.BaseData[SC_ASKBID_NUM_TRADES_DIFF_LOW]**: The array containing the minimum difference between the number of trades at the Ask price or higher and the number of trades at the Bid price or lower, for the bar at the specified Index. This is calculated at every tick during the creation of the bar. For the data in this array to be accurate, the **Intraday Data Storage Time Unit** setting in **Global Settings >> Data/Trade Service Settings** must be 1 Tick. The data in this array will not be available for historical data that does not have Ask and Bid volume.

If you are using this array in your study function, you must set **sc.MaintainAdditionalChartDataArrays** to 1 in the **sc.SetDefaults** code block.

- **sc.BaseData[SC_UPDOWN_VOL_DIFF_HIGH]**: The array containing the maximum difference between the total volume of trades for the bar where the trades occurred at a higher price than the trade before and the total volume of trades for the bar where the trades occurred at a lower price than the trade before. Also take note of the additional details explained in **sc.BaseData[SC_UPVOL/SC_DOWNVOL]**.

If you are using this array in your study function, you must set **sc.MaintainAdditionalChartDataArrays** to 1 in the **sc.SetDefaults** code block.

- **sc.BaseData[SC_UPDOWN_VOL_DIFF_LOW]**: The array containing the minimum difference between the total volume of trades for the bar where the trades occurred at a higher price than the trade before and the total volume of trades for the bar where the trades occurred at a lower price than the trade before. Also take note of the additional details explained in

sc.BaseData[SC_UPVOL/SC_DOWNVOL].

If you are using this array in your study function, you must set

sc.MaintainAdditionalChartDataArrays to 1 in the **sc.SetDefaults** code block.

- **sc.BaseData[SC_RENKO_OPEN]:** In the case of a Renko chart set through **Chart >> Chart Settings**, this array provides the Renko Open price. This array is used to draw the Renko bar body.
- **sc.BaseData[SC_RENKO_CLOSE]:** In the case of a Renko chart set through **Chart >> Chart Settings**, this array provides the Renko Close price. This array is used to draw the Renko bar body.
- **sc.BaseData[SC_BID_PRICE]:** This array contains the bid prices at the time of the last trade for each bar. By default, the Bid and Ask prices are only recorded when there is a trade.

When using this array in a study function, it is necessary to set

[sc.MaintainAdditionalChartDataArrays](#) to 1 in the [sc.SetDefaults](#) code block.

For there to be Bid prices in this array, Sierra Chart must be set to a [Tick by tick Data Configuration](#), the Intraday data file for the chart must contain tick by tick data, and contain a Bid and Ask prices. This is not supported with all Data and Trading services.

- **sc.BaseData[SC_ASK_PRICE]:** This array contains the ask prices at the time of the last trade for each bar. By default, the Bid and Ask prices are only recorded when there is a trade.

When using this array in a study function, it is necessary to set

[sc.MaintainAdditionalChartDataArrays](#) to 1 in the [sc.SetDefaults](#) code block.

For there to be Ask prices in this array, Sierra Chart must be set to a [Tick by tick Data Configuration](#), the Intraday data file for the chart must contain tick by tick data, and contain a Bid and Ask prices. This is not supported with all Data and Trading services.

- **sc.BaseData[SC_ASK_BID_VOL_DIFF_MOST_RECENT_CHANGE]:** This array contains a value indicating whether the **SC_ASKBID_DIFF_HIGH** or the **SC_ASKBID_DIFF_LOW** array was most recently changed.

It will be set to 1 if the **SC_ASKBID_DIFF_HIGH** array at the corresponding element was most recently changed. It will be set to -1 if the

SC_ASKBID_DIFF_LOW array at the corresponding element was most recently changed. When using this array in a study function, it is necessary to set

[sc.MaintainAdditionalChartDataArrays](#) to 1 in the [sc.SetDefaults](#) code block.

- **sc.BaseData[PF_DIRECTION_ARRAY]:** This array is only used with [Point and Figure chart bars](#). Otherwise, the array is empty.

At a particular chart bar index it will have a value of 1 if the Point and Figure chart bar is an Up bar. At a particular chart bar index it will have a value of -1 if the Point and Figure chart bar is an Down bar.

The index value returned by [sc.Input\(\).GetInputDataIndex\(\)](#), corresponds to the array index constants given above. You can use this Input to select any of these array index constants to use with `sc.BaseData[]`.

sc.BaseData is short-hand for **sc.BaseDataIn**.

Example

```
// Get the volume at the current index.  
// This requires sc.AutoLoop = 1;  
float Volume = sc.BaseData[SC_VOLUME][sc.Index];  
  
// Copy the Last price from the current index to sc.Subgraph 0  
sc.Subgraph[0][sc.Index]=sc.BaseData[SC_LAST][sc.Index];
```

sc.BaseData References

A useful technique to make it easier to work with a single array for main price graph is to use a reference to the array. A reference to one of these arrays would be declared as **SCFloatArrayRef**. **SCFloatArrayRef** is a reference to the type of the arrays that are used in `sc.BaseData[]`. Below is an example of using a reference to one of the arrays in the chart.

Example

```
// Make a reference to the array of High prices called Highs  
SCFloatArrayRef Highs = sc.BaseData[SC_HIGH];  
  
// Get the high price at the current index  
// This is the same as sc.BaseData[SC_HIGH][sc.Index]  
float HighValue = Highs[sc.Index];
```

sc.BaseDateTimeIn[]

Type: Read-only **SCDateTime** array.

sc.BaseDateTimeIn[] is an array of the Date-Time for each bar in the chart. Each element of the array is a [SCDateTime](#).

The word **In** at the end of this array means input, which signifies that this array is input data for your study.

The Date-Time for a chart bar is the starting time of that bar. To get the ending Date-Time of the chart bar, use [sc.BaseDataEndTime](#).

For information about array indexing and array sizes, refer to [Array Indexing and Sizes](#).

The [Time Zone](#) of the Date-Times in the array is the same time zone as the chart.

Example

example

```
// Get the DateTime at the current index.  
SCDateTime BarDateTime = sc.BaseDateTimeIn[sc.Index];
```

Since this is a [SCDateTime](#) array, you can use the **DateAt()** and **TimeAt()** member functions to get just the date or just the time at a single bar.

Example

```
// Get the date  
int BarDate = sc.BaseDateTimeIn.DateAt(sc.Index);  
  
// Get the time  
int BarTime = sc.BaseDateTimeIn.TimeAt(sc.Index);
```

sc.BasedOnGraphValueFormat

Type: Read-only integer variable.

This variable uses the same values as **sc.ValueFormat** and is set to the Value Format for the graph set with the **Based On** setting on an instance the study which has been applied to the chart.

sc.BaseGraphAutoScalePaddingPercentage

Type: Read/Write float variable.

The **sc.BaseGraphAutoScalePaddingPercentage** variable is identical to [sc.AutoScalePaddingPercentage](#) except it is for the main price graph in the chart and not the study instance itself.

sc.BaseGraphConstantRangeScaleMode

Type: Read/Write integer variable.

The **sc.BaseGraphConstantRangeScaleMode** variable is the constant range scale mode for the main price graph in the chart.

It can be any of the following constant values:

- `CONST_RANGE_MODE_MANUAL = 0`
- `CONST_RANGE_MODE_AUTO_CENTER_LAST_BAR = 1` (Equivalent to **Keep The Last Bar Within View**)
- `CONST_RANGE_MODE_AUTO_CENTER_LAST_PRICE = 2`
- `CONST_RANGE_MODE_AUTO_KEEP_LAST_BAR_WITHIN_TICKS_FROM_EDGE = 3`
- `CONST_RANGE_MODE_AUTO_CENTER_WHEN_BAR_BEYOND_TICKS_FROM_CENTE`

= 4

For more information, refer to the [Chart Scale](#) page.

sc.BaseGraphGraphDrawType

Type: Read/Write integer variable.

sc.BaseGraphGraphDrawType is set to the Graph Draw Type for the base graph in the chart. This is also known as the main price graph. This is a read/write value and can be changed.

Supported Values:

- GDT_CUSTOM
- GDT_OHLCBAR
- GDT_CANDLESTICK
- GDT_CANDLESTICK_BODY_ONLY
- GDT_LINEONCLOSE
- GDT_MOUNTAIN
- GDT_HLCBAR
- GDT_LINEONOPEN
- GDT_LINEONHLAVG
- GDT_STAIRSTEPONCLOSE
- GDT_HLBAR
- GDT_KAGI
- GDT_POINT_AND_FIGURE_BARS
- GDT_POINT_AND_FIGURE_XO
- GDT_BID_ASK_BAR
- GDT_PRICE_VOLUME
- GDT_CANDLE_PRICE_VOLUME_BAR
- GDT_BLANK
- GDT_NUMBERS_BARS
- GDT_NUMERIC_INFORMATION
- GDT_RENKO_BRICK
- GDT_RENKO_BRICK_WITH_WICKS
- GDT_CANDLESTICK_HOLLOW
- GDT_MARKET_DEPTH
- GDT_VOLUME_LEVEL_TRADES
- GDT_CANDLE_PRICE_VOLUME_BAR_HOLLOW

sc.BaseGraphHorizontalGridLineIncrement

Type: Read/Write float variable.

The **sc.BaseGraphHorizontalGridLineIncrement** variable is the increment between the horizontal grid lines for the main price graph in the chart, if the chart is set to display the horizontal grid lines. Zero means that the setting is automatic.

For additional information, refer to [Horizontal Grid Line Increment](#) on the Chart Scale page.

sc.BaseGraphScaleConstRange

Type: Read/write float variable.

The **sc.BaseGraphScaleConstRange** variable gets and sets the grayscale range for the chart base graph when it is using a Scale Type of **Constant Range**. For additional information, refer to the [Chart Scale and Scale Adjusting](#) documentation.

When changing the symbol of the chart, this constant range value gets reset to a default value based upon the **Tick Size** of the chart.

sc.BaseGraphScaleIncrement

Type: Read/write float variable.

This sets the Scale Increment for the main price graph. When it is set to a nonzero value, it will change the Scale Increment for the main price graph.

Example

```
sc.BaseGraphScaleIncrement= 0;
```

sc.BaseGraphScaleValueOffset

Type: Read/Write 32-bit float value.

The **sc.BaseGraphScaleValueOffset** variable is the percentage value, where 1% is .01, for the offset of the chart scale from the center. This applies to the main price graph for the study and not for the study itself.

This can be programmatically set and is also interactively set by the user through the [Interactive Scale Move](#) functionality.

Also refer to [sc.ScaleValueOffset](#).

sc.BaseGraphScaleRangeBottom

Type: Read/Write float variable.

The **sc.BaseGraphScaleRangeBottom** variable is the bottom/minimum value of a User Defined scale range for the main price graph in the chart.

In the user interface this setting is in

Chart >> Chart Settings >> Scale >> Scale Range >> User-Defined >> Bottom of Range.

This can be modified by the custom study.

To get and set the bottom of the User Defined scale range for the study itself, use [sc.ScaleRangeBottom](#).

Example

```
float ScaleBottom = sc.BaseGraphScaleRangeBottom;
```

sc.BaseGraphScaleRangeTop

Type: Read/Write float variable.

The **sc.BaseGraphScaleRangeTop** variable is the top/maximum value of a User Defined scale range for the main price graph in the chart.

In the user interface this setting is in

Chart >> Chart Settings >> Scale >> Scale Range >> User-Defined >> Top of Range.

This can be modified by the custom study.

To get and set the top of the User Defined scale range for the study itself, use [sc.ScaleRangeTop](#).

Example

```
float ScaleTop = sc.BaseGraphScaleRangeTop;
```

sc.BaseGraphScaleRangeType

Type: Read/Write integer variable.

Initial value: **SCALE_AUTO**

The **sc.BaseGraphScaleRangeType** member allows you to determine and set the vertical scale range type for the main price graph (base graph) in the chart. It can be any of the following values:

- SCALE_AUTO
- SCALE_USERDEFINED
- SCALE_INDEPENDENT
- SCALE_SAMEASREGION
- SCALE_CONSTRANGE
- SCALE_CONSTRANGECENTER

When using **SCALE_USERDEFINED**, it is necessary to set [sc.ScaleRangeTop](#) and [sc.ScaleRangeBottom](#).

Example

```
sc.BaseGraphScaleRangeType = SCALE_AUTO;
```

sc.BaseGraphValueFormat

Type: Read/Write integer variable.

The **sc.BaseGraphValueFormat** variable is an Integer indicating the **Price Display Format/Value Format** for the main graph in the chart.

This is set through **Chart >> Chart Settings**. It is most useful to use with the `sc.FormatGraphValue()` function.

Example

```
SCString CurrentHigh = sc.FormatGraphValue(sc.BaseData[SC_HIGH][CurrentVisibleIndex], sc.BaseGrap
```

sc.Bid

Type: Read-only float variable.

sc.Bid is the current Bid price value for the symbol if Bid and Ask data is available for the symbol.

If the chart is not replaying, then this will only be set and up-to-date when Sierra Chart is connected to the data feed.

During a replay, this value is set. For additional information, refer to [Trade Simulation Accuracy and Bid/Ask Prices During Replays](#).

sc.BidSize

Type: Read-only integer variable.

sc.BidSize is the current Bid size/quantity value for the symbol if Bid and Ask data is available for the symbol.

If the chart is not replaying, then this will only be set and up-to-date when Sierra Chart is connected to the data feed.

During a replay, this value is set. For additional information, refer to [Trade Simulation Accuracy and Bid/Ask Prices During Replays](#).

sc.BlockChartDrawingSelection

Type: Read/Write integer variable.

Refer to [sc.BlockChartDrawingSelection](#).

sc.CalculationPrecedence

Type: Read/Write integer variable.

Initial value: **STD_PREC_LEVEL**

sc.CalculationPrecedence is a variable that can be set to 3 possible values.

The default value is **STD_PREC_LEVEL** (standard precedence level), and will mean your study gets calculated relative to other studies on the same chart that your custom study is applied to, based on its position in the **Analysis >> Studies >> Studies to Graph** list for the chart.

A value of **LOW_PREC_LEVEL** (low precedence level) will mean your study gets calculated after all other studies with a standard precedence level.

A value of **VERY_LOW_PREC_LEVEL** (very low precedence level) will mean your study gets calculated after all other studies with standard and low precedence levels. You will want to use a very low precedence level if your study depends on other studies that have low precedence level, such as Study Moving Average or a study based on other study.

The reason why you would want to set this variable to a lower precedence level, is when you are using a function such as [sc.GetStudyArray\(\)](#).

It is not necessary to set a low precedence when you are internally calculating study formulas, like when using [sc.SimpleMovAvg\(\)](#).

For further information about study calculation precedence, refer to [Study Calculation Precedence And Related Issues](#).

Example

```
sc.CalculationPrecedence = STD_PREC_LEVEL;
```

sc.CancelAllOrdersOnEntries

Type: Read/Write integer variable.

For the documentation for **sc.CancelAllOrdersOnEntries**, refer to [CancelAllOrdersOnEntries](#) on the Automated Trading Management page.

sc.CancelAllOrdersOnReversals

Type: Read/Write integer variable.

For the documentation for **sc.CancelAllOrdersOnReversals**, refer to [CancelAllOrdersOnReversals](#) on the Automated Trading Management page.

sc.CharacterEventCode

Type: Read integer variable.

The **sc.CharacterEventCode** variable is set to the [ASCII](#) value of the corresponding character pressed on the keyboard when the study has requested these events by setting [sc.ReceiveCharacterEvents](#).

The study function will be called when a character key is pressed and the chart containing the study has the focus. Otherwise, this variable will not be set.

sc.ChartBackgroundColor

Type: Read/Write RGB integer color variable.

sc.ChartBackgroundColor sets the background color of the chart. This does not affect the global chart background color setting, only the chart specific setting. When setting this variable, it is also necessary to set **sc.UseGlobalChartColors = 0** outside of the **sc.SetDefaults** code block, otherwise the variable is ignored.

Example

```
sc.UseGlobalChartColors = 0;
sc.ChartBackgroundColor = RGB(123,123,123);
```

sc.ChartBarSpacing

Type: Read/Write integer variable.

The **sc.ChartBarSpacing** variable is the spacing between the chart bars in pixels. This is for the chart your custom study is applied to.

The bar spacing can be changed by the custom study.

Example

```
int ChartBarSpacing = sc.ChartBarSpacing;
```

sc.ChartbookName

Type: Function returning [SCString](#).

This member has been changed to a function effective with version 2152.

The **sc.ChartbookName** function which returns a text string which contains the name of the Chartbook that contains the chart the study instance is applied to.

sc.ChartDataEndDate

Type: Read/Write integer variable.

When **sc.LoadChartDataByDateRange** is set to a nonzero value, then the [sc.ChartDataEndDate](#) variable specifies the last date to load into the chart.

This variable corresponds to **Chart >> Chart Settings >> Use Date Range >> To**.

If this is set to 0, then the last date available in the chart data file is loaded into the chart. For further information, refer to [Use Date Range >> To](#).

Changes to this variable do not go into effect until the study function returns. The chart will then be reloaded.

sc.ChartDataStartDate

Type: Read/Write integer variable.

When **sc.LoadChartDataByDateRange** is set to a nonzero value, then the [sc.ChartDataStartDate](#) variable specifies the first date to load into the chart.

This variable corresponds to **Chart >> Chart Settings >> Use Date Range >> From**.

If this is set to 0, then the earliest date available in the chart data file is loaded into the chart. For further information, refer to [Use Date Range >> From](#).

Changes to this variable do not go into effect until the study function returns. The chart will then be reloaded.

sc.ChartDataType

Type: Read/write integer variable.

sc.ChartDataType is set to the data type of the underlying chart. This can be either Intraday data or Daily data.

Example

```
if (sc.ChartDataType == DAILY_DATA)
{
    // The chart is a Historical Daily chart
}
else if (sc.ChartDataType == INTRADAY_DATA)
{
    // The chart is an Intraday chart
}
```

sc.ChartNumber

Type: Read-only integer variable.

Every chart contained within a Chartbook has a unique number.

sc.ChartNumber is set to the identifying number of the chart that the study is applied to. This is the same number that is shown on the top [Region Data Line](#) of the chart and on the chart window title bar.

If the identifying number of the chart calling the study function is #1, then the value of **sc.ChartNumber** will be 1.

sc.ChartRegion1BottomCoordinate

Type: Read-only integer variable.

The **sc.ChartRegion1BottomCoordinate** variable contains the Y-axis pixel coordinate that represents the bottom side of the rectangle that makes up Chart Region 1. For more information on chart regions, refer to [Chart Window and Regions](#).

The top left pixel coordinate of the chart is at 0, 0. These coordinates increase moving towards the bottom and right.

sc.ChartRegion1LeftCoordinate

Type: Read-only integer variable.

The **sc.ChartRegion1LeftCoordinate** variable contains the X-axis pixel coordinate that represents the left side of the rectangle that makes up Chart Region 1. For more information on chart regions, refer to [Chart Window and Regions](#).

The top left pixel coordinate of the chart is at 0, 0. These coordinates increase moving towards the bottom and right.

sc.ChartRegion1RightCoordinate

Type: Read-only integer variable.

The **sc.ChartRegion1RightCoordinate** variable contains the X-axis pixel coordinate that represents the right side of the rectangle that makes up Chart Region 1. For more information on chart regions, refer to [Chart Window and Regions](#).

The top left pixel coordinate of the chart is at 0, 0. These coordinates increase moving towards the bottom and right.

sc.ChartRegion1TopCoordinate

Type: Read-only integer variable.

The **scChartRegion1TopCoordinate** variable contains the Y-axis pixel coordinate that represents the top side of the rectangle that makes up Chart Region 1. For more information on chart regions, refer to [Chart Window and Regions](#).

The top left pixel coordinate of the chart is at 0, 0. These coordinates increase moving towards the bottom and right.

sc.ChartTextFont

Type: Function returning [SCString](#).

This member has been changed to a function effective with version 2152.

The **sc.ChartTextFont** function returns a text string which contains the font name used by the chart. This is useful if you want to create a font that matches the chart text.

sc.ChartTradeModeEnabled

Type: Read-only integer variable.

The **sc.ChartTradeModeEnabled** variable indicates whether the [Trade >> Chart Trade Mode On](#) option is enabled or not.

When the **Chart Trade Mode On** is enabled, this variable has a value of **1**. Otherwise, it is **0**.

sc.ChartTradingOrderPrice

Type: Read-only double precision floating-point variable.

When in Chart Trade Mode, the **sc.ChartTradingOrderPrice** variable indicates the price level that the mouse pointer is at on the chart. This applies to the chart that the study function is applied to.

sc.ChartWindowHandle

Type: Read Only integer variable.

sc.ChartWindowHandle is the Windows API handle (HWND) for the chart window that the study is applied to. This is useful when making Windows API function calls that require a window handle. This is for advanced programming only.

The window handle is provided even when the study is first calculated after the Chartbook is opened that the study instance is contained within.

sc.ChartWindowsActive

Type: Read-only integer variable.

The **sc.ChartWindowsActive** variable is set to 1 when the chart that contains the study instance, is the active chart window within Sierra Chart. This means either that it has the focus, or it is considered the active chart window within Sierra Chart if no Sierra Chart window currently has the focus.

sc.ConnectToExternalServiceServer

Type: Write integer variable.

Set the **sc.ConnectToExternalServiceServer** variable to TRUE to perform the **File >> Connect to Data Feed** command from within your custom study. This is not performed immediately at the time you set this to TRUE. It is only done when your study function returns.

Also see [sc.DisconnectFromExternalServiceServer](#) and [sc.ReconnectToExternalServiceServer](#).

Example

```
sc.ConnectToExternalServiceServer = TRUE;
```

sc.ConstantRangeScaleModeTicksFromCenterOrEdge

Type: Read/Write Integer variable.

The **sc.ConstantRangeScaleModeTicksFromCenterOrEdge** variable is set to the **Ticks from Center** setting for the **Constant Range** [Scale Range](#).

This variable applies only to the main price graph scale and not to the study scale.

The study can change this variable.

sc.ContinuousFuturesContractLoading

Type: Read-only integer variable.

The **sc.ContinuousFuturesContractLoading** variable is set to 1 when a chart is in the process of loading the historical futures contract data when the **Chart >> Chart Settings >> Symbol >> Continuous Contract** option is enabled. Otherwise, if the option is not enabled or the chart is not loading the historical futures contract data, then this will be 0. This flag is useful to avoid certain processing in your study function during the loading process.

sc.ContinuousFuturesContractOption

Type: Read/Write integer variable.

The **sc.ContinuousFuturesContractOption** variable is equivalent to the [Continuous Contract](#) setting in Chart Settings.

This variable can be changed to update this Chart Setting.

The possible values are listed below.

- **CFCO_NONE** = 0
- **CFCO_DATE_RULE_ROLLOVER** = 1
- **CFCO_VOLUME_BASED_ROLLOVER** = 2

- **CFCO_DATE_RULE_ROLLOVER_BACK_ADJUSTED** = 3
- **CFCO_VOLUME_BASED_ROLLOVER_BACK_ADJUSTED** = 4
- **CFCO_FORWARD_CURVE_CURRENT_DAY** = 5
- **CFCO_ROLLOVER_EACH_YEAR_SAME_MONTH** = 6

sc.ContractRolloverDate

Type: Read Only [SCDateTime](#) variable.

The **sc.ContractRolloverDate** variable is the rollover date associated with the futures symbol for chart the study instance is applied to.

For an example to use this variable, refer to the `scsf_RolloverDateDisplay` function in the `/ACS_Source/studies2.cpp` file in the Sierra Chart installation folder.

sc.CurrencyValuePerTick

Type: Read/Write float variable.

sc.CurrencyValuePerTick is a variable that is set to the currency value per tick of the chart the study instance is applied to.

This is the same as the [Currency Value per Tick](#) setting for the chart.

sc.CurrentSystemDateTime

Type: Read Only [SCDateTime](#) variable.

sc.CurrentSystemDateTime is the current Date and Time from your local computer's date and time, provided as a [SCDateTime](#) variable.

The Time Zone setting in **Global Settings >> Data/Trade Service Settings** is applied to this `SCDateTime` variable.

This member is not affected by the chart the study is applied to, when being replayed. In other words, it will always indicate the current system Date-Time.

This variable does not contain milliseconds. The resolution is to the second.

If you want to perform a particular action in a custom study at a particular time and want to make sure the study function is called at that time, then you should use [sc.UpdateAlways = 1](#) in the **sc.SetDefaults** block of the study function.

Example

```
SCDateTime DateTime = sc.CurrentSystemDateTime;
```

sc.CurrentDateTimeForReplay

Type: Read Only [SCDateTime](#) variable.

The `sc.CurrentDateTimeForReplay` variable is only set when a chart is replaying. It contains the current Date-Time in relation to the replaying chart and is relative to the starting Date-Time of the replaying chart. It is based upon the elapsed amount of time since the replay was started and the replay speed.

If you want to perform a particular action in a custom study at a particular time and want to make sure the study function is called at that time, then you should use [sc.UpdateAlways = 1](#) in the **sc.SetDefaults** block of the study function.

However, when there is an accelerated replay, a study function will not necessarily be called at the interval specified by the [Chart Update Interval](#) relative to the replay times. For example, if the chart is set to update every 1000 milliseconds, during an accelerated replay it will not necessarily be called every theoretical second when using **sc.UpdateAlways = 1**.

During an accelerated replay, **sc.CurrentDateTimeForReplay** can be ahead of the expected time. For complete information, refer to the function [sc.GetCurrentDateTime](#). **sc.CurrentDateTimeForReplay** is used with that function during a replay.

Example

```
SCDateTime CurrentDateTime;  
if (sc.IsReplayRunning())  
    CurrentDateTime = sc.CurrentDateTimeForReplay;  
else  
    CurrentDateTime = sc.CurrentSystemDateTime;
```

sc.CurrentSystemDateTimeMS

Type: Read-only [SCDateTimeMS](#) variable.

sc.CurrentSystemDateTimeMS is the current Date and Time from your local computer's date and time, provided as a [SCDateTimeMS](#) variable.

SCDateTimeMS is derived from [SCDateTime](#). It contains all of the same functionality, except that it is specialized for milliseconds. Comparisons done using this type are down to the nearest millisecond unlike the nearest second for [SCDateTime](#).

The Time Zone setting in **Global Settings >> Data/Trade Service Settings** is applied to this [SCDateTimeMS](#) variable.

This member is not affected by the chart the study is applied to, when being replayed. In other words, it will always indicate the current system Date-Time.

This variable contains the Date-Time of the computer system down to the millisecond.

If you want to perform a particular action in a custom study at a particular time and want to make sure the study function is called at that time, then you should use [sc.UpdateAlways = 1](#) in the

sc.SetDefaults block of the study function.

Example

```
SCDateTimeMS DateTimeWithMilliseconds = sc.CurrentSystemDateTimeMS;
```

sc.CurrentlySelectedDrawingTool

Type: Read-only integer variable.

sc.CurrentlySelectedDrawingTool

Example

```
int CurrentlySelectedDrawingTool
```

sc.CurrentlySelectedDrawingToolState

Type: Read-only integer variable.

sc.CurrentlySelectedDrawingToolState set to a constant indicating the state of the active drawing tool. This will be 0 when the tool is not active and > 0 when the tool is active. For example, when actively drawing a line with the **Line** Tool, this will be set to 1.

Example

```
int ToolState = sc.CurrentlySelectedDrawingToolState;
```

sc.CustomAffiliateCode

Type: Function returning [SCString](#).

This member has been changed to a function effective with version 2152.

The **sc.CustomAffiliateCode** function returns a text string which contains the affiliate code associated with a particular Sierra Chart account.

This function has specialized uses and is not normally used.

sc.CustomChartTitleBarName

Type: Read/Write SCString variable.

The **sc.CustomChartTitleBarName** variable is for getting and setting the

Chart >> Chart Settings >> Display >> Title Bar Name setting.

sc.DailyHigh

Type: Read Only float variable.

sc.DailyHigh is the current daily high for the symbol of the chart.

sc.DailyHigh is the same Daily High value displayed in the **Window >> Current Quote Window**. This data comes from the Current Quote Data from the connected data feed.

sc.DailyHigh is only valid when connected to the data feed or during a chart replay.

sc.DailyLow

Type: Read-only float variable.

sc.DailyLow is the current daily low for the symbol of the chart.

sc.DailyLow is the same Daily Low value displayed in the **Window >> Current Quote Window**. This data comes from the Current Quote Data from the connected data feed.

sc.DailyLow is only valid when connected to the data feed or during a chart replay.

sc.DailyStatsResetTime

Type: Read-only [SCDateTime](#) variable.

This is no longer used.

sc.DailyVolume

Type: Read-only integer variable.

sc.DailyVolume is the current daily volume for the symbol.

sc.DailyVolume is the same Daily Volume value displayed in the **Window >> Current Quote Window**. This data comes from the Current Quote Data from the connect to data feed.

sc.DailyVolume is only valid when connected to the data feed.

sc.DailyVolume will be 0 during a chart replay.

sc.DataFeedActivityCounter

Type: Read-only integer variable.

The **sc.DataFeedActivityCounter** variable is the activity counter for the incoming data feed that Sierra Chart is currently connected to.

It is the same value shown on the [Status Bar](#) after **DF:**.

sc.DataFile

Type: Read/Write SCString variable.

sc.DataFile is a text string of the complete path and file name of the chart data file for the chart the study instance is applied to.

This text string can be changed to change the symbol of the chart.

When this is changed, the new data file will be loaded after the study function returns. So the data file will be changed before the next call into the study function which is called after the new data file is loaded.

Only the filename needs to be set. The path is ignored. The

Global Settings >> General Settings >> Data Files Folder setting is used for the path.

The file extension has to be set. Use **.dly** for Historical charts and **.scid** for Intraday charts.

Example

```
// Specify a new chart data file
sc.DataFile = "QQQQ.scid";
```

sc.DataFilesFolder

Type: Function returning [SCString](#).

This member has been changed to a function effective with version 2152.

The **sc.DataFilesFolder** function returns a text string which contains the complete path to the Data files folder used by Sierra Chart.

For further information, refer to [Data Files Folder](#).

sc.DataStartIndex

Type: Read/Write integer variable.

Initial value: 0

Set **sc.DataStartIndex** to the index of the first element at which subgraphs should start to be drawn. This is to prevent Sierra Chart from drawing elements at the beginning of the sc.Subgraph arrays that do not have enough prior data to be properly calculated. For example: If you have a 10-bar moving average, this value should be set to 9. It should be set to 9 instead of 10 because array index values always begin at 0.

Example

```
sc.DataStartIndex = 9; // Start drawing the subgraphs at element index 9 (the 10th bar)
```

sc.DateTimeOfLastFileRecord

Type: Read Only [SCDateTime](#) variable.

sc.DateTimeOfLastFileRecord is the Date-Time, as a [SCDateTime](#) variable, of the starting time of the last data record read from the chart data file which has been added to the chart.

sc.DateTimeOut[]

Type: Read/Write [SCDateTime](#) array.

sc.DateTimeOut[] is an array of the DateTimes for each of the bars you have created in a study which is set up as a custom chart. Each element is a [SCDateTime](#). This array is only used when [sc.IsCustomChart](#) is set to 1 (TRUE).

For additional information about indexing and array sizes, see [Array Indexing and Sizes](#).

Example

```
// Set the element at our CustomIndex in sc.DateTimeOut to match the current index in sc.BaseDateTimeIn  
sc.DateTimeOut[CustomIndex] = sc.BaseDateTimeIn[sc.Index];
```

sc.DaysToLoadInChart

Type: Read/Write integer variable.

The **sc.DaysToLoadInChart** variable is the number of days of data to load within the chart. It applies to both Intraday and Historical charts.

It is equivalent to **Chart >> Chart Settings >> Use Number of Days to Load >> Days to Load**.

Changes to this variable do not go into effect until the study function returns. At that time the chart will be reloaded and the new setting will go into effect.

If you are reducing the value of this variable, the chart needs to be reloaded to reduce the amount of data in the chart. There is not the capability to remove data in the chart without the chart needing to reload.

sc.DeltaVolumePerBar

Type: Read/Write integer variable.

This ACSIL structure member is considered out of date/deprecated. Instead use the [sc.GetBarPeriodParameters](#) and [sc.SetBarPeriodParameters](#) functions.

sc.DisconnectFromExternalServiceServer

Type: Write integer variable.

Set the **sc.DisconnectFromExternalServiceServer** variable to TRUE to perform the **File >> Disconnect** command from within your custom study. This is not performed immediately at the time you set this to TRUE. It is only done when your study function returns.

Also see [sc.ConnectToExternalServiceServer](#) and [sc.ReconnectToExternalServiceServer](#).

Example

```
sc.DisconnectFromExternalServiceServer = TRUE;
```

sc.DisplayAsMainPriceGraph

Type: Read/Write integer variable.

Initial value: 0 (FALSE)

If **sc.DisplayAsMainPriceGraph** is set to 1 (TRUE), then your study will become the main price graph for the chart. All other studies applied to the chart, will be based on it. You will also need to set [sc.GraphRegion](#) to 0, when setting this variable to 1 (TRUE). If it is set to 0 (FALSE), then your study is a standard study based on the existing main price graph.

Example

```
sc.DisplayAsMainPriceGraph = 1;
```

sc.DisplayStudyInputValues

Type: Read/Write integer variable.

sc.DisplayStudyInputValues is equivalent to the **Display Input Values** setting on the Study Settings window for the study. This variable can be set to either 1 to enable that option or to 0 to disable it.

Example

```
sc.DisplayStudyInputValues = 1;
```

sc.DisplayStudyName

Type: Read/Write integer variable.

sc.DisplayStudyName is equivalent to the **Display Study Name** setting on the Study Settings window for the study. This can be set to either 1 to enable that option or to 0 to disable it.

Example

```
sc.DisplayStudyName = 1;
```

sc.DLLNameUserServiceLevel

Type: Read-only 8-byte integer variable.

sc.DLLNameUserServiceLevel is used in conjunction with the [sc.IsUserAllowedForSCDLLName](#) variable.

This variable returns the Service Level setting for an authorized user for the custom studies DLL that the function is in and the user has been authorized to use.

This value is set through the **Custom DLL Studies Management** control panel accessed through your account on the Sierra Chart website.

This is a 64-bit integer variable which can be set to any value. You can set it to the numbers which represent, certain bits within the 64-bit variable being set.

Example

```
if (sc.DLLNameUserServiceLevel == 100)//Any number can be used.
{
    // Perform some action like executing the study
}
else
    return;// Do nothing and return
```

sc.DocumentationImageURL

Type: Read/Write SCString variable.

sc.DocumentationImageURL is a text string variable that can be set to an internet URL to an image providing an example of your study. For complete documentation refer to [ACSIL Study Documentation Interface Members](#).

Example

```
sc.DocumentationImageURL = "http://www.sierrachart.com/images/HomePageImages/HomePage_ChartW
```



sc.DoNotRedrawChartAfterStudyReturns

Type: Read/Write Integer variable.

The **sc.DoNotRedrawChartAfterStudyReturns** variable can be set to 1 or a nonzero value to prevent the chart from being redrawn after the study function returns.

Every time the studies on a chart are calculated either as part of a full recalculation or an update calculation, the chart is redrawn. When this variable is set to 1, it prevents the chart from being redrawn. Even if other studies have not set this variable to 1 during the studies calculation, the chart will still be prevented from being redrawn if one of the studies has set this to 1.

The purpose of this variable is to improve performance (reduce CPU usage) when handling certain events that the study is notified of when the study function is called. This includes key events indicated through variables like [sc.KeyboardKeyEventCode](#) and [Pointer Events](#). Some of these events should not cause the chart to be redrawn immediately but rather after some series of events as determined by the study function.

sc.DownloadingHistoricalData

Type: Read-only integer variable.

sc.DownloadingHistoricalData is considered out of date and has been replaced by the [sc.ChartsDownloadingHistoricalData](#) function.

sc.DrawACSDrawingsAboveOtherDrawings

Type: Read/Write integer variable.

When the **sc.DrawACSDrawingsAboveOtherDrawings** variable is set to a non-zero value, Chart Drawings added by an Advanced Custom Study for the chart will be drawn above other User Drawn drawings.

When this is set to **0**, the Chart Drawings added by an Advanced Custom Study for the chart are drawn in the same way with respect to other Chart Drawing priorities.

sc.DrawBaseGraphOverStudies

Type: Read/Write **Integer** variable.

The **sc.DrawBaseGraphOverStudies** variable is the same as the **Chart >> Chart Settings >> Display Main Chart Graph on Top of Studies** setting. When it is set to a nonzero value, then the main chart graph will be displayed on top of other studies on the chart. Otherwise, the studies will be on top of the main chart graph.

Example

```
sc.DrawBaseGraphOverStudies = 1;
```

sc.DrawStudyUnderneathMainPriceGraph

Type: Read/Write integer variable.

When the **sc.DrawStudyUnderneathMainPriceGraph** variable is set to 1 (TRUE) the study will be displayed underneath the main price graph in the chart. Assuming **sc.GraphRegion** is set to 0. Otherwise, this variable is not relevant. When **sc.DrawStudyUnderneathMainPriceGraph** is set to 0 (FALSE), then the study is displayed above the main price graph in the chart.

Example

```
sc.DrawStudyUnderneathMainPriceGraph = 1;
```

sc.EarliestUpdateSubgraphDataArrayIndex

Type: Read/Write Integer Variable

sc.EarliestUpdateSubgraphDataArrayIndex is an index variable which can optionally be set to the earliest **sc.Subgraph[].Data[]** array index being modified during a study update in your study function.

There is no need to set this variable if a study does not make changes to a **sc.Subgraph[].Data[]** array at an index earlier than [sc.UpdateStartIndex](#) for manual looping, or the earliest [sc.Index](#) set for a study during an update when using automatic looping.

sc.EarliestUpdateSubgraphDataArrayIndex is referenced by the **Study/Price Overlay** study, the **Color Bar Based on Alert Condition** study, the **Spreadsheet Studies**, and other studies which depend on other studies so that they are aware of the earliest index that changes have occurred at in a source study in order for them to properly process that data.

This variable should not be set when [sc.IsFullRecalculation](#) is TRUE/1.

sc.EndTime1

Type: Read/Write Integer Variable

sc.EndTime1 is the first end time for a day on the chart.

This is equal to the [Session Times >> End Time](#) setting for the chart.

This variable is a SCDatetime [TimeValue](#) in seconds.

Also refer to [sc.StartTime1](#).

sc.EndTime2

Type: Read/Write Integer Variable

sc.EndTime2 is the second end time for a day on the chart. This is only used if

[sc.UseSecondStartEndTimes](#) is set to 1 (TRUE).

This is equal to the [Session Times >> Evening End Time](#) setting for the chart.

This variable is a SCDatetime [TimeValue](#) in seconds.

Also refer to [sc.StartTime2](#).

sc.ExternalServiceUsername

Type: Read-only SCString variable.

The `sc.ExternalServiceUsername` is a text string containing the username for the currently selected external Data or Trading service in Sierra Chart.

sc.FileRecordIndexOfLastDataRecordInChart

Type: Read-only integer variable.

The `sc.FileRecordIndexOfLastDataRecordInChart` variable is the index of the last Intraday data record read from the Intraday data file or the file cache, used in the chart.

This is not necessarily the very last data record in the file. For example, if a replay is in progress, it can be earlier than the last record in the file or file cache.

In newer versions of Sierra Chart, due to file caching, this variable could refer to an index which has not yet been written to the Intraday data file.

sc.FilterChartVolumeGreaterThanOrEqualTo

Type: Read/Write float variable.

`sc.FilterChartVolumeGreaterThanOrEqualTo` is for setting and getting the Volume Filter value which excludes volume greater than or equal to the specified value.

This is the same **Volume Filter >=** value set through **Chart >> Chart Settings**.

sc.FilterChartVolumeLessThanOrEqualTo

Type: Read/Write float variable.

`sc.FilterChartVolumeLessThanOrEqualTo` is for setting and getting the Volume Filter value which excludes volume less than or equal to the specified value.

This is the same **Volume Filter <=** set through **Chart >> Chart Settings**.

sc.FilterChartVolumeTradeCompletely

Type: Read/Write integer variable.

The `sc.FilterChartVolumeTradeCompletely` variable corresponds to the [Filter Trade Completely](#) Chart Setting.

It is set to 1/TRUE when the setting is enabled. Otherwise, it is 0.

This variable can be changed by the study function and goes into effect after the study function returns.

sc.FlagFullRecalculate

Write only integer variable.

The **sc.FlagFullRecalculate** variable can be set to 1 or a nonzero value to cause a full recalculation of all studies on the chart to occur after returning from the study function.

A full recalculation means in the case of automatic looping that the study function will be called for each bar in the chart starting at **sc.Index** 0. In the case of manual looping, the study function will be called and **sc.UpdateStartIndex** will be set to 0.

It is not recommended to use this because it is highly inefficient. Generally the reason this is used is due to programming problems within the study function itself and this is not an appropriate solution in those cases.

sc.FlagToReloadChartData

Write only integer variable.

The **sc.FlagToReloadChartData** variable can be set to 1 or a nonzero value to cause a reload of the chart data from the data file after the study function returns. The reload does not happen immediately.

This reload is the same as selecting **Chart >> Reload and Recalculate** on the menu.

sc.FreeDLL

Type: Read/Write integer variable.

Initial value: 0 (FALSE)

sc.FreeDLL is no longer required as of version 1836 and higher. As of that version, setting this has no effect.

sc.GlobalDisplayStudySubgraphsNameAndValue

Type: Read/Write integer variable.

sc.GlobalDisplayStudySubgraphsNameAndValue is equivalent to the **Display Study Name, Subgraph Names and Subgraph Values - Global** setting on the Study Settings window for the study. This can be set to either 1 to enable that option or to 0 to disable it.

Example

```
sc.GlobalDisplayStudySubgraphsNameAndValue = 1;
```

sc.GlobalTradeSimulationIsOn

Type: Read-only integer variable.

sc.GlobalTradeSimulationIsOn is set to 1, when **Trade >> Trade Simulation Mode On** is enabled. Otherwise, it is set to zero.

Example

```
int TradeSimulationIsOn = sc.GlobalTradeSimulationIsOn;
```

sc.GraphDrawType

Type: Read/Write integer variable.

Initial value: GDT_CUSTOM

sc.GraphDrawType can be set to one of the values in the list below.

When you use **GDT_CUSTOM**, which is the default, then the study will be able to use the [sc.Subgraph\[\].DrawStyle](#) member to set the [Draw Style](#) for each Subgraph.

When you use a setting other than GDT_CUSTOM, then the Graph Draw Type will be a type of price bar which uses the data in the [sc.Subgraph\[0 through 4\].Data](#) arrays. Use the: [sc.Subgraph\[SC_OPEN\]\[\]](#) array for Open values, [sc.Subgraph\[SC_HIGH\]\[\]](#) array for High values, [sc.Subgraph\[SC_LOW\]\[\]](#) array for Low values, and the [sc.Subgraph\[SC_LAST\]\[\]](#) array for Close or Last values.

In the case of **GDT_CANDLE_VOLUME_BAR**, use the [sc.Subgraph\[SC_VOLUME\].Data\[\]](#) array to control the width of the candlestick.

If you set this to a value other than **GDT_CUSTOM** and you also set the [sc.DisplayAsMainPriceGraph](#) variable to 1 (TRUE), you should calculate the average values and fill in the [sc.Subgraph\[SC_VOLUME\]\[\]](#) and [sc.Subgraph\[SC_NUM_TRADES\]\[\]](#) arrays. To calculate the averages, make a call to [sc.CalculateOHLCAverages\(\)](#) near the end of the study function.

When you use a **sc.GraphDrawType** setting value other than **GDT_CUSTOM**, then the [sc.Subgraph\[\].DrawStyle](#) variable is automatically set for each of the relevant [sc.Subgraphs](#) needed by the [sc.GraphDrawType](#). You cannot change them.

Additionally, it is not possible when you are drawing a price bar type of graph ([sc.GraphDrawType](#) not equal to GDT_CUSTOM), to also use standard study lines or other Draw Styles using [sc.Subgraph\[4\]](#) and higher which are not used by the price bar graph draw types. In this case you

will need to use a separate study.

Colors when using `sc.GraphDrawType != GDT_CUSTOM`: When using a `sc.GraphDrawType` that is not equal to `GDT_CUSTOM`, the **PrimaryColor** and **SecondaryColor** members of `sc.Subgraph` set the color of the bars. For more information, refer to [Color Settings for Graph Draw Types](#). If you wish to color certain bars a certain type of color when using a price bar graph draw type, then you can use the `sc.Subgraph[0-4].DataColor[]` arrays.

Supported Values:

- `GDT_CUSTOM`
- `GDT_OHLCBAR`
- `GDT_CANDLESTICK`
- `GDT_CANDLESTICK_BODY_ONLY`
- `GDT_LINEONCLOSE`
- `GDT_MOUNTAIN`
- `GDT_HLCBAR`
- `GDT_LINEONOPEN`
- `GDT_LINEONHLAVG`
- `GDT_STAIRSTEPONCLOSE`
- `GDT_HLBAR`
- `GDT_KAGI`
- `GDT_POINT_AND_FIGURE_BARS`
- `GDT_POINT_AND_FIGURE_XO`
- `GDT_BID_ASK_BAR`
- `GDT_PRICE_VOLUME`
- `GDT_CANDLE_PRICE_VOLUME_BAR`
- `GDT_BLANK`
- `GDT_NUMBERS_BARS`
- `GDT_NUMERIC_INFORMATION` (For complete documentation, refer to [Numeric Information Table Graph Draw Type](#))
- `GDT_RENKO_BRICK`
- `GDT_RENKO_BRICK_WITH_WICKS`
- `GDT_CANDLESTICK_HOLLOW`
- `GDT_MARKET_DEPTH`
- `GDT_VOLUME_LEVEL_TRADES`
- `GDT_CANDLE_PRICE_VOLUME_BAR_HOLLOW`

In the case of when using `GDT_POINT_AND_FIGURE_XO`, if the Box Size is greater than the chart **Tick Size**, then it is necessary to set [sc.PointAndFigureXOGraphDrawTypeBoxSize](#) to the Box Size.

To set the Graph Draw Type for the main price graph, it is necessary to use [sc.BaseGraphGraphDrawType](#) instead.

Example

```
sc.GraphDrawType = GDT_OHLCBAR;
```

sc.GraphName

Type: Read/Write SCString variable.

sc.GraphName is the name of your study. This must be set when [sc.SetDefaults](#) is 1 (TRUE).

Example

```
sc.GraphName = "My Study";
```

sc.GraphRegion

Type: Read/Write integer variable.

Initial value: 1 or the next unused Chart Region.

sc.GraphRegion is the zero-based index of the Chart Region for the study graph to be displayed in. A value of 0 means Chart Region 1, which is where the main price graph is drawn. A value of 1 means Chart Region 2, which is directly under the main price graph.

Currently there are 8 Chart Regions available. Therefore, the maximum value can be 7.

It is not possible for a single study to draw in a Chart Region outside of what is specified with **sc.GraphRegion**. A separate study is required for each Chart Region.

If you want to force a particular Chart Region to be used and not allow it to be automatically selected by Sierra Chart when adding a new study instance to the chart, when you have set **sc.GraphRegion** to 1, then set **sc.GraphRegion** below the **sc.SetDefaults** code block in the study function, to the particular Graph Region you want to display the study in.

Example

```
sc.GraphRegion = 0; // Use the main price graph region
```

sc.GraphShortName

Type: Read/write SCString variable.

sc.GraphShortName is a text string that is set to the **Short Name** for the instance of the study function applied to the chart.

Example

```
SCString ShortName = sc.GraphShortName;
```

sc.GraphUsesChartColors

Type: Read/Write integer variable.

The **sc.GraphUsesChartColors** variable only applies when the **sc.GraphDrawType** is set to one of the price bar types like **GDT_OHLCBAR**.

When **sc.GraphUsesChartColors** is set to TRUE (1), this means that the colors of the drawn price bars will be set according to the colors specified for the chart itself and not the study **sc.Subgraph[]** colors. These chart colors can either be the global color settings, or the chart specific color settings.

When **sc.GraphUsesChartColors** is set to FALSE (0), this means that the colors of the drawn price bars will be set according to the colors specified by the study **sc.Subgraph[]** colors.

For additional details, refer to [Color Settings for Graph Draw Types](#).

Example

```
sc.GraphUsesChartColors = TRUE;
```

sc.HideDLLAndFunctionNames

Type: Read/Write integer variable.

When **sc.HideDLLAndFunctionNames** is set to 1/TRUE, then the DLL file name and study function name is not displayed on the [Study Settings](#) window. Otherwise, there is a text field which displays these names.

sc.HideStudy

Type: Read/Write integer variable.

Initial Value: 0.

sc.HideStudy can be set to 1 (TRUE) to prevent the study from being displayed on the chart. Or set it to 0 (FALSE), to display the study.

Example

```
sc.HideStudy = 1;
```

sc.HistoricalHighPullbackVolumeAtPriceForBars

Type: Read-only custom data array object of type **c_VAPContainer**.

The **sc.HistoricalHighPullbackVolumeAtPriceForBars** is like the [sc.VolumeAtPriceForBars](#) array. The data that it contains is the Volume at Price data for each bar in the chart, since the last price pullback from the bar High.

The data in this array will only be maintained when you have enabled **Historical Pullback Data** in **Chart >> Chart Settings**. For complete details, refer to the [Historical Pullback Data](#) section in the Chart Settings documentation.

sc.HistoricalLowPullbackVolumeAtPriceForBars

Type: Read-only custom data array object of type **c_VAPContainer**.

The **sc.HistoricalHighPullbackVolumeAtPriceForBars** is like the [sc.VolumeAtPriceForBars](#) array. The data that it contains is the Volume at Price data for each bar in the chart, since the last price pullback from the bar Low.

The data in this array will only be maintained when you have enabled **Historical Pullback Data** in **Chart >> Chart Settings**. For complete details, refer to the [Historical Pullback Data](#) section in the Chart Settings documentation.

sc.HistoricalPriceMultiplier

Type: Read-only float variable.

The **sc.HistoricalPriceMultiplier** variable contains the value of the [Historical Price Multiplier](#) Chart Setting for the chart.

sc.HTTPBinaryResponse

Type: Read-only character array (SCConstCharArray)

The **sc.HTTPBinaryResponse** character string array contains the resulting response from a request.

sc.HTTPRequestID

Type: Read-only integer variable.

The **sc.HTTPRequestID** variable is set to the request identifier for a completed [HTTP request](#).

sc.IncludeInSpreadsheet

Type: Read/Write integer variable.

The **sc.IncludeInSpreadsheet** variable controls the [Include In Spreadsheet](#) Study Setting for the study.

Setting this variable to 1 means the option is enabled. Setting this variable to 0 means the option is disabled.

sc.IncludeInStudySummary

Type: Read/Write integer variable.

The **sc.IncludeInStudySummary** variable controls the [Include In Study Summary](#) Study Setting for the study.

Setting this variable to 1 means the option is enabled. Setting this variable to 0 means the option is disabled.

sc.CurrentIndex / sc.Index

Type: Read-only integer variable.

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 used with automatic looping and is equal to the elements in the [sc.BaseDataIn\[\]\[\]](#) arrays that need to be processed and/or the elements in the [sc.Subgraph\[\].Data\[\]](#) arrays that need to be filled in.

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.BaseDataIn\[\]\[\]](#) arrays to process, assuming your custom chart function uses the [sc.BaseDataIn\[\]\[\]](#) arrays.

For complete information, refer to [Automatic Looping/Iterating](#).

The range of **sc.Index** is from 0 through and including **sc.ArraySize** -1.

During chart updating when a new bar is added to the chart, **sc.Index** will always start the prior value of **sc.ArraySize** -1 and not the current value.

sc.IndexOfFirstVisibleBar

Type: Read-only integer variable.

sc.IndexOfFirstVisibleBar is the index into the **sc.BaseData[][]** arrays for the first visible bar that is drawn on the chart. When the chart is first loaded, this index will be 0.

If the study function relies on this member and it needs to be aware of changes such as when the user scrolls the chart, then you will need to set [sc.UpdateAlways](#) to 1 (TRUE).

In the case where the study is replacing the main price graph because the study function has set **sc.DisplayAsMainPriceGraph** = 1 or it has set **sc.IsCustomChart** = 1, then as explained

previously, this will be set to the index into the **sc.BaseData[][]** arrays, and not to the index into the resulting output array.

Example

```
// Get the Close value of the first bar that is drawn  
float Value = sc.BaseData[SC_LAST][sc.IndexOfFirstVisibleBar];
```

sc.IndexOfLastVisibleBar

Type: Read-only integer variable.

sc.IndexOfLastVisibleBar is the index into the **sc.BaseData[][]** arrays for the last visible bar that is drawn on the chart. When the chart is first loaded, this index will be 0. This value is different than **sc.ArraySize**.

If the study function relies on this member and it needs to be aware of changes such as when the user scrolls the chart, then you will need to set [sc.UpdateAlways](#) to 1 (TRUE).

In the case where the study is replacing the main price graph because it has set **sc.DisplayAsMainPriceGraph** = 1 or it has set **sc.IsCustomChart** = 1, then as explained previously, this will be set to the index into the **sc.BaseData[][]** arrays, and not to the index into the resulting output array.

Example

```
// Get the Close value of the last bar that is drawn  
float Value = sc.BaseData[SC_LAST][sc.IndexOfLastVisibleBar];
```

sc.IntradayDataStorageTimeUnit

Type: Read-only integer variable.

sc.IntradayDataStorageTimeUnit is the Intraday Data Storage Time Unit setting in **Global Settings >> Data/Trade Service Settings**. For more information, refer to [Data/Trade Service Settings](#).

Example

```
int IDSTU = sc.IntradayDataStorageTimeUnit;
```

sc.IntradayChartRecordingState

Type: Read-only integer variable.

The **sc.IntradayChartRecordingState** variable indicates the current state related to the writing of data to the Intraday data file for the symbol of the chart the study instance is applied to. This is useful to prevent certain processing from occurring in the study function based on this state.

It can be one of the following values:

- IDFRS_NOT_RECORDING_DATA
- IDFRS_HISTORICAL_DATA_DOWNLOAD_PENDING
- IDFRS_DOWNLOADING_HISTORICAL_DATA
- IDFRS_RECEIVING_REALTIME_DATA
- IDFRS_FINISHED_RECEIVING_DATA

sc.IsAutoTradingEnabled

Type: Read-only integer variable.

The **sc.IsAutoTradingEnabled** variable indicates the state of the

Trade >> Auto trading Enabled - Global menu command. A value of 1 means the option is enabled. A value of 0 means it is disabled.

Example

```
if(sc.IsAutoTradingEnabled)
{
}
```

sc.IsAutoTradingOptionEnabledForChart

Type: Read-only integer variable.

The **sc.IsAutoTradingOptionEnabledForChart** variable indicates the state of the

Trade >> Auto Trading Enabled - Chart menu command. A value of 1 means the option is enabled. A value of 0 means it is disabled.

sc.IsChartbookBeingSaved

Type: Read Only integer variable.

The **sc.IsChartbookBeingSaved** variable is set to **1** when the Chartbook that contains the study that uses this variable is being actively saved to the file. Otherwise, the variable is set to **0**.

The study functions are called when a Chartbook is being saved in order for a study to take some action during that time. This would be for more specialized purposes. It is not typically used.

sc.IsChartTradeModeOn

Type: Read-only integer variable.

The **sc.IsChartTradeModeOn** variable is set to **1** if the [Trade >> Chart Trade Mode On](#) is active. Otherwise it is set to **0**.

sc.IsCustomChart

Type: Read/Write integer variable.

Initial value: 0 (FALSE)

Set **sc.IsCustomChart** to 1 (TRUE) in the **sc.SetDefaults** code block to make the study work as a custom chart.

A custom chart is used when you need to create a bar chart or some other style chart that is of a different size, either smaller or greater, than the underlying data it is based on. In other words, either smaller or greater than the existing price bars in the chart.

When the study functions as a custom chart, the study needs to control the size of its **sc.Subgraph[].Data** arrays by using the [sc.ResizeArrays\(\)](#) and [sc.AddElements\(\)](#) functions. The study will also need to set the Date-Times of each chart bar in the [sc.DateTimeOut\[\]](#) array.

For a complete example, refer to the **scsf_CopyOfBaseGraph()** function in the **/ACS_Source/CustomChart.cpp** file in the Sierra Chart installation folder.

You may want to use a custom chart to create your own specialized Range bar chart or Point and Figure chart.

The **/ACS_Source/CustomChart.cpp** file in the folder where Sierra Chart is installed to, provides an example of a simple custom chart. The **scsf_PointAndFigureChart** function in the **/ACS_Source/studies8.cpp** file provides a more advanced example.

Example

```
sc.IsCustomChart = 1; // Set this study as a custom chart

sc.GraphRegion = 0; // Custom charts need to be set to use chart region 0
```

sc.IsFullRecalculation

Type: Read-only integer variable.

The **sc.IsFullRecalculation** variable will be set to 1 (TRUE) when the chart is performing a full recalculation of the studies.

For more information about a full recalculation, refer to the [When the Study Function is Called](#) section.

This flag is useful to not perform certain processing in the study function when there is a full recalculation for efficiency or to perform certain initializations in the study function when there is a full recalculation.

However, when performing certain initializations in the study function when there is a full recalculation when using [Automatic Looping](#), also check that **sc.Index** equals 0 so that the initialization is not performed at every chart bar during the full recalculation. This is essential.

Example

```
int& Variable1 = sc.GetPersistentInt(1);
if(sc.IsFullRecalculation)
{
    if (sc.Index == 0)//This line is needed for automatic looping
        Variable1 = 0;
}
```

sc.IsKeyPressed_Alt

Type: Read-only Integer variable.

The **sc.IsKeyPressed_Alt** variable is used to know when the Alt key has been pressed. The variable is set to 1 when the Alt key is depressed, otherwise it is set to 0.

For this variable to be set, it is necessary to set [sc.SupportKeyboardModifierStates](#) to a value of 1, otherwise the state of this keyboard modifier cannot be checked.

Whether a chart is active or not, it will still have this variable set if the key is pressed. To see if a chart is active, use [sc.ChartWindowIsActive](#).

sc.IsKeyPressed_Control

Type: Read-only Integer variable.

The **sc.IsKeyPressed_Control** variable is used to know when the Control key has been pressed. The variable is set to 1 when the Control key is depressed, otherwise it is set to 0.

For this variable to be set, it is necessary to set [sc.SupportKeyboardModifierStates](#) to a value of 1, otherwise the state of this keyboard modifier cannot be checked.

Whether a chart is active or not, it will still have this variable set if the key is pressed. To see if a chart is active, use [sc.ChartWindowIsActive](#).

sc.IsKeyPressed_Shift

Type: Read-only Integer variable.

The **sc.IsKeyPressed_Shift** variable is used to know when the Shift key has been pressed. The variable is set to 1 when the Shift key is depressed, otherwise it is set to 0.

For this variable to be set, it is necessary to set [sc.SupportKeyboardModifierStates](#) to a value of 1, otherwise the state of this keyboard modifier cannot be checked.

Whether a chart is active or not, it will still have this variable set if the key is pressed. To see if a

chart is active, use [sc.ChartWindowsIsActive](#).

sc.IsUserAllowedForSCDLLName

Type: Read-only integer variable.

sc.IsUserAllowedForSCDLLName is used to authorize access to a study. For more information refer to [Redistributing and Allowing Use Only by a Defined List of Users](#).

It is also possible to specify a Service-Level for an authorized user for a particular **SCDLLName** in the **Custom DLL Studies Management** control panel accessed through your account on the Sierra Chart website. To access this service level programmatically, refer to [sc.DLLNameUserServiceLevel](#).

sc.KeyboardKeyEventCode

Type: Read integer variable.

The **sc.KeyboardKeyEventCode** variable is set to the standard Windows virtual key code when there is a keyboard keypress event and the study has requested these events by setting [sc.ReceiveKeyboardKeyEvents](#).

Also refer to [sc.SupportKeyboardModifierStates](#) and [sc.CharacterEventCode](#).

sc.LastCallToFunction

Type: Read-only integer variable.

sc.LastCallToFunction is set to 1 (TRUE) when an Advanced Custom Study instance is in the process of being removed from the chart or the chart is being closed.

This variable is useful if you want to do something before the study is removed from the chart or when the chart is being closed.

Example

```
if (sc.LastCallToFunction)
{
    // This study is being removed from the chart or the chart is being closed
    // Insert cleanup code here
}
```

sc.LastFullCalculationTimeInMicroseconds

Type: Read-only integer variable.

The **sc.LastFullCalculationTimeInMicroseconds** variable indicates the calculation time in microseconds of the last time the study was [fully recalculated](#).

This variable does not indicate the time of an update calculation.

An example of a full recalculation is when study settings are modified through **Analysis >> Studies**. When the **OK** or **Apply** buttons are pressed, all studies will be fully recalculated.

sc.LastSize

Type: Read-only integer variable.

sc.LastSize is the volume of the last trade. This member will only be set when Sierra Chart is connected to the data feed and receiving current market data for the symbol of the chart, or during a chart replay.

This value for the chart is displayed in the **Window >> Current Quote Window**.

Example

```
int LastTradeSize = sc.LastSize;
```

sc.LastTradePrice

Type: Read-only float variable.

sc.LastTradePrice is the last trade price for the symbol of the chart the study is applied to.

This is the same value as **LastPrice** in **Window >> Current Quote Window**.

sc.LastTradePrice usually will be equal to **sc.BaseData[SC_LAST][sc.ArraySize -1]** except when the Session Times in Chart Settings are excluding the data from the current time. In this last case, it will be set to the actual last trade price.

This member will only be set when Sierra Chart is connected to the data feed and a nonzero last trade price in the current quote data is being provided by the data feed.

This member will be set during a chart replay.

sc.LatestDateTimeForLastBar

Type: Read Only [SCDateTime](#) variable.

The **sc.LatestDateTimeForLastBar** variable is set to the Date-Time, as a [SCDateTime](#) value, of the very latest trade from the data feed that is included in the last bar in the chart. The data feed has a resolution down to the second. Or, it will be set to the starting Date-Time of the very latest chart data file record which has been read into the chart, whichever is greater.

The Date-Time of the latest data file record read into the chart, is affected by the **Intraday Data Storage Time Unit**. If this is set to greater than 1 second, then the Date-Time of that data file record may not be set to the most recent second received in that record.

During a replay, **sc.LatestDateTimeForLastBar** will always be set to the Date-Time of the latest data file record read into the chart.

sc.LatestDateTimeForLastBar is adjusted to the Sierra Chart Time Zone setting.

Example

```
SCDateTime DateTime = sc.LatestDateTimeForLastBar;
```

sc.LoadChartDataByDateRange

Type: Read/Write integer variable.

When **sc.LoadChartDataByDateRange** is set to a nonzero value, then the [sc.ChartDataStartDate](#) and [sc.ChartDataEndDate](#) variables specify the date range to load into the chart.

This variable corresponds to **Chart >> Chart Settings >> Use Date Range**.

sc.MaintainAdditionalChartDataArrays

Type: Read/Write integer variable.

sc.MaintainAdditionalChartDataArrays needs to be set to 1 (TRUE) to flag that the chart needs to maintain the following **sc.BaseGraph[]** arrays. This should be set in the **sc.SetDefaults** code block at the beginning of the function. These are the constants for these arrays:

- SC_UPVOL
- SC_DOWNVOL
- SC_BIDNT
- SC_ASKNT
- SC_ASKBID_DIFF_HIGH
- SC_ASKBID_DIFF_LOW
- SC_ASKBID_NUM_TRADES_DIFF_HIGH
- SC_ASKBID_NUM_TRADES_DIFF_LOW

When **sc.MaintainAdditionalChartDataArrays** is set to 0 (FALSE), these arrays are not maintained by the chart.

sc.MaintainAdditionalChartDataArrays also causes the **sc.BaseDataEndTime[]** array to be filled in and maintained.

Example

sc.MaintainHistoricalMarketDepthData

Type: Read/Write integer variable.

When this variable is set to TRUE/1, then historical market depth data will be loaded into the chart the study instance is applied to. The chart will need to be reloaded once after this has been set if the historical market depth data is not already loaded in the chart. Real-time market depth data will also be stored when this variable is TRUE.

It is necessary to set this to TRUE/1 when using [c_ACSILDepthBars](#) in the study function.

The default for this variable is FALSE/0.

sc.MaintainReferenceToOtherChartsForPersistentVariables

Type: 16-bit Integer variable.

The default for **sc.MaintainReferenceToOtherChartsForPersistentVariables** is 1. This means that when using the [sc.GetPersistent*FromChartStudy](#) functions, when there is any update to the chart being referenced, then the chart that had a study that called the **sc.GetPersistent*FromChartStudy** function will be calculated so that the studies are aware of changes in the source chart. All of the studies will be calculated in this case.

When **sc.MaintainReferenceToOtherChartsForPersistentVariables** is set to 0, the above does not happen.

sc.MaintainTradeStatisticsAndTradesData

Type: Read/Write integer variable.

If your study function uses the ACSIL Trading functions, uses the **s_SCPositionData** structure, or uses the [sc.GetTradeStatisticsForSymbol\(\)](#) function, then

sc.MaintainTradeStatisticsAndTradesData needs to be set to TRUE in the study function in order to maintain the necessary data for this functionality to all work properly.

Example

```
if(sc.SetDefaults)
{
    sc.MaintainTradeStatisticsAndTradesData = TRUE;
}
```

sc.MaintainVolumeAtPriceData

Type: Read/Write integer variable.

Set **sc.MaintainVolumeAtPriceData** to 1 (TRUE), to have Sierra Chart maintain detailed volume at price data for the loaded bars in the chart your custom study is applied to.

Example

```
if (sc.SetDefaults)
{
    sc.MaintainVolumeAtPriceData = 1;
}
```

sc.NewBarAtSessionStart

Type: Read/Write integer variable.

The **sc.NewBarAtSessionStart** variable indicates the state of the **Chart >> Chart Settings >> Session Times >> New Bar at Session Start** setting. The variable will be 1 if this option is enabled or 0 if it is disabled.

Example

```
if (sc.NewBarAtSessionStart)
{
    //Do something when true
}
```

sc.NumberOfArrays

Type: Read-only integer variable.

Initial value: 1

sc.NumberOfArrays is the number of [sc.Subgraph\[\].Data\[\]](#) arrays that are allocated for the study to use. In other words this is the number of **sc.Subgraph[]** objects which have an allocated Data array starting at index 0. There are not going to be any skipped Subgraphs which would have an unallocated array. They will all be allocated up to the specified number starting 0.

sc.NumberOfArrays gets set automatically as the study function uses the [sc.Subgraph\[\].Data\[\]](#) arrays.

For example, if **sc.Subgraph[3].Data[]** is accessed, then **sc.NumberOfArrays** becomes set to 4. Although this value does not update until after the study function returns.

sc.NumberOfForwardColumns

Type: Read Only Integer variable.

The **sc.NumberOfForwardColumns** is set to the same value as the [Number of Forward Columns](#) setting in the **Chart >> Chart Settings** for the chart the study is applied to.

sc.TicksPerBar / sc.NumberOfTradesPerBar

Type: Read/Write integer variable.

This ACSIL structure member is considered out of date/deprecated. Instead use the [sc.GetBarPeriodParameters](#) and [sc.SetBarPeriodParameters](#) functions.

sc.NumFillSpaceBars

Type: Read/Write integer variable.

sc.NumFillSpaceBars is the number of bars in the fill space on the right side of the chart. For additional information, refer to [Right Side Fill Space](#).

Example

```
// Make sure there are at least 10 bars of fill space
if (sc.NumFillSpaceBars < 10)

    sc.NumFillSpaceBars = 10;
```

sc.OutArraySize

Type: Read-only integer variable.

sc.OutArraySize is the number of array elements (equivalent to chart bars) that are in the output arrays for the chart the study is applied to.

This includes the [sc.Subgraph\[\].Data\[\]](#) arrays, the [sc.Subgraph\[\].DataColor\[\]](#) arrays if used, and the [sc.DateTimeOut\[\]](#) array.

Normally **sc.OutArraySize** is the same as [sc.ArraySize](#). However, in the case when [sc.IsCustomChart](#) is used, the output arrays can have a different number of elements than the **sc.BaseData[][]** arrays.

sc.OutArraySize is updated when you use the [sc.ResizeArrays\(\)](#) and [sc.AddElements\(\)](#) functions.

Example

```
// Resize the output arrays to be half the current size of the output arrays
sc.ResizeArrays(sc.OutArraySize / 2);
```

sc.p_VolumeLevelAtPriceForBars

Type: Read-only custom data array object pointer of type **c_VAPContainer**.

When using the [Large Volume Trade Indicator](#) study, this causes volume data for trades with a volume over a particular volume threshold to be maintained for the price levels of those trades.

This data is contained within the **sc.p_VolumeLevelAtPriceForBars** member.

It is essential that the [Large Volume Trade Indicator](#) be added to the chart for the **sc.p_VolumeLevelAtPriceForBars** object to contain data.

sc.VolumeAtPriceForBars is a pointer to a c_VAPContainer. To access a function member of c_VAPContainer requires that you use the member of pointer operator ->. Refer to [Member access operators](#).

A c_VAPContainer container contains many member functions to access the data within. These functions are documented in the [sc.VolumeAtPriceForBars](#) section on this page.

Also refer to the /ACS_Source/VAPContainer.h header file for all of the available functions.

sc.PersistVars

sc.PersistVars is no longer supported. It is necessary to use the [GetPersistent*](#) functions instead.

Existing compiled code will continue to work which uses the old sc.PersistVars structure, but it will not recompile. It will need to be updated to use the new functions.

sc.PlaceACSCChartShortcutMenuItemsAtTopOfMenu

Refer to the [sc.PlaceACSCChartShortcutMenuItemsAtTopOfMenu](#) page for information on this variable, as it is part of the [Advanced Custom Study Interaction With Menus, Control Bar Buttons, Pointer Events](#) documentation.

sc.PointAndFigureBoxSizeInTicks

Type: Read Only Integer variable.

This ACSIL structure member is considered out of date/deprecated. Instead use the [sc.GetBarPeriodParameters](#) and [sc.SetBarPeriodParameters](#) functions.

sc.PointAndFigureReversalSizeNumBoxes

Type: Read Only Integer variable.

This ACSIL structure member is considered out of date/deprecated. Instead use the [sc.GetBarPeriodParameters](#) and [sc.SetBarPeriodParameters](#) functions.

sc.PointAndFigureXOGraphDrawTypeBoxSize

Type: Read/Write Float variable.

In the case of when [sc.GraphDrawType](#) is set to **GDT_POINT_AND_FIGURE_XO**, **sc.PointAndFigureXOGraphDrawTypeBoxSize** sets the Box Size for the X and O boxes.

sc.PointerHorzWindowCoord

Type: Read Only integer variable.

The **sc.PointerHorzWindowCoord** is the X coordinate of the pointer in the chart window coordinates.

The chart window coordinate system uses the upper left hand corner to define the (0,0) point and increases to the right and down.

sc.PointerVertWindowCoord

Type: Read Only integer variable.

The **sc.PointerVertWindowCoord** is the Y coordinate of the pointer in the chart window coordinates.

The chart window coordinate system uses the upper left hand corner to define the (0,0) point and increases to the right and down.

sc.PreserveFillSpace

Type: Read/Write integer variable.

This is the same as the **Lock Fill Space** command on the **Chart** menu.

Set **sc.PreserveFillSpace** to 1 (TRUE) to prevent the fill space from being filled in as new bars are added to the chart or when the chart is being scrolled. Set it to 0 (FALSE) to allow the fill space to be filled in.

sc.PreviousClose

Type: Read-only float variable.

The **sc.PreviousClose** variable contains the current Settlement Price obtained from the Current Quote data. For more information, refer to [Current Quote Window](#).

sc.PriceChangesPerBar

Type: Read/Write integer variable

This ACSIL structure member is considered out of date/deprecated. Instead use the [sc.GetBarPeriodParameters](#) and [sc.SetBarPeriodParameters](#) functions.

sc.ProcessIdentifier

Type: Read-only integer variable.

sc.ProcessIdentifier is the ID of Sierra Chart process. This variable is for advanced programming only.

sc.ProtectStudy

Type: Read/Write integer variable.

When **sc.ProtectStudy** is set to 1, it will prevent the data from the Subgraph arrays from being accessed by other ACSIL studies and also prevents the study from being outputted to Spreadsheets using any of the Spreadsheet Studies.

When this is set to 0, none of the above happens and the data from the study is accessible normally.

sc.PullbackVolumeAtPrice

Type: Read-only custom data array object of type **c_VAPContainer**.

The **sc.PullbackVolumeAtPrice** array has the same type of data that is contained within the [sc.VolumeAtPriceForBars](#) array. It only contains data for the last bar in the chart. It contains the volume data for each price level since the last price pullback from the last bar High or Low.

The data contained in this array is what is displayed in the Numbers Bars study in the Pullback column, and could be a high or low pullback based on the current bar state. This data is maintained when **sc.MaintainVolumeAtPriceData** is set to 1 (TRUE) in your custom study function.

When specifying the **BarIndex** parameter with the various VAP container functions, you must always use 0 for this parameter, since this array only contains data for a single bar.

sc.RangeBarType

Type: Read/Write integer variable.

This ACSIL structure member is considered out of date/deprecated. Instead use the [sc.GetBarPeriodParameters](#) and [sc.SetBarPeriodParameters](#) functions.

sc.RangeBarValue

Type: Read/Write float variable.

This ACSIL structure member is considered out of date/deprecated. Instead use the [sc.GetBarPeriodParameters](#) and [sc.SetBarPeriodParameters](#) functions.

sc.RealTimePriceMultiplier

Type: Read/Write float variable.

The **sc.RealTimePriceMultiplier** variable contains the value of the [Real-Time Price Multiplier](#) for the chart.

sc.ReceiveCharacterEvents

Type: Read/Write integer variable.

When the **sc.ReceiveCharacterEvents** variable is set to 1 or a nonzero value, then the custom

study function will be called immediately for any character key press events from the keyboard.

The actual ASCII character code can be determined with the [sc.CharacterEventCode](#) variable.

When the custom study function is called when a character key is pressed on the keyboard, this study function call is an efficient standard update calculation where `sc.Index` and `sc.UpdateStartIndex` be equal to `sc.ArraySize - 1`.

The study function will know the reason it is being called by checking that `sc.CharacterEventCode` is nonzero. If it is nonzero, the study function only has to handle the character event and does not need to do anything else. It can still do whatever it requires, but does not have to do any calculations because at the usual time a calculation needs to be performed when the chart is updated, the study function will be called again.

The flag variable [sc.DoNotRedrawChartAfterStudyReturns](#) can be set to specify to not redraw the chart window after a study function is called. This can be set when **sc.CharacterEventCode** is nonzero. This will make the process of handling these types of events very efficient.

sc.ReceiveKeyboardKeyEvents

Type: Read/Write integer variable.

When the **sc.ReceiveKeyboardKeyEvents** variable is set to 1 or a nonzero value, then the custom study function will be called immediately for any key press events from the keyboard. This also includes the letter keys as well.

The actual key code can be determined with the [sc.KeyboardKeyEventCode](#) variable. The actual value is a standard Windows virtual key code.

For a listing of the virtual key codes, refer to [Virtual-Key Codes](#).

In the case of an alphanumeric key, you can receive the actual ASCII code for the key pressed with the [sc.CharacterEventCode](#) variable.

When the custom study function is called when a key is pressed on the keyboard, this study function call is an efficient standard update calculation where `sc.Index` and `sc.UpdateStartIndex` be equal to `sc.ArraySize - 1`.

The study function will know the reason it is being called by checking that **sc.KeyboardKeyEventCode** or [sc.CharacterEventCode](#) is nonzero.

If either of these is nonzero, the study function only has to handle the key press event and does not need to do anything else. It can still do whatever it requires, but does not have to do any calculations because at the usual time a calculation needs to be performed when the chart is updated, the study function will be called again.

The flag variable [sc.DoNotRedrawChartAfterStudyReturns](#) can be set to specify to not redraw the chart window after a study function is called. This can be set when **sc.KeyboardKeyEventCode** or [sc.CharacterEventCode](#) is nonzero. This will make the process of handling these types of events very efficient.

If a single keyboard key is pressed, like an alphanumeric key, you can use either [sc.KeyboardKeyEventCode](#) or the [sc.CharacterEventCode](#) variables to obtain that particular key value.

When detecting key combinations by using for example [sc.IsKeyPressed_Control](#), and detecting that state along with an alphanumeric key, it is best to use **sc.CharacterEventCode**.

When detecting key combinations which have already been mapped to particular commands through **Global Settings >> Customize Keyboard Shortcuts**, is not possible to detect the usage of these keyboard combinations through ACSIL.

sc.ReceivePointerEvents

Type: Read/Write integer variable.

sc.ReceivePointerEvents can be set to **ACS_RECEIVE_NO_POINTER_EVENTS**, **ACS_RECEIVE_POINTER_EVENTS_WHEN_ACS_BUTTON_ENABLED**, **ACS_RECEIVE_POINTER_EVENTS_ALWAYS**, **ACS_RECEIVE_POINTER_EVENTS_ALWAYS_FOR_ALL_TOOLS** for the study function to receive mouse Pointer related events.

For complete details, refer to [Receiving Pointer Events](#).

sc.ReconnectToExternalServiceServer

Type: Write integer variable.

Set the **sc.ReconnectToExternalServiceServer** variable to TRUE to perform the **File >> Disconnect** and then the **File >> Connect to Data Feed** commands from within your custom study. This is not performed immediately at the time you set this to TRUE. It is only done when your study function returns.

Also see [sc.DisconnectFromExternalServiceServer](#) and [sc.ConnectToExternalServiceServer](#).

Example

```
sc.ReconnectToExternalServiceServer = TRUE;
```

sc.RenkoNewBarWhenExceeded

Type: Read/Write integer variable.

This ACSIL structure member is considered out of date/deprecated. Instead use the [sc.GetBarPeriodParameters](#) and [sc.SetBarPeriodParameters](#) functions.

sc.RenkoReversalOpenOffsetInTicks

Type: Read/Write integer variable.

This ACSIL structure member is considered out of date/deprecated. Instead use the [sc.GetBarPeriodParameters](#) and [sc.SetBarPeriodParameters](#) functions.

sc.RenkoTicksPerBar

Type: Read/Write integer variable.

This ACSIL structure member is considered out of date/deprecated. Instead use the [sc.GetBarPeriodParameters](#) and [sc.SetBarPeriodParameters](#) functions.

sc.RenkoTrendOpenOffsetInTicks

The **sc.RenkoTrendOpenOffsetInTicks** is the number of Ticks for the Trend Offset for Flex Renko chart bars. Each tick is equivalent to the chart **Tick Size**.

This ACSIL structure member is considered out of date/deprecated. Instead use the [sc.GetBarPeriodParameters](#) and [sc.SetBarPeriodParameters](#) functions.

sc.ReplayStatus

Type: Read-only integer variable.

The **sc.ReplayStatus** variable indicates the current replay status for the chart the study is applied to. It can be one of the following constants:

- **REPLAY_STOPPED:** The chart is not replaying.
- **REPLAY_RUNNING:** The chart is replaying.
- **REPLAY_PAUSED:** The chart is considered in a replaying state, however the replay is paused.

Example

```
int ReplayStatus = sc.ReplayStatus;
```

sc.ResetAlertOnNewBar

Type: Read/Write integer variable.

sc.AlertOnlyOncePerBar can be set to 1 or 0 (TRUE/FALSE) to force an alert set with [sc.SetAlert\(\)](#) to reset when there is a new bar. This works with the [sc.SetAlert\(\)](#) function.

sc.ResetAllScales

Type: Read/Write integer variable.

When the **sc.ResetAllScales** variable is set to a value of **1**, all the graphs in the chart associated with the study that called this function will have their scales reset to their default values.

sc.ReversalTicksPerBar

Type: Read/Write integer variable.

This ACSIL structure member is considered out of date/deprecated. Instead use the [sc.GetBarPeriodParameters](#) and [sc.SetBarPeriodParameters](#) functions.

sc.RightValuesScaleLeftCoordinate

Type: Read-only integer variable.

sc.RightValuesScaleLeftCoordinate is set to the pixel coordinate for the left of the Values Scale on the right side of the chart that the study function instance is applied to.

Also refer to [sc.RightValuesScaleRightCoordinate](#).

sc.RightValuesScaleRightCoordinate

Type: Read-only integer variable.

sc.RightValuesScaleRightCoordinate is set to the pixel coordinate for the right of the Values Scale on the right side of the chart that the study function instance is applied to.

This right coordinate is included in the right side Values Scale. It is not one pixel beyond the right side which would be the case if bounding coordinates were being provided.

Also refer to [sc.RightValuesScaleLeftCoordinate](#).

sc.RoundTurnCommission

Type: Read-only integer variable.

sc.RoundTurnCommission is the Round Turn Commission set for the symbol or symbol pattern in the [Global Symbol Settings](#).

If the chart the study instance is on is open at the time that you have set the **Round Turn Commission** or changed this value in the [Global Symbol Settings](#), it is necessary to go to that chart and select **Chart >> Reload and Recalculate**, to be able to access the value using **sc.RoundTurnCommission**.

sc.SaveChartImageToFile

Write-only integer variable.

Set **sc.SaveChartImageToFile** to 1 or a nonzero number, to flag to save an image of the chart the study is applied to, to a file in the **Images** subfolder in the folder where Sierra Chart is installed to.

This operation is performed after your study function returns and all other studies on a chart are calculated. The saving operation is performed when the chart is next updated which occurs at the **Chart Update Interval** setting in **Global Settings >> General Settings** or the chart specific

setting in **Chart >> Chart Settings**.

The Filename is the Symbol of the chart plus a timestamp which includes resolution down to the millisecond.

Also refer to [sc.SaveChartImageToFileExtended\(\)](#) and [sc.UploadChartImage\(\)](#).

Example

```
sc.SaveChartImageToFile = 1; // Save the chart image to a file when the study function returns
```

sc.ScaleBorderColor

Type: Read/Write integer variable.

sc.ScaleBorderColor

Example

```
unsigned int ScaleBorderColor;
```

sc.ScaleConstRange

Type: Read/Write integer variable.

Initial value: 0

sc.ScaleConstRange is the range, that is the difference between the high and the low, for the constant range scale types. This is only used when [sc.ScaleRangeType](#) is either **SCALE_CONSTRANGE** or **SCALE_CONSTRANGECENTER**. If [sc.ScaleRangeType](#) is set to one of these constant range types, **sc.ScaleConstRange** needs to be greater than 0.

Example

```
// Set this study to use a centered constant range scale  
sc.ScaleRangeType = SCALE_CONSTRANGECENTER;  
// Set the range of the scale to 10 points  
sc.ScaleConstRange = 10.0f;
```

sc.ScaleIncrement

Type: Read/Write integer variable.

Initial value: 0 (auto)

sc.ScaleIncrement is the increment at which values in the right side vertical scale on the chart are displayed. This is the same as the **Scale Increment** setting in the **Scale Window** that can be opened from the **Settings and Inputs** tab in the **Study Settings** window for the study. A value of 0 means the scale increment is automatically determined by Sierra Chart.

Example

```
sc.ScaleIncrement = 0.01f; // Draw values on the scale at every 0.01 point
```

sc.ScaleRangeBottom

Type: Read/Write integer variable.

Initial value: 0

sc.ScaleRangeBottom is the minimum value for the scale range when using a user-defined scale range ([sc.ScaleRangeType](#) = **SCALE_USERDEFINED**;). When this variable is set, it must always be less than or equal to [sc.ScaleRangeTop](#).

Example

```
sc.ScaleRangeType = SCALE_USERDEFINED;  
  
sc.ScaleRangeTop = 100.0f;  
sc.ScaleRangeBottom = -100.0f;
```

sc.ScaleRangeTop

Type: Read/Write integer variable.

Initial value: 0

sc.ScaleRangeTop is the maximum value for the scale range when using a user-defined scale range ([sc.ScaleRangeType](#) = **SCALE_USERDEFINED**;). When this variable is set, it must always be greater than or equal to [sc.ScaleRangeBottom](#).

Example

```
sc.ScaleRangeType = SCALE_USERDEFINED;  
  
sc.ScaleRangeTop = 100.0f;  
sc.ScaleRangeBottom = -100.0f;
```

sc.ScaleRangeType

Type: Read/Write integer variable.

Type: Initial value: **SCALE_AUTO**

The **sc.ScaleRangeType** member allows you to get and set the vertical Scale Range Type for the study. It can be any of the following constant values:

- SCALE_AUTO
- SCALE_USERDEFINED
- SCALE_INDEPENDENT
- SCALE_SAMEASREGION
- SCALE_CONSTRANGE
- SCALE_CONSTRANGECENTER
- SCALE_ZEROBASED R

Example

```
sc.ScaleRangeType = SCALE_INDEPENDENT; // Set this study to use an independent scale
```

sc.ScaleType

Type: Read/Write integer variable.

Initial value: **SCALE_LINEAR**

sc.ScaleType is the type of value scaling that is used on the study. This can be set to either **SCALE_LINEAR** for a basic linear scale, or **SCALE_LOGARITHMIC** for a logarithmic scale.

Example

```
sc.ScaleType = SCALE_LOGARITHMIC; // Set this study to use a logarithmic scale
```

sc.ScaleValueOffset

Type: Read/Write integer variable.

Initial value: 0

sc.ScaleValueOffset is a percentage value with a range from -1 to 1 for the offset of the scale from the center. A value of 1 means the graph will be pushed all the way off the bottom, and a value of -1 means the graph will be pushed all the way off the top. This is the value that gets changed when you use the Interactive Scale Move feature when you left click and drag the scale on the right side of the chart.

Example

```
sc.ScaleValueOffset = 0.25f
```

sc.SCDataFeedSymbol

Type: Function returning [SCString](#).

This member has been changed to a function effective with version 2152.

The **sc.SCDataFeedSymbol** function returns a text string which contains the symbol pattern used by the [Real-Time Exchange Data Feeds Available from Sierra Chart](#) which corresponds to the symbol of the chart the study is on.

This symbol pattern has specialized uses and is not typically used.

sc.ScrollToDateTime

Type: Write-only SCDatetime variable.

Initial Value: 0.

sc.ScrollToDateTime is a [SCDateTime](#) variable that can be set to a specific Date and Time to scroll the chart to.

As soon as the study function returns, the chart will scroll to the specified Date and Time and then the variable is cleared for next use, if required.

sc.SecondsPerBar

Type: Read/Write integer variable.

This ACSIL structure member is considered out of date/deprecated. Instead use the [sc.GetBarPeriodParameters](#) and [sc.SetBarPeriodParameters](#) functions.

sc.SelectedAlertSound

Type: Read-only integer variable.

The **sc.SelectedAlertSound** function returns the value of the [Alert Sound / Number](#) setting on the **Alerts** tab of the [Study Settings](#) window.

Example

```
int AlertSoundNumber = sc.SelectedAlertSound;
```

sc.SelectedTradeAccount

Type: SCString. Read/Write Text string variable.

The **sc.SelectedTradeAccount** is set to the same Trade Account which is selected and displayed on the Trade Window of the chart the study is applied to. For more information, refer to [Selecting Trade Account](#).

This can also be set to a different Trade Account to change the Trade Account for the chart. The change goes into effect when the study function returns.

However, this must be set to a valid Trade Account which is also valid based upon whether Trade Simulation Mode is enabled or not. Otherwise, the new Trade Account will be ignored.

sc.ServerConnectionState

Type: Read-only integer variable.

sc.ServerConnectionState is an Integer variable which can be one of the constants listed below. It indicates the connection state to the Data and Trade (in the case of a Trading service) servers.

The connection and disconnection to the servers is performed through

File >> Connect to Data Feed and **File >> Disconnect**.

- SCS_DISCONNECTED
- SCS_CONNECTING
- SCS_RECONNECTING
- SCS_CONNECTED
- SCS_CONNECTION_LOST
- SCS_DISCONNECTING

When the connection state is disconnected and then the connection becomes connected, the study function will be called. When the connection state is connected and then it is no longer connected, the study function will be called. At either of these times the study function can check the state of the **sc.ServerConnectionState** variable.

sc.ServiceCodeForSelectedDataTradingService

Type: Function returning [SCString](#).

This member has been changed to a function effective with version 2152.

The **sc.ServiceCodeForSelectedDataTradingService** function returns a text string which contains the internal Sierra Chart service code for the currently selected Data/Trade service set in **Global Settings >> Data/Trade Service Settings**.

sc.SetDefaults

Type: Read-only integer variable.

sc.SetDefaults is a TRUE (1) or FALSE (0) value that gets set to 1 (TRUE) when a study instance is manually added to a chart through **Analysis >> Studies >> Add >>**, a study instance

is added to a chart through a Study Collection which has just been applied to a chart, or when a Chartbook is opened and the study instance is contained on a chart in the Chartbook.

If the study is on more than one chart in the Chartbook or there are multiple instances of it, then the study function will be called once for each instance of it, with **sc.SetDefaults** set to 1.

This variable is also TRUE (1) and your study function is called when the

Analysis >> Studies >> Add Custom Study window is opened by a user. This is so that the Names and Descriptions of your studies can be listed for selection.

All study functions must include a code block at the top of the function that checks this member before doing any further processing within the study function. Refer to the example below. If this value is 1 (TRUE), then the study function must return at the end of the **sc.SetDefaults** code block.

The purpose of **sc.SetDefaults** is to allow the study function to configure itself and set defaults.

The **sc.SetDefaults** code block for setting your configuration and defaults is not meant to be used for anything other than configuring the study and setting the defaults of the study.

If you modify the source code for a study and rebuild the DLL when an instance of the study is on a chart and the chart is open, and you do not remove the study from the chart and add it again, then all of the default settings like Input and Subgraphs settings in the **sc.SetDefaults** code block will not change back to defaults. They will remain as they were previously were.

What Should Be Located in the **sc.SetDefaults** Code Block

1. **sc** interface structure members that configure the study and should only be set once. An example would be **sc.AutoLoop = 1**. It would not make sense to set a variable like this outside of **sc.SetDefaults** because in some cases it is too late to set them if used outside the **sc.SetDefaults** code block and it would be inefficient to keep setting them.
2. Setting of defaults. For example, you will want to set properties of a **sc.Subgraph[]** like the **sc.Subgraph[].DrawStyle** and the colors. You will want to set default study Input values with the **sc.Input[].Set*** functions. Setting of defaults must be done in the **sc.SetDefaults** code block because otherwise when the settings are changed through the **Study Settings** window for the study, those changes will get reverted back to what the study function is setting them to if the code to change the **sc.Subgraph**, **sc.Input** and other settings is done outside of the **sc.SetDefaults** code block.
3. **sc** members that need to be set only once. For example, you will want to set **sc.GraphName**, **sc.Subgraph[].Name** and **sc.Input[].Name** in the **sc.SetDefaults** code block because usually these names will not change during chart updating.
4. Unless otherwise noted in the documentation for the particular ACSIL interface structure member (those that begin with **sc.**), any member used in the **sc.SetDefaults** code block can also be read, set or used outside of

sc.SetDefaults block. For example, [sc.GraphName](#) can be changed later on by also setting it outside of the **sc.SetDefaults** code block.

5. There are some ACSIL members like **sc.BaseData[][]**, where it would not make sense to interact with them within the **sc.SetDefaults** code block because the arrays will be empty.
6. If an **sc** member is associated with the chart, it must not be set in the **sc.SetDefaults** code block because it will have no effect. For example, you cannot set [sc.StartTime1](#) inside of **sc.SetDefaults**. However, it is possible to read **sc.StartTime1** from within the **sc.SetDefaults** code block. All ACSIL members which are related to the chart itself, like **sc.StartTime1**, are set and valid when **sc.SetDefaults** is TRUE. However, they are read-only.
7. When using [ACSIL Functions](#) within the **sc.SetDefaults** code block and those functions interact with the chart, they will have no effect. They will return and do nothing.

Example

```
SCSFExport scsf_UniqueFunctionName(SCStudyInterfaceRef sc)
{
    if (sc.SetDefaults)
    {
        // Set the defaults
        sc.GraphName = "My New Study Function";

        sc.Subgraph[0].Name = "Subgraph name";
        sc.Subgraph[0].DrawStyle = DRAWSTYLE_LINE;

        sc.AutoLoop = 1;

        // You can also enter additional configuration code
        return;
    }

    // Perform your data processing here.

    //This is an example that multiplies the last price by 10.
    sc.Subgraph[0][sc.Index] = sc.BaseData[SC_LAST][sc.Index] * 10;

    return;
}
```

sc.StandardChartHeader

Type: Read/Write integer variable.

Initial value: 0 (FALSE)

If **sc.StandardChartHeader** is set to 1 (TRUE), then your study will use the standard main Price Graph text header. This text header can be configured by going to

Global Settings >> Customize Chart Header on the menu. If you set this to 1 (TRUE), you must also set [sc.DisplayAsMainPriceGraph](#) to 1 (TRUE), otherwise it will be ignored.

Example

```
sc.DisplayAsMainPriceGraph = 1; // Set the study to the main Price Graph  
sc.StandardChartHeader = 1;
```

sc.StartTime1

Read/write integer variable.

sc.StartTime1 is the first start time for a day on the chart.

This is equal to the [Session Times >> Start Time](#) setting for the chart.

This variable is a `SCDateTime` [TimeValue](#) in seconds.

Also refer to [sc.EndTime1](#).

sc.StartTime2

Read/write integer variable.

sc.StartTime2 is the second start time for a day on the chart. This is only used if [sc.UseSecondStartEndTimes](#) is set to 1 (TRUE).

This is equal to the [Session Times >> Evening Start Time](#) setting for the chart.

This variable is a `SCDateTime` [TimeValue](#) in seconds.

Also refer to [sc.EndTime2](#).

sc.StartTimeOfDay

Type: Read-only integer variable.

The **sc.StartTimeOfDay** variable is set to the starting time of the trading day based on the [Session Times](#) settings for the chart the study instance is on.

The value returned is a [Time Value](#).

It can be converted to a [SCDateTime](#) type by using the `SCDateTime::SetTime()` function. Refer to the example below.

Example

```
SCDateTime StartTime;  
StartTime.SetTime(sc.StartTimeOfDay);
```

sc.StorageBlock

Type: Read/Write block of bytes.

sc.StorageBlock is a pointer to a block of 4096 bytes of permanent memory storage. This block of memory can be used for your custom data storage, and it is saved to disk when the chartbook is saved.

This storage block is unique to each individual study instance.

Depending upon what you are using the Storage Block for, you may want to enable

Global Settings >> General Settings >> Charts >> Chartbooks >> Autosave Chartbooks Interval to automatically save your Chartbook periodically.

Example

```
// 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");
```

sc.StudyDescription

Type: Read/Write string variable.

sc.StudyDescription is a text string that can be set to a description of the study. This is optional.

If you use this, it needs to be in the [sc.SetDefaults](#) code block. This study description will be displayed on a webpage when using the **Show Study Description** or **Description** buttons in the Chart Studies and Study Settings windows respectively.

Once **sc.StudyDescription** is set in the [sc.SetDefaults](#) code block. It cannot later be accessed in the ACSIL function. The string will then be blank.

Example

```
sc.StudyDescription = "Here is a description for this study. This description will be shown in the Add Custom
```

sc.StudyGraphInstanceID

Type: Read-only **Integer** variable.

The StudyGraphInstanceID is a unique number assigned to each study on a chart. This identifier number is for the particular instance of your custom study which is applied to the chart. This ID number can be seen in the **Chart Studies** window on the right side of the study name in the **Studies to Graph** list.

This is the same study identifier which is used with functions like **sc.GetStudyArrayUsingID**.

The underlying main price graph in the chart will always have a StudyGraphInstanceID itself of 0. If there is another study which is acting as the main price graph, then it will have a value higher than 0.

Example

```
int StudyGraphInstanceID = sc.StudyGraphInstanceID;
```

sc.StudyRegionBottomCoordinate

Type: Read-only integer variable.

sc.StudyRegionBottomCoordinate is the Y-coordinate of the bottom of the Chart Region that the study is in. This value is given in the coordinate system of the client window and is in pixels.

This variable will only provide a valid value when read outside of the **sc.SetDefaults** code block.

sc.StudyRegionLeftCoordinate

Type: Read Only integer variable.

sc.StudyRegionLeftCoordinate is the X-coordinate of the left side of the Chart Region that the study is in. This value is given in the coordinate system of the client window and is in pixels.

This variable will only provide a valid value when read outside of the **sc.SetDefaults** code block.

sc.StudyRegionRightCoordinate

Type: Read Only integer variable.

sc.StudyRegionRightCoordinate is the X-coordinate of the right side of the Chart Region that the study is in. This value is given in the coordinate system of the client window and is in pixels.

This variable will only provide a valid value when read outside of the **sc.SetDefaults** code block.

sc.StudyRegionTopCoordinate

Type: Read-only integer variable.

sc.StudyRegionTopCoordinate is the Y-coordinate of the top of the Chart Region that the study is in. This value is given in the coordinate system of the client window and is in pixels.

This variable will only provide a valid value when read outside of the **sc.SetDefaults** code block.

sc.StudyVersion

Type: Unsigned integer variable.

sc.StudyVersion can be set to any integer value which is then displayed on the [Settings and Inputs](#) tab of the **Study Settings** window for the study. It is prefixed with the text **Study Version:**.

This simply provides version information to the user and serves no other purpose.

SC_SUBGRAPHS_AVAILABLE

Type: Constant integer variable.

SC_SUBGRAPHS_AVAILABLE is the number of study Subgraphs there is available for a custom study. For additional information, refer to [sc.Subgraphs](#).

sc.SupportAttachedOrdersForTrading

Type: integer variable.

When **sc.SupportAttachedOrdersForTrading** is set to TRUE(1), then the [Attached Orders](#) set on the Trade Window for the chart the trading study is applied to will be used no matter what the **Use Attached Orders** setting is set to on the Trade Window.

However, this setting is irrelevant when Targets or Stops are set on the [s_SCNewOrder](#) data structure. In this case it is implied to be on for those Attached Orders.

sc.SupportKeyboardModifierStates

Type: Read/Write Integer variable.

The **sc.SupportKeyboardModifierStates** variable is used to allow the [sc.IsKeyPressed_Control](#), [sc.IsKeyPressed_Shift](#), [sc.IsKeyPressed_Alt](#) variables to be set to indicate the state of these keyboard modifiers.

Set this variable to 1 to allow support, or set to 0 to not allow support. If this is set to 0, then it is not possible for ACSIL function to determine the state of these keyboard modifiers. The default is 0.

sc.SupportTradingScaleIn

Type: Read/Write integer variable.

The **sc.SupportTradingScaleIn** variable is set to 1 when the Scaling In option for trading is enabled for the chart. 0 means it is disabled. This variable can also be set by the study function to enable [Scaling In](#).

This variable must be set outside of and below the **sc.SetDefaults** code block in the study function. Otherwise, it will have no effect.

sc.SupportTradingScaleIn is ignored when submitting an order for a different Symbol or Trade Account than the chart is currently set to.

Example

```
sc.SupportTradingScaleIn = 1;
```

sc.SupportTradingScaleOut

Type: Read/Write integer variable.

The **sc.SupportTradingScaleOut** variable is set to 1 when the Scaling Out option for trading is enabled for the chart. 0 means it is disabled. This variable can also be set by the study function to enable [Scaling Out](#).

This variable must be set outside of and below the **sc.SetDefaults** code block in the study function. Otherwise, it will have no effect.

sc.SupportTradingScaleOut is ignored when submitting an order for a different Symbol or Trade Account than the chart is currently set to.

Example

```
sc.SupportTradingScaleOut = 1;
```

sc.Symbol

Type: Read-only SCString variable.

sc.Symbol is the symbol of the chart.

To perform comparisons to this symbol, or to directly access it see [How To Compare Strings](#) and [Directly Accessing a SCString](#), respectively.

To change the symbol of the chart use [sc.DataFile](#).

sc.SymbolData

Type: Read Only data structure.

sc.SymbolData is a pointer to a data structure containing all of the current pricing and related data for the symbol of the chart.

The symbol data structure does not include the market depth data. For market depth data, use

the [sc.GetBidMarketDepthEntryAtLevel](#) and [sc.GetBidMarketDepthEntryAtLevel](#) functions.

The data in this structure is only valid when Sierra Chart is connected to the data feed and the symbol is receiving data from the data feed or the chart is replaying.

Refer to the **ACS_Source/SCSymbolData.h** file in the folder that Sierra Chart is installed to, for the structure definition.

sc.TextInput

Type: Read/Write SCString variable.

Initial value: "" (empty string)

sc.TextInput is the text input string that is made available in the study's inputs. The text input can be found at the bottom of the list of inputs shown on the Inputs and Settings tab in the Technical Study Settings window. This string can not be longer than 255 characters.

You must have the [sc.TextInputName](#) set if you want to have the text input available to the user.

sc.TextInput is considered out of date. It is recommended to use [sc.Input\[\].SetString\(\)](#) and [sc.Input\[\].GetString\(\)](#) instead.

Example

```
// Make a copy of the text input into our own c-style string. Be very careful
// when using this and make sure you will not use an invalid index into the
// string array. If this string is not used properly, the study could crash.
char TextInputCopy[256];

strncpy(TextInputCopy, sc.TextInput.GetChars(), sizeof(TextInputCopy) - 1);
```

sc.TextInputName

Type: Read/Write SCString variable.

Initial value: "" (empty string)

sc.TextInputName is a string of the name that is shown in the table of inputs on the Inputs and Settings tab in the Technical Study Settings window. You must set this to make the text input available to the user.

Example

```
sc.TextInputName = "Letters"; // Show the text input with the name Letters
```

sc.TickSize

Type: Read/Write float variable.

sc.TickSize is a variable that is set to the Tick Size of the chart the study instance is applied to.

This is the same as the [Tick Size](#) setting set in the **Chart >> Chart Settings** window.

This variable can be modified by the custom study. When it is modified, to cause the chart to reload which is necessary, the study must set [sc.FlagToReloadChartData](#) = 1.

sc.TimeScaleAdjustment

Type: Read-only SCDatetime variable.

sc.TimeScaleAdjustment is the difference between the global [Time Zone](#) setting in Sierra Chart and GMT/UTC time. Or if there is a separate Time Zone setting on the chart, then it is the difference between the Time Zone setting for the chart and GMT/UTC time.

This value uses the Daylight Savings Time adjustment based upon whether it is currently in effect or not.

This variable is a [SCDateTime](#) variable. You can add this variable to or subtract this variable from any [SCDateTime](#) variable.

This variable is only useful for when working with Date-Times for the latest day in the chart. For prior days, use the [sc.ConvertDateTimeFromChartTimeZone](#) function.

Example

```
// Figure out the Date-Time of the last bar without the time offset applied.  
// The Date-Time values of sc.BaseDateTimeln[] are already adjusted to the Sierra Chart Time Zone setting  
// Subtracting sc.TimeScaleAdjustment will remove the adjustment.  
SCDateTime AdjustedDateTime = sc.BaseDateTimeln[sc.ArraySize - 1] - sc.TimeScaleAdjustment;
```

Example

```
// Adjust a time and sales record date-time  
// TimeSales is a s_TimeAndSales record requested with sc.GetTimeAndSales().  
SCDateTime TempDateTime = TimeSales[TSIndex].DateTime;  
TempDateTime += sc.TimeScaleAdjustment; // Apply the time zone offset for the chart
```

sc.TradeAndCurrentQuoteSymbol

Type: Read/Write SCString variable.

sc.TradeAndCurrentQuoteSymbol is a text string containing the **Trade and Current Quote Symbol** set in **Chart >> Chart Settings**.

This symbol can be changed and one purpose for this is to change it to a Trade and Current

Quote Symbol which corresponds with the `sc.Symbol` text string.

When this is changed, it does not go into effect until after the study function returns. At that time, the [Trades List](#) in the chart is rebuilt if necessary.

This symbol is what is used when trading from a chart. All trading related functionality uses this Trade and Current Quote symbol. Although after changing this symbol it is not possible during the same call into the study function to then submit a trading order which will then use this symbol because the symbol has not yet gone into effect. To avoid this limitation, either wait to submit the trading order until the next call into the study function or it is necessary to use the functionality to submit an order for a different Symbol and/or Trade Account. Refer to [Submitting and Managing Orders for Different Symbol and/or Trade Account](#).

sc.TradeServiceAccountBalance

Type: Read-only double precision float variable.

This variable is no longer supported since version 2395 and higher. Instead use the [sc.GetTradeAccountData](#) function.

sc.TradeServiceAvailableFundsForNewPositions

Type: Read-only double precision float variable.

This variable is no longer supported since version 2395 and higher. Instead use the [sc.GetTradeAccountData](#) function.

sc.TradeWindowConfigFileName

Type: Read/Write SCString variable.

The **sc.TradeWindowConfigFileName** string variable can be set to the file name, not including the folder path, to a previously saved Trade Window configuration file that the custom study on the chart wants to configure the Trade Window to.

For further details about Trade Window Configurations, refer to [Using and Changing Between Different Trade Window and Attached Orders Configurations](#).

sc.TradeWindowConfigFileName is set to the current Trade Window configuration file name.

For further information about using this variable, refer to [SupportAttachedOrdersForTrading](#).

One useful purpose of this is to create a custom study which uses Control Bar buttons and/or Keyboard Shortcuts to change to your own custom Trade Window configuration. For additional information, refer to [Advanced Custom Study Interaction With Menus, Control Bars, Pointer Events](#).

Example

```
// This line can be within the sc.SetDefaults code block or below it.  
sc.TradeWindowConfigFileName = "SimpleBracket.twconfig";
```

sc.TradeWindowOrderQuantity

Type: Read/Write integer (32-bit version) / double (64-bit version) variable.

sc.TradeWindowOrderQuantity is set to the order quantity on the Trade Window for the chart that the study instance is applied to.

Example

```
int OrderQuantity = sc.TradeWindowOrderQuantity;
```

sc.TradingIsLocked

Type: Read-only integer variable.

The **sc.TradingIsLocked** variable is set to 1 when **Trade >> Trading Locked** is enabled. Otherwise, it is set to 0.

To change the state of trading locked, use the [sc.SetTradingLockState](#) function.

sc.TransparencyLevel

Type: Read/Write integer variable.

This is the Transparency Level for Subgraph Draw Styles that require transparency. For additional information, refer to [Transparency Level](#).

sc.UpdateAlways

Type: Read/Write integer variable.

Initial value: 0 (FALSE)

When the **sc.UpdateAlways** variable is set to 1 (TRUE), the study instance will update continuously rather than only when new market or order data is available for the symbol.

This means your study function will be called continuously at the update interval.

The update interval is set through

Global Settings >> General Settings >> Chart Update Interval.

It is possible to set an individual chart to update at a different interval than the global **Chart Update Interval**. To do this, set a nonzero number for

Chart >> Chart Settings >> Display >> Chart Update Interval in Milliseconds.

One use of this setting is to simulate real time updating during testing. This setting does not affect other studies on the same chart.

To maintain efficiency, the study function will not be called more often than every 200 ms when setting this variable to TRUE and new data is not available. Otherwise, setting this variable to 1 (TRUE) could significantly increase CPU usage if the Chart Update Interval is set very low. Effective with version 1340, the preceding does not apply if the chart has set its own Chart Update Interval.

Example

```
sc.UpdateAlways = 1; // Set this study to continuously update
```

sc.UpdateStartIndex

Type: Read-only integer variable.

sc.UpdateStartIndex is only used with Manual Looping. For complete information on Manual Looping, refer to [Working with ACSIL Arrays and Understanding Looping](#).

The array indexing variables **sc.Index**, **sc.CurrentIndex** and **sc.UpdateStartIndex** are all set to the same value when the study function instance is called. All three of these variables can be used interchangeably. However, the behavior of **sc.Index**, **sc.CurrentIndex** and **sc.UpdateStartIndex** differs significantly depending whether you are using Auto Looping or Manual Looping.

sc.UpdateStartIndex is set by Sierra Chart to the index where the primary **for** loop in the study function will start looping from. This is the index into the [sc.BaseDataIn\[\]](#) arrays where data 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 index into the [sc.BaseDataIn\[\]](#) arrays to begin processing at.

For example: When a chart is opened, reloaded, recalculated, or a replay is started, the study function instance will be called and **sc.UpdateStartIndex** will be 0. This means the study function needs to update all elements in the [sc.Subgraph\[\].Data\[\]](#) arrays it is using.

During chart updating, if there are 100 bars in a chart and a trade occurs and either the last bar is updated or a new bar is added to the chart, then **sc.UpdateStartIndex** variable will be set to 99 indicating the value at index 99 (bar 100) has been updated.

For example: **sc.BaseDataIn[SC_LAST][99]** (bar 100) has been updated. During normal chart updating **sc.UpdateStartIndex** will always be equal to the last index of the prior **sc.ArraySize** before the chart update. If a new bar was added, then there will also be a new array element at **sc.BaseDataIn[][100]** (bar 101). In this case, the study function needs to update the

sc.Subgraph[].Data[] arrays at index 99 and index 100. The reason you need to use **sc.UpdateStartIndex** is to only perform calculations on the modified data and only update the study values starting from this index. This makes your study very efficient.

It is very possible that more than 1 bar will be added to a chart between updates. There could be hundreds or thousands of bars potentially added when historical data is downloaded into the chart after the initial data load from the local data file on your system. Therefore, you need to update from **sc.UpdateStartIndex** to the last array element in all the [sc.Subgraph\[\]\[\]](#) Data arrays you are using, and not just the element at this index.

The index value provided by **sc.UpdateStartIndex** will never go backwards. However, during a full recalculation of the study, such as when the chart is reloaded or for other reasons, it can start back at 0. Therefore, you can rely that it will not go backwards other than to 0.

It is possible that a chart containing a study instance is making references to other charts which are triggering a full recalculation in the chart which is making the reference. In this particular case, during the full recalculation **sc.UpdateStartIndex** will be set to 0 for the study instances using Manual Looping. For additional information, refer to [References to Other Charts and Tagging](#).

sc.UseGlobalChartColors

Type: Read/Write integer variable.

When **sc.UseGlobalChartColors** is set to 1 or a nonzero number, then the global color settings defined in **Global Settings >> Graphics Settings** are used for the chart.

When **sc.UseGlobalChartColors** is set to 0, then the chart specific color settings defined in **Chart >> Graphics Settings** are used for the chart.

Since **sc.UseGlobalChartColors** affects the chart itself, it must be located outside and below the **sc.SetDefaults** code block.

Example

```
sc.UseGlobalChartColors = 0;
```

sc.UseGUIAttachedOrderSetting

Type: Read/Write integer variable.

When **sc.UseGUIAttachedOrderSetting** is set to a nonzero number, then the [Attached Orders](#) defined on the Trade Window for the chart, if any, will be attached to [Buy Entry or Sell Entry](#) trade orders submitted from the study function if the **Use Attached Orders** setting is also enabled on the Trade Window.

sc.UseHighResolutionWindowRelativeCoordinatesForChartDrawings

Type:

sc.UserName

Type: Function returning [SCString](#).

This member has been changed to a function effective with version 2152.

The **sc.UserName** function returns a text string which contains the Account Name that was entered at the Sierra Chart Login window, which displays when Sierra Chart starts.

sc.UseSecondStartEndTimes

Type: Read/Write integer variable.

The **sc.UseSecondStartEndTimes** variable can be set to 1 (TRUE) or 0 (FALSE). This is equivalent to **Chart >> Chart Settings >> Use Evening Session**.

When it is set to 1, then the [sc.StartTime2](#) and [sc.EndTime2](#) variables are used by the chart the study is on. These variables represent the Evening Session times in **Chart >> Chart Settings**.

sc.UsesMarketDepthData

Type: Read/Write integer variable.

The **sc.UsesMarketDepthData** variable when set to 1 will cause market depth data to be subscribed to for the Symbol of the chart the custom study applied to.

Sierra Chart needs to be connected to the data feed with **File >> Connect to Data Feed** to receive market depth data.

sc.ValueFormat

Type: Read/Write integer variable.

Initial value: 2.

sc.ValueFormat sets the numeric display format for all values in the `sc.Subgraph[]`.Data arrays for the study. This can be one of the following values:

- 0 = Zero decimal places. Example output: 1
- 1 = One decimal place. Example output: 0.1
- 2 = Two decimal places. Example output: 0.01
- 3 = Three decimal places. Example output: 0.001
- 4 = Four decimal places. Example output: 0.0001
- 5 = Five decimal places. Example output: 0.00001
- 6 = Six decimal places. Example output: 0.000001
- 7 = Seven decimal places. Example output: 0.0000001
- 8 = Eight decimal places. Example output: 0.00000001
- 9 = Nine decimal places. Example output: 0.000000001

- 19 = Maximum Digits
- 20 = Time value. Where the value is as a floating-point fractional value where one millisecond is 1/86400000. Example output: 12:30:00
- 21 = Date
- 21 = Date
- 22 = Date-Time
- 23 = Currency
- 24 = Percent
- 25 = Scientific Notation
- 26 = Large Integer with Suffix
- 27 = Boolean
- VALUEFORMAT_INHERITED = Same Value Format as is set in **Chart >> Chart Settings** .
- 102 = Halves. Example output: 1/2
- 104 = Quarters. Example output: 1/4
- 108 = Eighths. Example output: 1/8
- 116 = Sixteenths. Example output: 1/16
- 132 = Thirty-seconds. Example output: 1/32
- 134 = Half Thirty-seconds. Example output: .5/32
- 136 = Quarter Thirty-seconds. Example output: .25/32
- 140 = Eighth Thirty-seconds. Example output: .125/32
- 164 = Sixty-fourths. Example output: 1/64
- 228 = One-hundred-twenty-eighths. Example output: 1/128
- 356 = Two-hundred-fifty-sixths. Example output: 1/256

Example

```
sc.ValueFormat = 4;
```

sc.ValueIncrementPerBar

Type: Read-only float variable.

sc.ValueIncrementPerBar is the **Value Increment per Bar in Ticks (Price Unit per Bar in Ticks)** setting in **Chart >> Chart Settings >> Chart Drawings** .

For complete details, refer to [Value Increment per Bar in Ticks \(Price Unit per Bar in Ticks\)](#) in the Chart Settings documentation.

Example

```
float ValueIncrementPerBar = sc.ValueIncrementPerBar;
```

sc.VersionNumber

Type: Function returning [SCString](#).

This member has been changed to a function effective with version 2152.

The **sc.VersionNumber** function returns a text string which contains the current version number of Sierra Chart the user is using.

Note: If your Advanced Custom Study is used on a version of Sierra Chart that does not support all of the interface members compared with the version it was built under, then a warning message will be given and the study cannot be used.

Therefore, using this version number member to perform a check to see if your study will work on a certain version, is unnecessary.

Example

```
if (atoi(sc.VersionNumber().GetChars()) < 2100)
{
    //do something
}
```

sc.VolumeAtPriceForBars

Type: Read-only custom data array object pointer of type **c_VAPContainer**.

Sierra Chart internally maintains volume at price data for each price tick of each loaded bar. The tick size is controlled by the **Tick Size** setting for the chart.

This data is maintained when **sc.MaintainVolumeAtPriceData** is set to 1 (TRUE) in the custom study function. Or, when there is a study on the chart which requires this data such as the **Volume by Price** study. This data is fully accessible through the ACSIL using the member **sc.VolumeAtPriceForBars**.

It is essential that **sc.MaintainVolumeAtPriceData** be set to 1 in the study function in the **sc.SetDefaults** code block to ensure that the necessary volume at price data is being maintained for the chart.

For an example on how to use **sc.VolumeAtPriceForBars**, refer to the function **scsf_VolumeAtPriceArrayTest** in the **/ACS_Source/studies8.cpp** file in the folder where Sierra Chart is installed to. Or refer to **scsf_VolumeWeightedAveragePrice** in **/ACS_Source/studies2.cpp**. Or refer to **scsf_VolumeAtPriceThresholdAlertV2** in **/ACS_Source/studies8.cpp**.

All of these are good examples. The basic method to access the volume by price data for a chart bar is given in the code example below.

sc.VolumeAtPriceForBars is a pointer to a **c_VAPContainer**. To access a function member of

c_VAPContainer requires that you use the member of pointer operator ->. Refer to [Member access operators](#).

It is no longer recommended to use the **GetNextHigherVAPElement** or the **GetNextLowerVAPElement** member functions to access the data. These are less efficient than the example below.

The function parameter that is commonly used in the c_VAPContainer is **PriceInTicks**. To calculate what you should use for **PriceInTicks**, divide the price by **sc.TickSize**.

Example

```
unsigned int TotalVolume = 0;

const s_VolumeAtPriceV2 *p_VolumeAtPrice=NULL;

int VAPSizeAtBarIndex = sc.VolumeAtPriceForBars->GetSizeAtBarIndex(sc.Index);

for (int VAPIndex = 0; VAPIndex < VAPSizeAtBarIndex; VAPIndex++)
{
    if (!sc.VolumeAtPriceForBars->GetVAPElementAtIndex(sc.Index, VAPIndex, &p_VolumeAtPrice))
        break;

    TotalVolume += p_VolumeAtPrice->Volume;
}
```

sc.VolumeAtPriceForBars Member Function Descriptions

The following are descriptions of some of the member functions of sc.VolumeAtPriceForBars.

- **int sc.VolumeAtPriceForBars->GetNumberOfBars()**: This function returns the last chart bar index + 1 that the sc.VolumeAtPriceForBars container has volume at price data for. As long as volume at price data is being maintained for the chart, which will be the case when **sc.MaintainVolumeAtPriceData = 1**, the return value of this function will equal sc.ArraySize. This function takes no parameters. If the **sc.VolumeAtPriceForBars** container is empty, then the return value will be 0.
- **int sc.VolumeAtPriceForBars->GetSizeAtBarIndex(unsigned int BarIndex)**: This function returns the number of prices that have Volume at Price data for the given **BarIndex**.
- **bool sc.VolumeAtPriceForBars->GetVAPElementAtIndex(unsigned int BarIndex, int VAPDataIndex, s_VolumeAtPriceV2** p_VAP)**: This function fills out the given **s_VolumeAtPriceV2** data structure pointer (**p_VAP**) for the specified **BarIndex** and **VAPDataIndex**. In the case of automatic looping **BarIndex** normally needs to be set to **sc.Index**. **VAPDataIndex** is the zero-based price index within the chart bar to return the s_VolumeAtPriceV2 data for. 0 is the lowest price within the chart bar. Higher indexes return higher values. The number of prices can be determined with the

sc.VolumeAtPriceForBars->GetSizeAtBarIndex function. The maximum value of **VAPDataIndex** can be set to will be the number returned by **sc.VolumeAtPriceForBars->GetSizeAtBarIndex** - 1.

Always pass the address of the given **s_VolumeAtPriceV2** structure pointer (**p_VAP**). If **p_VAP** is 0 upon returning from this function, then no data was returned. This function returns **true** upon success and **false** upon no data found for the given parameters.

There is another overload of this function which takes a const **s_VolumeAtPriceV2** p_VAP**.

- **const s_VolumeAtPriceV2& sc.VolumeAtPriceForBars->GetVAPElementAtPrice**(const unsigned int **BarIndex**, const int **PriceInTicks**):

The **GetVAPElementAtPrice** function returns a reference to a **s_VolumeAtPriceV2** element for the bar index specified by **BarIndex** and at the price specified by **PriceInTicks**. **PriceInTicks** is an integer specifying the price in ticks, where each tick is specified by chart Tick Size and is equal to 1. So if the Tick Size is .25, then the price value of 10 would be 40 (=10/.25).

- **bool sc.VolumeAtPriceForBars->GetVAPElementForPriceIfExists**(const unsigned int **BarIndex**, const int **PriceInTicks**, **s_VolumeAtPrice** p_VAP** , unsigned int& **r_InsertionIndex**):

The **GetVAPElementForPriceIfExists** function fills out the given **s_VolumeAtPriceV2** data structure pointer (**p_VAP**) for the specified bar index specified by **BarIndex** and at the price specified by **PriceInTicks**, if the price exists. **PriceInTicks** is an integer specifying the price in ticks, where each tick is specified by chart Tick Size and is equal to 1. So if the Tick Size is .25, then the price value of 10 would be 40 (=10/.25).

In the case when the cam volume at Price element is not found and the function returns false, the **r_InsertionIndex** parameter which is an integer, will be set to the index in the Volume at Price container, where the Volume at Price data will be inserted for the specified **BarIndex** and **PriceInTicks** parameters.

- **sc.VolumeAtPriceForBars->SetVolumeAtPrice**(const int **PriceInTicks**, const unsigned int **BarIndex**, const **t_VolumeAtPrice& VolumeAtPrice**):

The **VolumeAtPrice** volume at price data structure will be placed at the specified **PriceInTicks** and chart **BarIndex**. If **PriceInTicks** does not exist, the function will do nothing and return false.

sc.VolumeAtPriceForStudy

Type: Read/write custom data array object pointer of type **c_VAPContainer**.

The **sc.VolumeAtPriceForStudy** variable is the same as [sc.VolumeAtPriceForBars](#) except it is the volume at price data for the study itself instead of the main price graph of the chart. Normally this is not applicable and contains no data but it is used for special purposes.

There is no further documentation available at this time.

sc.VolumeAtPriceMultiplier

Type: Read/Write integer variable.

The **sc.VolumeAtPriceMultiplier** variable is set to the same value as the **Volume At Price Multiplier** setting on the **Chart >> Chart Settings >> Main Settings** tab.

When the value is changed, the chart will reload using this new value, after the study function returns.

sc.VolumePerBar

Type: Read/Write integer variable.

This ACSIL structure member is considered out of date/deprecated. Instead use the [sc.GetBarPeriodParameters](#) and [sc.SetBarPeriodParameters](#) functions.

sc.VolumeValueFormat

Type: Read/Write integer variable.

The **sc.VolumeValueFormat** variable specifies the format for fractional volume values. This variable is the same as the setting in **Chart >> Chart Settings >> Symbol >> Volume Value Format**.

Normally this will be set to 1. It is only used when a particular symbol trades in a fractional quantity.

*Last modified Wednesday, 05th July, 2023.