Quant Script™

Quant Script™

Service Disclaimer

This manual was written for use with the ***QuantScriptTM*** programming language. This manual and the product described in it are copyrighted, with all rights reserved. This manual and the ***QuantScriptTM*** outputs (charts, images, data, market quotes, and other features belonging to the product) may not be copied, except as otherwise provided in your license or as expressly permitted in writing by S-Trader.com and/or its development partners.

Export of this technology may be controlled by the United States Government and/or the Canadian Government. Diversion contrary to U.S. and/or Canadian law prohibited. Use of the ***QuantScriptTM*** product and other services accompanying your license and its documentation are governed by the terms set forth in your license. Such use is at your sole risk. The service and its documentation (including this manual) are provided «AS IS» and without warranty of any kind.

S-Trader.com, its development partners AND ITS LICENSORS (HEREINAFTER COLLECTIVELY REFERRED TO AS "S-TRADER") EXPRESSLY DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE AND AGAINST INFRINGEMENT. WHILE WE MAKE AN EFFORT TO ENSURE THE BEST QUALITY OF SERVICE POSSIBLE, S-TRADER DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE SERVICE WILL MEET YOUR REQUIREMENTS, OR THAT THE OPERATION OF THE SERVICE WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT DEFECTS IN THE SERVICE OR ERRORS IN THE DATA WILL BE CORRECTED. FURTHERMORE, S-TRADER DOES NOT WARRANT OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE SERVICE OR ITS DOCUMENTATION IN TERMS OF THEIR CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE. NO ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY S-TRADER OR AN S-TRADER AUTHORIZED REPRESENTATIVE SHALL CREATE A WARRANTY OR IN ANY WAY INCREASE THE SCOPE OF THIS WARRANTY.

SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY. UNDER NO CIRCUMSTANCES INCLUDING NEGLIGENCE, SHALL S-TRADER, ITS LICENSORS OR THEIR DIRECTORS, OFFICERS, EMPLOYEES OR AGENTS BE LIABLE FOR ANY INCIDENTAL, SPECIAL OR CONSEQUENTIAL DAMAGES (INCLUDING DAMAGES FOR LOSS OF BUSINESS, LOSS OF PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION AND THE LIKE) ARISING OUT OF THE USE OR THE INABILITY TO USE THE SERVICE OR ITS DOCUMENTATION, EVEN IF S-TRADER OR AN S-TRADER AUTHORIZED REPRESENTATIVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME JURISDICTIONS DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY. In no event shall S-TRADER's total liability to you for all damages, losses, and causes of action (whether in contract, tort, including negligence, or otherwise) exceed the monthly lease fee paid for the product and its documentation.

Trading Disclaimer

No offer or solicitation to buy or sell cash or derivative securities, including but not limited to futures, options or options on futures is being made. The service is not and should thus not be construed as trading or investment advice and no recommendation or strategy is being made, given or in any manner endorsed by S-TRADER or any of its affiliates, partners, employees or developing partners. Past performance, whether actual or indicated by historical tests of strategies, is no guarantee of future performance or success. Active trading is generally not appropriate for someone of limited resources, limited investment or trading experience, or low-risk tolerance, or who does not have capital to risk. There is a risk of loss in stock, futures and forex trading. Market data may be delayed or unavailable at times due to system and software errors, Internet traffic, outages and other factors. Trading carries a high level of risk and may not be suitable for all investors. There is a possibility that you may sustain a loss equal to or greater than your entire investment; therefore, you should not invest or risk money that you cannot afford to lose. You should be aware of all risks associated with trading your particular markets and products.

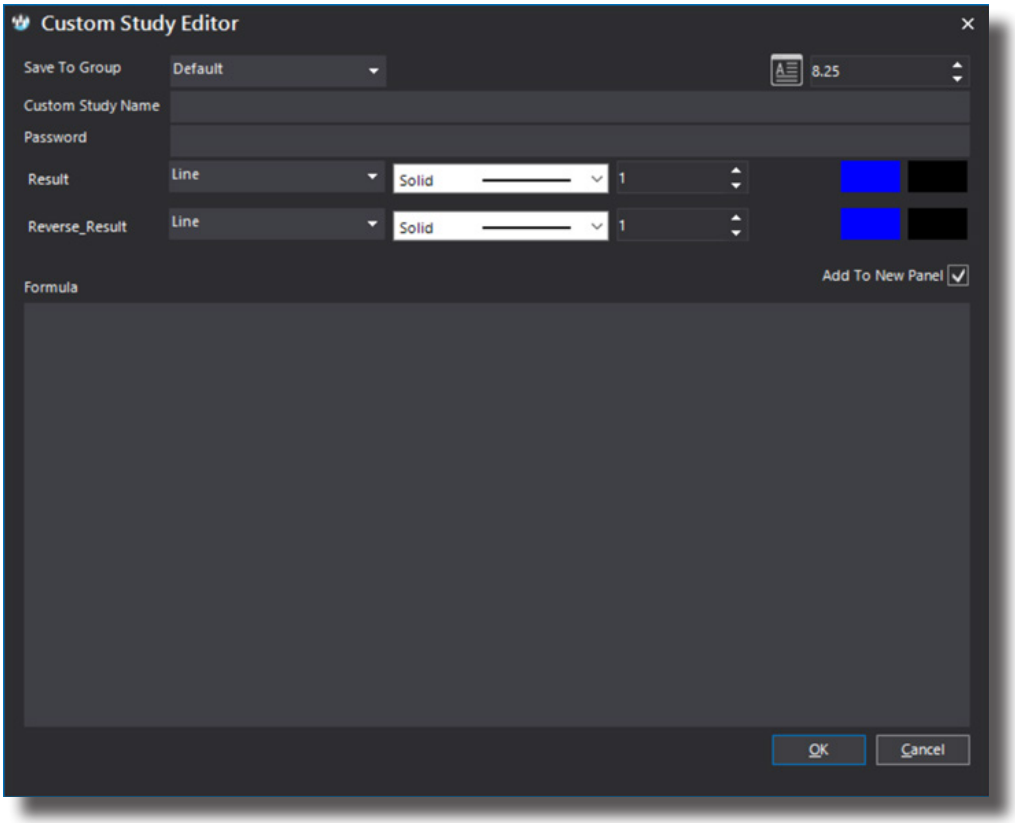
Special Thanks

Thanks to all the users and testers of ***QuantScriptTM***, whose suggestions have made it a much better programming language than it would have otherwise been.

Contents

HOW THIS GUIDE IS ORGANIZED

Chapters 1 through 4 of this guide contain references to the core functions used in performing common, basic tasks such as identifying securities within a specific price range, increasing in volatility, crossing over an indicator and so forth. You can cut and paste many of these examples right into the ***Quant Script*** code editor in your software.



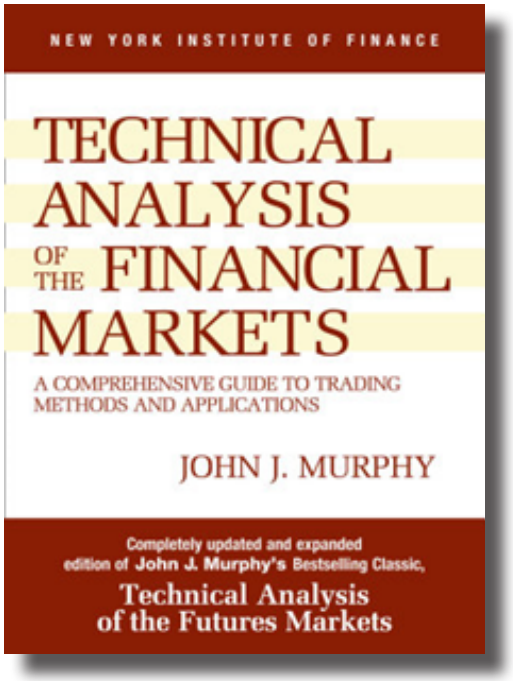
Chapters 5 through 13 of this guide contain references to functions, properties, and constants supported by the ***Quant Script*** language as well as hands-on trading system examples. Many of the functions available in ***Quant Script*** are also built in the S-Trader chart component and have detailed spec sheets explaining all parameters, formulas, long and short-form syntaxes. We also provide samples of the code editor, code wizard and chart component dialogs.

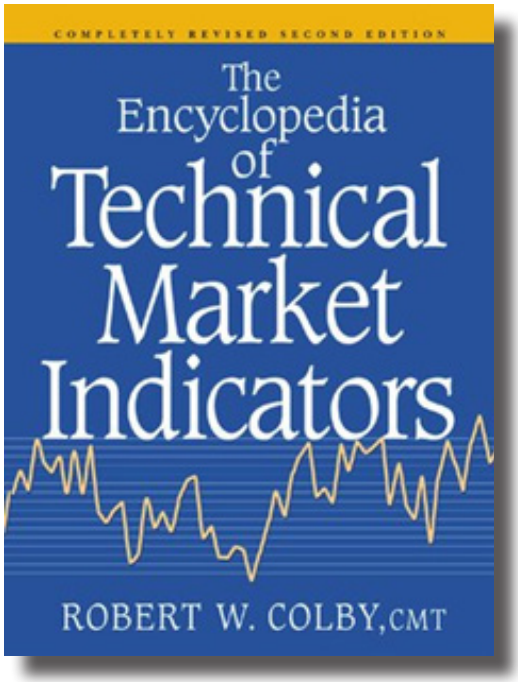
This method of organization allows the beginner programmer to see results immediately while learning at his or her own pace.

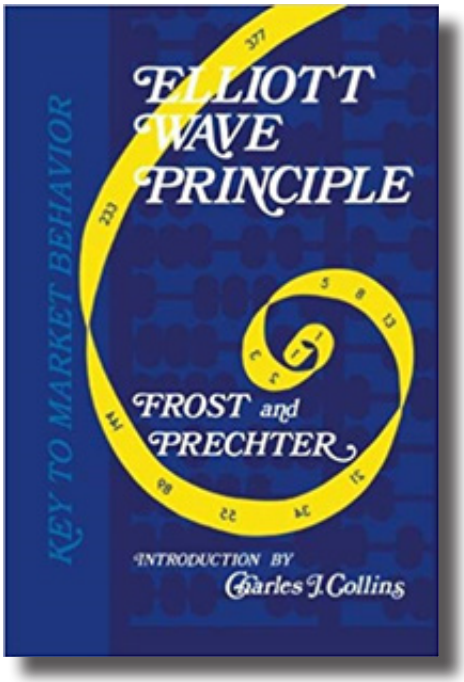
PREREQUISITES

A basic understanding of technical analysis is the only pre-requisite for using this programming guide. For a thorough understanding of how to use previously defined scripts with modules of the S-trader application please consult the S-Trader application manual or the S-Trader website.

Recommended readings



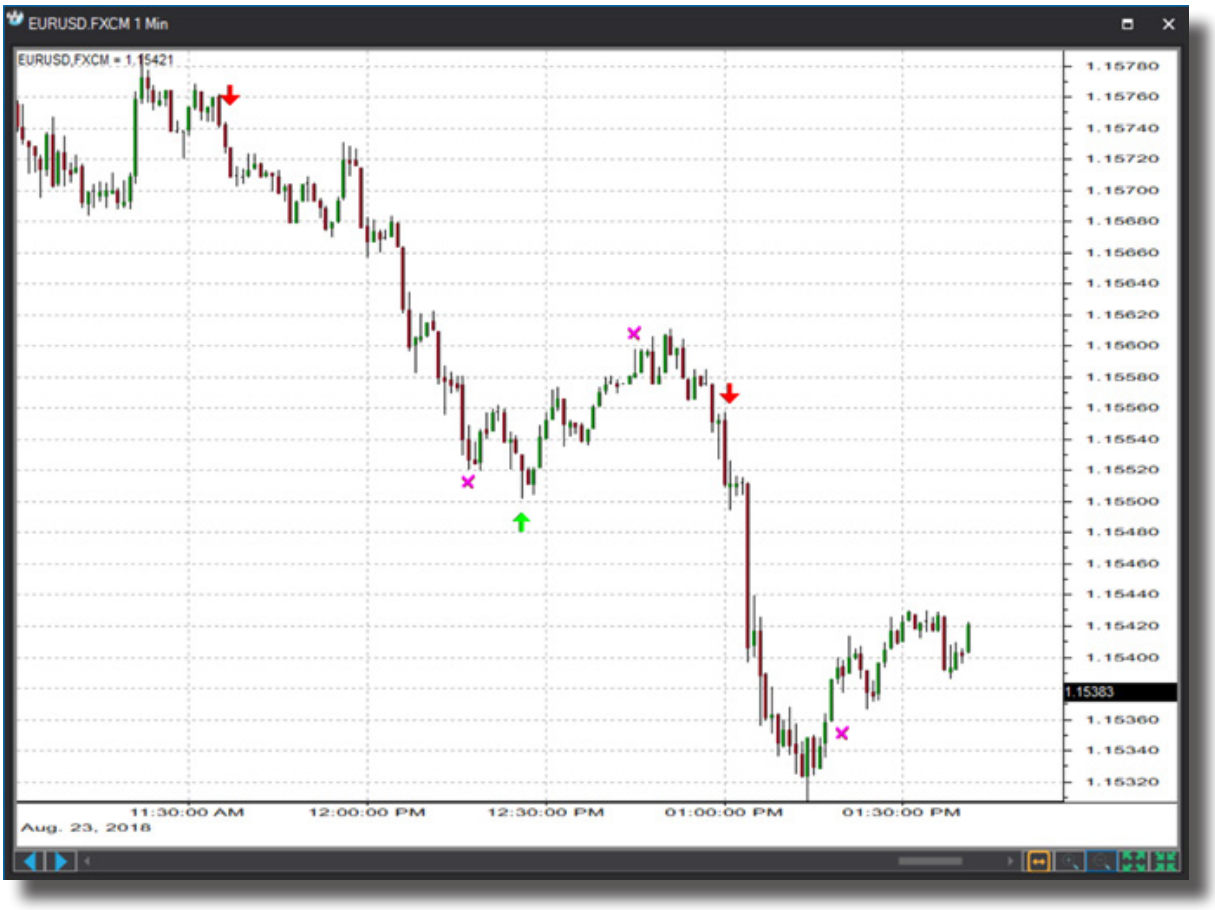




The Quant Script Programming Language

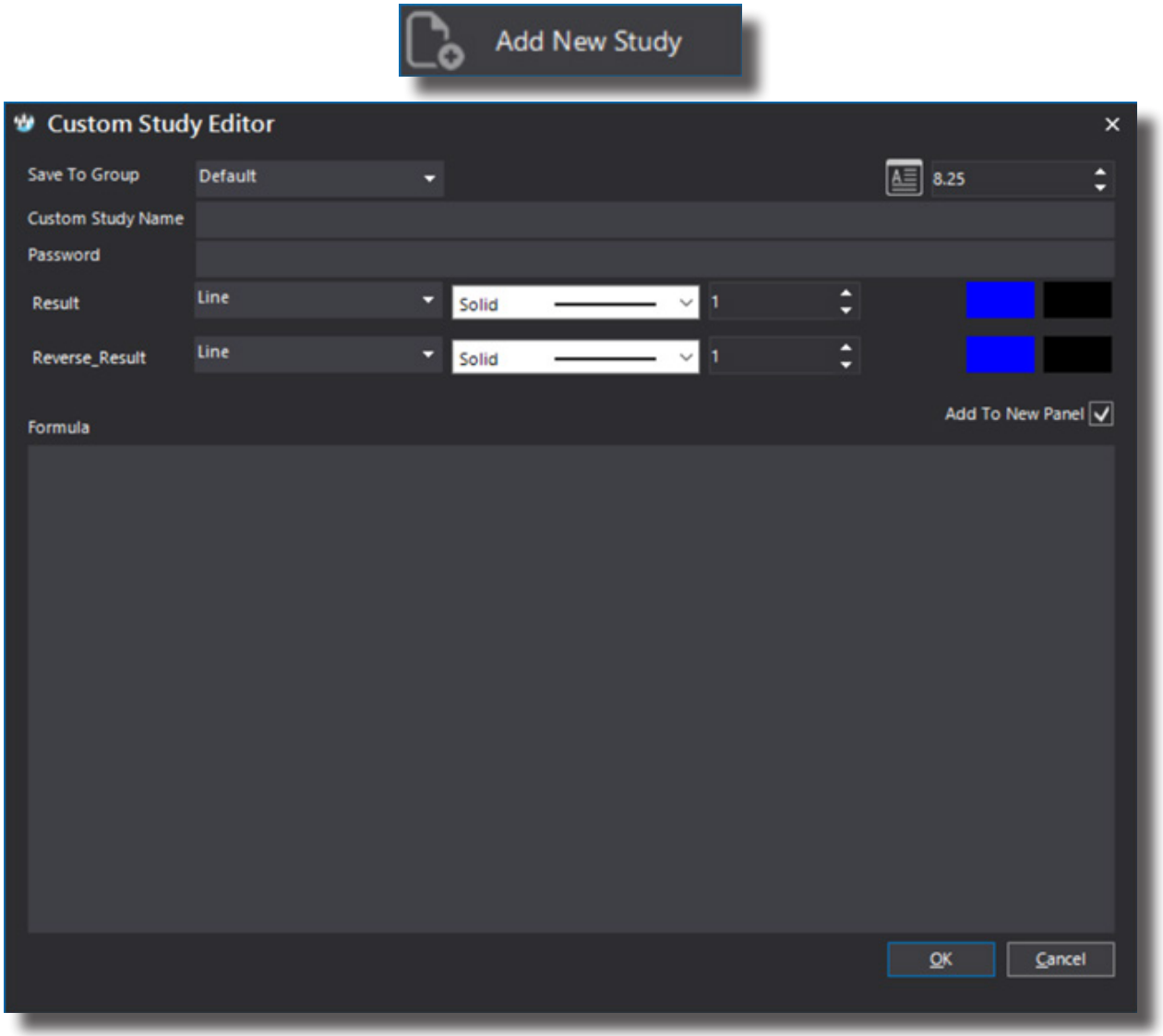
Introduction

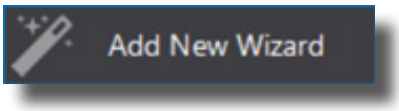
Quant Script is the engine that drives the scripting language in your S-Trader trading software. It is a non-procedural scientific vector programming language that was designed specifically for developing trading systems. A script is simply a set of instructions that tell the ***Quant Script*** engine to do something you deem useful, such as provide an alert when the price of one stock reaches a new high, crosses over a moving average, or drops by a certain percentage. There are many uses.

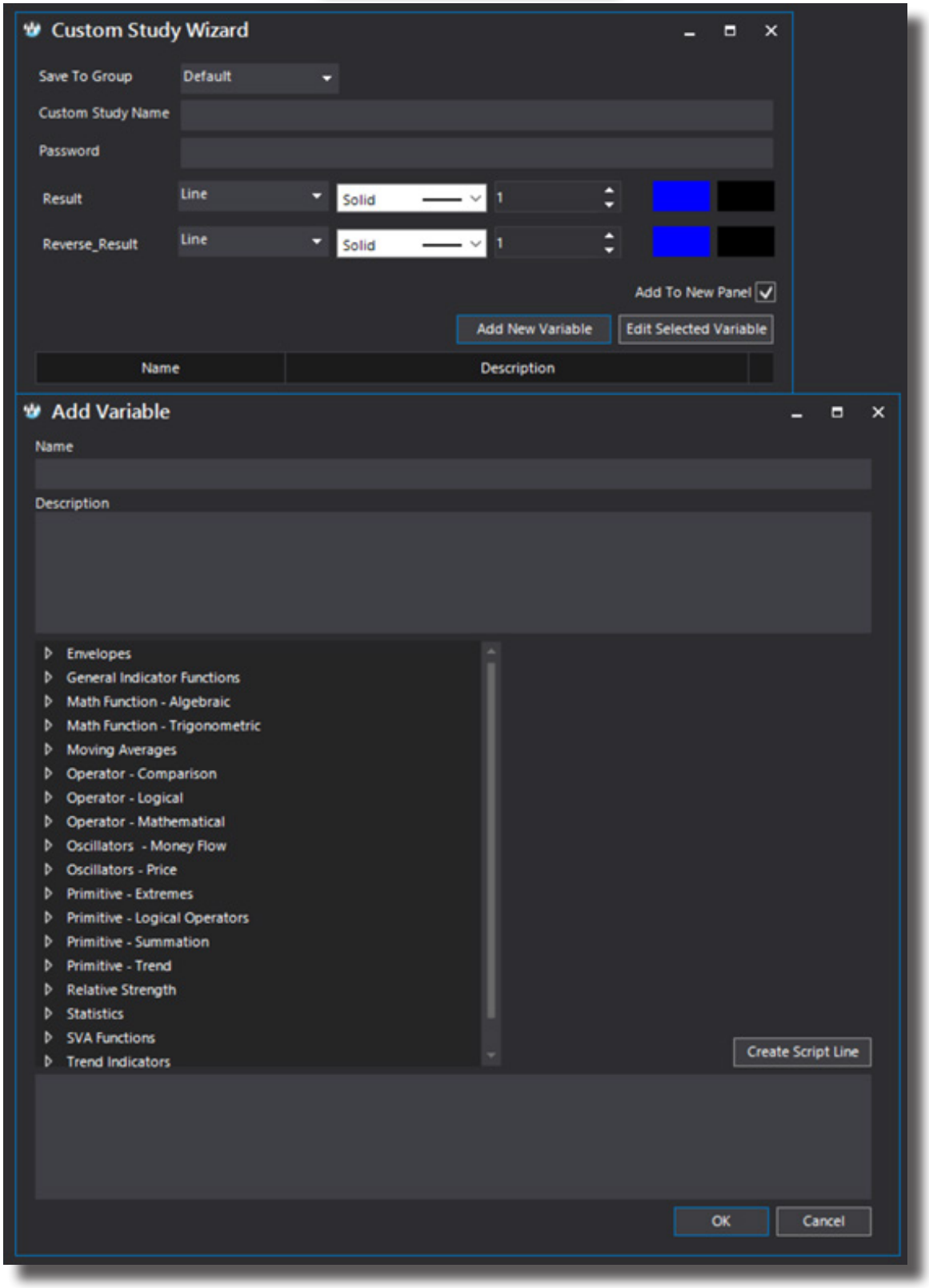


Important Concepts

Quant Script is a powerful and versatile programming language for traders. The language provides the framework required to build sophisticated trading programs piece by piece without extensive training or programming experience. Not only can the ***Quant Script*** syntax be mastered with almost no learning curve, the S-Trader application also allows you to create scripts using a versatile ***Quant Script*** Wizard.




Add New Wizard


Custom Study Wizard
Save To Group
Default
Custom Study Name
Password
Result
Line
Solid
1
Reverse_Result
Line
Solid
1
Add To New Panel
Add New Variable
Edit Selected Variable
Name
Description
Add Variable
Name
Description
Envelopes
General Indicator Functions
Math Function - Algebraic
Math Function - Trigonometric
Moving Averages
Operator - Comparison
Operator - Logical
Operator - Mathematical
Oscillators - Money Flow
Oscillators - Price
Primitive - Extremes
Primitive - Logical Operators
Primitive - Summation
Primitive - Trend
Relative Strength
Statistics
SVA Functions
Trend Indicators
Create Script Line
OK
Cancel

The following script is a very simple example that identifies markets that are trading higher than the opening price:

LAST > OPEN

It almost goes without saying that the purpose of this script is to identify when the last price is trading higher than the open price. It is nearly as plain as English.

Just as a spoken language gives you many ways to express each idea, the ***Quant Script*** programming language provides a wide variety of ways to program a trading system. Scripts can be very simple as just shown in the example above or extremely complex, consisting of many hundreds of lines of instructions. For most systems, individual scripts usually consist of just a few lines of code. Scripts can be further combined and utilized in unique ways using the existing S-Trader infrastructure and modules to create the most powerful trading systems or trade supporting elements you can imagine.

The examples outlined in the first section of this guide are relatively short and simple but provide a solid foundation for the development of more complex scripts.

Boolean Logic

The scripts shown in this first section may be linked together using Boolean logic just by adding the **AND** or the **OR** keyword. For example :

Script 1 evaluates to **TRUE** when the last price is higher than the open price:

```
LAST > OPEN
```

Script 2 evaluates to **TRUE** when volume is two times the previous period's volume:

```
VOLUME >REF(VOLUME, 1) * 2
```

You can aggregate scripts so that your script returns results for securities that are higher than the open and with the volume two times the previous volume:

```
LAST > OPEN AND VOLUME >REF(VOLUME, 1) * 2
```

Likewise, you can change the AND into an OR to find securities that are either trading higher than the open or have a volume two times the previous volume:

```
LAST > OPEN OR VOLUME >REF(VOLUME, 1) * 2
```

Once again, the instructions are nearly as plain as the English language. The use of Boolean logic with the **AND** and **OR** keywords is a very important concept that is used extensively by the ***Quant Script*** programming language.

Program Structure

It does not matter if your code is all on a single line or on multiple lines. It is often easier to read a script where the code is broken into multiple lines. The following script will work exactly as the previous example, but is somewhat easier to read:

```
LAST > OPEN OR
VOLUME >REF(VOLUME, 1) * 2
```

It is good practice to structure your scripts to make them as intuitive as possible for future reference. In some cases it may be useful to add comments to a very complex script. A comment is used to include explanatory remarks in a script.

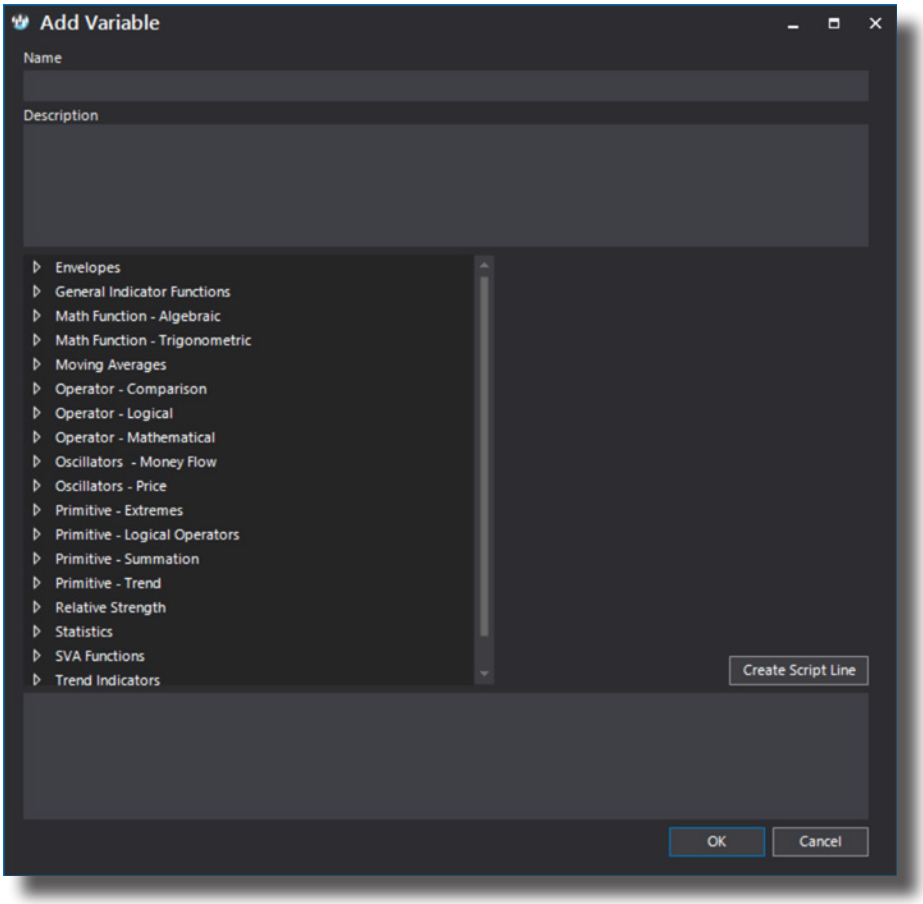
Whenever the pound sign is placed at the beginning of a line, the script will ignore the words that follow. The words will only serve as a comment or note to make the script more understandable:

```
# Evaluates to true when the last price is higher than the open or
LAST > OPEN OR

# the volume is 2 X's the previous volume:
VOLUME >REF(VOLUME, 1) * 2
```

The script runs just as it did before with the only difference being that you can more easily understand the design and purpose of the script.

It may not be a bad idea to maintain your scripts in TXT files managed by a known code editor such as NotePad++ or Crimson Editor and then copy the content inside S-Trader custom study files. Click on the logos below to download the installers.

Functions



The **Quant Script** language provides many built-in functions that make programming easier. When functions are built into the core of a programming language they are referred to as primitives. The TREND function is one example:

TREND(CLOSE, 30) = UP In this example, the TREND function tells **Quant Script** to identify trades where the closing price is in a 30-day uptrend.

The values that are contained inside a function (such as the **REF** function or the **TREND** function) are called arguments. For instance, there are two ***arguments*** in the TREND function.
Argument #1 is the **closing price**, and argument #2 is **30**, as in "30 days" or "30 periods".

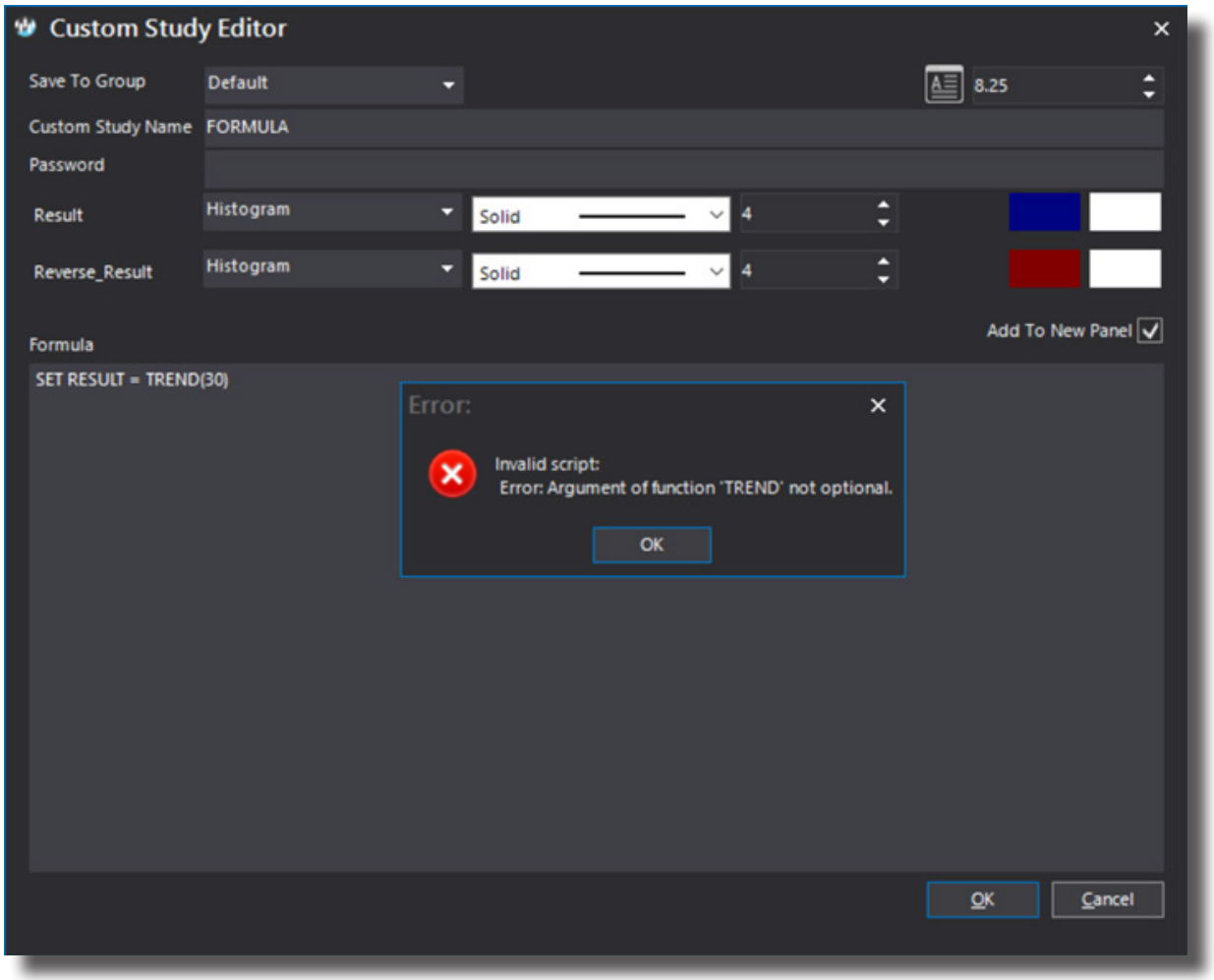
Only one of two things will occur if you use a function incorrectly:
- ***Quant Script*** will automatically fix the problem and the script will still run, or;
- ***Quant Script*** will report an error, telling you what's wrong with the script and allowing you to fix the problem and try again.

In other words, user input errors will never cause ***QuantScript*** to break or return erroneous results without first warning you about a potential problem.

Let's take **CLOSE** out of the TREND function and then try to run the script again:

SET RESULT = TREND(30)

The following error occurs:
Error: argument of **‹TREND›** function not optional.



We are given the option to fix the script and try again.

Vector Programming

Vector programming languages (also known as array or multidimensional languages) generalize operations on scalars to apply transparently to vectors, matrices, and higher dimensional arrays.

The fundamental idea behind vector programming is that operations apply at once to an entire set of values (a vector or field). This allows you to think and operate on whole aggregates of data, without having to resort to explicit loops of individual scalar operations.

As an example, to calculate a simple moving average based on the median price of a stock over 30 days, in a traditional programming language such as BASIC you would be required to write a program similar to this:

```
For each symbol
For bar = 30 to max
Average = 0
For n = bar - 30 to bar
median = (CLOSE + OPEN) / 2
Average = Average + median
Next
MedianAverages(bar) = Average / 30 Next bar
Next symbol
```

Nine to ten lines of code would be required to create the "MedianAverages" vector. But with ***Quant Script***, you can effectively accomplish the same thing using only one line:

```
SET MedianAverage = SimpleMovingAverage((CLOSE + OPEN) / 2, 30)
```

And now MedianAverage is actually a new vector that contains the 30-period simple moving average of the median price of the stock at each point. It is not uncommon to find array programming language "one-liners" that require more than a couple of pages of BASIC, Java or C++ code. **QuantScript thus allows traders to put their ideas at work and test them or run them with much less effort.**

The REF Function

At this point you may be wondering what "**REF**" and "**TREND**" are. These are two of the very useful ***primitives*** that are built into the ***Quant Script*** language.

The REF function is used whenever you want to reference a value at any specific point in a vector. Assume the MedianAverage vector contains the average median price of a stock. In order to access a particular element in the vector using a traditional programming language, you would write:

SET A = MedianAverage[n]

Using ***Quant Script*** you would write:

SET A = REF(MedianAverage, n)

The main difference other than a variation in syntax is that traditional languages reference the points in a vector starting from the beginning, or 0 if the vectors are zero-based. Quant Script on the other hand references values backwards, from the end. This is most convenient since the purpose of ***Quant Script*** is of course, to develop trading systems. It is always the last, most recent value that is of most importance. To get the most recent value in the MedianAverage vector we could write:

SET A = REF(MedianAverage, 0)
This is the same as not using the REF function at all. Therefore the preferred way to get the last value (the most recent value) in a vector is to simply write:

SET A = MedianAverage

The last value of a vector is always assumed when the REF function is absent. To get the value as of one bar ago, we would write:

SET A = REF(MedianAverage, 1)

Or two bars ago: **SET A = REF(MedianAverage, 2)**

The TREND Function

Traders often refer to "trending" as a state when the price of a stock has been increasing (up-trending) or decreasing (down-trending) for several days, weeks, months, or years. The typical investor or trader would avoid opening a new long position of a stock that has been in a downtrend for many months. ***Quant Script*** provides a primitive function aptly named **TREND** especially for detecting trends in stock price, volume, or indicators:

```
TREND(CLOSE, 30) = UP
```

This tells ***Quant Script*** to identify trades where the closing price is in a 30-period uptrend. Similarly, you could also use the TREND function to find trends in volume or technical indicators:

```
# the volume has been in a downtrend for at least 10 periods:
TREND(VOLUME, 10) = DOWN

#the 14-day CMO indicator has been up-trending for at least 20 days:
TREND(CMO(CLOSE, 14), 20) = UP
```

It is useful to use the **TREND** function for confirming a trading system signal. Suppose we have a trading system that buys when the close price crosses above a 20-day Simple Moving Average. The script may look similar to this:

```
# Gives a buy signal when the close price crosses above the 20-day SMA
CROSSOVER(CLOSE, SimpleMovingAverage(CLOSE, 20)) = TRUE
```

It would be helpful in this case to narrow the script down to only the securities that have been in a general downtrend for some time. We can add the following line of code to achieve this:

```
AND TREND(CLOSE, 40) = DOWN
```

TREND tells us if a vector has been trending upwards, downwards, or sideways, but does not tell us the degree of which it has been trending. We can use the REF function in order to determine the range in which the data has been trending. To find the change from the most current price and the price 40 bars ago, we could write:

```
SET A = LAST - REF(CLOSE, 40)
```

Price Gaps and Volatility

Although the **TREND** function can be used for identifying trends and the **REF** function can be used for determining the degree to which a stock has moved, it is often very useful to identify gaps in prices and extreme volume changes, which may be early indications of a change in trend. We can achieve this by writing:

```
# Returns true when the price has gapped up
LOW >REF(HIGH, 1)
```

Or:

```
# Returns true when the price has gapped down
HIGH <REF(LOW, 1)
```

You can further specify a minimum percentage for the price gap:

```
# Returns true when the price has gapped up at least 1%
LOW >REF(HIGH, 1) * 1.01
```

And with a slight variation we can also check whether the volume is either up or down by a large margin:

```
# the volume is up 1000%
VOLUME >REF(VOLUME, 1) * 10
```

Or by the average volume:

```
# the volume is up 1000% over average volume
VOLUME >SimpleMovingAverage(VOLUME, 30) * 10
```

We can also measure volatility in price or volume by using any one of the built-in technical indicators such as the Volume Oscillator, Chaikin Volatility Index, Coefficient of Determination, Price Rate of Change, Historical Volatility Index, etc. These technical indicators are described in Chapters 512-.

Equity Valuation (SVA) Functions

Unique to the S-Trader platform and the ***Quant Script*** engine are the SVA Equity Valuation functions. SVA stems from Strategic Valuation Analysis and is a suite of indicators used to determine the intrinsic value and the stability of companies based on a proprietary balance sheet analysis methodology.

```
SVA_function(line acronym) / FMV_function() / SR_function()
```

```
# the Close Price is between Normal and High Mid breakpoints
CLOSE >= SVA_function({N}) AND CLOSE<=SVA_function({HM})

# the Stability Ratio has been trending UP
TREND(SR_function(),30) = UP

# the Close is below the Fair Market Value
Close < FMV_function()
```

Technical Analysis

Quant Script provides many built-in technical analysis functions. Using only a single line of code you can calculate functions such as Moving Averages, Bollinger Bands, Directional Movement Index or validate Japanese Candlestick patterns. A complete list of technical analysis functions is covered in chapters 5 through 13.

The following is a simple example of how to use one of the most common technical analysis functions, the simple moving average:

```
LAST > SimpleMovingAverage(CLOSE, 20)
```

The script will return **TRUE** if the last price is over the 20-day moving average of the close price.

The CLOSE variable is actually a vector of closing prices, not just the most recent close price. You can use the OPEN, HIGH, LOW, CLOSE and VOLUME vectors to create your own calculated vectors using the SET keyword:

```
SET Median = (CLOSE + OPEN) / 2
```

This code creates a vector containing the median price for each trading day. We can then use the Median vector inside any function that requires a vector:

LAST > SimpleMovingAverage(Median, 20)

This condition will evaluate to **TRUE** when the last price is greater than a 20-day moving average of the median price.

Because functions return vectors, functions can also be used as valid arguments within other functions:

LAST > SimpleMovingAverage(SimpleMovingAverage(CLOSE, 30), 20)

This evaluates to **TRUE** when the last price is greater than the 20-day moving average of the 30-day moving average of the close price.

It is much better, however, to first initialize the functions you want to use as arguments:

SET A = SimpleMovingAverage(CLOSE, 30)
SET RESULT = LAST > SimpleMovingAverage(A, 20)

Please note that many technical indicator names are quite long, therefore function abbreviations have been conveniently provided.

Long name, i.e. long-form function: SimpleMovingAverage(CLOSE, 30)
Abbreviated name, i.e. short-form function: SMA(CLOSE, 30)

SMA is the same function as **SimpleMovingAverage** and both methods work the same way. You will note each function has a long-form and a short-form Quant Script syntax.

Crossovers

You may be familiar with the term "crossover", which is what happens when one series crosses over the top of another series as depicted in the image below. Many technical indicators such as the MACD for example, have a "signal line". A buy or sell signal is generated when the signal line crosses over or under the technical indicator.

The **CROSSOVER** function helps you identify when one series has crossed over another.
For example, we can find the exact point in time when one moving average crossed over another by using the **CROSSOVER** function:

```
SET MA1 = SimpleMovingAverage(CLOSE, 28)

SET MA2 = SimpleMovingAverage(CLOSE, 14)

CROSSOVER(MA1, MA2) = TRUE
```

The script above will evaluate to **TRUE** when the MA1 vector most recently crossed over the MA2 vector. And we can reverse the script to the MA1 vector crossed below the MA2 vector:
CROSSOVER(MA2, MA1) = TRUE

This would be equivalent to writing:

```
SET MA1 = SimpleMovingAverage(CLOSE, 28)

SET MA2 = SimpleMovingAverage(CLOSE, 14)

MA1 > MA2 AND REF(MA1,1) < REF(MA2, 1)
```

Key Reversal Script

Finally, before we move into the technical reference section of this guide let's create a script that finds **Key Reversals**, so that you can see firsthand how ***Quant Script*** can be used to create trading systems based upon complex rules. The definition of a **Key Reversal** is that after an uptrend, the open must be above the previous close, the most current bar must make a new high, and the last price must be below the previous low. Let's translate that into script form:

```
#First make sure that the stock is in an uptrend
TREND(CLOSE, 30) = UP

#The open must be above yesterday's close
AND OPEN >REF(CLOSE, 1)

#Today must be making a new high
AND HIGH >= REF(HIGH,1)

# And the last price must be below yesterday's low
AND LAST <REF(LOW, 1)
```

Ironically, the script minus comments is actually shorter than the English definition of this trading system. Key Reversals do not occur frequently but they are very reliable when they do occur. You can experiment by removing the line **AND HIGH >= REF(HIGH,1),** or you can replace it with other criteria. This script can also be reversed:

```
#First make sure that the stock is in a downtrend
TREND(CLOSE, 30) = DOWN

#The open must be below yesterday's close
AND OPEN <REF(CLOSE, 1)

#Today must be making a new low
AND LOW <= REF(LOW,1)

 # And the last price must be above yesterday's high
AND LAST >REF(HIGH, 1)
```

Again, the signal seldom occurs but is very reliable when it does.

Primitive Functions & Operators

Primitives

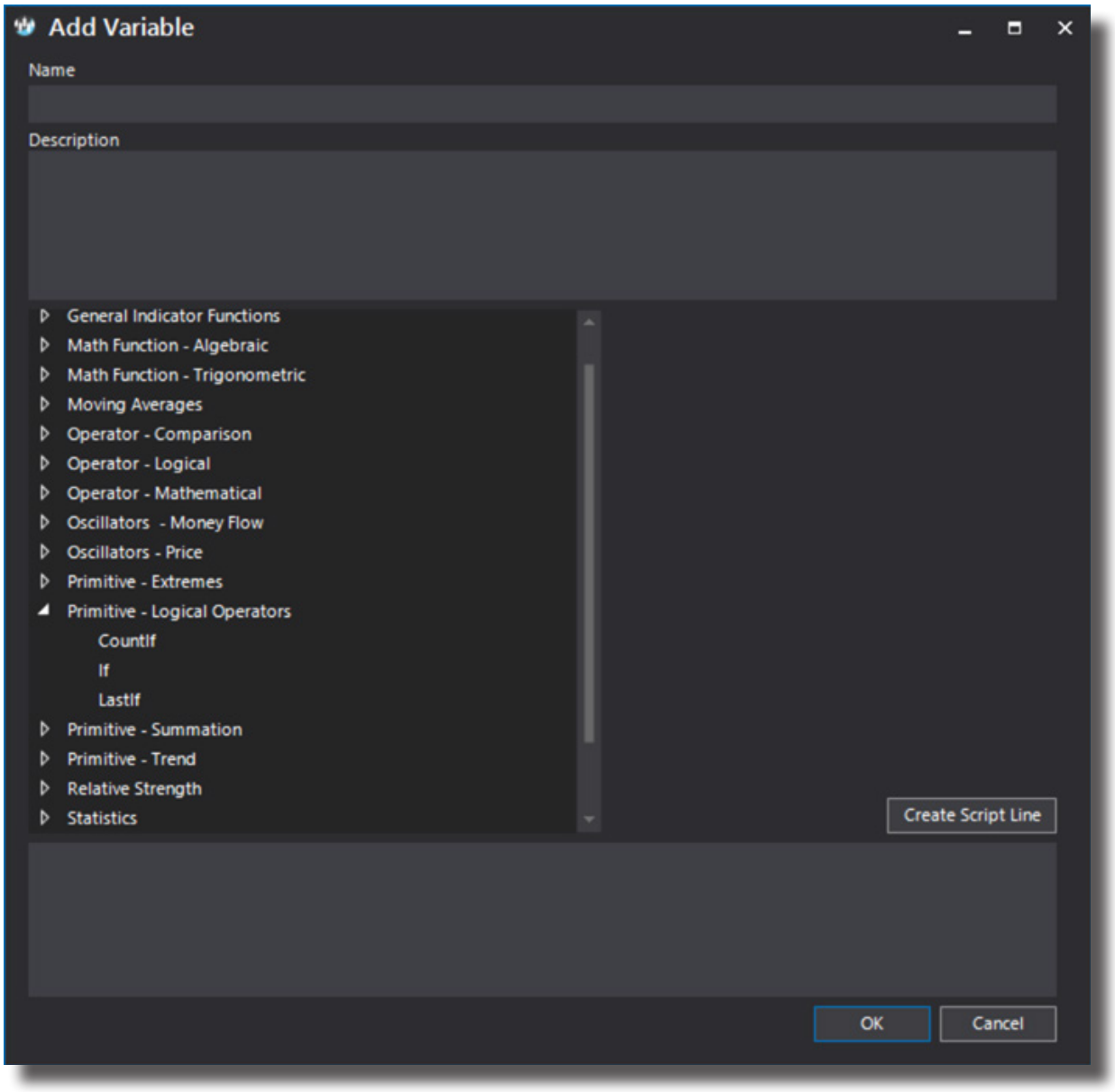
This chapter covers the core functions of ***Quant Script***, also known as primitives. These important functions define the ***Quant Script*** programming language and provide the basic framework required to build complex trading systems from the ground up.

Literally any type of trading system can be developed using the ***Quant Script*** programming language with minimal effort. If a system can be expressed in mathematical terms or programmed in any structured, procedural language such as C++, VB, or Java for example, you can rest assured that the same formulas can also be programmed using the ***Quant Script*** programming language.

Sometimes technical analysis formulas can be very complex. For example, technical analysis functions exist that require recursive calculations and complicated IF – Then - ELSE structures as part of their formula. These complex trading systems are traditionally developed in a low level programming language but most if not all of them can also be developed using ***Quant Script***. We have successfully built such scripts consisting of many hundreds of lines and have successfully ran them live on tick charts.

This chapter outlines how ***Quant Script*** can be used to perform these same calculations in a much simpler way by means of vector operations and simulated control structure.

Logical Operator Primitives



COUNTIF Function

COUNTIF(Condition)

Returns a vector representing the total number of times the specified condition evaluated to TRUE.

Example:

```
SET RESULT = COUNTIF(CROSSOVER(SimpleMovingAverage(CLOSE, 14), CLOSE))
```

The script returns a vector with increasing values expressing the number of times the 14-day Simple Moving Average crossed over the closing price.

LASTIF Function

LASTIF(Condition)

Similar to **COUNTIF**, except **LASTIF** returns a vector containing the number of periods since the last time the specified condition evaluated to **TRUE**. The count is reset to zero each time the condition evaluates to **TRUE**.

Example:

```
SET RESULT = LASTIF(CLOSE < REF(CLOSE, 1))
```

The script returns a vector that increases in value for each bar where the closing price was not less than the previous closing price. When the condition evaluates to **TRUE**, meaning the closing price was less than the previous closing price, the reference count is reset to zero.

"IF" Conditional Function

IF(Condition, True part, False part)

The conditional "IF" function allows you to design complex Boolean logic filters.

Example:

If you paste the following script into the Script area in your trading software application, you will see a column of numbers that oscillate between 1 and -1, depending on when the closing price is greater than the opening price:

```
SET A = IF(CLOSE > OPEN, 1, -1)
```

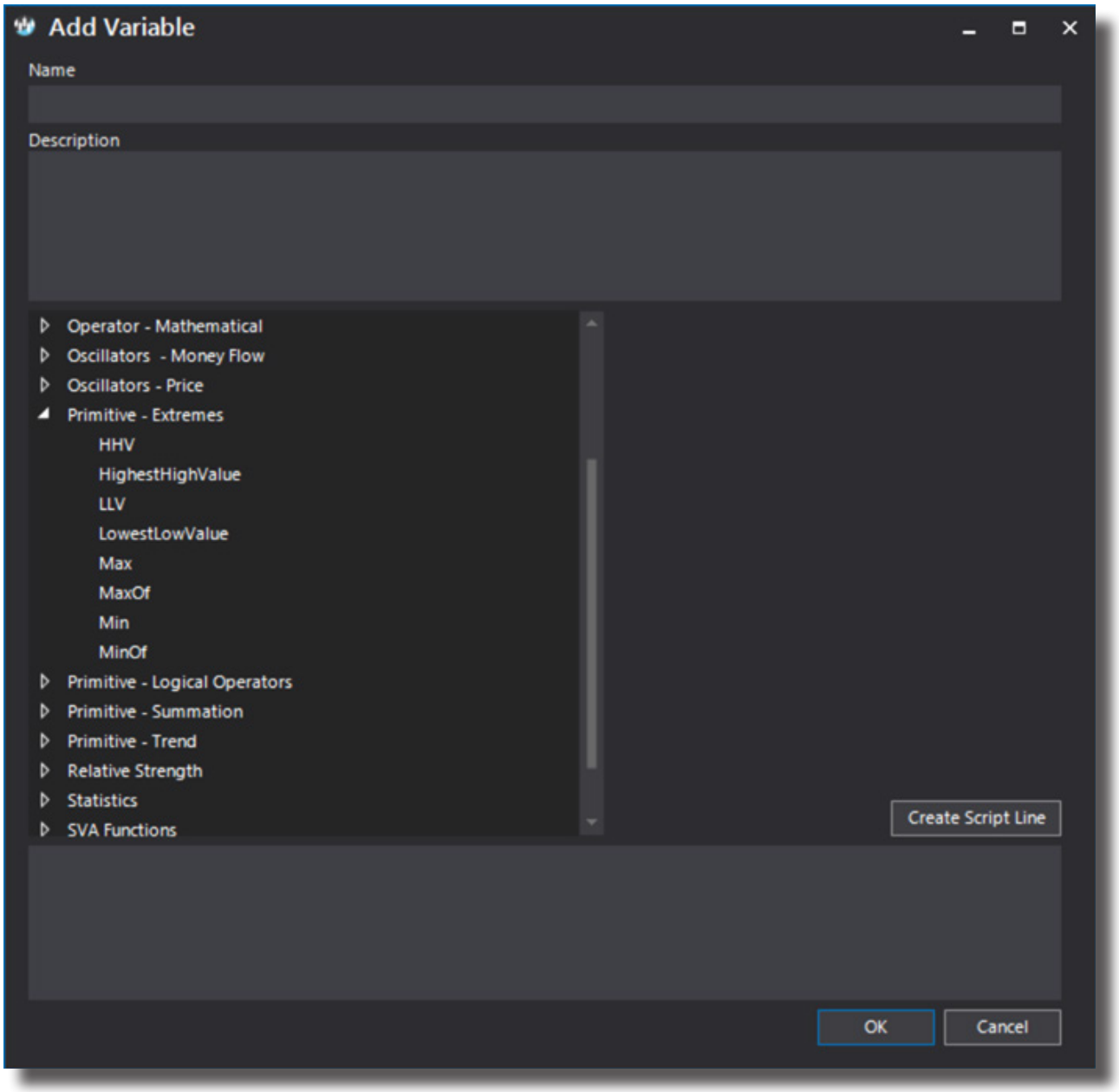
- The first argument of the "IF" function is a logical test.
- The second argument is the value that will be used if the condition evaluates to **TRUE**.
- Conversely, the third argument is the value that will be used if the condition evaluates to **FALSE**.

The logical test may be any value or expression that can be evaluated to **TRUE** or **FALSE**. For example, CLOSE = OPEN is a logical expression; if the close price is the same as the opening price, the expression evaluates to **TRUE**. Otherwise, the expression evaluates to **FALSE**.

Conditional IF functions can be nested, provided the conditions evaluated are properly initialized:

```
SET Con1 = (CLOSE>OPEN)

SET Con2 = (CLOSE=HIGH)

SET A = If(Con1, 1, If(Con2, 2, 0))
```

Extremes Primitives



HHV Function

HighestHighValue(Periods) / HHV(Periods)

This returns the highest value of the high price over the specified number of periods.

HIGH = HHV(21) evaluates to **TRUE** when the high is the highest high in the past 21 bars.

LLV Function

LowestLowValue(Periods) / LLV(Periods)

This returns the lowest value of the low price over the specified number of periods.

LOW = LLV(21) evaluates to **TRUE** when the low is the lowest low in the past 21 bars.

MAX Function

MAX(Vector, Periods)

This returns a vector containing a running maximum, as specified by the Periods argument. The values represent the maximum value for each window.

MAX(CLOSE, 10) returns a vector of maximum values based on a 10- period window.

MAXOF Function

MAXOF(Vector1, Vector2, [Vector3]...[Vector8])

This returns a vector containing a maximum value of all specified vectors, for up to eight vectors. Vector1 and Vector2 are required and vectors 3 through 8 are optional.

MAXOF(CLOSE, OPEN) returns a vector containing the maximum value for each bar, which is either the opening price or the closing price in this example.

MIN Function

MIN(Vector, Periods)

This returns a vector containing a running minimum, as specified by the Periods argument. The values represent the minimum value for each window.

MIN(CLOSE, 10) returns a vector of minimum values based on a 10- period window.

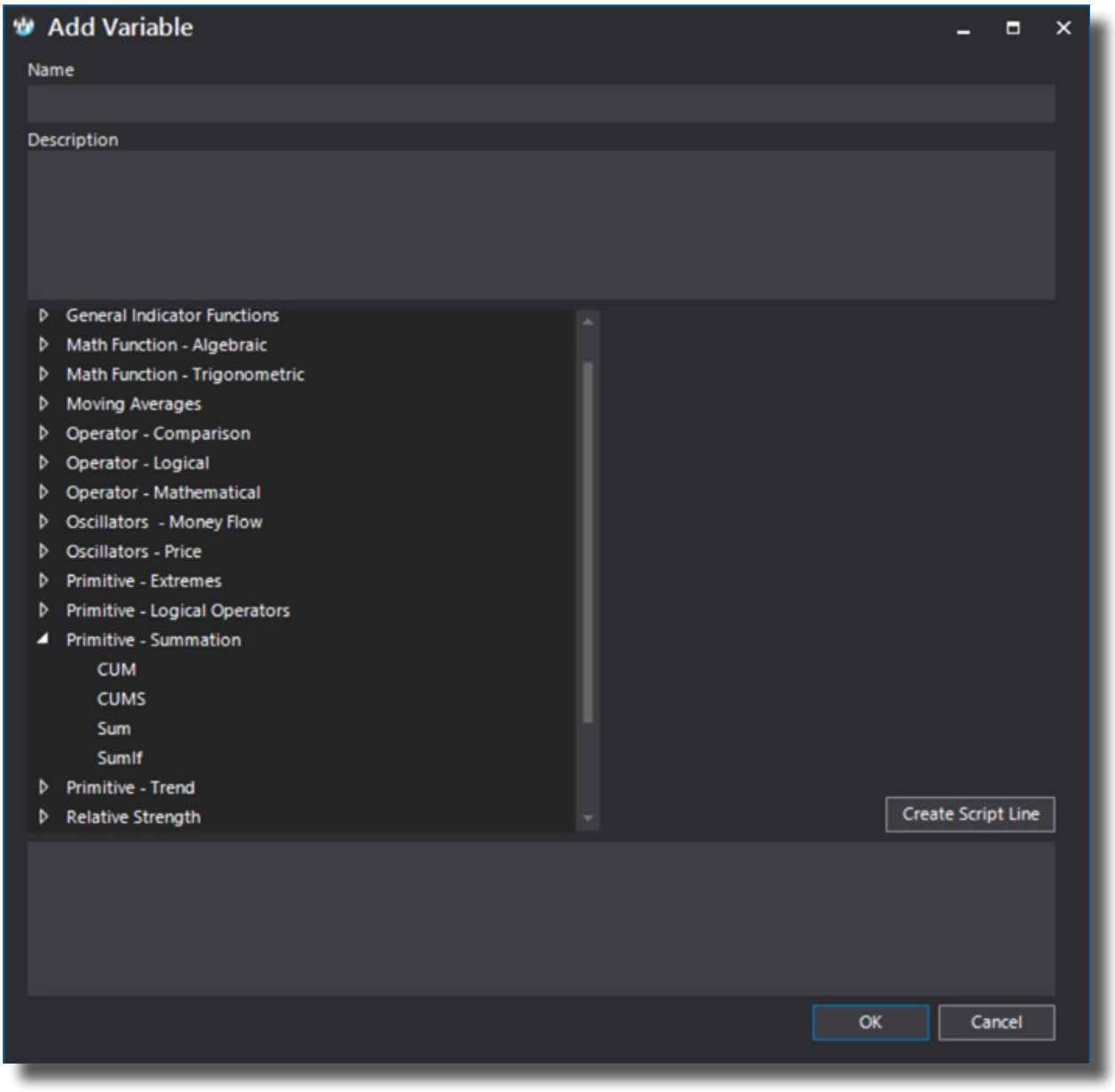
MINOF Function

MINOF(Vector1, Vector2, [Vector3]...[Vector8])

This returns a vector containing a minimum value of all specified vectors, for up to eight vectors. Vector1 and Vector2 are required and vectors 3 through 8 are optional.

MINOF(CLOSE, OPEN) returns a vector containing the minimum value for each bar, which is either the opening price or the closing price in this example.

Summation Primitives



CUM Function

CUM(Vector, Periods)

Cummulative(vector, periods)

The CUMMULATIVE function outputs a vector containing a running sum, as specified by the Periods argument.

CUM(Close, 10) outputs the running sum of the past 10 periods closing prices.

CUMS Function

CUMS(Vector)

CummulativeSeries(Vector)

The CUMMULATIVE SERIES function outputs a vector containing a running sum of the entire series

CUMS(Volume) outputs the running sum of all periods' volume.

SUM Function

SUM(Vector, Periods)

The SUM function (not to be confused with the SUMIF function) outputs a vector containing a running sum, as specified by the Periods argument.

SUM(CLOSE, 10) returns a vector of closing price sums based on a 10-period window.

SUMIF Function

SUMIF(Condition, Vector)

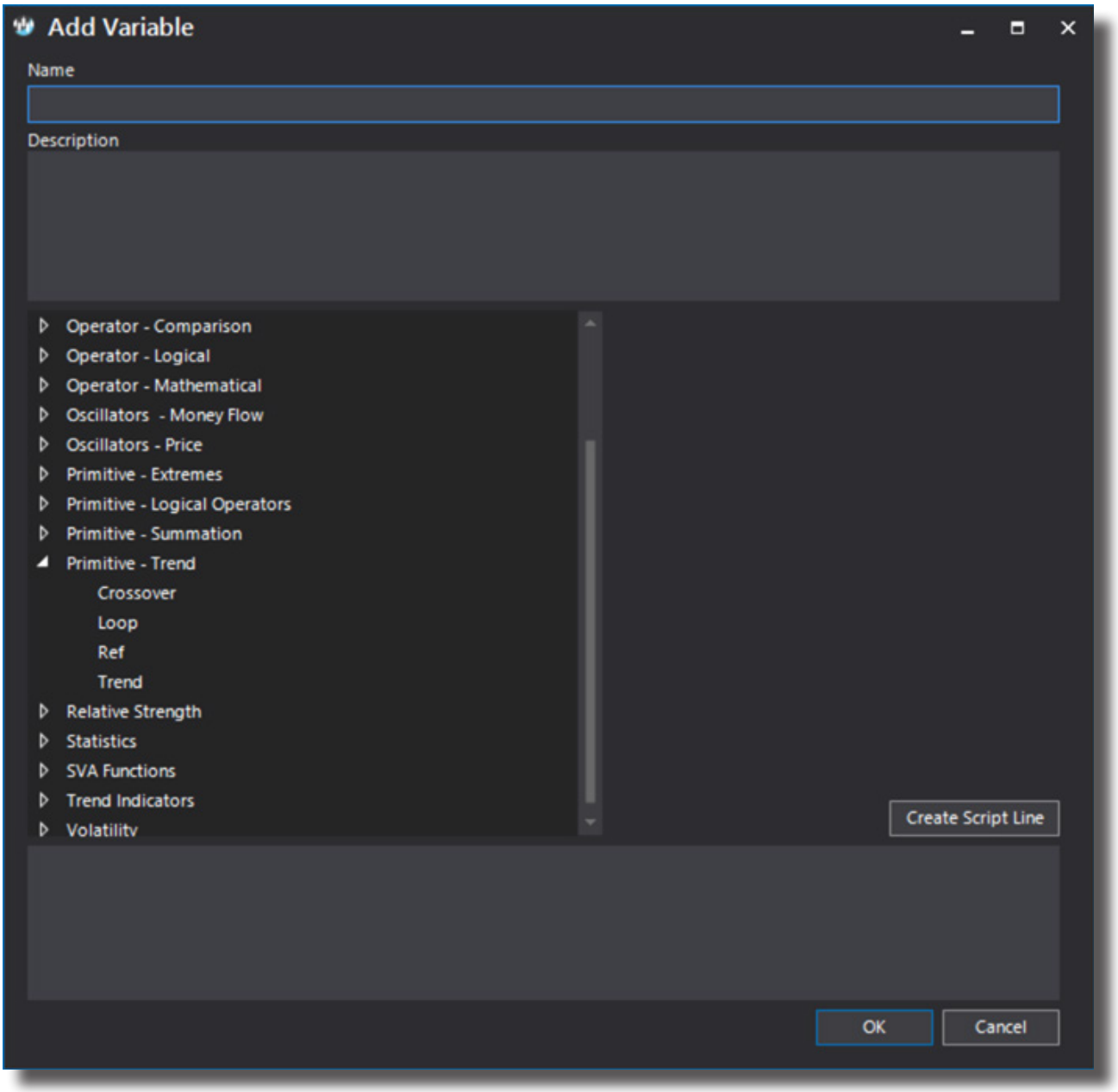
This is a hybrid between an "IF" logical operator primitive and a summation primitive. This function outputs a running sum of all values in the supplied Vector wherever the supplied Condition evaluates to **TRUE**.

For example if we wanted a vector containing the sum of volume for all periods where the closing price closed up 0.05%, we could write:

```
SET A = CLOSE > (REF(CLOSE,1)*(1.0005))

SET RESULT = SUMIF(A, VOLUME)
```

The RESULT will be a vector containing a running sum of volume for each period where the closing price closed up at least 0.05%.

Trend Primitives



CROSSOVER

CROSSOVER(Vector1, Vector2) = TRUE / FALSE

Many technical indicators such as the MACD for example, have a "signal line". Traditionally a buy or sell signal is generated when the signal line crosses over or under the technical indicator.

The CROSSOVER function helps you when one series has crossed over another. For example, we can find the exact point in time when one moving average crossed over another by using the CROSSOVER function:

```
SET MA1 = SimpleMovingAverage(CLOSE, 28)

SET MA2 = SimpleMovingAverage(CLOSE, 14)

CROSSOVER(MA1, MA2) = TRUE
```

The script above will evaluate to **TRUE** when the MA1 vector most recently crossed over the MA2 vector. And we can reverse the script to the MA1 vector crossed below the MA2 vector:

```
CROSSOVER(MA2, MA1) = TRUE
```

A different way to write this script would be:

```
SET MA1 = SimpleMovingAverage(CLOSE, 28)

SET MA2 = SimpleMovingAverage(CLOSE, 14)

MA1 > MA2 AND REF(MA1,1)<REF(MA2,1)
```

LOOP Function

LOOP(Vector1, Vector2, Offset1, Offset2, Operator)

LOOP provides simulated control structure by means of a single function call.

Vector1 is the vector to initialize the calculation from. **Offset1** is the offset where values are referenced in **Vector1** for the incremental calculation, and **Offset2** is the offset where values are referenced from in **Vector2**.

Example 1:

LOOP(Vector1, Vector2, 1, 2, Multiply) is a series that can only be calculated from index number 3 and each term results by multiplying a 1 period offset Vector 1 value with a 2 period offset Vector 2 value.

Vector 1	Vector 2	LOOP
1.25	1	
2.25	2	
3.25	3	2.25*1
4.25	4	3.25*2
5.25	5	4.25*3
6.25	6	5.25*4
7.25	7	6.25*5

REF Function

REF(Vector, Periods)

By default all calculations are performed on the last, most recent value of a vector. The following script evaluates to **TRUE** when the last open price (the current bar's open price) is less than $30:

OPEN < 30

OPEN is assumed to be the current bar's open by default. You can reference a previous value of a vector by using the REF function:

REF(OPEN, 1) < 30

And now the script will evaluate whether previous bar's open price was less than $30. The number 1 (the second argument) tells the REF function to reference values as of one bar ago. To reference values two bars ago, simply use 2 instead of 1. The valid range for the Periods argument is 1 - 250 unless otherwise noted.

TREND

TREND(Vector)

The **TREND** function can be used to determine if data is trending upwards, downwards, or sideways. This function can be used on the price (open, high, low, close), volume, or any other vector. The **TREND** function returns a constant of either UP, DOWN or SIDEWAYS.

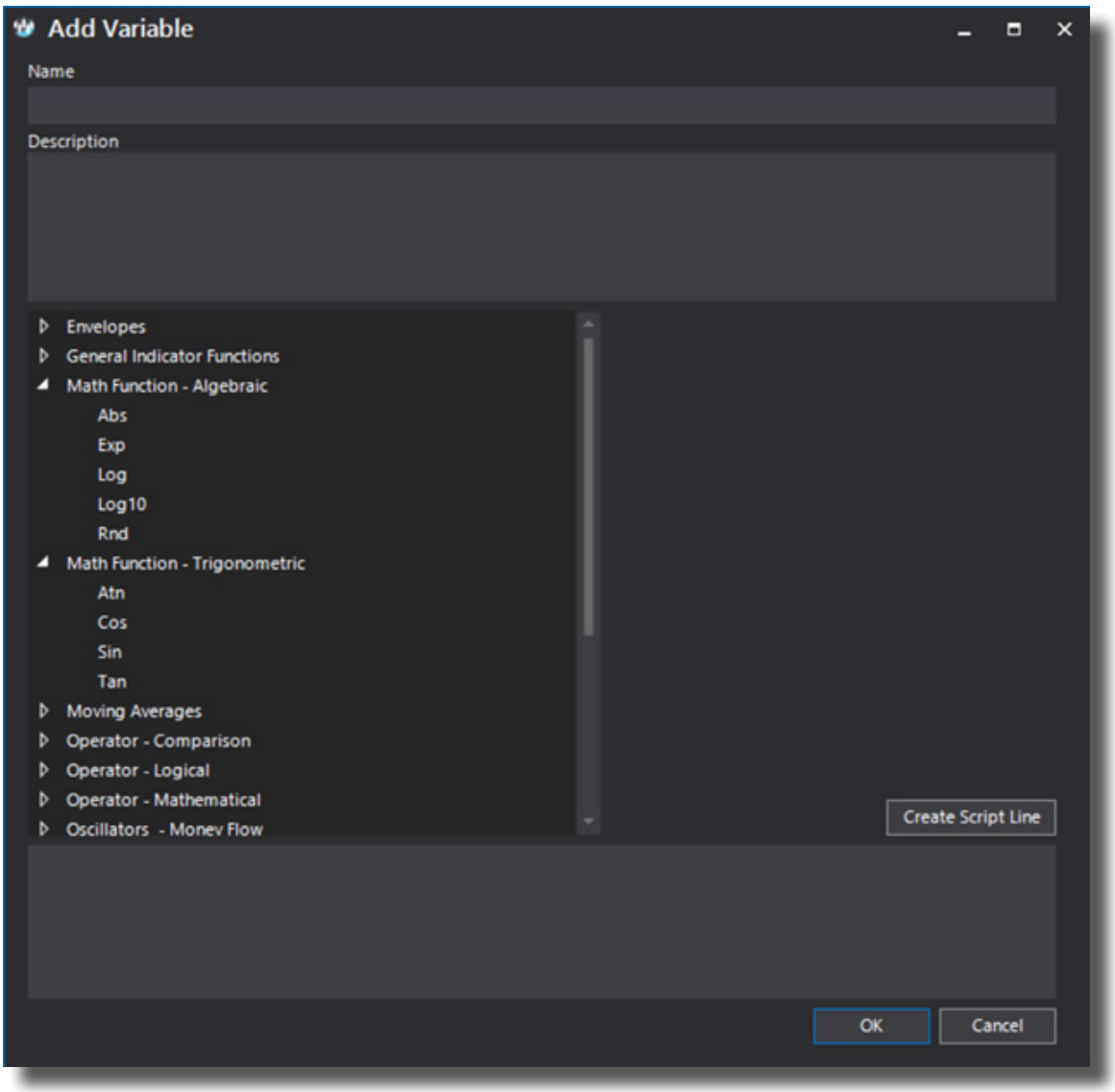
Example: **TREND(CLOSE) = UP AND TREND(VOLUME) = DOWN**

TREND is often the first function used as a means of filtering securities that are not trending in the desired direction.

Math Functions

Introduction

Note that all math functions return a vector. For example **ABS(CLOSE - OPEN)** returns a vector of the ABS value of CLOSE - OPEN (one record per bar). The RND function returns a vector of random values, one for each bar, and so forth.



Algebraic Functions

ABS – Absolute Value Function

The ABS function returns the absolute value for a number. Negative numbers become positive and positive numbers remain positive.

Example: ABS(CLOSE - OPEN). The script always evaluates to a positive number, even if the opening price is greater than the closing price.

EXP – Exponential Function

EXP raises ***e*** to the power of a number. The LOG function is the inverse of this function.

Example: EXP(3.26). The script outputs 26.28

LOG – Logarithmic Function

This returns the natural logarithm of a positive number. The EXP function is the reverse of this function. Also see LOG10.

Example: LOG(26.28). The script outputs 3.26

LOG10 – Log of 10 Function

Returns the base 10 logarithm of a positive number. Also see LOG.

Example: LOG10(26.28). The script outputs 1.42

RND – Random Function

The RND function returns a random number from 0 to a maximum value.

Example: RND(100). Outputs a random number from 0 to 100.

Trigonometric Functions

ATN – Arctangent Function

Returns the arctangent of a number.

Example: ATN(45). The script outputs 1.548

COS – Cosinus Function

COS returns the cosine for a number (angle).

Example:COS(45). The script outputs 0.525

SIN – Sinus Function

The SIN function returns the sine for a number (angle).

Example: SIN(45). The script outputs 0.851

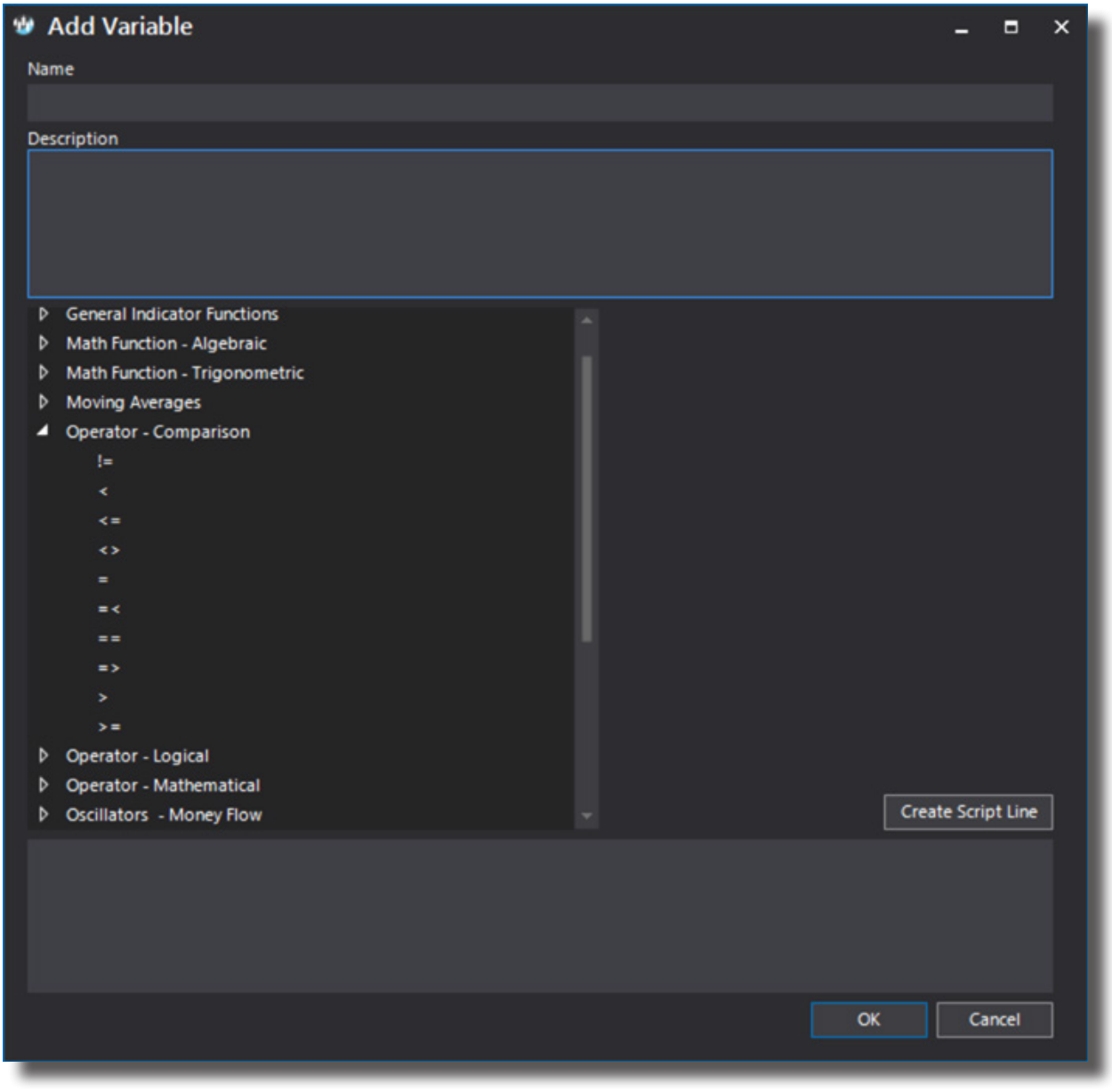
TAN – Tangent Function

The TAN function returns the tangent for a number (angle).

Example: TAN(45). The script outputs 1.619

Operators

Comparison Operators



Equal (= or ==)

The equal operator is used to assign a value to a variable or vector, or to compare values.

When used for assignment, a single variable or vector on the left side of the = operator is given the value determined by one or more variables, vectors, and/or expressions on the right side. Also, the **SET** keyword must precede the variable name when the = operator is used for an assignment:

```
SET A = 123
SET B = 123
A = B = TRUE
```

Greater Than (>)

The > operator determines if the first expression is greater-than the second expression.

Example:

```
SET A = 124
SET B = 123
A > B = TRUE
```

Less Than (<)

The < operator determines if the first expression is less-than the second expression.

Example:

```
SET A = 123
SET B = 124
A > B = TRUE
```

Greater Than Or Equal To (>= or =>)

The >= operator determines if the first expression is greater-than or equal to the second expression.

Example:

```
SET A = 123
SET B = 123
A >= B = TRUE
```

AND

```
SET A = 124
SET B = 123
A >= B = TRUE
```

Less Than Or Equal To (<= or =<)

The <= operator determines if the first expression is less-than or equal to the second expression.

Example:

```
SET A = 123
SET B = 123
A <= B = TRUE
```

AND

```
SET A = 123
SET B = 124
A <= B = TRUE
```

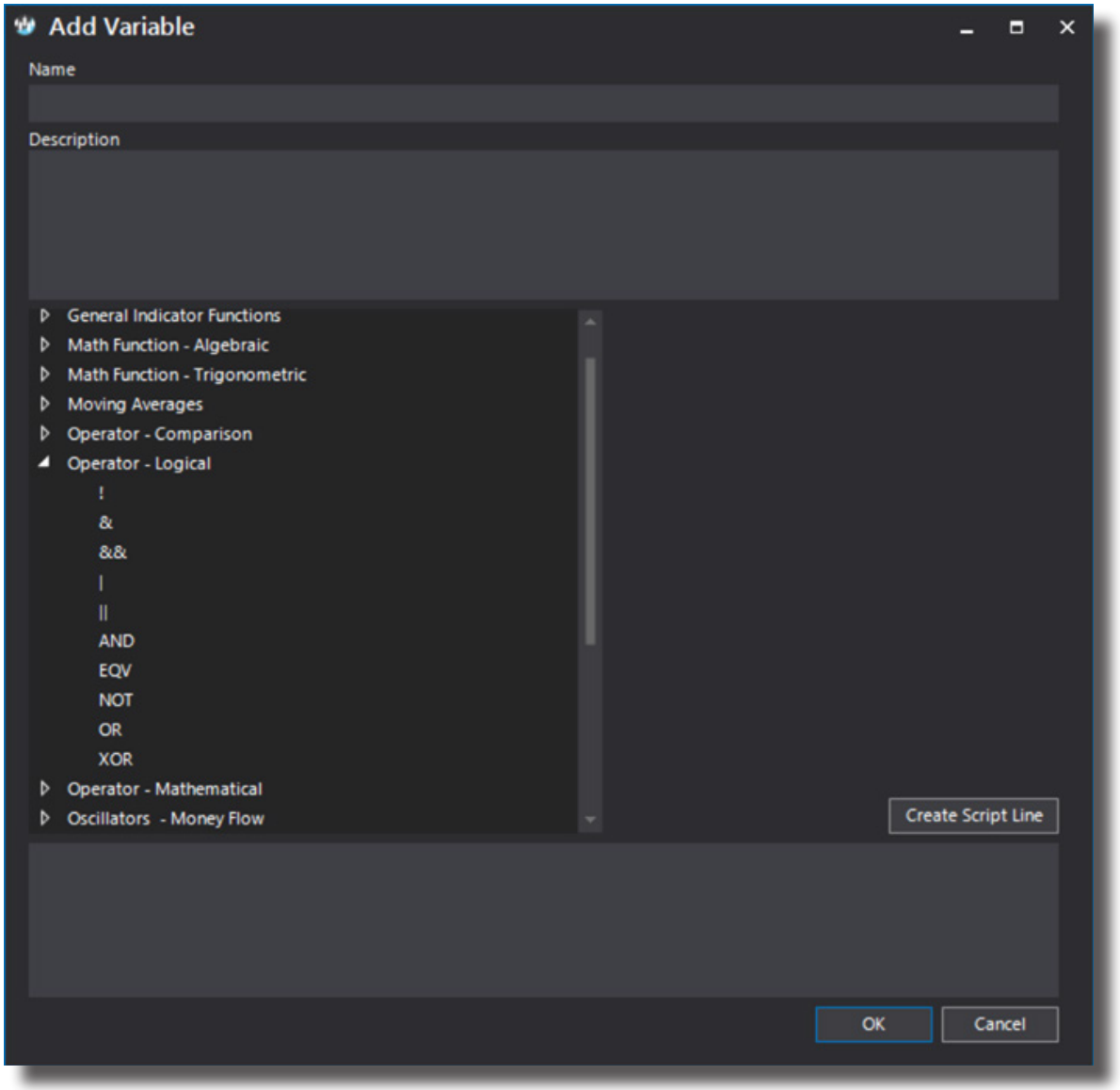
Not Equal (<> or !=)

Both the != and the <> inequality operators determine if the first expression is not equal to the second expression.

Example:

```
SET A = 123
SET B = 124
A != B = TRUE
```

Logical Operators



AND(&&)

The AND operator is used to perform a logical conjunction on two expressions, where the expressions are Null, or are of Boolean subtype and have a value of True or False.

The AND operator can also be used as a «bitwise operator» to make a bit-by-bit comparison of two integers. If both bits in the comparison are 1, then a 1 is returned. Otherwise, a 0 is returned.

When using the AND to compare Boolean expressions, the order of the expressions is not important.

Example:

(TRUE = TRUE AND FALSE = FALSE) = TRUE

AND

(TRUE = TRUE AND FALSE = TRUE) = FALSE

EQV(&)

The EQV operator is used to perform a logical comparison on two expressions (i.e., are the two expressions identical), where the expressions are Null, or are of Boolean subtype and have a value of True or False.

The EQV operator can also be used a «bitwise operator» to make a bit-by-bit comparison of two integers. If both bits in the comparison are the same (both are 0›s or 1›s), then a 1 is returned. Otherwise, a 0 is returned.

The order of the expressions in the comparison is not important.

Example:

TRUE EQV TRUE = TRUE

AND

TRUE EQV FALSE = FALSE

MOD

The MOD operator divides two numbers and returns the remainder. In the example below, 5 divides into 21, 4 times with a remainder of 1.

Example:

21 MOD 5 = 1 AND 22 MOD 5 = 2

NOT

The NOT operator is used to perform a logical negation on an expression. The expression must be of Boolean subtype and have a value of True or False. This operator causes a True expression to become False, and a False expression to become True.

Example:

NOT (TRUE = FALSE) = TRUE

AND

NOT (TRUE = TRUE) = FALSE

OR(||)

The **OR** operator is used to perform a logical disjunction on two expressions, where the expressions are Null, or are of Boolean subtype and have a value of True or False.

The **OR** operator can also be used a «bitwise operator» to make a bit-by-bit comparison of two integers. If one or both bits in the comparison are 1, then a 1 is returned. Otherwise, a 0 is returned.

When using the OR to compare Boolean expressions, the order of the expressions is important.

Example:

(TRUE = TRUE OR TRUE = FALSE) = TRUE

AND

(FALSE = TRUE OR TRUE = FALSE) = FALSE

XOR(|)

The XOR operator is used to perform a logical exclusion on two expressions, where the expressions are Null, or are of Boolean subtype and have a value of True or False.

The XOR operator can also be used a «bitwise operator» to make a bit-by-bit comparison of two integers. If both bits are the same in the comparison (both are 0›s or 1›s), then a 0 is returned. Otherwise, a 1 is returned.

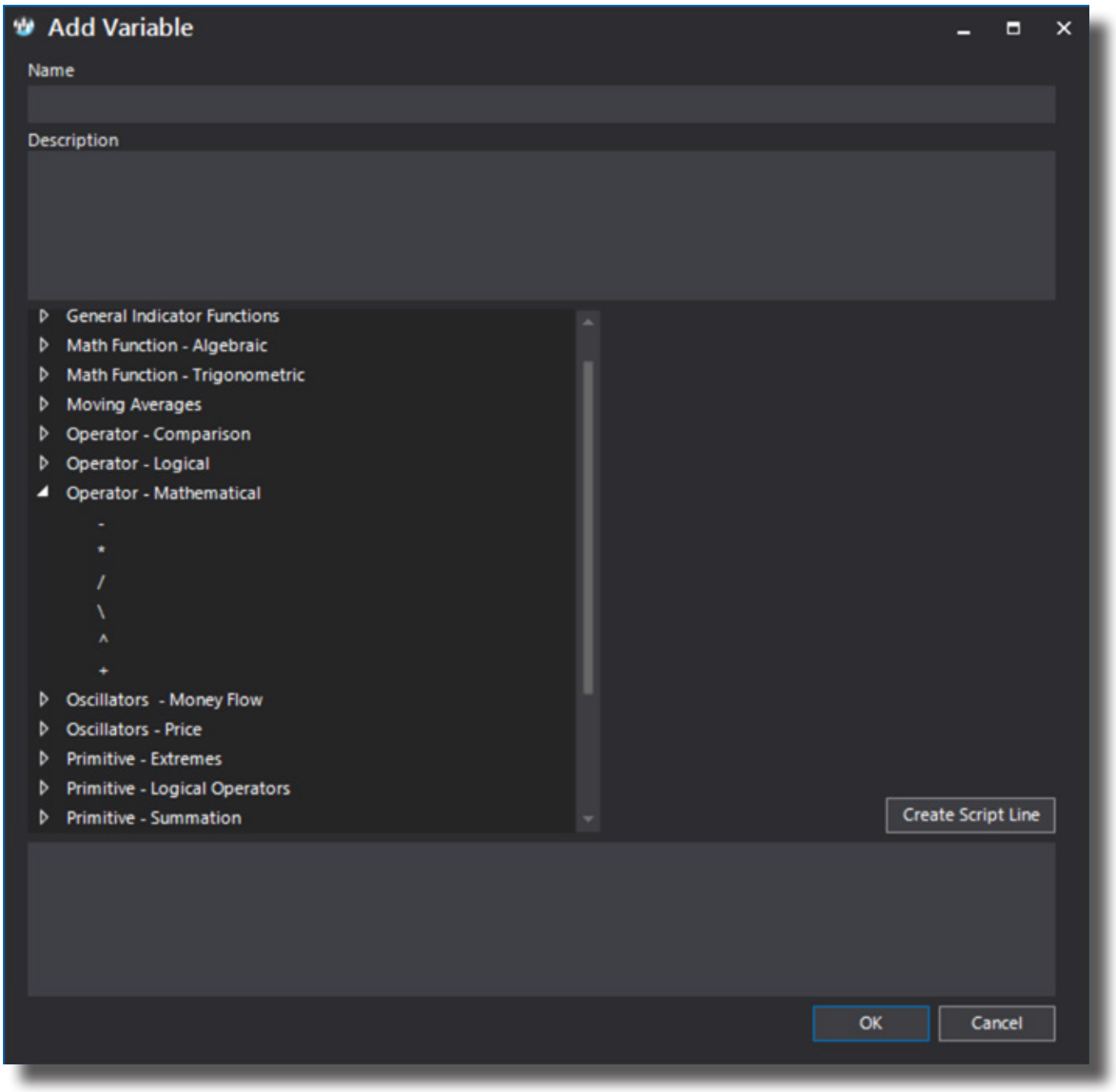
Example:

(TRUE XOR FALSE) = TRUE

AND

(FALSE XOR FALSE) = FALSE

Mathematical Operators



Addition operator (+)

Performs addition of n number of vectors: **$Vector_1$ + $Vector_2$+ .. + $Vector_n$**

Division operator (/ \ )

Performs division of n number of vectors: **$Vector_1$/ $Vector_2$/ .. + $Vector_n$**

Multiplication operator (*)

Performs multiplication of n number of vectors: **$Vector_1$* $Vector_2$* ..*$Vector_n$**

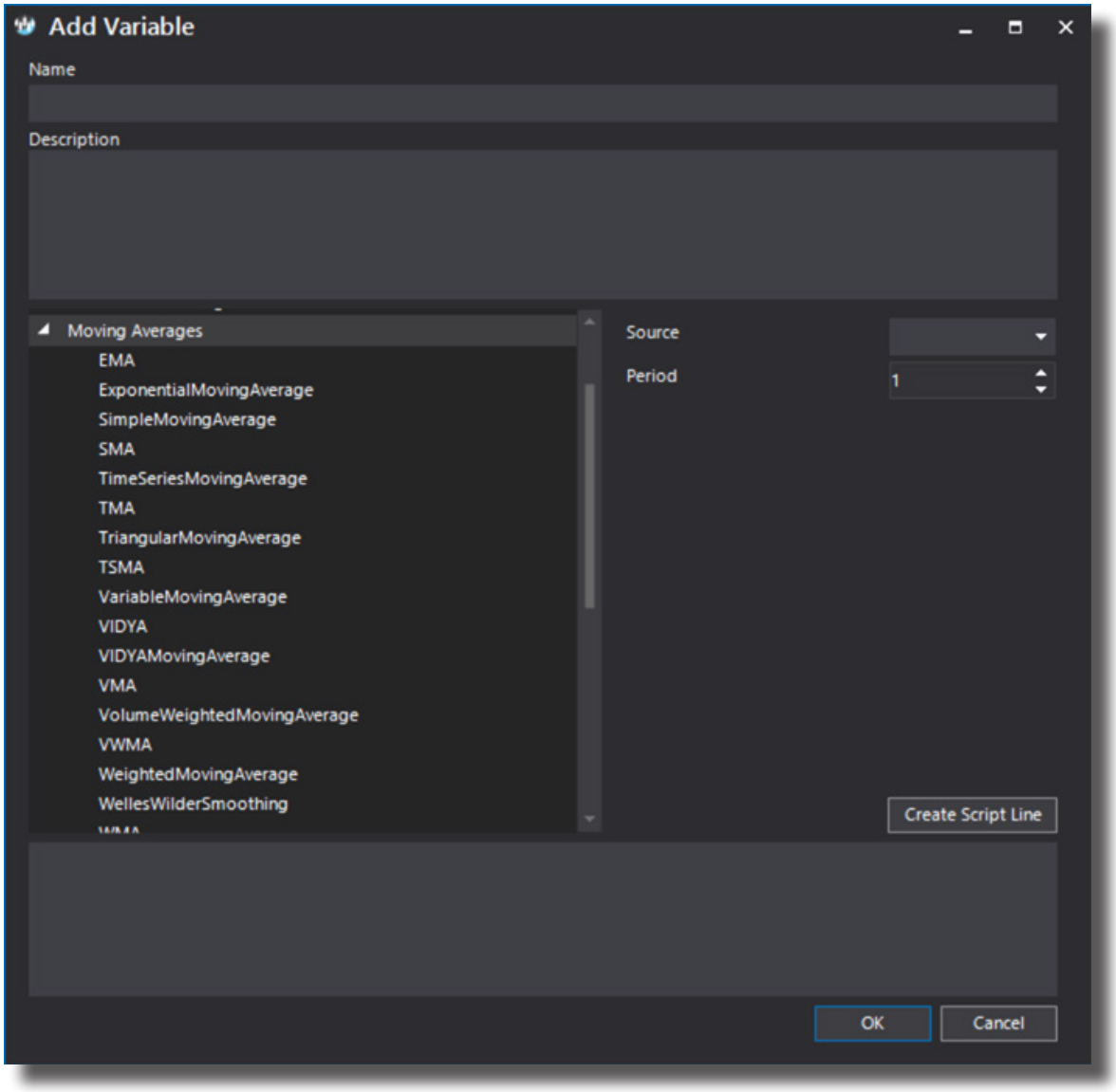
Subtraction operator (-)

Performs subtraction of n number of vectors: **$Vector_1$- $Vector_2$- ..-$Vector_n$**

Power operator(^)

Rises a number to a power: **2^3 = 2*2*2 = 8**

Moving Averages



Introduction

Moving averages are the foundation of quantitative technical analysis. These functions calculate averages or variations of averages of the underlying vector.

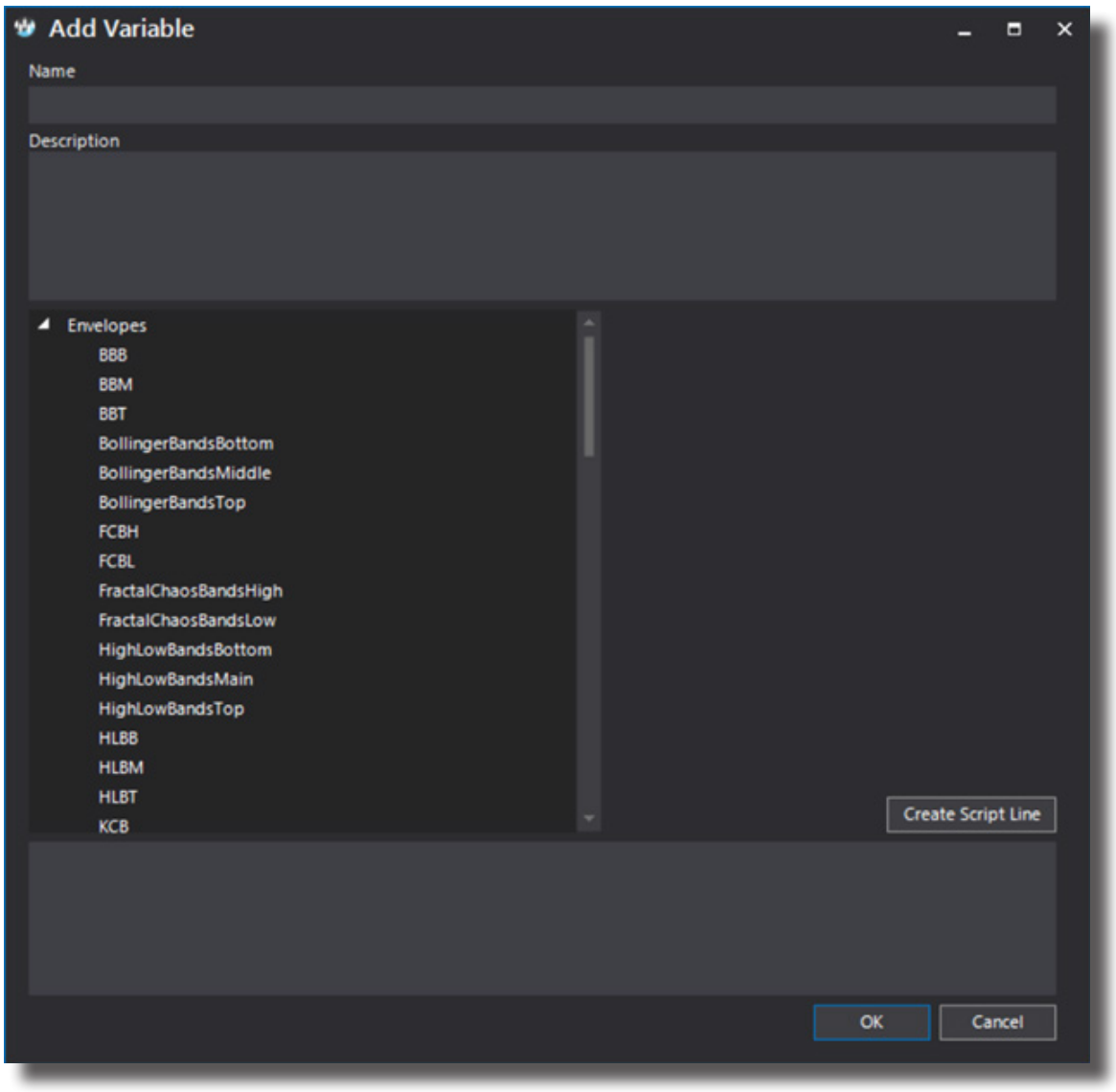
Many technical indicators rely upon the smoothing features of moving averages as part of their calculation.

The S-Trader trading platform and the Quant Script engine use no less than nine (9) Moving Average types. You will find detailed specification sheets for all of them in the **Help** section of the S-Trader.

Moving Averages List

Long Form	Short Form	Arguments
ExponentialMovingAverage	EMA	(Vector, Periods)
SimpleMovingAverage	SMA	(Vector, Periods)
TimeSeriesMovingAverage	TSMA	(Vector, Periods)
TriangularMovingAverage	TMA	(Vector, Periods)
VariableMovingAverage	VMA	(Vector, Periods)
VIDYAMovingAverage	VIDYA	(Vector, Periods)
VolumeWeightedMovingAverage	VWMA	(Vector, Periods)
WeightedMovingAverage	WMA	(Vector, Periods)
WellesWilderSmoothing	WWS	(Vector, Periods)

Envelopes


Add Variable
Name
Description
Envelopes
BBB
BBM
BBT
BollingerBandsBottom
BollingerBandsMiddle
BollingerBandsTop
FCBH
FCBL
FractalChaosBandsHigh
FractalChaosBandsLow
HighLowBandsBottom
HighLowBandsMain
HighLowBandsTop
HLBB
HLBM
HLBT
KCB
Create Script Line
OK
Cancel

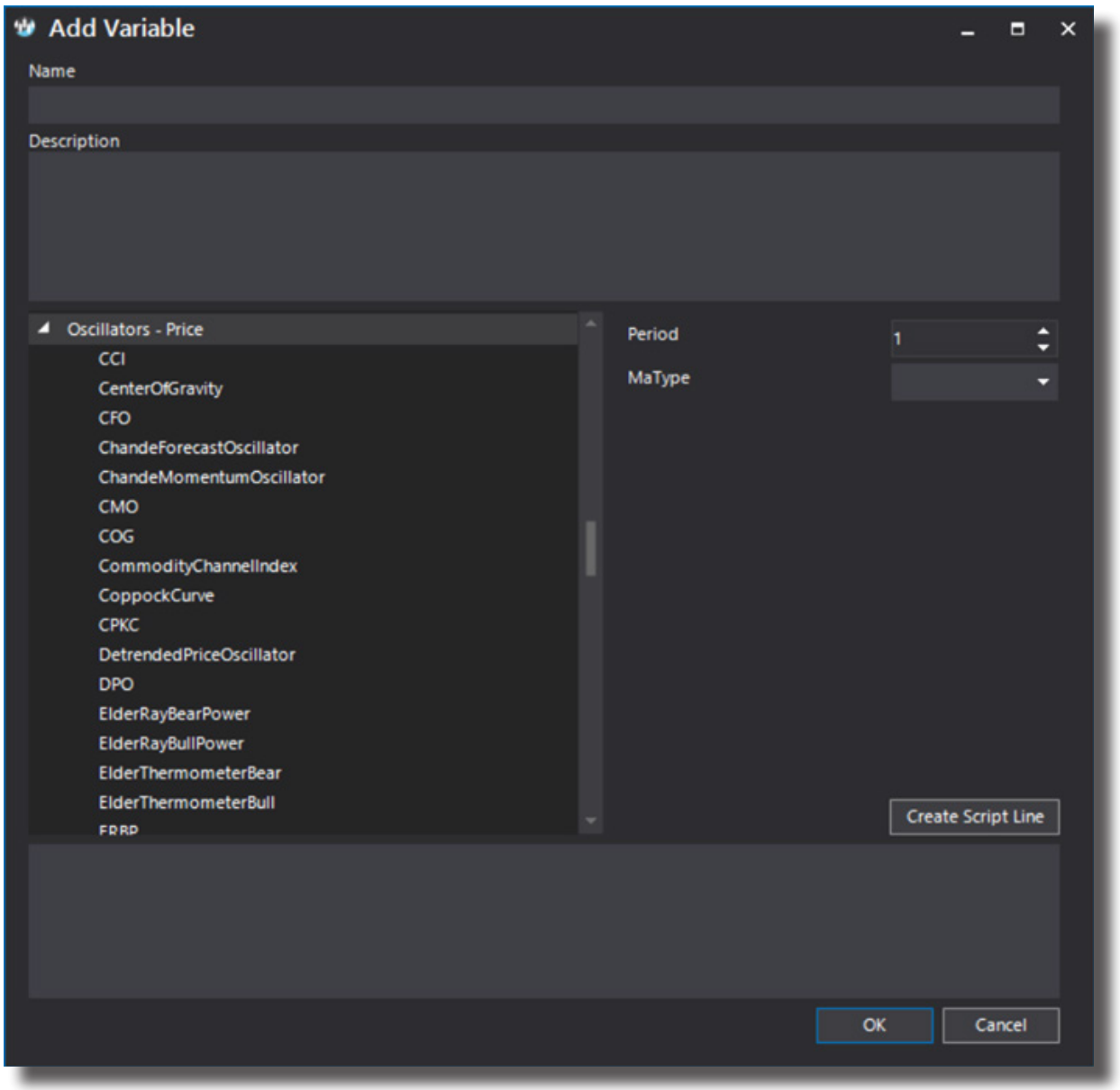
Introduction

Certain technical indicators are designed for overlaying on price charts to form an envelope or band around the underlying price. A change in trend is normally indicated if the underlying price breaks through one of the bands or retreats after briefly touching a band. The most popular band indicator is the Bollinger Bands, developed by stock trader John Bollinger in the early 1980's. You can find detailed specification sheets on all envelope studies in the Help section of the S-Trader.

Envelopes List

Long Form	Short Form	Arguments
BollingerBandsTop BollingerBandsMiddle BollingerBandsBottom	BBT BBM BBB	(Vector, Periods, StDevs, MA Type)
FractalChaosBandsHigh FractalChaosBandsLow	FCBH FCBL	(Periods)
HighLowBandsTop HighLowBandsMain HighLowBandsBottom	HLBT HLBM HLBB	(Vector, Periods, Shift)
KeltnerChannelTop KeltnerChannelMedian KeltnerChannelBottom	KCT KCM KCB	(Periods, MA Type, ATR Period, ATR MA Type, ATR Shift)
KeltnerChannelTopST KeltnerChannelMedianST KeltnerChannelBottomST	KCTST KCMST KCBST	(Source, Periods, MA Type, ATR Period, ATR MA Type, ATR Shift)
MovingAverageEnvelopeTop MovingAverageEnvelopeMain MovingAverageEnvelopeBottom	MAET MAEM MAEB	(Periods, MA Type, Shift)
PrimeNumberBandsTop PrimeNumberBandsBottom	PNBT PNBB	()
StollerAverageRangeChannelTop StollerAverageRangeChannelMedian StollerAverageRangeChannelBottom	STARCT STARCM STARCB	(Vector, Periods, MA Type, ATR Periods, ATR MA Type, Multiplier)
StandardErrorBandsTop StandardErrorBandsMiddle StandardErrorBandsBottom	SEBT SEBM SEBB	(Vector, Periods, MA Type, Multiplier)

Oscillators (Price)



Introduction

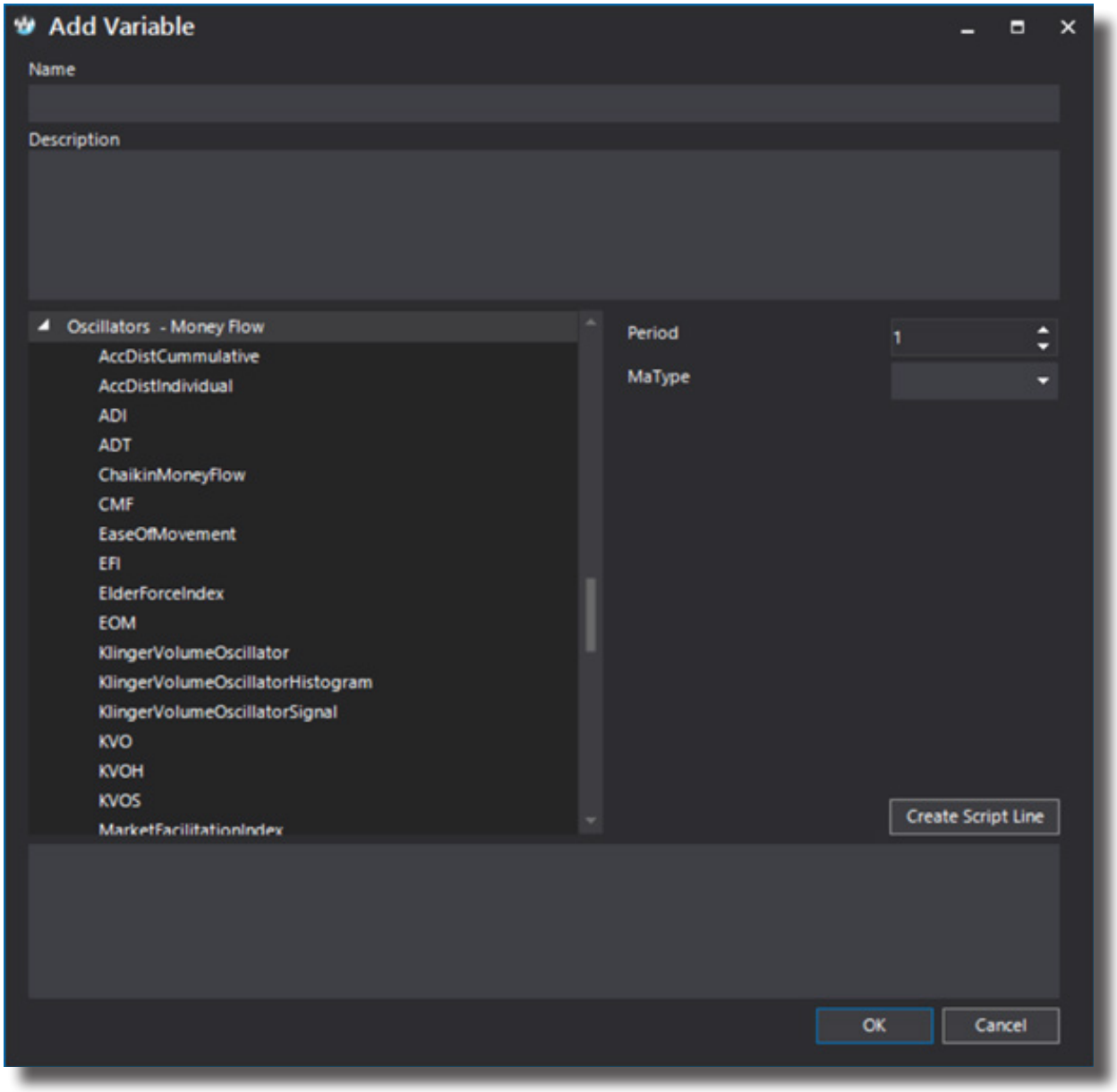
An oscillator is a technical analysis tool that is banded between two extreme values and built with the results from a trend indicator for discovering short-term overbought or oversold conditions. As the value of the oscillator approaches the upper extreme value, the asset is deemed to be overbought, and, as it approaches the lower extreme, it is deemed to be oversold. The slope of the oscillator is usually proportional to the velocity of the move. Likewise, the distance the oscillator moves up or down is usually proportional to the magnitude of the move. In this section we will take a look at the Price Oscillators available in the S-Trader and Quant Script engine. As usual detailed spec sheets for each study can be found in the **Help** section of the S-Trader.

Price Oscillators List

Long Form	Short Form	Arguments
CenterOfGravity	COG	(Vector, Periods)
ChandeForecast Oscillator	CFO	(Vector, Periods)
ChandeMomentumOscillator	CMO	(Vector, Periods)
CommodityChannelIndex	CCI	(Periods, MA Type)
CoppockCurve	CPKC	(Vector, Periods 1, Periods 2, Periods 3, MA Type)
DetrendedPriceOscillator	DPO	(Vector, Periods, MA Type)
ElderRayBullPower ElderRayBearPower	ERBP ERRP	(Periods, MA Type)
ElderThermometerBull ElderThermometerBear	ETHB ETHR	(Periods)
FractalChaosOscillator	FCO	(Periods)
IntradayMomentumIndex	IMI	(Periods)
MACDStudy MACDSignal MACDHistogram	MACD MACDS MACDH	(Vector, Periods 1, Periods 2, Periods 3, MA Type)
MACDSTraderStudy MACDSTraderSignal MACDSTraderHistogram	MACDST MACDSTS MACDSTH	(Vector1, Periods 1, MA Type 1, Vector 2, Periods 2, MA Type 2, Periods 3, MA Type 3)
MomentumOscillator	MO	(Vector, Periods)

Long Form	Short Form	Arguments
ParabollicSAR	PSAR	(Min AF, Max AF)
PerformanceIndex	PFI	(Vector)
PrettyGoodOscillator	PGO	(Vector, MA Periods, MA Type, ATR Period, ATR MA Type)
PercentagePriceOscillator PercentagePriceOscillatorSignal PercentagePriceOscillatorHistogram	PPO PPOS PPOH	(Vector, Periods 1, Periods 2, MA Type)
RateOfChange	ROC	(Vector, Periods)
PrimeNumberOscillator	PNO	(Vector)
Qstick	QSTK	(Periods, MA Type)
RainbowOscillator	RBO	(Vector, Levels, MA Type)
RelativeStrengthIndex	RSI	(Vector, Periods)
StochasticMomentumIndexK StochasticMomentumIndexD	SMIK SMID	(K Periods, K Smooth, K Double Smooth, D Periods, MA Type 1, MA Type 2)
StochasticOscillatorPCTK StochasticOscillatorPCTD	SOPK SOPD	(K Periods, %K Periods, %D Periods, MA Type)
StochasticOscillatorPCTKST StochasticOscillatorPCTDST	SOPKST SOPDST	(K Periods, %K Periods, %D Periods, MA Type 1, MA Type 2)
TrueStrengthIndex TrueStrengthIndexSmooth	TSI TSIS	(Vector, Period, FSPC Periods, FSPC MA Type, DSPC Periods, DSPC MA Type, FSAPC Periods, FSAPC MA Type, DSAPC Periods, DSAPC MA Type, TSI Smooth Per, TSI Smooth MA Type)
UltimateOscillator	UO	(Periods 1, Periods 2, Periods 3)
UltimateOscillatorST	UOST	(Periods 1, Periods 2, Periods 3, MA Type)
WilliamsPctR	WPR	(Periods)

Oscillators - Money Flow



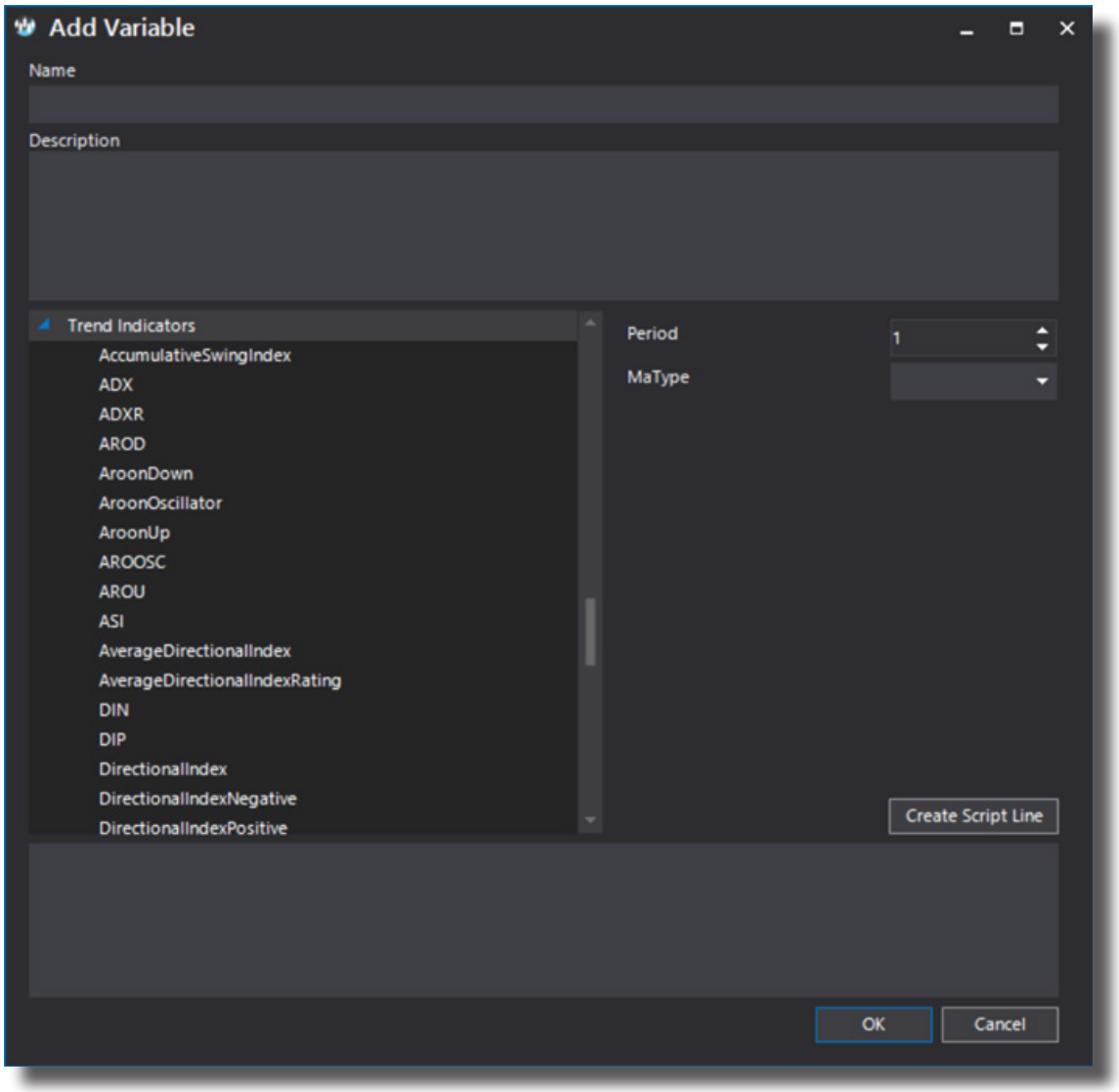
Introduction

As said before, an oscillator is a technical analysis tool that is banded between two extreme values and built with the results from a trend indicator for discovering short-term overbought or oversold conditions. As the value of the oscillator approaches the upper extreme value, the asset is deemed to be overbought, and, as it approaches the lower extreme, it is deemed to be oversold. The slope of the oscillator is usually proportional to the velocity of the price move. Likewise, the distance the oscillator moves up or down is usually proportional to the magnitude of the move. In this section we will take a look at the oscillators that measure money flow, i.e. volume and volume momentum. As usual detailed spec sheets for each study can be found in the **Help** section of the S-Trader.

Money Flow Oscillators List

Long Form	Short Form	Arguments
AccDistIndividual AccDistCummulative	ADI ADT	(Periods)
ChaikinMoneyFlow	CMF	(Periods)
EaseOfMovement	EOM	(Periods, MA Type)
ElderForceIndex	EFI	(Periods, MA Type)
KlingerVolumeOscillator KlingerVolumeOscillatorSignal KlingerVolumeHistogram	KVO KVOS KVOH	(Periods 1, MA Type1, Periods 2, MA Type 2, Periods 3, MA Type3)
MarketFacilitationIndex	MKTFI	()
MoneyFlowIndex	MFI	(Periods)
NegativeVolumeIndex	NVI	(Vector)
OnBalanceVolume	OBV	(Vector)
PositiveVolumeIndex	PVI	(Vector)
PriceVolumeTrend	PVT	(Vector)
TradeVolumeIndex	TVI	(Vector, Minimum Tick Move)
TwiggsMoneyFlow	TMF	(Periods)
VolumeOscillator VolumeOscillatorSignal VolumeOscillatorHistogram	VO VOS VOH	(Periods 1, Periods 2, MA Type, PointsPercent, Periods Signal, Signal MA Type)
VolumeRateOfChange	VROC	(Periods)
WilliamsAccumulationDistribution	WAD	()
WilliamsVariableAccDist	WVAD	(Periods, MA Type)

Trend Indicators



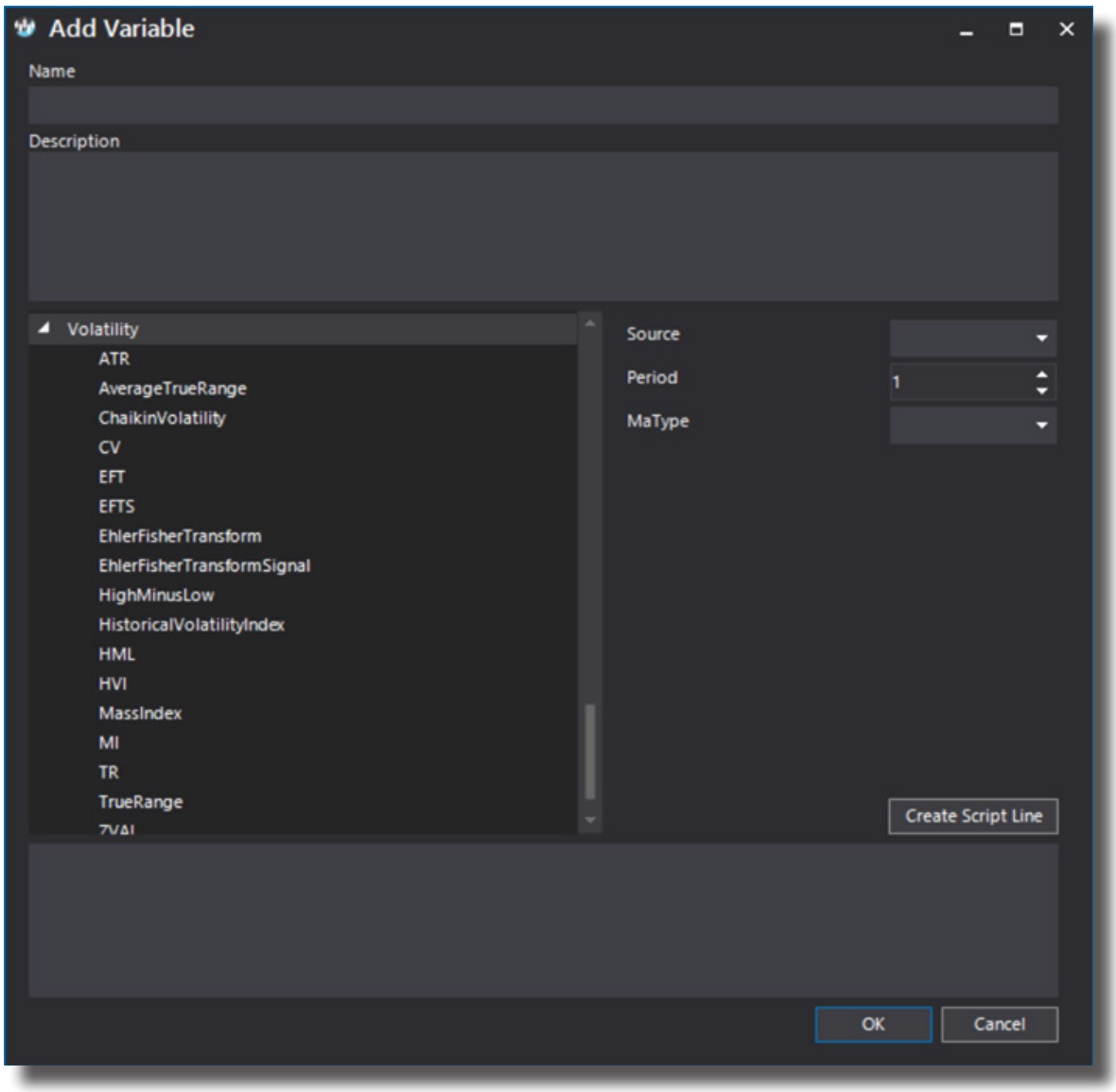
Introduction

As their name reveals, trend indicators measure the direction and intensity of price trends. They are exceptionally important in separating trending phases of the market from non-trending ones which is perhaps one of the most important distinction to make when deciding how to approach and model market risk and market opportunity. As usual detailed spec sheets for each study can be found in the **Help** section of the S-Trader.

Trend Indicators List

Long Form	Short Form	Arguments
Swing Index AccumulativeSwingIndex	SI ASI	() / (Periods)
AroonUp AroonDown AroonOscillator	AROU AROD AROOSC	(Periods)
AverageDirectionalIndex AverageDirectionalIndexRating DirectionalIndexPositive DirectionalIndexNegative DirectionalIndex TrueRangeSummation SmoothedPositiveDM SmoothedNegativeDM	ADX ADXR DIP DIN DX TRSUM SPDM SNDM	(Periods)
GopalakrishnanRangeIndex	GOPRI	(Periods)
Forecast TimeSeriesForecast Intercept Slope Rsquared	LRF TSF LRI LRS R2	(Vector, Periods)
RandomWalkIndexUp RandomWalkIndexDown	RWIU RWID	(Periods, MA Type)
RangeActionVerificationIndex	RAVI	(Vector, Periods 1, MA Type 1, Periods 2, MA Type 2)
SchaffTrendCycleMACD	STCMACD	(Vector, Per 1, MA Type 1, Per 2, MA Type 2, Signal Periods, MA Type 3, K Per 1, %K Per 1, %D Per 1, MA Type 4, MA Type 5, K Per 2, %K Per 2, %D Per 2, MA Type 6, MA Type 7)
SchaffTrendCycleMACDS	STCMACDS	(Vector, Per 1, MA Type 1, Per 2, MA Type 2, Signal Periods, MA Type 3, K Per 1, %K Per 1, %D Per 1, MA Type 4, MA Type 5, K Per 2, %K Per 2, %D Per 2, MA Type 6, MA Type 7)
TRIXIndicator	TRIX	(Periods)
VerticalHorizontalFilter	VHF	(Vector, Periods)

Volatility Indicators



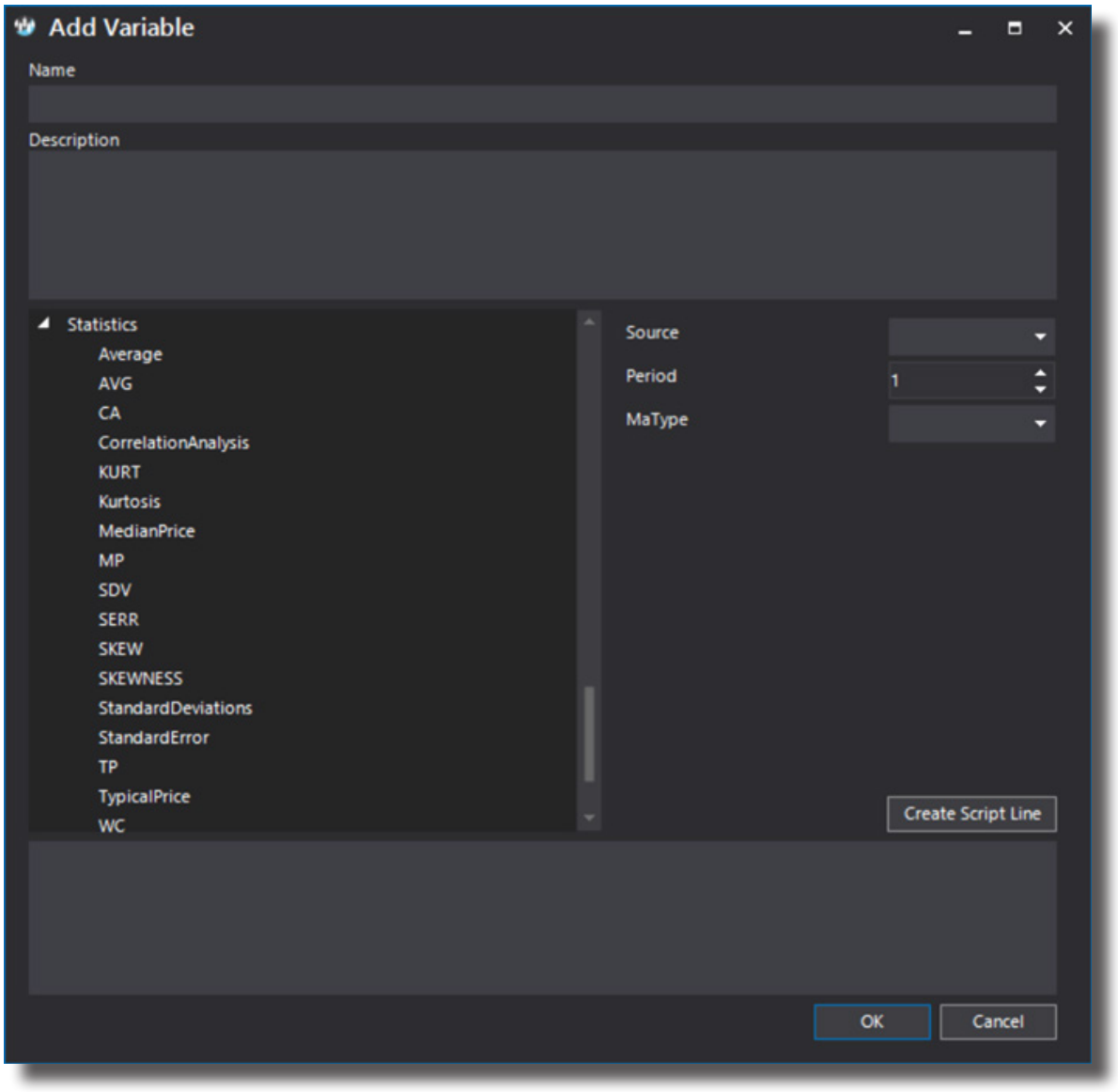
Introduction

Volatility is the one coin whose two sides represent what is significant about financial markets, i.e. opportunity and risk. As usual detailed spec sheets for each volatility study can be found in the **Help** section of the S-Trader.

Volatility Indicators List

Long Form	Short Form	Arguments
ChaikinVolatility	CV	(Periods, ROC Periods, MA Type)
EhlerFisherTransform EhlerFisherTransformSignal	EFT EFTS	(Lookup Periods, Raw Smooth Periods, Fisher Smooth Periods, Fisher Signal Periods)
HighMinusLow	HML	()
HistoricalVolatilityIndex	HVI	(Vector, Periods, Annualized Bar Periods, Multiplier)
MassIndex	MI	(EMA Periods, Summation Periods)
TrueRange	TR	()
AverageTrueRange	ATR	(Vector, Periods, MA Type)
ZValue	ZVAL	(Vector, Periods, MA Type)

Statistical Functions



Introduction

A statistic (singular) or sample statistic is a single measure of some attribute of a sample (e.g., its arithmetic mean value). It is calculated by applying a function (statistical algorithm) to the values of the items of the sample, which are known together as a set of data.

Descriptive statistics are used to describe the basic features of the data in a study. They provide simple summaries about the sample and the measures. Together with simple graphics analysis, they form the basis of virtually every quantitative analysis of data.

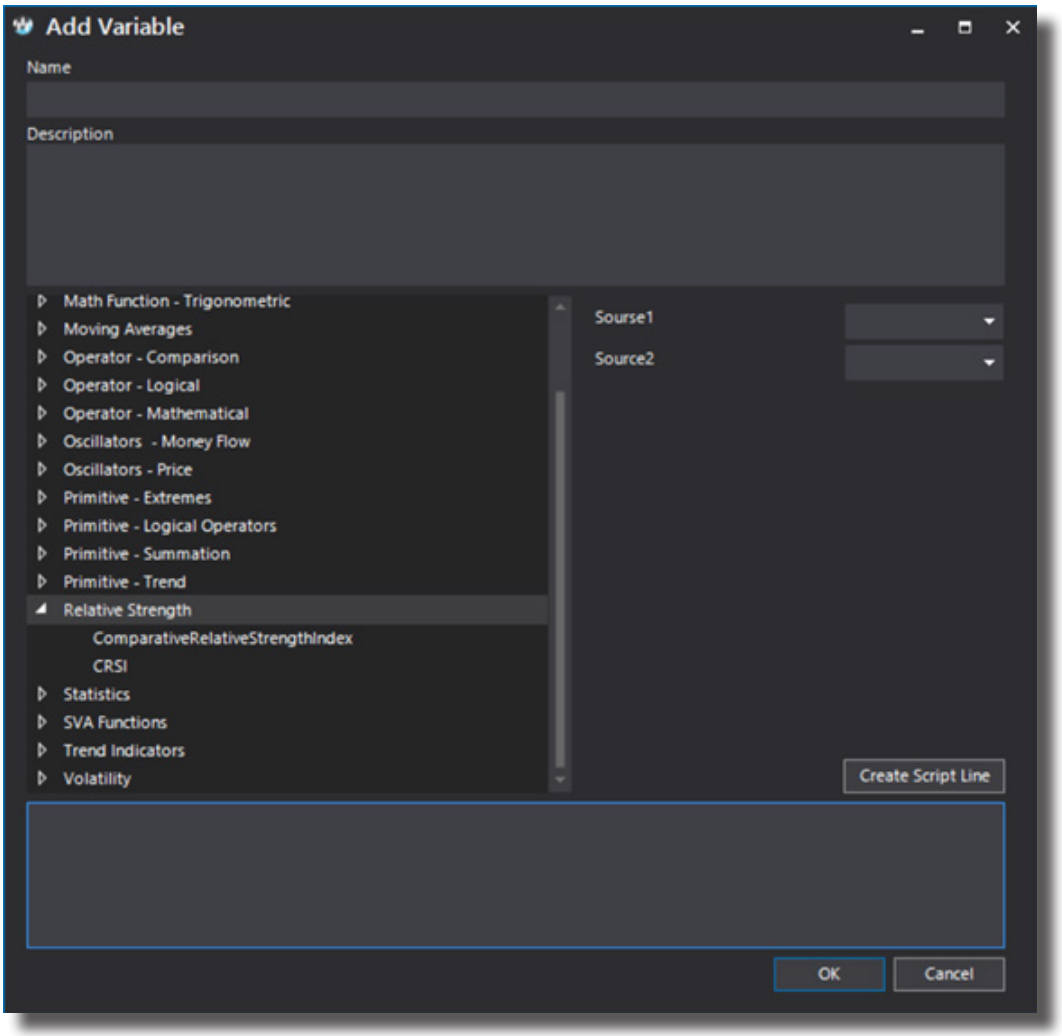
Descriptive statistics are typically distinguished from inferential statistics. With descriptive statistics you are simply describing what is or what the data shows. With inferential statistics, you are trying to reach conclusions that extend beyond the immediate data alone.

Current Quant Script capabilities pertain to descriptive statistics. Inferential statistics capabilities are planned in future versions of the S-Trader application.

Statistical Functions List

Long Form	Short Form	Arguments
Average	CV	(Vector, Periods)
Median	MP	()
TypicalPrice	TP	()
WeightedClose	WC	()
Kurtosis	KURT	(Vector, Periods, MA Type, Excess Threshold)
Skewness	SKEW	(Vector, Periods, MA Type)
StandardDeviation	SDV	(Vector, Periods, Multiplier, MA Type)
StandardError	SERR	(Vector, Periods, Multiplier, MA Type)
CorrelationAnalysis	CA	(Vector 1, Vector 2, MA Type, Periods, Offset V2)

Relative Strength Functions

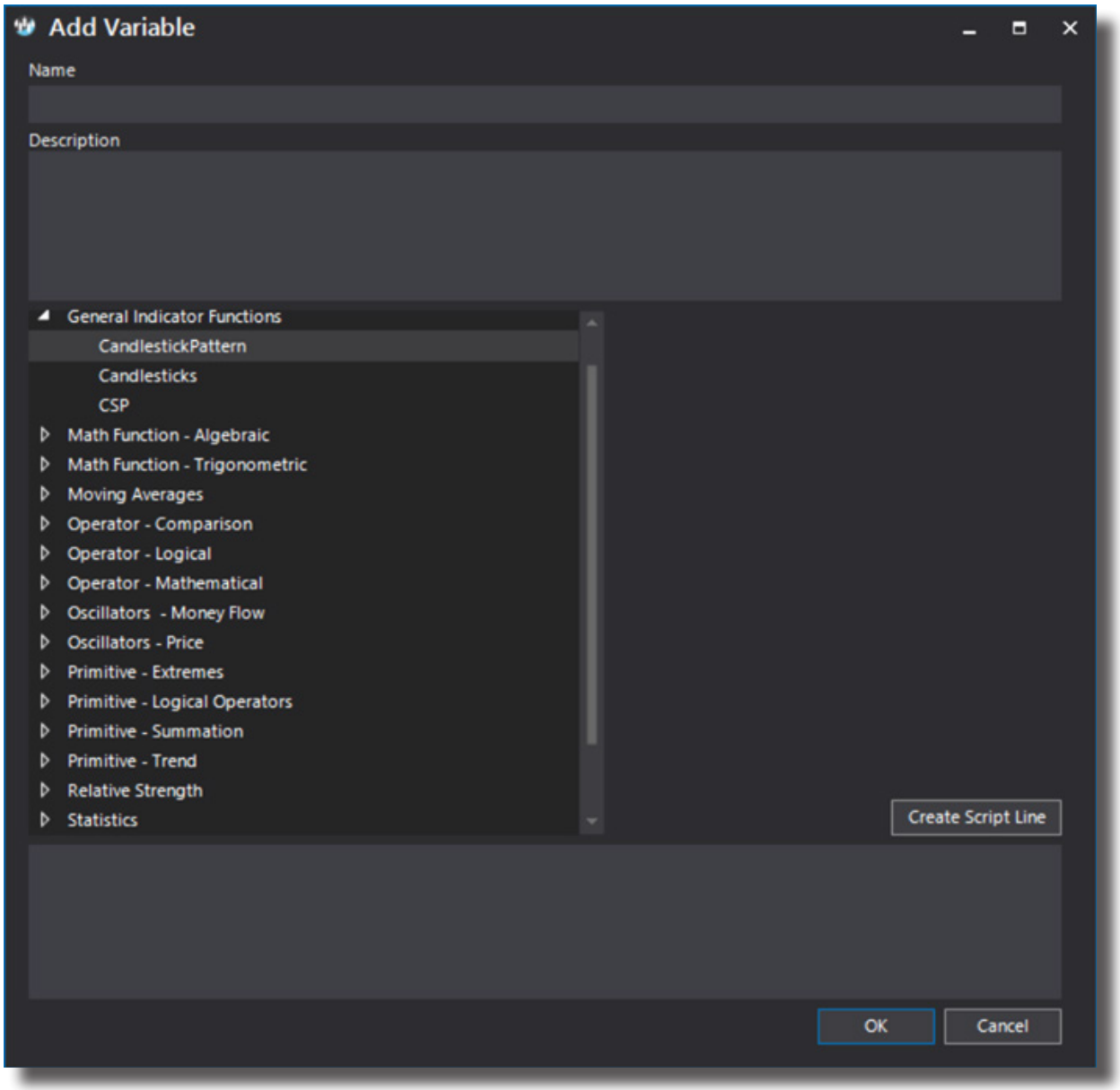


Relative Strength Functions List

Long Form	Short Form	Arguments
ComparativeRelativeStrengthIndex	CRSI	(Vector 1, Vector 2)

This function returns the ratio between Vector 1 and Vector 2.

General Indicator Functions



Candlesticks Patterns

What is a ‹Candlestick›

A candlestick is a chart that displays the high, low, opening and closing prices of a security for a specific time frame (i.e. 1 hour, 1 day, etc) over a specific period of time. The wide part of the candlestick is called the «real body» and tells investors whether the closing price was higher or lower than the opening price. Black/red indicates that the stock closed lower and white/green indicates that the stock closed higher.

The candlestick›s shadows show the day›s high and low and how they compare to the open and close. A candlestick›s shape varies based on the relationship between the day›s high, low, opening and closing prices.

Candlesticks reflect the impact of investor sentiment on security prices and are used by technical analysts to determine when to enter and exit trades. Candlestick charting is based on a technique developed in Japan in the 1700s for tracking the price of rice. Candlesticks are a suitable technique for trading any liquid financial asset such as stocks, foreign exchange and futures.

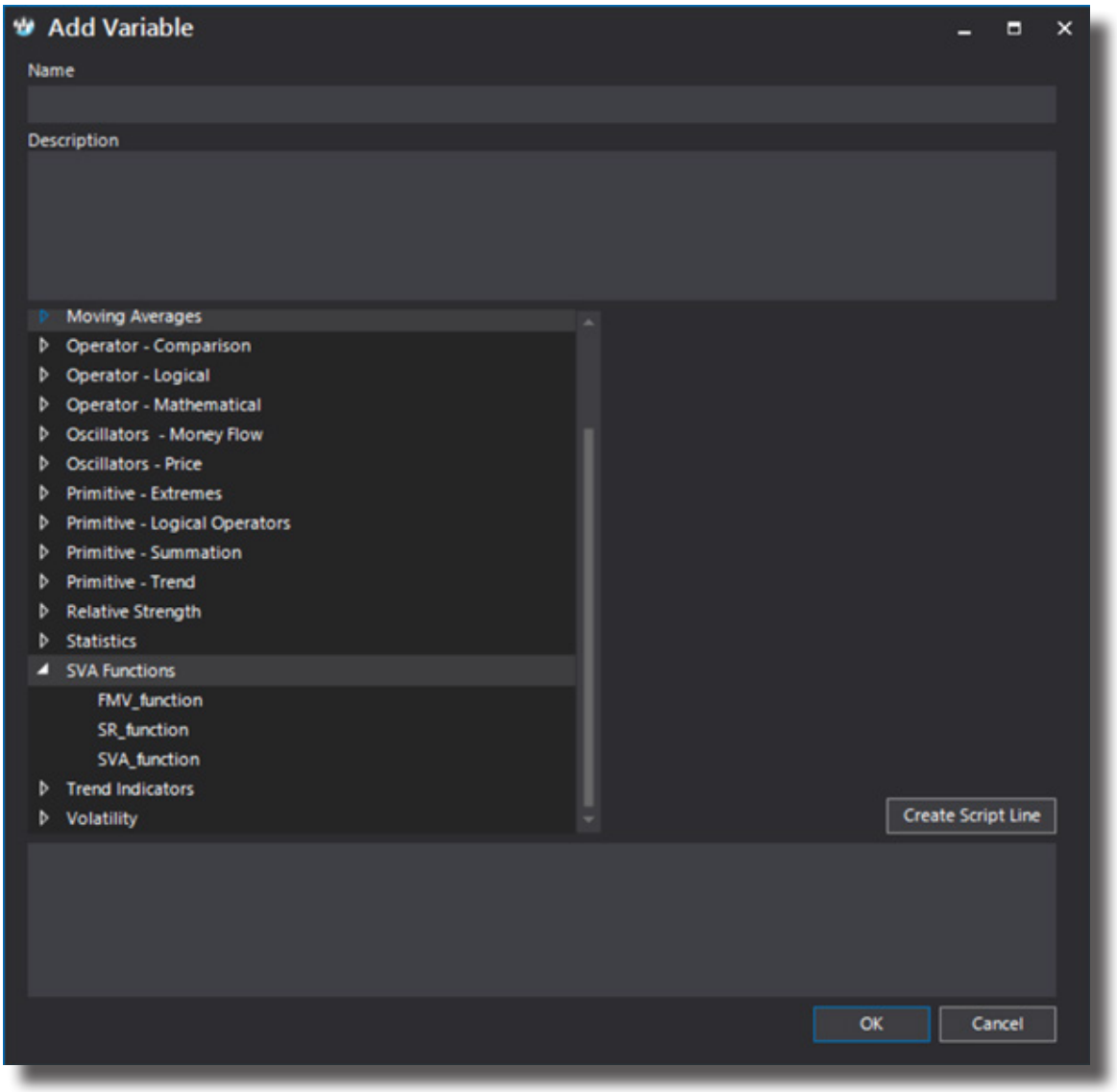
Long white/green candlesticks indicate there is strong buying pressure; this typically indicates price is bullish, however, they should be looked at in the context of the market structure as opposed to individually. For example, a long white candle is likely to have more significance if it forms at a major price support level. Long black/red candlesticks indicate there is significant selling pressure. This suggests the price is bearish. A common bullish candlestick reversal pattern, referred to as a hammer, forms when price moves substantially lower after the open, then rallies to close near the high. The equivalent bearish candlestick is known as a hanging man. These candlesticks have a similar appearance to a square lollipop, and are often used by traders attempting to pick a top or bottom in a market.

Candlestick Pattern Function

This function returns a value based on the identified candlestick pattern:

- LONG_BODY = 1
- DOJI = 2
- HAMMER = 3
- HARAMI = 4
- STAR = 5
- DOJI_STAR = 6
- MORNING_STAR = 7
- EVENING_STAR = 8
- PIERCING_LINE = 9
- BULLISH_ENGULFING_LINE = 10
- HANGING_MAN = 11
- DARK_CLOUD_COVER = 12
- BEARISH_ENGULFING_LINE = 13
- BEARISH_DOJI_STAR = 14
- BEARISH_SHOOTING_STAR = 15
- SPINNING_TOPS = 16
- HARAMI_CROSS = 17
- BULLISH_TRISTAR = 18
- THREE_WHITE_SOLDIERS = 19
- THREE_BLACK_CROWS = 20
- ABANDONED_BABY = 21
- BULLISH_UPSIDE_GAP = 22
- BULLISH_HAMMER = 23
- BULLISH_KICKING = 24
- BEARISH_KICKING = 25
- BEARISH_BELT_HOLD = 26
- BULLISH_BELT_HOLD = 27
- BEARISH_TWO_CROWS = 28
- BULLISH_MATCHING_LOW = 29

SVA Equity Valuation Functions



Introduction

Structural Valuation Analysis (SVA) was first detailed by Dr. Verne Atrill in his manuscript, ***How All Economies Work***, and has since been refined into an investment research system by the Strategic Analysis Corporation (SAC).

The key to using SVA is the understanding that the stock market is broken into a spectrum of "valuation zones". That is to say, the stock market does not treat the valuation process as a continuum, but as a spectrum. These valuation zones are bounded by Breakpoints, which correspond to fixed multiples of an entity's adjusted book value per share, referred to as the Normal Price. These multiples have their origin in Dr. Atrill's theory of Accounting Dynamics, which explored how physical constants emerge to govern the financial structure of a firm. Zones are characterized by entity stability, risk, and investor expectations. At the Breakpoints, there is a natural tendency for prices to be turned back, thereby providing signals for optimal buying and selling points.

SVA Functions List

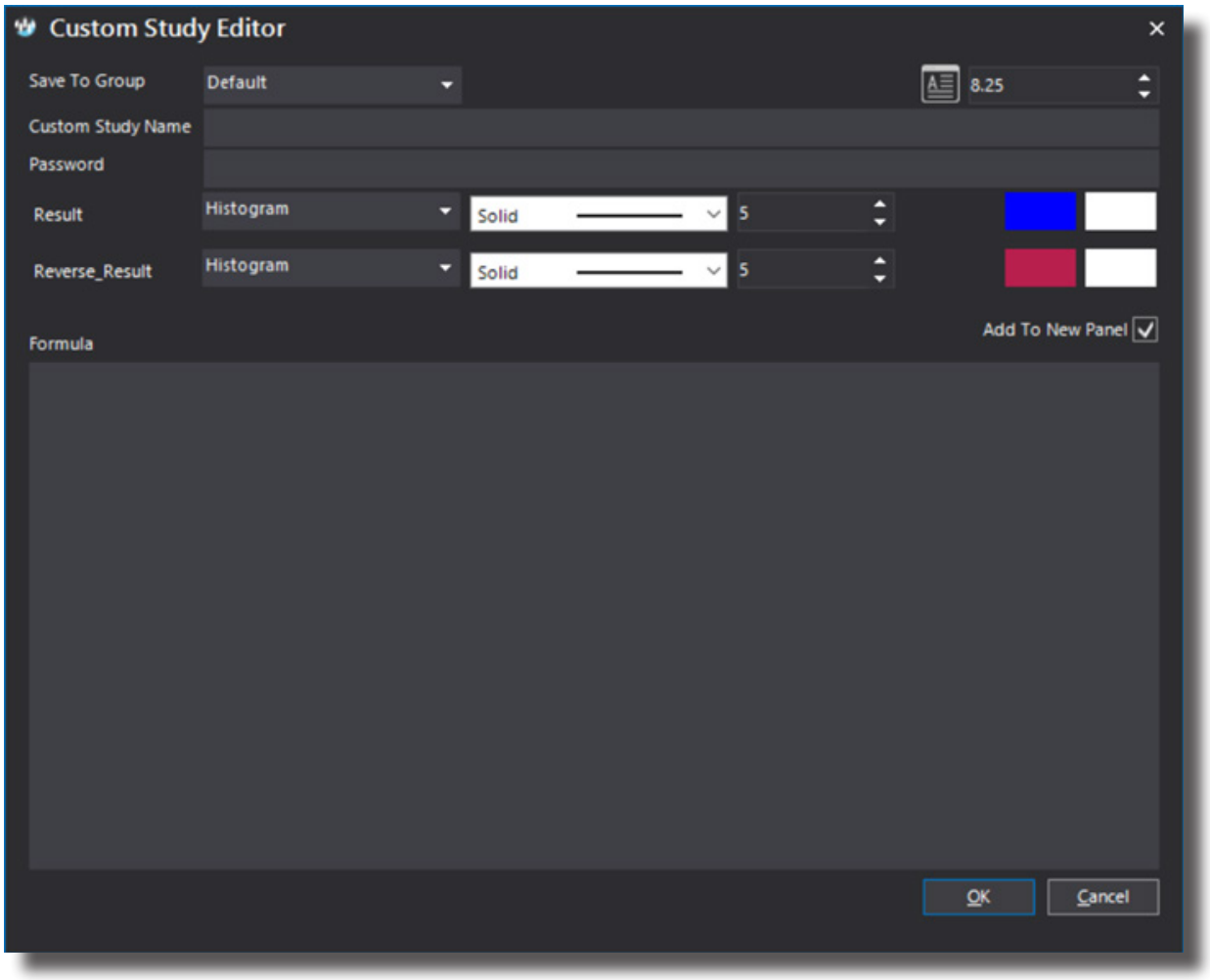
Long Form	Short Form	Arguments
FMV_function		()
SR_function		()
SVA_function		({Breakpoint Acronym})

For the SVA Function, the Breakpoints are:

High Bubble 8	HB8	Growth	G
High Bubble 7	HB7	High Conservation	HC
High Bubble 6	HB6	High-Mid	HM
High Bubble 5	HB5	Normal	N
High Bubble 4	HB4	Low-Mid	LM
High Bubble 3	HB3	Low Conservation	LC
High Bubble 2	HB2	Blue	BL
High Bubble 1	HB1	Deep Blue 1	DB1
Bubble	BB	Deep Blue 2	DB2
Mid Super Growth	MSG	Deep Blue 3	DB3
Low Super Growth	LSG	Deep Blue 4	DB4
Super Growth	SG	Deep Blue 5	DB5
Mid-Growth	MG	Deep Blue 6	DB6

Sample Scripts

The quant script engine allows you to create scripts using either an **editor** or a **wizard**. The editor is typically required in the case of Expert Advisers and Scripted Alerts and is optional under the Custom Studies.



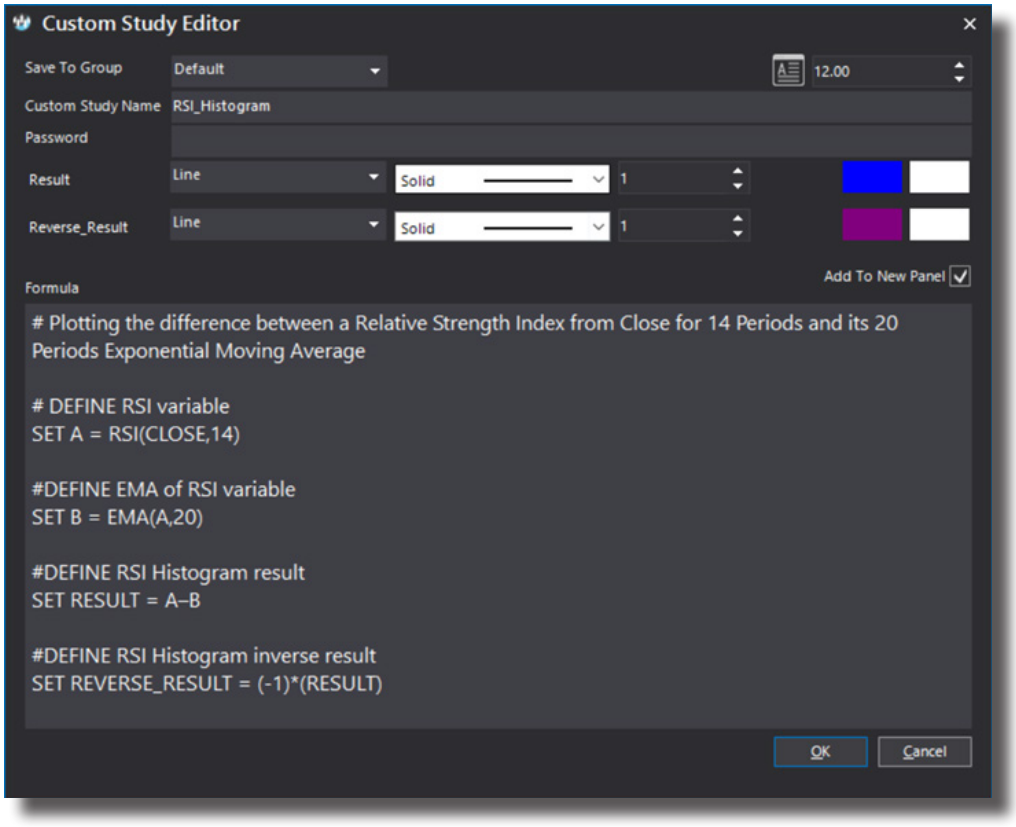
The custom study script editor is showcased above. You can type code in that dialog or in any other text editor such as **Notepad ++** or **Crimson editor** and copy / paste that code in the dialog.

Sample Custom Study 1 – RSI Histogram

======COPY / PASTE this code inside your code editor

```
# Plotting the difference between a Relative Strength Index from Close for 14 Periods and its 20 Periods Exponential Moving Average

# DEFINE RSI variable
SET A = RSI(CLOSE,14)

#DEFINE EMA of RSI variable
SET B = EMA(A,20)

#DEFINE RSI Histogram result
SET RESULT = A-B

#DEFINE RSI Histogram inverse result
SET REVERSE_RESULT = (-1)* (RESULT)
```

This is how the editor would look like:



When plotted on a chart, the custom study **RSI_Histogram** would look like this:

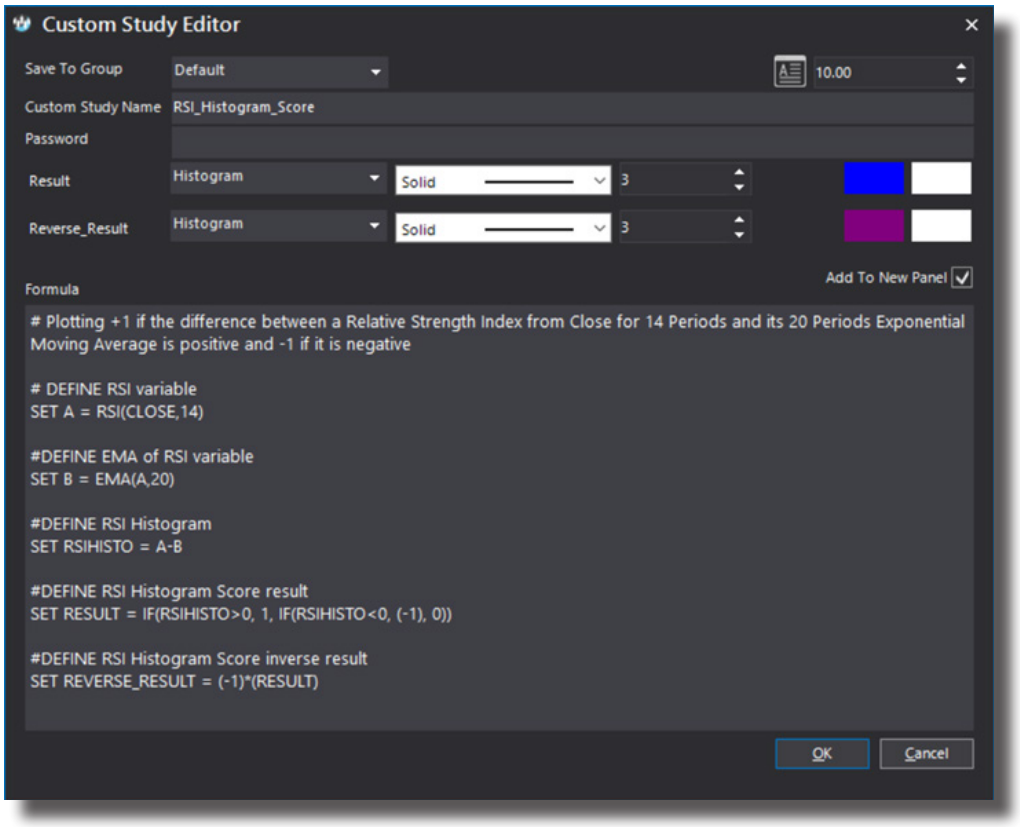
Sample Custom Study 2 – RSI Histogram Scoring

Expanding on the previous example, we will now create a custom study that will return +1 when the RSI Histogram is positive and -1 when it is negative.

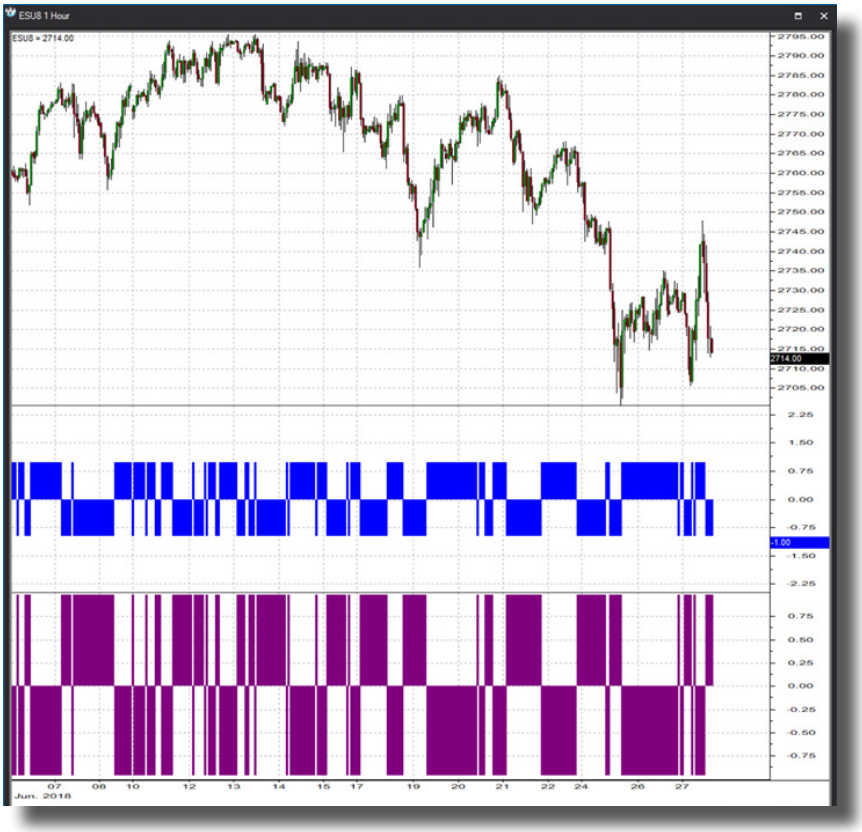
======COPY / PASTE this code inside your code editor

```
# Plotting +1 if the difference between a Relative Strength Index from Close for 14 Periods and its 20 Periods Exponential Moving Average is positive and -1 if it is negative

# DEFINE RSI variable
SET A = RSI(CLOSE,14)

#DEFINE EMA of RSI variable
SET B = EMA(A,20)

#DEFINE RSI Histogram
SET RSIHISTO = A-B

#DEFINE RSI Histogram Score result
SET RESULT = IF(RSIHISTO>0, 1, IF(RSIHISTO<0, (-1), 0))

#DEFINE RSI Histogram Score inverse result
SET REVERSE_RESULT = (-1)*(RESULT)
```

This is how the editor would look like:



When plotted on a chart, the custom study **RSI_Histogram_Score** would look like this:



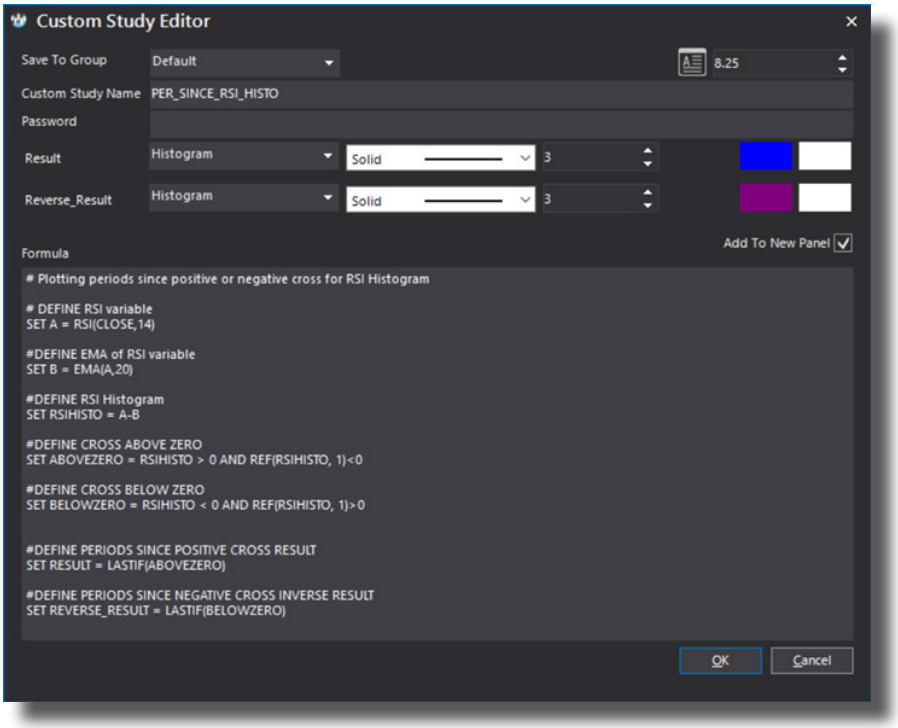
Sample Custom Study 3 – RSI Histogram days since turning positive and negative

Expanding on the previous two examples, we will now create a custom study that will return the number of periods since the RSI_Histogram last crossed into positive and the number of days since it last crossed into negative.

======COPY / PASTE this code inside your code editor

```
# Plotting periods since positive or negative cross for RSI Histogram

# DEFINE RSI variable
SET A = RSI(CLOSE,14)

#DEFINE EMA of RSI variable
SET B = EMA(A,20)

#DEFINE RSI Histogram
SET RSIHISTO = A-B

#DEFINE CROSS ABOVE ZERO
SET ABOVEZERO = RSIHISTO > 0 AND REF(RSIHISTO, 1)<0

#DEFINE CROSS BELOW ZERO
SET BELOWZERO = RSIHISTO < 0 AND REF(RSIHISTO, 1)>0

#DEFINE PERIODS SINCE POSITIVE CROSS RESULT
SET RESULT = LASTIF(ABOVEZERO)

#DEFINE PERIODS SINCE NEGATIVE DAYS RESULT
SET REVERSE_RESULT = LASTIF(BELOWZERO)
```

This is how the editor would look like:

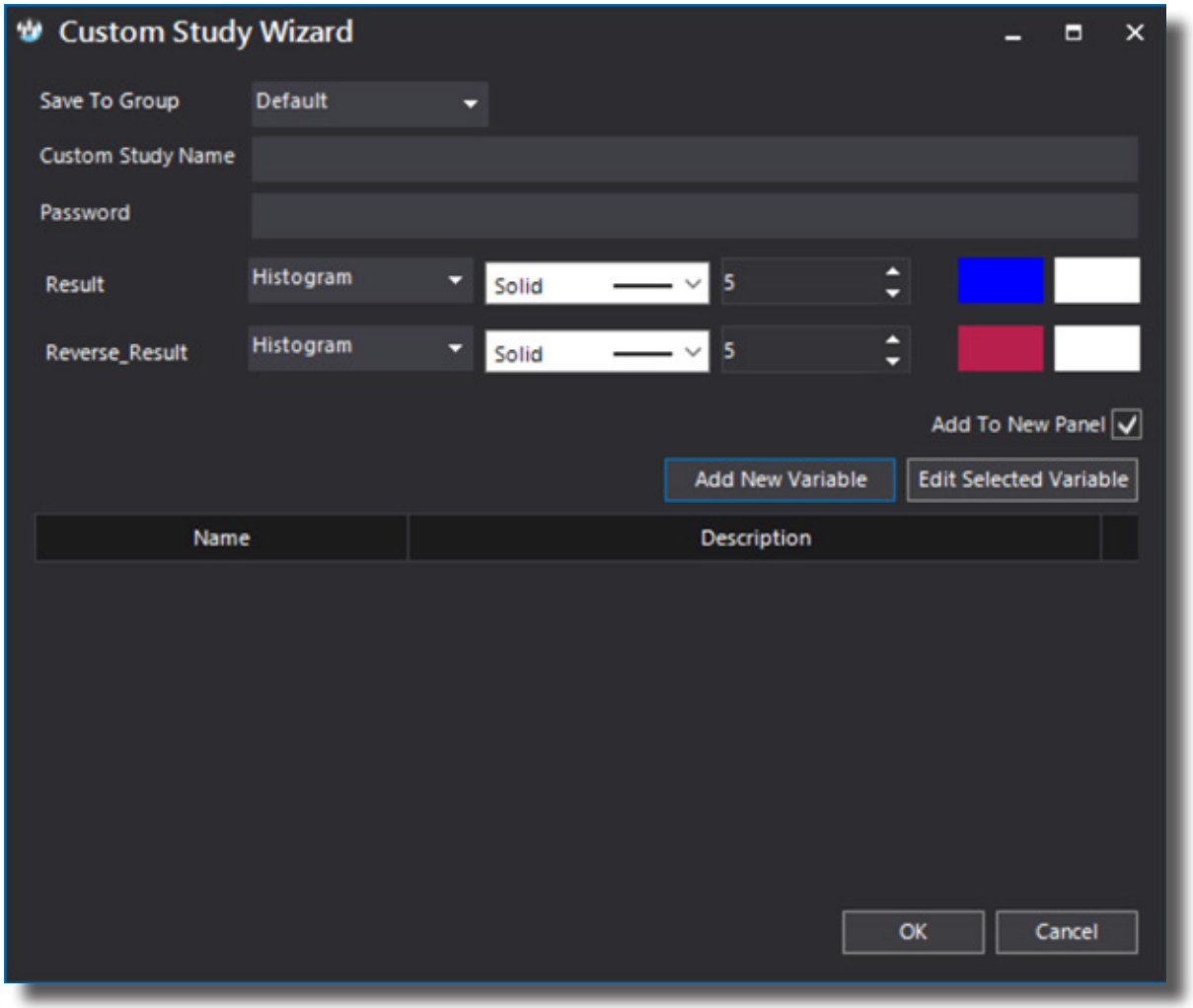


When plotted on a chart, the custom study **PER_SINCE_RSI_HISTO** would look like this:

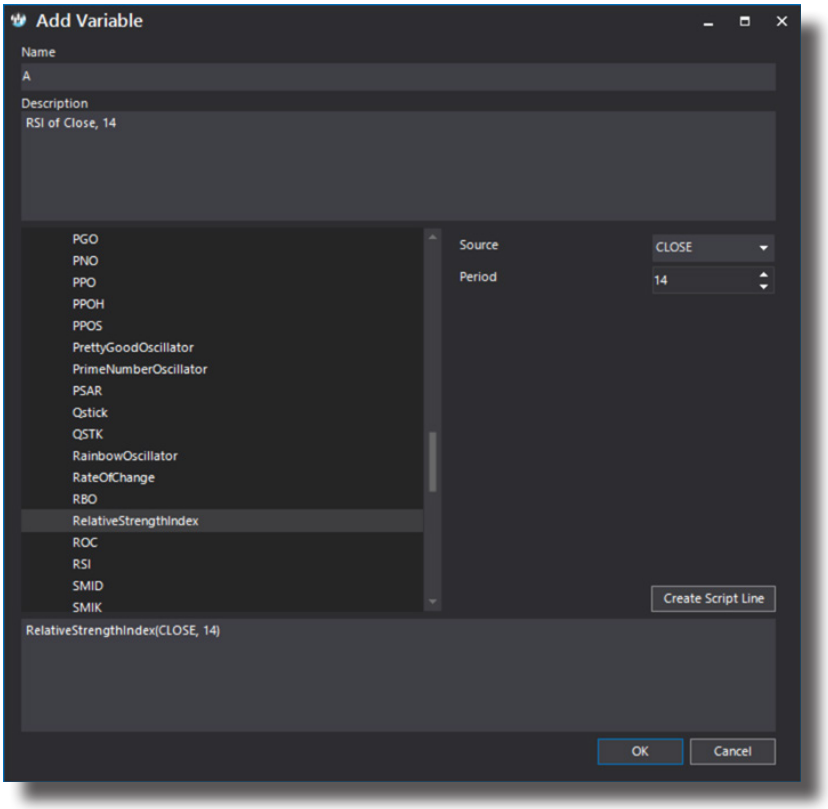
Using the Script Wizard

The exact same scripts built by the code editor above can be built using the Script Wizard. You would go and create the script line by line, choosing the right function from its group, setting the desired arguments and pressing the “Create Script Line” button.

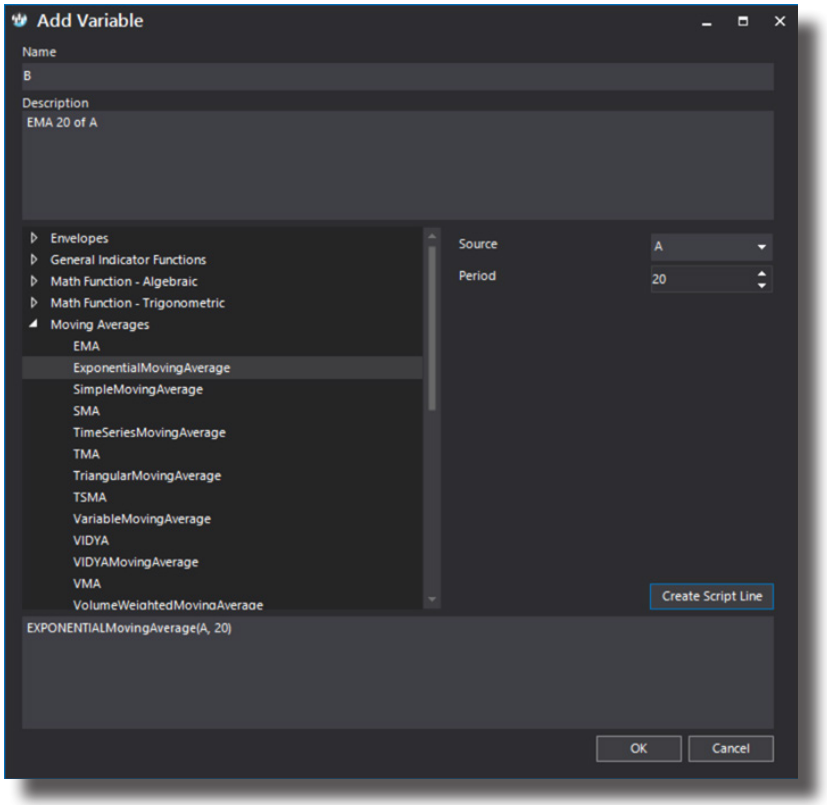
Sample Custom Study 4 – RSI Histogram by Script Wizard



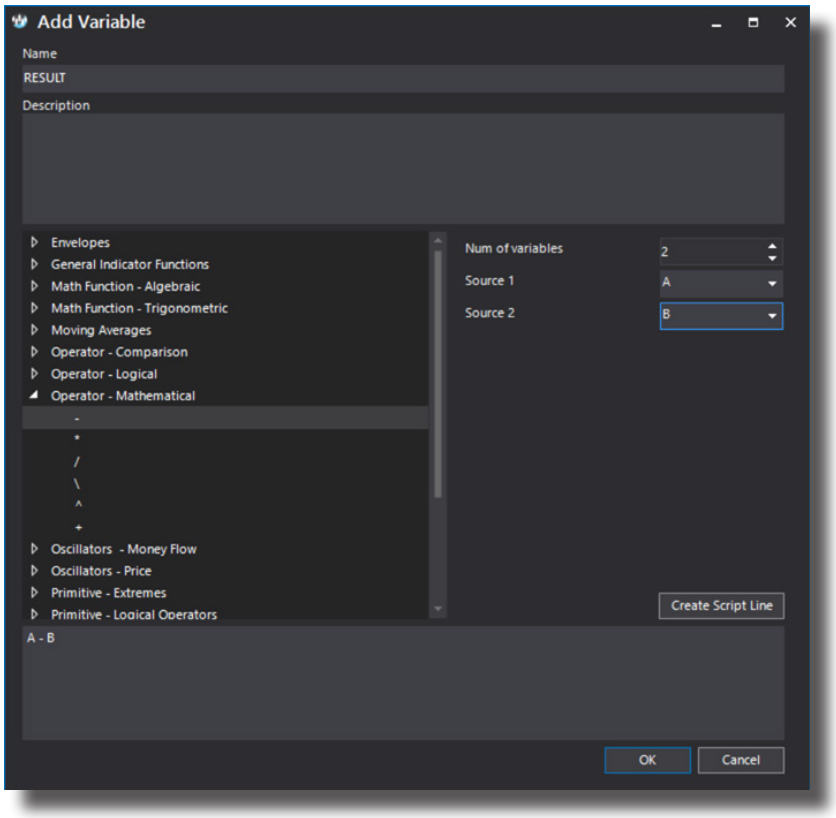
SET A = RSI (CLOSE, 14)



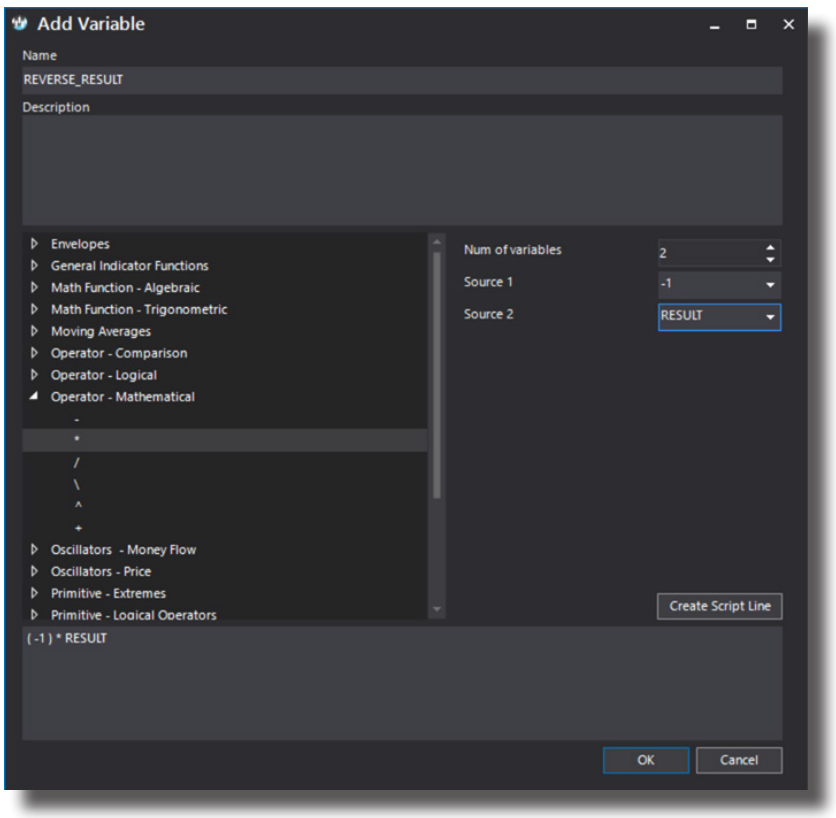
SET B = EMA (A, 20)



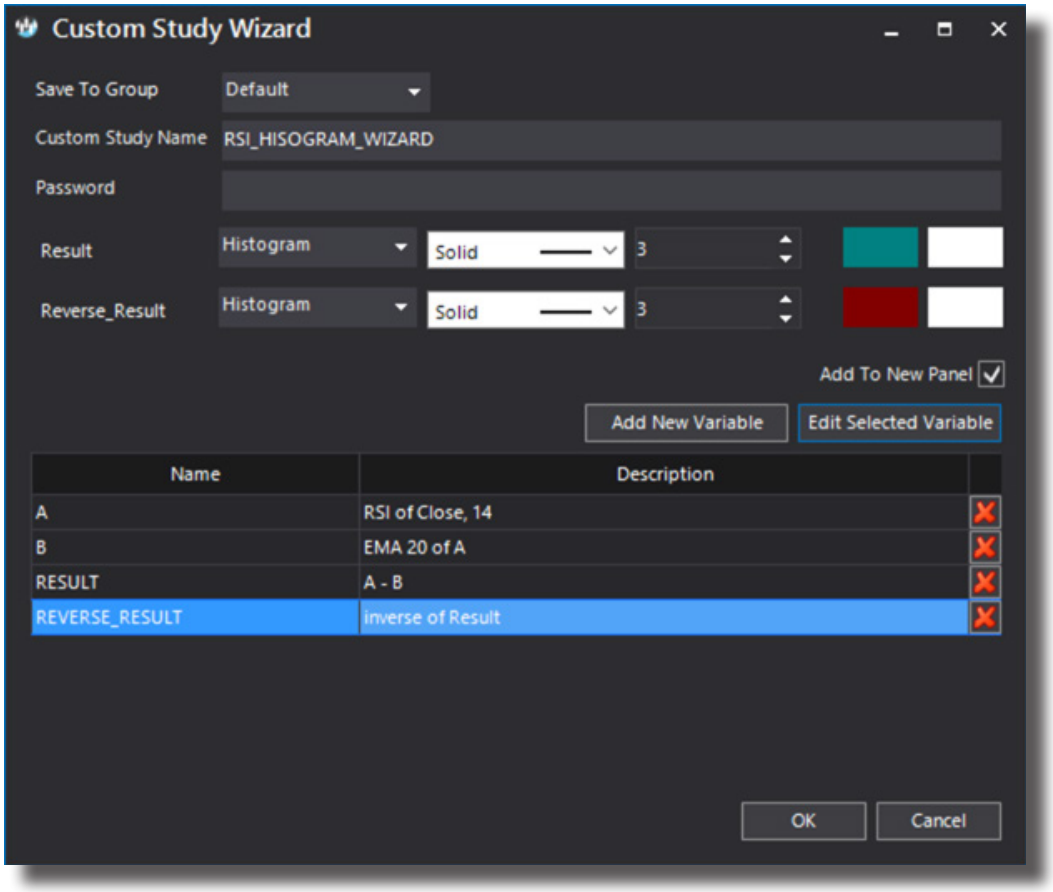
SET RESULT = A-B



SET RE SET REVERSE_RESULT = (-1)*(RESULT) SULT = A-B



Your Script Wizard Custom Study looks like this:



When plotted on a chart, the custom study **RSI_HISTO_WIZARD** would look like this:

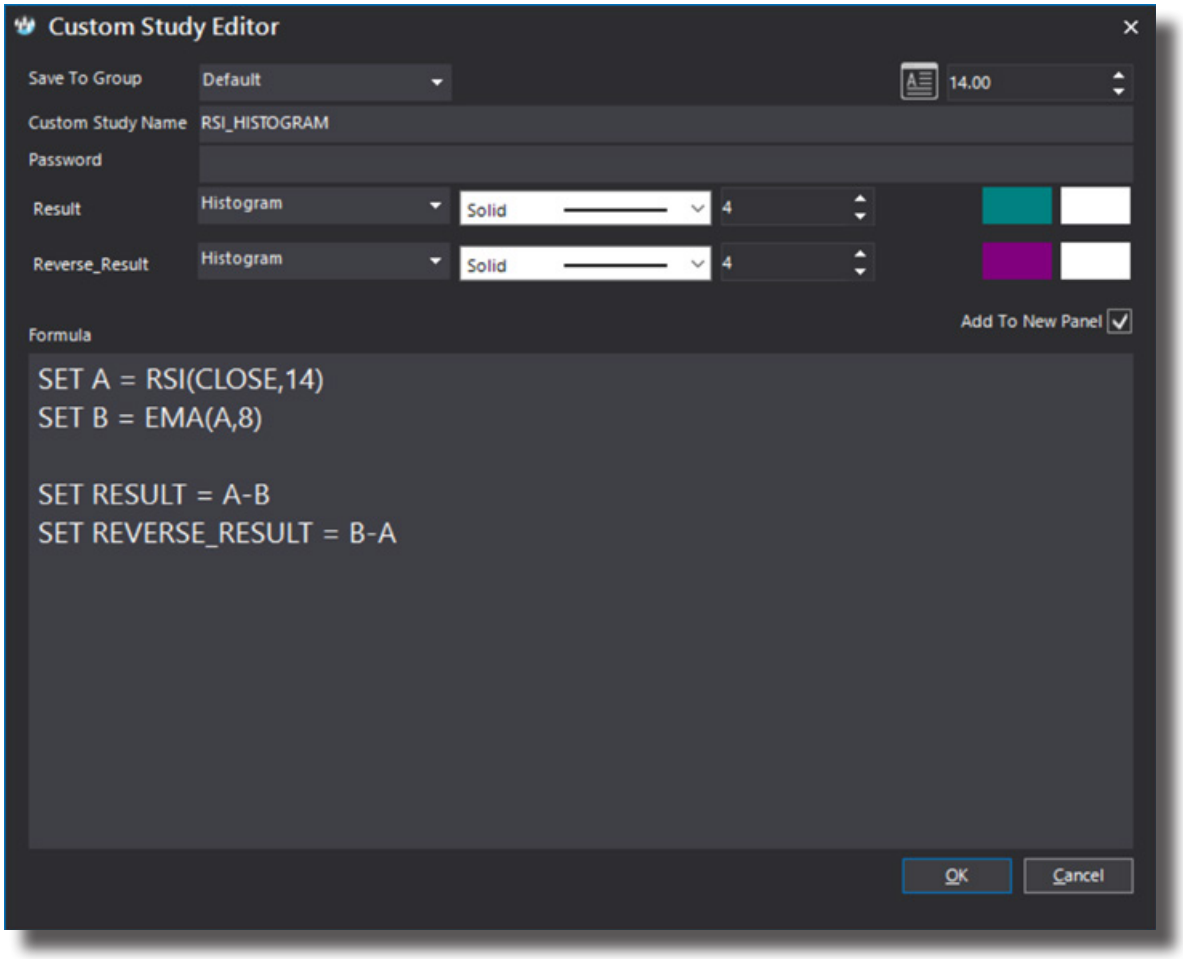
If compared to the equivalent custom study built by editor, you will notice the values are identical:

Using Quant Script in the S-Trader

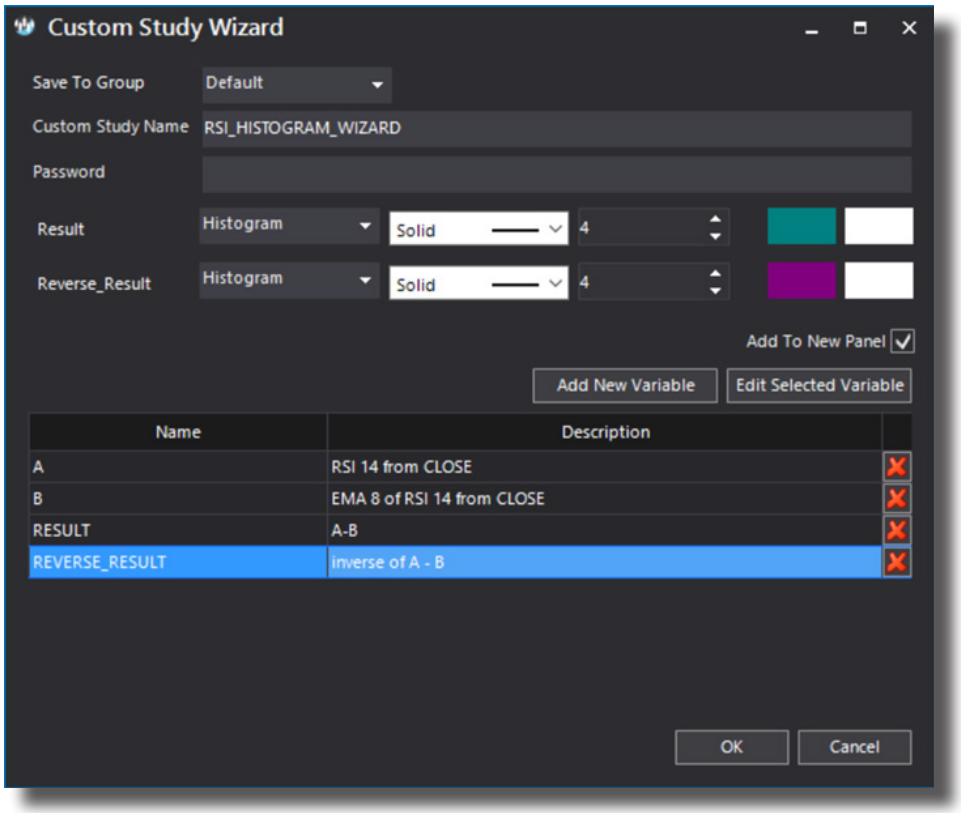
There are many uses for ***Quant Script*** inside the S-Trader application. Below we are giving you a couple of sample uses to show you how powerful ***Quant Script*** really is.

Custom Studies

Build Custom Studies by Code Editor



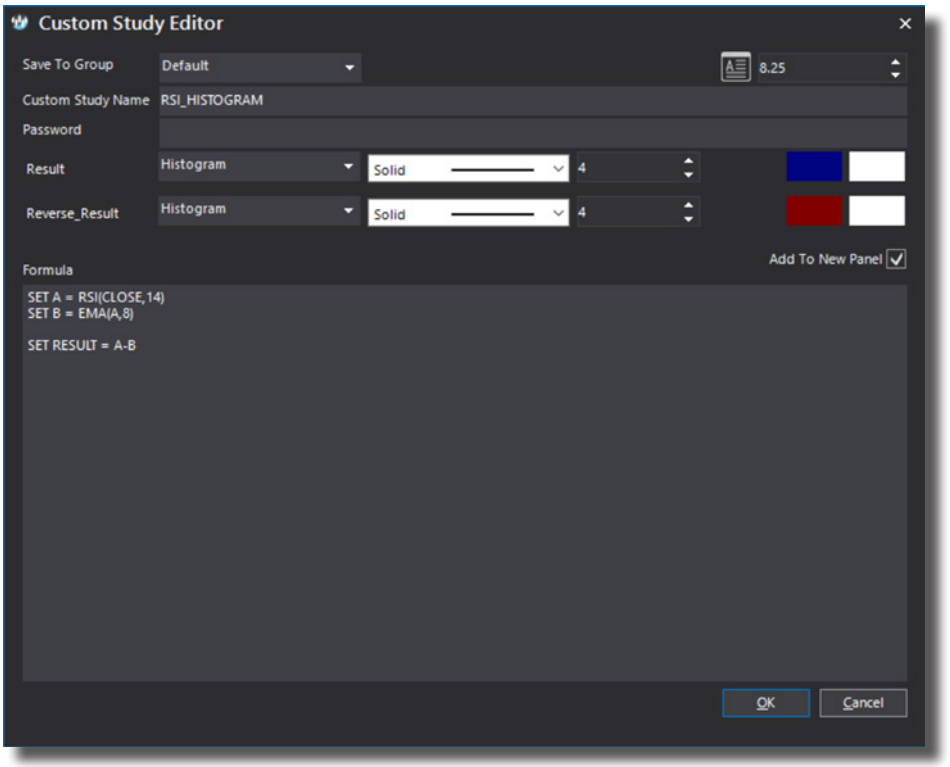
Build Custom Studies by Code Wizard

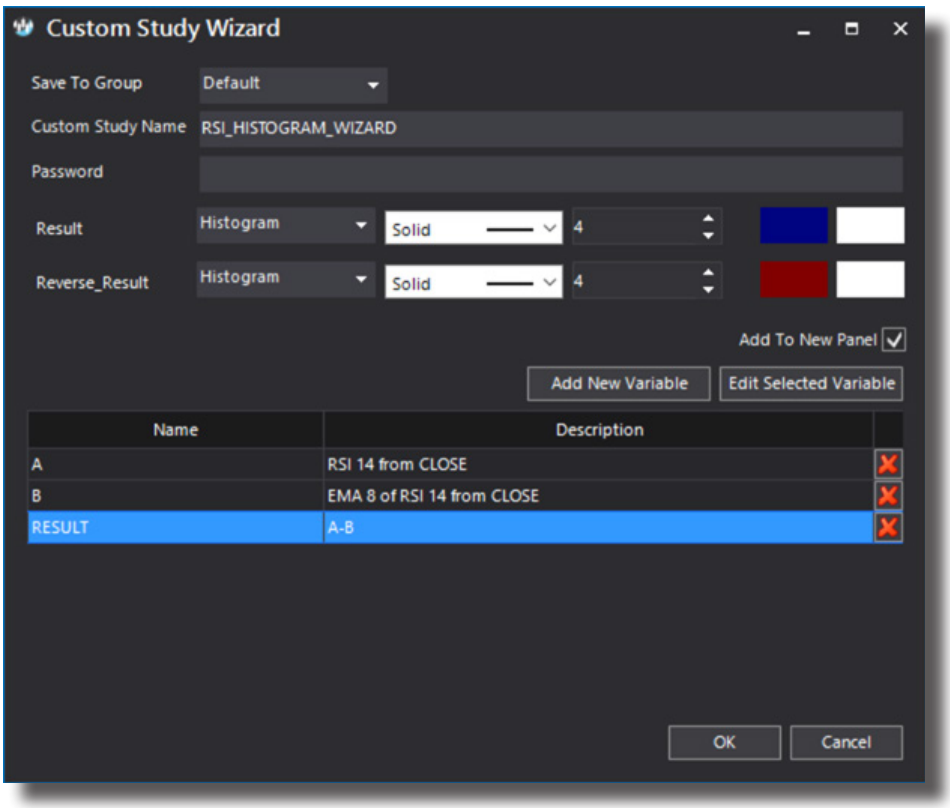


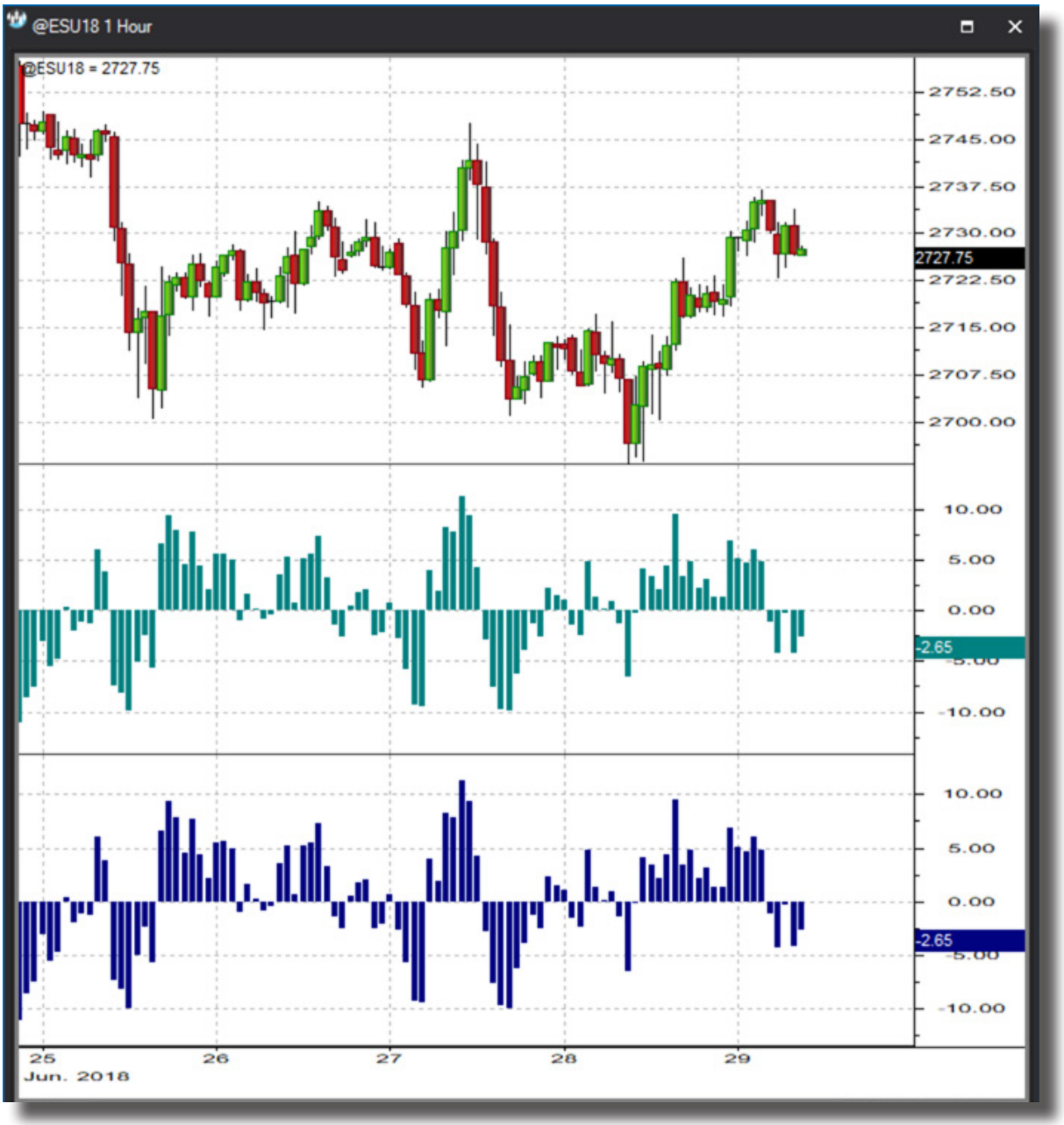
Plot Custom Studies on charts

Plot Custom Studies on charts – RESULT function only

If you are willing to plot only one function, i.e. RESULT, then set your Custom Study as such. This is only advisable for the Custom Studies you plan to use on charts

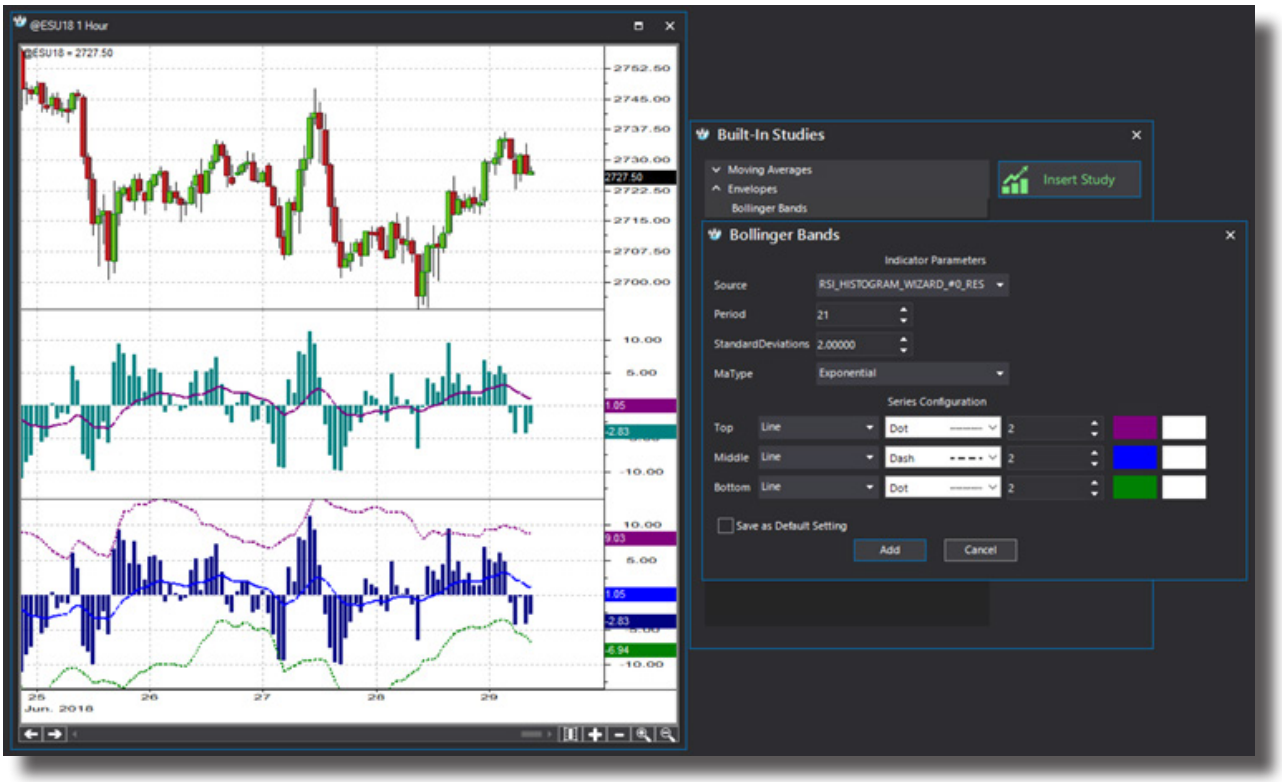





@ESU18 1 Hour
@ESU18 = 2727.75
2752.50
2745.00
2737.50
2730.00
2727.75
2722.50
2715.00
2707.50
2700.00
10.00
5.00
0.00
-2.65
-10.00
10.00
5.00
0.00
-2.65
-10.00
25
26
27
28
29
Jun. 2018

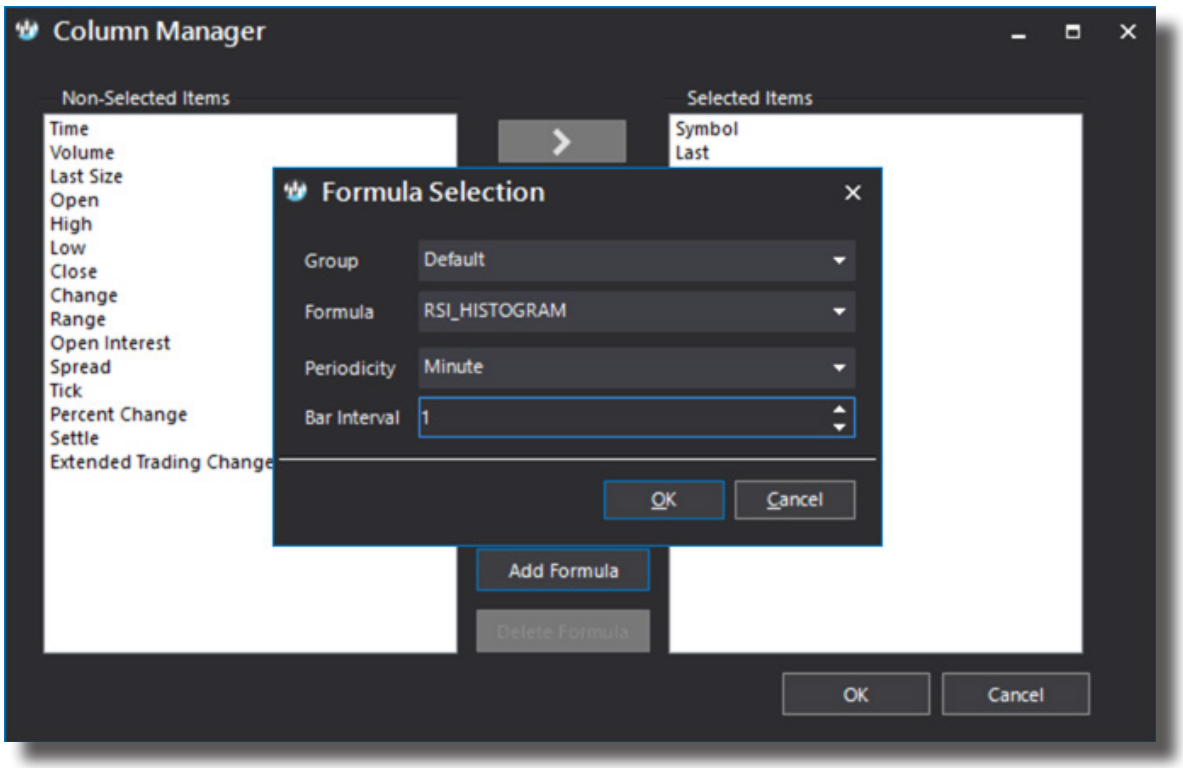
Nest Custom Studies inside other Built-in Studies

Custom studies can be nested, i.e. used as vectors, inside a lot of other built-in function. In the examples below you see an Exponential moving average and a Bollinger Band indicator plotted on the RSI_HISTOGRAM Custom Study built by Code Editor and Code Wizard, respectively.



Run Custom Studies inside Watch List columns

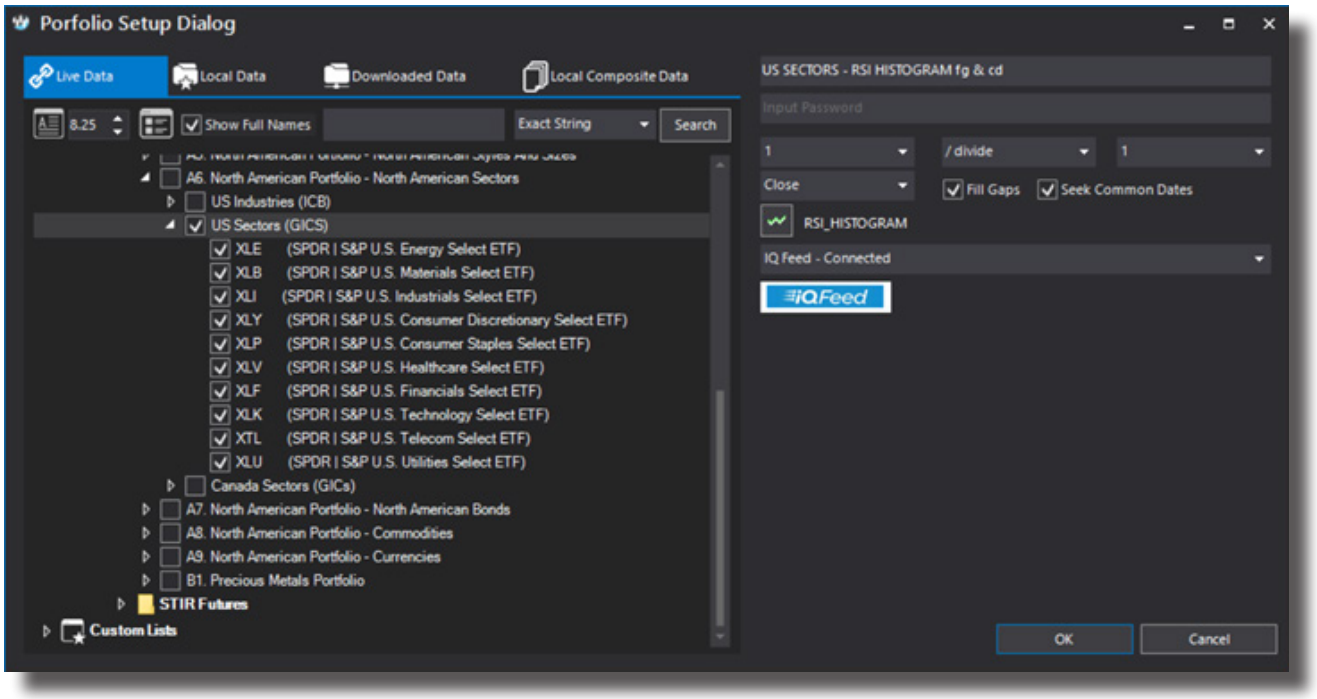
You can calculate any script for any time frame and see its on-the-fly result updated in real time by choosing to run it inside watch list columns:



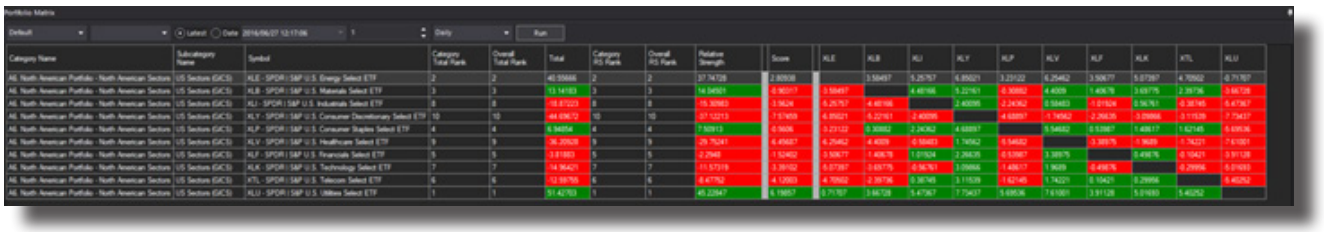
Symbol	Last	BidSize	Bid	Ask	AskSize	RSI_HISTOGRAM 1...
Default						
@ESU18 - E-MINI SP 500 FUTURE SEP 2018	2743.75	275	2743.50	2743.75	148	-1.63
USDCAD.FXCM - FXCM USD CAD Spot	1.31480	0	1.31480	1.31490	0	-1.71953

Use Custom Studies inside Portfolio Systems

You can use Custom Studies inside Portfolio Systems to build matrices and Relative Strength numbers for any portfolio in any time frame:



Overall RS Rank	Relative Strength	Score	XLE	XLB	XLI	XLY	XLP	XLV	XLF	XLK	XTL	XLU
2	37.74728	2.80938		3.58497	5.25757	6.85021	3.23122	6.25462	3.50677	5.07397	4.70502	-0.71707
3	14.04501	-0.90317	-3.58497		4.48166	5.22161	-0.30882	4.4009	1.40678	3.69775	2.39736	-3.66728
8	-15.30983	-3.5624	-5.25757	-4.48166		2.40095	-2.24362	0.58483	-1.01924	0.56761	-0.38745	-5.47367
10	-37.12213	-7.57459	-6.85021	-5.22161	-2.40095		-4.68897	-1.74562	-2.26635	-3.09866	-3.11539	-7.73437
4	7.50913	-0.5606	-3.23122	0.30882	2.24362	4.68897		5.54682	0.53987	1.48617	1.62145	-5.69536
9	-29.75241	-6.45687	-6.25462	-4.4009	-0.58483	1.74562	-5.54682		-3.38975	-1.9689	-1.74221	-7.61001
5	-2.2948	-1.52402	-3.50677	-1.40678	1.01924	2.26635	-0.53987	3.38975		0.49876	-0.10421	-3.91128
7	-11.57319	-3.39102	-5.07397	-3.69775	-0.56761	3.09866	-1.48617	1.9689	-0.49876		-0.29956	-5.01693
6	-8.47752	-4.12003	-4.70502	-2.39736	0.38745	3.11539	-1.62145	1.74221	0.10421	0.29956		-5.40252
1	45.22847	6.19857	0.71707	3.66728	5.47367	7.73437	5.69536	7.61001	3.91128	5.01693	5.40252	



Overall RS Rank	Relative Strength	Score	XLE	XLB	XLI	XLY	XLP	XLV	XLF	XLK	XTL	XLU
8	-15.57599	0.87216		0.97844	-5.34675	-6.21978	-0.9849	-2.35024	-0.29476	-0.91403	-2.61847	2.17451
9	-29.807	2.31537	-0.97844		-2.93416	-3.60842	-4.30458	-5.53698	-2.87355	-3.62952	-6.19036	0.24901
4	3.70932	2.66562	5.34675	2.93416		-1.48357	-5.80712	-6.10198	1.73926	1.13888	-1.00141	6.94436
7	-13.78195	3.75602	6.21978	3.60842	1.48357		-3.10854	-4.13603	-3.96476	-5.05299	-7.96301	-0.86838
6	-5.12321	-0.07285	0.9849	4.30458	5.80712	3.10854		-3.346	-3.78346	-5.12382	-10.38079	3.30571
5	-1.23808	3.52148	2.35024	5.53698	6.10198	4.13603	3.346		-5.86734	-6.81121	-10.14092	0.11016
3	13.96118	3.98041	0.29476	2.87355	-1.73926	3.96476	3.78346	5.86734		-1.60518	-5.03661	5.55836
2	25.55148	5.80789	0.91403	3.62952	-1.13888	5.05299	5.12382	6.81121	1.60518		-4.23782	7.79142
1	61.28638	16.8747	2.61847	6.19036	1.00141	7.96301	10.38079	10.14092	5.03661	4.23782		13.71698
10	-38.98212	-6.74261	-2.17451	-0.24901	-6.94436	0.86838	-3.30571	-0.11016	-5.55836	-7.79142	-13.71698	

Essentially this gives you an opportunity to evaluate universe of investments according to your preferred formulas and algorithms, for any desired list of instruments or securities over any time frame you want.

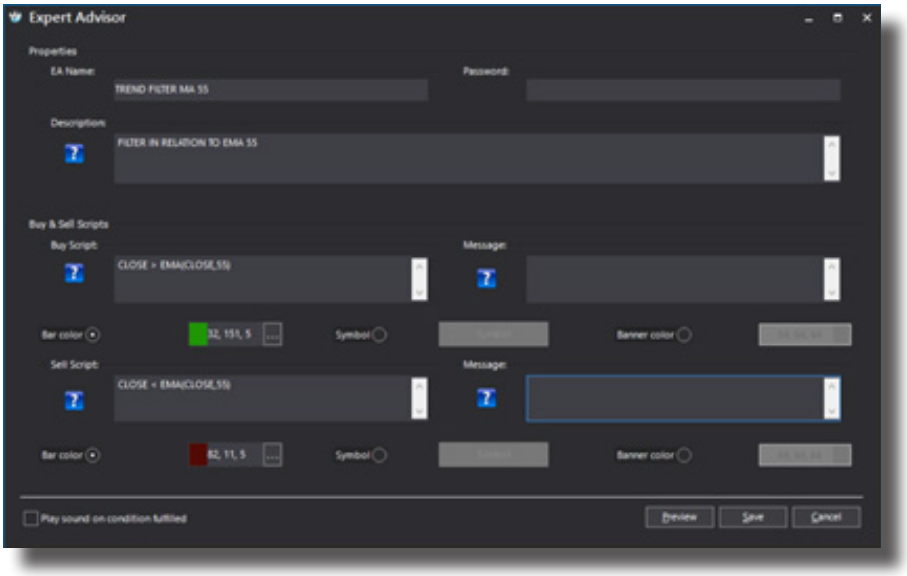
Use Custom Studies inside Loop Systems

Use Custom Studies inside Loop Trading Systems for validation of customized trading algorithms inside totally flexible binomial tree logical sequences;

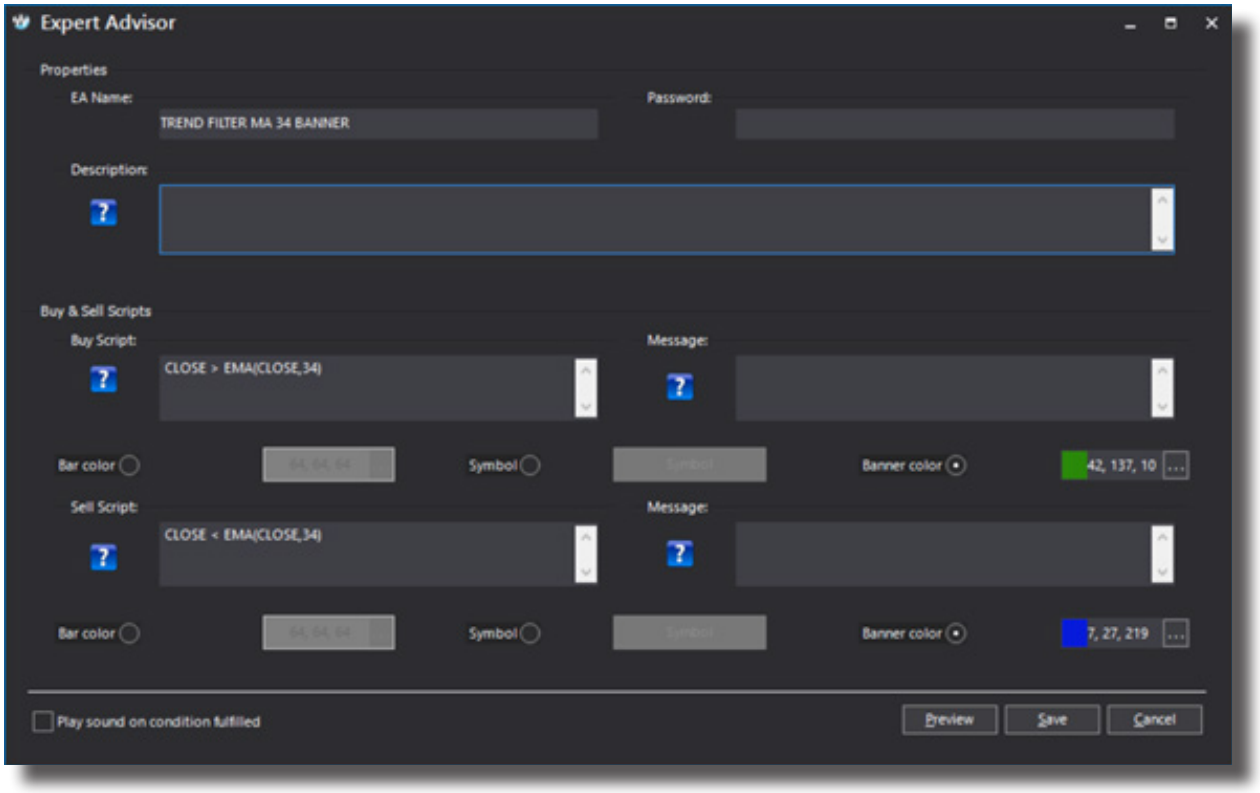
Expert Advisers

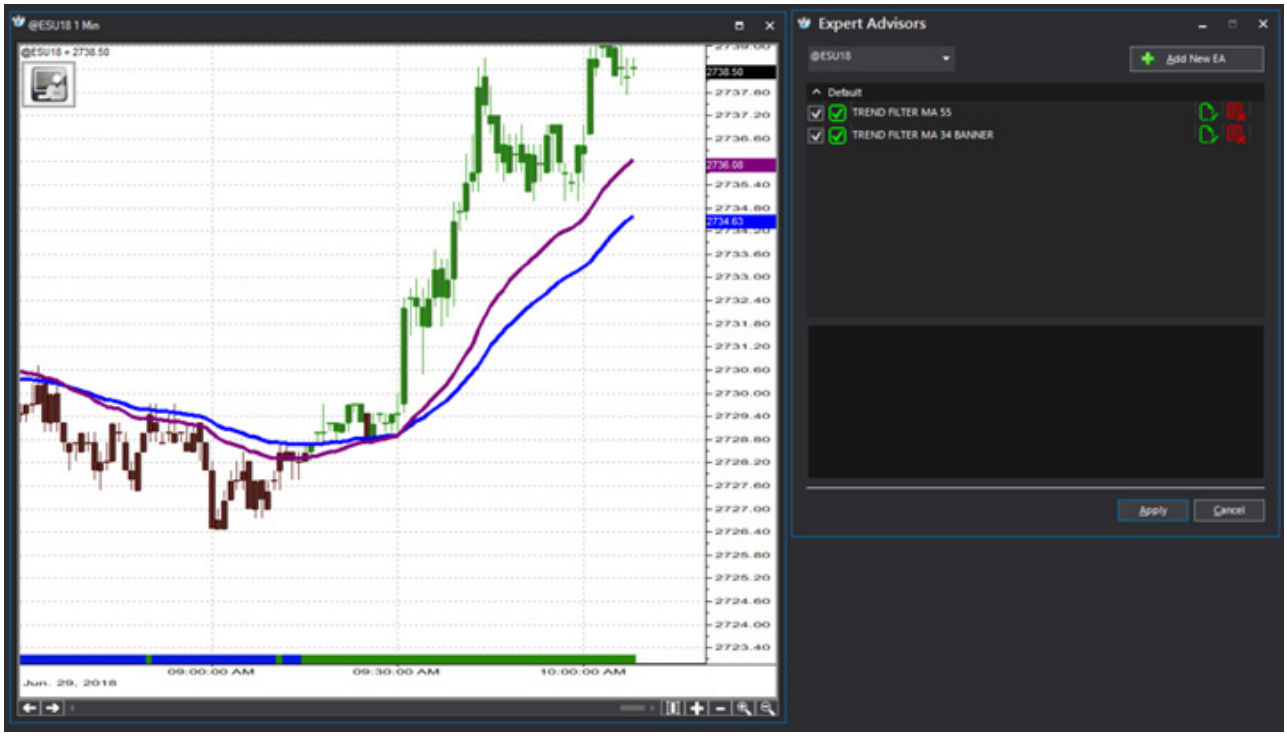
Build filters or triggers to plot on charts

One of the things you can do with expert advisers is visually plot occurrences of certain events on charts. You can, for instance, set trend filters to color your bars with specific colors:

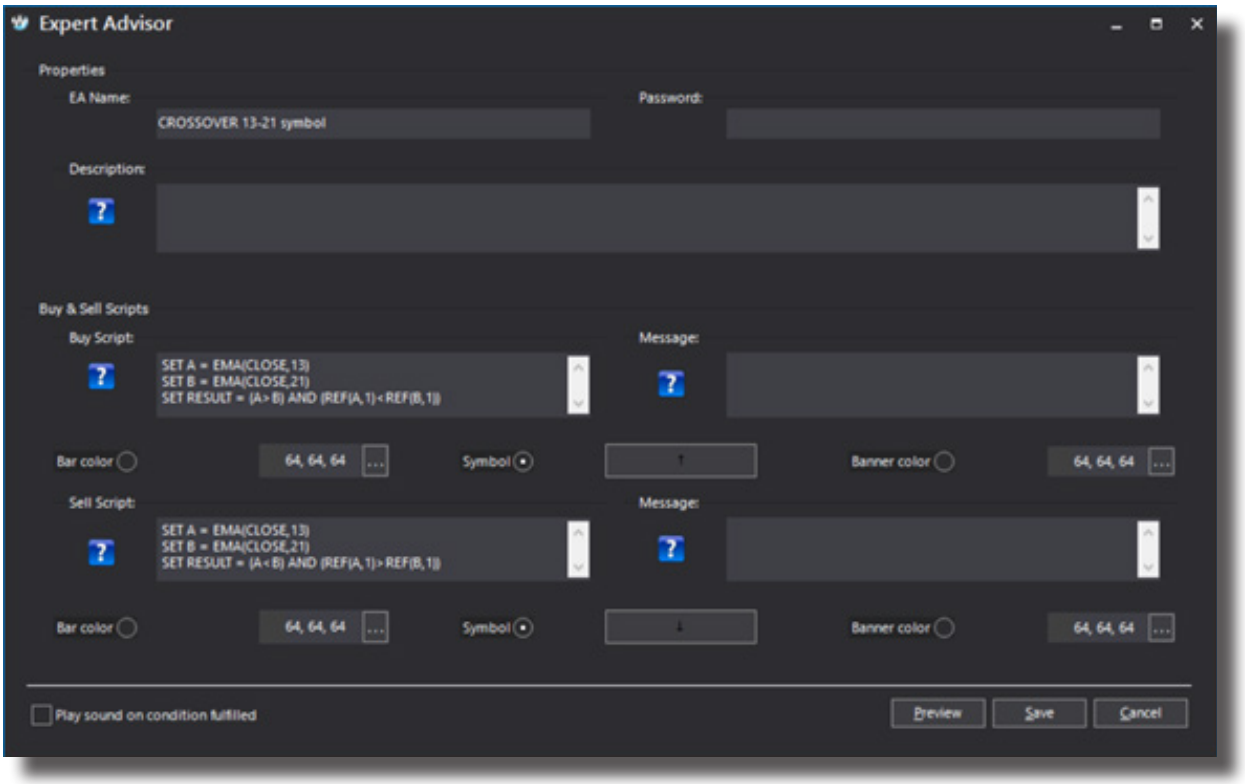


You can also set trend filters to plot at the bottom of the chart as a banner:



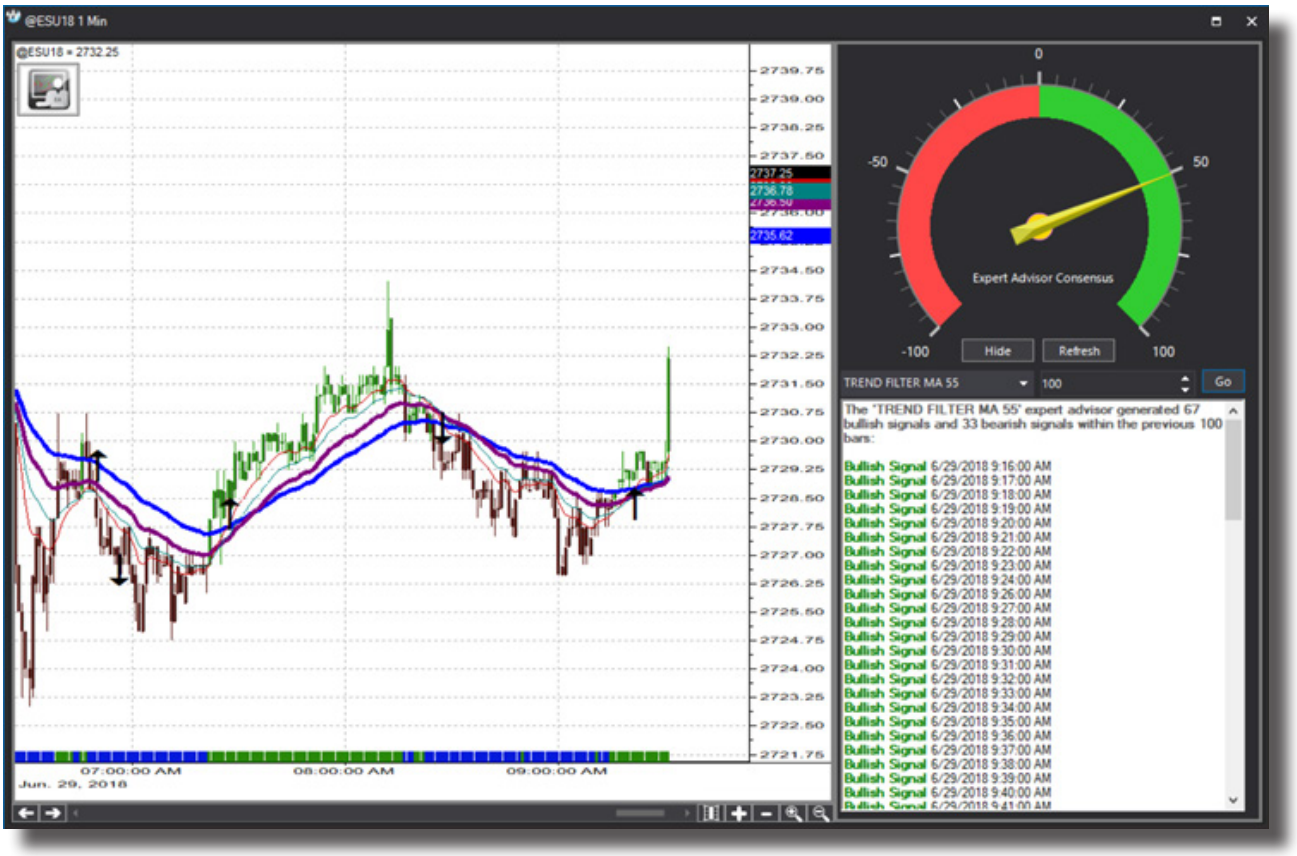


Finally, you can place symbols on charts to illustrate occurrence of events such as crossovers:



Consensus Reports

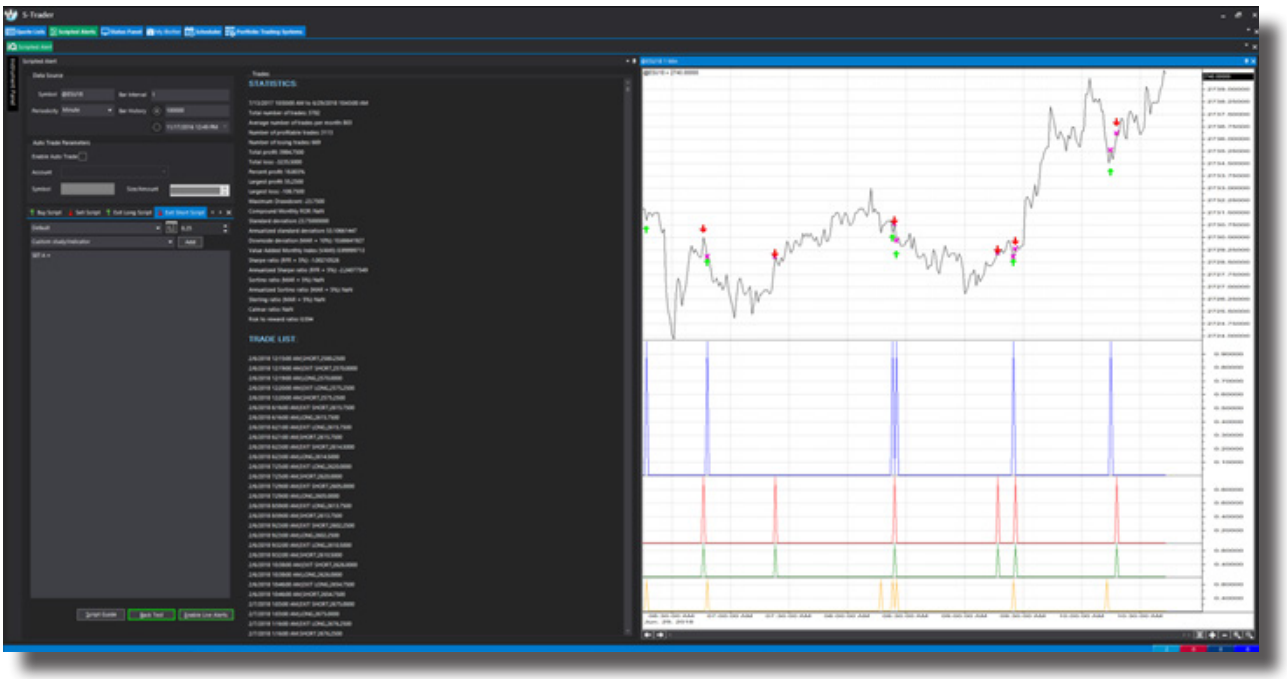
If you want to see the frequency of occurrence of certain events over a specific bar count, you can run a Consensus Report based on any Expert Advisor:



Scripted Alerts

Back-test & Live Run

Build any simple “Long / Short / Exit Long / Exit Short” algorithms with total flexibility and analyze the results over extended period of time:



The exact same way you can choose to run live algorithms you are comfortable with.