EasyScript® User's Guide for beeTrader® v 1.10.10.34
latest update: 20/08/2019

EASYSCRIPT

Table of Contents

Chapter 1

Introduction to EasyScript®

1

Foreword

beeTrader® has been designed to send orders to market in the instant when the script condition is verified, thus no Stop and/or Limit orders will be left unmanaged on broker's servers. **Unless explicitly specified, orders will be sent as "At Market"!**
In this way beeTrader® avoids having orders placed early at market but not filled because the bid/ask quantities are insufficient: the implicit risk is to incur in a higher than anticipated slippage, but the detected trend won't be lost because of pending entry orders.
Another important feature in beeTrader® is the ability to choose if Strategy has to be run "On Close" or "Tick by Tick". Default method is to run Strategy "Tick by Tick", but it's possible to select a different setting using the Advanced Settings button on beeTrader® historical chart Sidebar. Consult beeTrader® User's Guide for an in-depth explanation of all Strategy settings.

Prerequisites

A basic understanding of technical analysis is the only prerequisite for using this programming guide.

How This Guide Is Organized

The first section of this guide contains short examples that demonstrate how to perform common, basic tasks such as identifying securities within 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 EasyScript® editor in beeTrader®.

The last section of this guide contains a reference of functions, properties, and constants supported by the EasyScript® language as well as hands-on trading system examples. This method of organization allows the beginning programmer to see results immediately while learning at his or her own pace.

The EasyScript® Programming Language

EasyScript® is a non-procedural scientific vector programming language integrated in beeTrader® specifically designed for developing trading systems using scripts. A script is simply a set of instructions that tell the EasyScript® engine to do something 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. Possibilities are virtually infinite.
EasyScript® is a "case insensitive" language, it doesn't distinguish between upper case and lower case letters: when developing scripts with it, programmers have to pay particular attention to duplicated variables (for example, a variable named MovingAverage and a second variable named MOVINGAVERAGE will overlap).

Important Concepts

EasyScript® 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.

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 EasyScript® programming language provides a wide variety of ways to program a trading system.

Scripts can be very simple as just shown or extremely complex, consisting of many hundreds of lines of instructions. But for most systems, scripts usually consist of just a few lines of code: having many conditions to satisfy in order to generate a signal implies a restricted number of times the same conditions will be met in the future.

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 bar's volume:

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

It's possible to 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 EasyScript® 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 the
# volume is 2 times the prvious volume
LAST > OPEN OR
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.

Functions

The EasyScript® 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 EasyScript® to identify trades where the closing price is in a 30-periods uptrend. The values that are contained inside a function (such as the REF function or the TREND function) are called arguments. Here there are two arguments in the TREND function. Argument #1 is the closing price, and argument #2 is 30, as in "30 periods" or "30 days".

If not all required arguments are provided to a function, EasyScript® will report an error, unless the function have default parameters values, a feature useful to speed-up scripts creation. In the first case, EasyScript® will highlight what's wrong in the script, let you fix the problem and try again.
In other words, user input errors will never cause EasyScript® to break or return erroneous results without first warning you about a potential problem.

Parameters / Variables

EasyScript® use the concept of user parameters. Using a parameter, the user can inform the software that a value passed to a function is not fixed, but can be changed and needs to be retrieved looking up at the INPUTS lines in the script. Recalling the previous example:

```
TREND (CLOSE, 30) = UP

#using user parameters
INPUTS: @PRICE (CLOSE), @LENGHT (30)
TREND (@PRICE, @LENGTH) = UP
```

In this way, instead of directly entering the periods value (30), it's possible to use a parameter. A parameter is a user-defined name preceded by the @

character, and include a default value, written inside parenthesis. When using a script with user-defined parameters, beeTrader will show all script parameters along with their default value in the software user interface.
Note: this feature is necessary for trading system optimization, allowing the user to specify the range of values to use for each of the parameters of the Signal script.

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 the sum of the median price of a stock over 14 days, in a traditional programming language such as BASIC you would be required to write a program similar to this:

```
For bar = 14 to max
        Sum = 0
        For n = (bar - 14) to bar
                Sum = Sum + (CLOSE + OPEN) / 2
        Next n
Next bar
```

But with EasyScript®, you can effectively accomplish the same thing using just one line:

```
SET Summation = Sum ((CLOSE + OPEN) / 2, 14)
```

And now Summation is actually a new vector that contains the 14-period sum 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.

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 EasyScript® 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 EasyScript® 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. EasyScript® on the other hand references values backwards, from the end. This is most convenient since the purpose of EasyScript® 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)
```

Which 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)
```

Stock 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.
EasyScript® provides a primitive function aptly named TREND especially for detecting trends in stock price, volume, or indicators :

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

This tells EasyScript® to identify trades where the closing price is in a 30-day 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 days
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) = UP
```

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 in 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)
```

```
# 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 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 chapter 3.

Technical Analysis

EasyScript® provides many built-in technical analysis functions. Using only a single line of code you can calculate functions such as Moving Averages, Bollinger Bands, Japanese Candlesticks, and so on.
A complete list of technical analysis functions is covered in chapter 5.

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 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 use the Median vector inside any function that requires a vector:

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

And this evaluates 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.

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 on the right. 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 find 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)
```

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)
```

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 EasyScript® 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. Key Reversals do not occur frequently but they are very reliable when they do occur. 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 >= HighestHighValue (30)
# And the last price must be below yesterday's low
AND LAST < REF (LOW, 1)
```

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 <= LowestLowValue
# 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.

Chapter 2

Primitive Variables, Constants & Parameters

2

This chapter serves as a look-up reference for primitive variables, constants, flags, candlestick patterns, and scripts parameters.

Price Vector

Basic price data such as open, high, low, close and volume can be used as a vector by any function or expression.

OPEN	Opening price of the period
HIGH	Highest price of the period
LOW	Lowest price of the period
CLOSE	Closing price of the period
HAOPEN	Heikin Ashi – Opening price of the period
HAHIGH	Heikin Ashi – Highest price of the period
HALOW	Heikin Ashi – Lowest price of the period
HACLOSE	Heikin Ashi – Closing price of the period
LAST	The last price (same value as CLOSE if after trading hours)
VOLUME	Trading volume of the period
TICKSCOUNT	Number of trades of the period
BARNUMBER	Progressive period number, starting from 0

Time Vectors

Basic time related data such as the date, the time or the day of the week can be used as a vector by any function or expression.

DATE	Closing date of the period, format YYYYMMDD
TIME	Closing time of the period, format is variable: HHMM for times higher than 10:00; HMM for times lower than 10:00; MM for times lower than 01:00; M for times lower than 00:10.
DAY	Day of the month, in numeric format (1, 2, 3, 4, …, 31)
MONTH	Month of the year, in numeric format (1, 2, 3, 4, …, 12)
YEAR	Year, format YYYY (2018, 2019, 2020, ….)
DAYOFWEEK	Day of the week, in numeric format. The value is determined with the following table:

	SUNDAY: 0 MONDAY: 1 TUESDAY: 2 WEDNESDAY: 3 THURSDAY: 4 FRIDAY: 5 SATURDAY: 6
TODAY	Current date, format YYYYMMDD

Colors Constants

Colors constants represents the predefined colors to use in your own Indicators scripts. There are 22 predefined colors:

COLOR_BLACK	Black
COLOR_WHITE	White
COLOR_SILVER	Light Gray, Silver
COLOR_GRAY	Dark Gray
COLOR_LIGHT_RED	Light Red
COLOR_RED	Red
COLOR_DARK_RED	Dark Red
COLOR_LIGHT_GREEN	Light Green
COLOR_GREEN	Green
COLOR_DARK_GREEN	Dark Green
COLOR_LIGHT_BLUE	Light Blue
COLOR_BLUE	Blue
COLOR_DARK_BLUE	Dark Blue
COLOR_LIGHT_YELLOW	Light Yellow
COLOR_YELLOW	Yellow
COLOR_DARK_YELLOW	Dark Yellow
COLOR_LIGHT_CYAN	Light Cyan
COLOR_CYAN	Cyan
COLOR_DARK_CYAN	Dark Cyan
COLOR_LIGHT_MAGENTA	Light Magenta
COLOR_MAGENTA	Magenta
COLOR_DARK_MAGENTA	Dark Magenta

Chromatic Constants

The chromatic constants are used in the COLORIZE function to determine chromatic gradient direction.

COLORIZE_ASCENDING	Ascending
COLORIZE_DESCENDING	Descending

Basic Constants

Basic constants for the most common mathematical values.

TRUE	1
FALSE	0
PI **PI_M** **PI_MATH**	3.14159265358979323846264338327950
NULL	An empty data vector
NAN	Not a Number

Operators Constants (used in LOOP and BARLOOP functions)

Operator constants can be used to determine the mathematical operation to perform in a LOOP or BARLOOP function.

ADD	Addition
SUBTRACT	Substract
MULTIPLY	Multiplication
DIVIDE	Division
MAXIMUM	Highest Value
MINUMUM	Lowest Value

Moving Averages Constants

Moving averages constants can be used to determine which kind of moving average a function must perform. Functions with configurable moving average type usually have a parameter named MATYPE for this purpose.

SIMPLE	1 (Simple Moving Average)
EXPONENTIAL	2 (Exponential Moving Average)
TIME_SERIES	3 (Time Series Moving Average)
VARIABLE	4 (Variable Moving Average)
TRIANGULAR	5 (Triangular Moving Average)

WEIGHTED	6 (Weighted Moving Average)
VOLATILITY	7 (VIDYA Moving Average)
WILDER	8 (Welles Wilder Smoothing)
DOUBLE	9 (Double Exponential Moving Average)
TRIPLE	10 (Triple Exponential Moving Average)

Trend Constants (used in the TREND function)

UP	1 (Uptrend)
DOWN	2 (Downtrend)
SIDEWAYS	3 (Sideways)

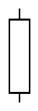
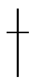
Points or Percent Constants (used in Volume Oscillator indicator)

POINTS	1 (indicator output is expressed in points)
PERCENT	2 (indicator output is expressed in percent)

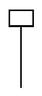
Interpolation Constants (used in INTERPOLATE Function)

LINEAR	1 (Linear Interpolation)
SPLINE	2 (Bi-cubic spline Interpolation)

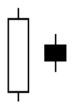
Candlestick Patterns Constants

LONG_BODY	
DOJI	

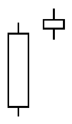
HAMMER



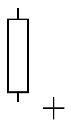
HARAMI



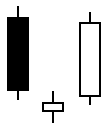
STAR



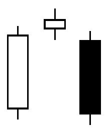
DOJI_STAR



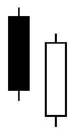
MORNING_STAR



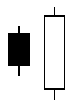
EVENING_STAR



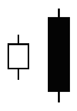
PIERCING_LINE



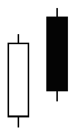
BULLISH_ENGULFING_LINE



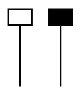
BEARISH_ENGULFING_LINE



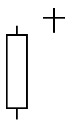
DARK_CLOUD_COVER



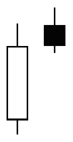
HANGING_MAN



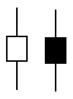
BEARISH_DOJI_STAR



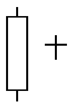
BEARISH_SHOOTING_STAR



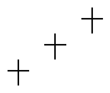
SPINNING_TOPS



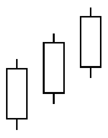
HARAMI_CROSS



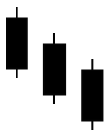
BULLISH_TRISTAR



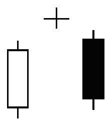
THREE_WHITE_SOLDIERS



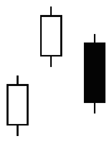
THREE_BLACK_CROWS



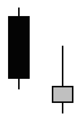
ABANDONED_BABY



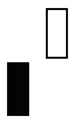
BULLISH_UPSIDE_GAP



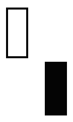
BULLISH_HAMMER



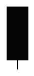
BULLISH_KICKING



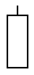
BEARISH_KICKING



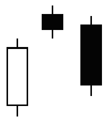
BEARISH_BELT_HOLD



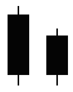
BULLISH_BELT_HOLD



BEARISH_TWO_CROWS



BULLISH_MATCHING_LOW



Function Scripts Variables

RETURN	The RETURN variable can be used to return the result of a user-defined Function. Alternatively, it's possible to use a variable with the same name as the of the Function itself. Example: **SET RETURN = SMA (CLOSE, 10) - SMA (CLOSE, 21)** The Function returns the difference between the 10-periods simple moving average and the 21-periods simple moving average.

Signal or Indicator Scripts Parameters

REQUIRED_BARS	The REQUIRED_BARS parameter can be used to define the number of bars used to initialize the scripts calculation. Default value is 50. Example: **SET REQUIRED_BARS = 250** Sets the number of bars required to inizialize script calculations to 250 bars.

Money Management Parameters

Money Management parameters must be specified at the start of the Signal scripts, before any other calculation. These parameters can be preceded only by Inputs specifications.

Example:

```
# Correct code

SET STOP_LOSS = 100

CLOSE < MIN (CLOSE, 30)
```

```
# Wrong code

CLOSE < MIN (CLOSE, 30)

SET STOP_LOSS = 100
```

MAX_POSITION_OPEN	When configured with the SET keyword, all open positions, long and short, will be closed after the specified number of periods. Example: **SET MAX_POSITION_OPEN = 20** Once the open position reaches a duration of 20 periods since entry, that position will be closed.
STOP_LOSS **STOP_LOSS_PERCENT**	When configured with the SET keyword, open positions will be closed whenever the loss reaches the specified amount/percent. Percent is calculated from the entry price. Example: **SET STOP_LOSS = 100** When the open position reaches a loss of 100, it will be closed.
TAKE_PROFIT **TAKE_PROFIT_PERCENT**	When configured with the SET keyword, open positions will be closed whenever the profit reaches the specified amount/percent. Percent is calculated from the entry price. Example: **SET TAKE_PROFIT = 100** When the open position reaches a profit of 100, it will be closed.
TRAILING_STOP & **TRAILING_PERCENT** **NOTE: These parameters must always be used together**	When both these parameters are configured using the SET keyword, if the open position profit reaches the TRAILING_STOP value an exit price is calculated using the TRAILING_PERCENT value. Example: **SET TRAILING_STOP = 400**

	SET TRAILING_PERCENT = 50 When the open position reaches a profit of 400, the exit price is calculated to maintain a minimum gain of 50%, equal to 200. If profit decreases, when it reaches 200 the open position will be closed. If instead the profit increase, a new exit price is recalculated using the same procedure as before, but starting from a higher initial profit, and the process restarts.

CONTRACTS	When configured with the SET keyword, the generated Signals will Buy or Sell the specified number of contracts, regardless of sidebar settings or current open position. If configured with 0, than the value specified in the sidebar or the number of open contracts will be used. This parameter can be especially useful when pyramiding is enabled.
	
	Example:
	Buy Script
	`# Use 5 contracts in all Buy operations`
	SET CONTRACTS = 5
	ExitLong Script
	`# Use number of contracts specified in sidebar for Exit Long operations`
	SET CONTRACTS = 0
	In this example, if the quantity specified in the sidebar is 1 contract, Buy operations will use 5 contracts at a time, while Exit Long operations will use just 1 contract at a time.

User-defined Input Variables

```
#User-defined inputs
INPUTS: @USERVAR1 (VALUE1), @USERVAR2 (VALUE2)

#User-defined inputs with predefined optimization range
INPUTS: @USERVAR1 (VALUE1 [,FROM [,TO [,STEP]]]), @USERVAR2 (VALUE2)
```

The keyword INPUTS is used to declare user-defined input variables that can be used as parameters in EasyScript® functions and instructions.
Input variables are identified by a name starting with the @ character, and must always provide a default value enclosed in braces after the name. Multiple input variables can be declared in a single line, using the comma character as separator.
Every input variable in Indicators, Alerts, Experts and Signals scripts is displayed to the user in beeTrader® when applying the script to a chart, so input variables values can be tailored to each specific use case. In case of Signal scripts used in a beeTrader® chart for Backtest, input variables can be optimized to achieve the best trading results. Optimization can take place ONLY on numeric inputs.
Note: if you have a long list of inputs variables, you can split it in multiple lines, adding the INPUTS keyword to each line.

Example:

```
# TREND function with fixed parameters
TREND (CLOSE, 30) = UP

# TREND function with variable parameters
INPUTS: @PRICE (CLOSE), @LENGTH (30)
TREND (@PRICE, @LENGTH) = UP
```

In the above example, whenever the @PRICE variable is encoutered, the script engine will use the CLOSE variable, and whenever the @LENGTH variable is encountered, the script engine will use the value 30.

Chapter 3

Mathematical and Logical Operators

3

This chapter introduces the Mathematical and Logical Operators available in EasyScript®.

Mathematical Operators

Multiplication (*)

The multiplication operator is used to multiply two vectors, two numerical values, or a vector and a numerical value.

Division (/ or \)

The division operator is used to divide two vectors, two numerical values, or a vector and a numerical value.

Power (^)

The power operator is used to elevate to power a vector or a numerical value. Exponent can be itself a vector or a numerical value.

Substraction (-)

The substraction operator is used to substract two vectors, two numerical values, or a vector and a numerical value.

Sum (+)

The sum operator is used to sum two vectors, two numerical values, or a vector and a numerical value.

Logical Operators

Equal (=)

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. **The SET keyword must precede the variable name when the = operator is used for an assignment**, as in the following example:

```
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 Then (<)

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 (>=)

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 (<=)

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
```

And:

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

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 "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 operator 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
```

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
```

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
```

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
```

Chaper 4

Primitive and Mathematical Functions

4

This chapter covers the built-in functions of EasyScript®, also known as primitives. These important functions define the EasyScript® 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 EasyScript® 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 EasyScript® 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.

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

Primitive Functions

AVG

AVG (@vector, @periods)

Returns a vector containing a running average, as specified by the Periods argument. The AVERAGE function can also be referenced by AVG for short.

Example:

AVG (CLOSE, 10)

The script returns a vector of averages based on a 10-period window.

AVGOF

AVGOF (@vector1, @vector2, [@vector3], …, [@vector8])

Returns a vector containing a point-per-point average of the specified vectors. Up to 8 vectors can be specified.

Example:

AVGOF (CLOSE, REF(CLOSE, 1) , REF(CLOSE, 2), REF(CLOSE, 3), REF(CLOSE, 4))

The script returns a vector with the averager of today's CLOSE, yesterday's CLOSE, 2 days ago CLOSE, and 3 days ago CLOSE.

BARLOOP

BARLOOP (@initialValue, @offset, @operator, @operand, @minimumValue, @maximumValue)

Returns a numerical value calculated progressively bar per bar. The *initialValue* parameter specifies the initial value to use to prime the output result: it can be any numerical value. The *offset* parameter specifies the number of bars to look-back starting from the current calculation bar, and must be a positive value. The *operator* parameter identifies the mathematical operator to use in the calculation: available operators are ADD, SUBSTRACT, MULTIPLY and DIVIDE (see LOOP function for an extended explanation). The *operand* parameter specifies the second operand of the mathematical operation to perform, can be any expression (numerical value, the result of another function, etc.). The parameters *minimumValue* and *maximumValue* are optional parameters used to limit the results to a range of values: it's possible to set these parameters to NAN (Not-A-Number) to avoid performing output range limitation.

Example:

```
SET EntryLB = BARLOOP (20, 1, MULTIPLY, (1 + DeltaHistVol), MinLB, MaxLB)
```

The result will be a numerical value progressively calculated, bar per bar. The calculation uses 20 as initial value, the number 1 indicates the previous bar data will be used. The calculation is a multiplication between the current value and the previous bar value of (1 + DeltaHistVol). The results are range limited between MinLB and MaxLB.

CHANGEIF

```
CHANGEIF (@condtion, @vector)
```

Returns a vector containing a copy of the @vector input when the condition evaluates to true, and the previous bar output when the condition evaluates to false.

Example:

```
CHANGEIF (CLOSE > OPEN, HIGH)
```

The result will be a vector containing the value HIGH if CLOSE > OPEN, or the previous calculated value otherwise.

Example:

```
SET PREV = REF(HIGH, 1)
CHANGEIF (HIGH > PREV, HIGH)
```

In the above example, EasyScript® uses the current bar HIGH when it's raising from the previous bar. Paired with similar instructions for the LOW price, it can be used to identify price channels.

NOTE: the CHANGEIF function can contain only a single logical condition, thus the condition expression cannot contain AND, OR, NOT or other logical operators. The logical condition doesn't support nesting.

COUNTIF

```
COUNTIF (@condition)
```

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

Example:

```
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.

NOTE: the COUNTIF function can contain only a single logical condition, thus the condition expression cannot contain AND, OR, NOT or other logical operators. The logical condition doesn't support nesting.

CROSSOVER

```
CROSSOVER (@vector1, @vector2)
```

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 find 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)
```

The script above will evaluate to true when the MA1 vector has crossed over the MA2 vector.
We can reverse the script to the MA1 vector crossed *below* the MA2 vector, swapping the inputs parameters or using the CROSSUNDER function:

```
# These two lines are equivalent
CROSSOVER (MA2, MA1)
CROSSUNDER (MA1, MA2)
```

CROSSUNDER

```
CROSSUNDER (@vector1, @vector2)
```

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

```
SET MA1 = SimpleMovingAverage (CLOSE, 28)
SET MA2 = SimpleMovingAverage (CLOSE, 14)
```

```
CROSSUNDER (MA1, MA2)
```

The script above will evaluate to true when the MA1 vector has crossed under the MA2 vector.
We can reverse the script to the MA1 vector crossed *above* the MA2 vector, swapping the inputs parameters or using the CROSSOVER function:

```
# These two lines are equivalent
CROSSUNDER (MA2, MA1)
CROSSOVER (MA1, MA2)
```

FOLLOWME

```
FOLLOWME ()
```

Returns a vector containing the results of the FOLLOWME function, which are always in one of the following two ranges:
For up-trends, the results will be in the range from +50 to +100;
For down-trends, the results will be in the range from -50 to -100.
The numerical value represents the strength of the trend, the sign represent the direction.

Example:

```
FOLLOWME () > 0
```

The above script returns TRUE when the FOLLOWME function is in up-trend territory.

IF

IF (@condition, @truePart, @falsePart)

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

NOTE: the IF function can contain only a single logical condition, thus the condition expression cannot contain AND, OR, NOT or other logical operators. The logical condition doesn't support nesting.

LASTIF

LASTIF (@condtion)

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

Example:

```
LASTIF (CLOSE < OPEN)
```

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.

NOTE: the LASTIF function can contain only a single logical condition, thus the condition expression cannot contain AND, OR, NOT or other logical operators. The logical condition doesn't support nesting.

LOOP

LOOP (@vector1, @vector2, @offset1, @offset2, @operator)

LOOP provides simulated control structure by means of a single function call. Consider the following:

```
SET X = CLOSE
SET X = REF(X, 1) + X
```

This script simply ads the previous close to the most current close. REF(X, 1) is evaluated once. This is expected behavior for a vector programming language: vectors are calculated independently in a step-wise fashion and are not recursive.
Now by changing CLOSE to 0, logically we would expect X to equal the previous X value plus one, and therefore expect REF(X, 1) to be evaluated once for each record in the vector:

```
SET X = 0
SET X = REF(X, 1) + X
```

Although we are looking at the exact same formula, because we are initializing X with a scalar and X is not related to any existing vector we would now expect X to be calculated as a series: 1,2,3,4,5,6,...n
We are now exceeding the limits of a vector programming language by requiring control structure.

Anytime we assign a variable to itself such as SET X = F(X) we are expecting F(X) to be recursive. In the first example we write SET X = CLOSE. CLOSE is a variable, not a function and does not have any relationship with X. Our expectations change when we initialize X with anything other than an existing vector.

The LOOP function overcomes this inherent limitation by simulating a structured programming construct, the for-loop iteration:

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

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:
X (Vector1) is a series from 5.25 to 11.25. If we write:

```
LOOP(X, 2, 1, 0, MULTIPLY)
```

the vector returned will contain values initialized by X, offset by 1 and multiplied by 2:

X	LOOP
5.25	5.25
6.25	10.5
7.25	21
8.25	42
9.25	84
10.25	168
11.25	336

Example 2:
The LOOP function can be used to generate a vector containing a series. If we write:

```
SET X = LOOP(X, 1, 1, 0, ADD)
```

the results will be the series 1, 2, 3, 4, 5, 6, … n.

Example 3:
Consider the following script:

```
SET X = REF(CLOSE,1)
SET Y = (REF(Y, 3) - X) * 2
```

Because Y requires control structure we must instead write:

```
SET X = REF(CLOSE,1)
SET Y = LOOP(Y, X, 3, 0, SUBTRACT) * 2
```

We could reduce that to:

```
SET Y = LOOP(Y, CLOSE, 3, 1, SUBTRACT) * 2
```

Valid operators are **ADD**, **SUBTRACT**, **MULTIPLY** and **DIVIDE**.

MAX

MAX (@vector, @periods)

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

Example:

MAX (CLOSE, 10)

Returns a vector of maximum values based on a 10-period window.

MAXOF

MAXOF (@vector1, @vector2, [@vector3], …, [@vector8])

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.

Example:

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

MIN (@vector, @periods)

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

Example:

MIN (CLOSE, 10)

Returns a vector of minimum values based on a 10-period window.

MINOF

MINOF (@vector1, @vector2, [@vector3], …, [@vector8])

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.

Example:

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.

NTHMAXOF

NTHMAXOF (@position, @vector1, [@vector2], [@vector3], …, [@vector7])

Returns a vector containing the nth maximum value of all specified vectors, for up to 7 vectors. Vector1 and Vector2 are required and vectors 3 through 7 are optional. The Position parameter identifies the required nth value.

Example:

NTHMAXOF (2, CLOSE, REF(CLOSE, 1) , REF(CLOSE, 2))

Returns a vector containing the second most higher value between CLOSE, CLOSE of 1 bar ago and CLOSE of 2 bars ago.

NTHMINOF

NTHMINOF (@position, @vector1, [@vector2], …, [@vector7])

Returns a vector containing the nth minimum value of all specified vectors, for up to 7 vectors. Vector1 and Vector2 are required and vectors 3 through 7 are optional. The Position parameter identifies the required nth value.

Example:

NTHMINOF (2, CLOSE, REF(CLOSE, 1) , REF(CLOSE, 2))

Returns a vector containing the second most lowest value between CLOSE, CLOSE of 1 bar ago and CLOSE of 2 bars ago.

PRINT

PRINT (@vector1, [@vector2], [@vector3], …, [@vector8])

Writes the contents of the specified vectors to the Debug Window of EasyScript Editor. Vector1 is required, vectors 2 through 8 are optional.

Example:

```
SET A = CLOSE
PRINT(A)
```

Writes the A variable (equal to CLOSE) to the Debug Window.

NOTE: it's strongly suggested to use the PRINT function in a single sub-script at a time in the case of Expert Advisor or Signal scripts (Buy Script or Sell Script for Expert Advisors; Buy Script, Sell Script, ExitLong Script or ExitShort Script for Signals).

REF

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 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.

SUM

```
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.

Example:

```
SUM (CLOSE, 10)
```

The script returns a vector of sums based on a 10-period window.

SUMIF

```
SUMIF (@condition, @vector)
```

Last in the "IF" function lineup is the SUMIF function. 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 the days where the closing price closed up 5%, we could write:

```
SET VALORE = REF(CLOSE,1) * 1.05
SUMIF (CLOSE > VALORE, VOLUME)
```

The result will be a vector containing a running sum of volume for each day where the closing price closed up at least 5%.

NOTE: the SUMIF function can contain only a single logical condition, thus the condition expression cannot contain AND, OR, NOT or other logical operators. The logical condition doesn't support nesting.

SUMOF

```
SUMOF (@vector1, @vector2, [@vector3], …, [@vector8])
```

Returns a vector containing the sum of all specified vectors. Vector1 and Vector2 are required, vectors 3 through 8 are optional.

Example:

```
SUMOF (CLOSE, REF(CLOSE, 1) ,  REF(CLOSE, 2), REF(CLOSE, 3), REF(CLOSE, 4))
```

The script returns a vector containing the sum of the CLOSE prices of the last 5 bars.

TREND

TREND (@vector, @periods, [@percent])

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, and needs a period on which to calculate the result. The Percent parameter is optional, and identifies the minimum difference between two consecutive values required to determine the trend, in percentage: if not supplied the default value of 0.5 is used.
The TREND function returns a constant of either UP, DOWN or SIDEWAYS.

Example:

```
TREND (CLOSE, 30, 0.1) = UP AND TREND (VOLUME, 30) = DOWN
```

Returns TRUE if the TREND of CLOSE in the last 30 periods is UP by at least 0.1% and at the same time the TREND of VOLUME in the last 30 periods is DOWN by at least 0.5%.
TREND is often the first function used as a means of filtering securities that are not trending in the desired direction.

Example:

```
SET T = TREND (CLOSE, 30)
SET PLOT1 = IF (T = UP, 1, 0)
SET PLOT1 = IF (T = DOWN, -1, PLOT1)
```

In the above example TREND is used to build an Indicator, outputting 1 if the TREND is UP, -1 if the TREND is DOWN, or 0 if the TREND is SIDEWAYS.

TRUEHIGH

TRUEHIGH ()

Returns a vector containing the highest value between the CLOSE of the previous bar and the HIGH of the current bar.

Example:

```
TRUEHIGH () > SMA (CLOSE, 14)
```

Returns TRUE if the TRUEHIGH is higher than the 14-period Simple Moving Average on CLOSE.

TRUELOW

TRUELOW ()

Returns a vector containing the lowest value between the CLOSE of the previous bar and the LOW of the current bar.

Example:

TRUELOW () < SMA (CLOSE, 14)

Returns TRUE if the TRUELOW is lower than the 14-period Simple Moving Average on CLOSE.

Mathematical Functions

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.

ABS or POS

```
ABS (@vector)
POS (@vector)
```

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

Example:

```
ABS (CLOSE – OPEN)
POS (CLOSE – OPEN)
```

The script always evaluates to a positive number, even if the opening price is greater than the closing price.

ABSOLUTEMAXIMUM

```
ABSOLUTEMAXIMUM(@vector)
```

The Absolute Maximum function returns the highest value in the entire vector.

Example:

```
ABSOLUTEMAXIMUM(HIGH)
```

The script returns the highest high calculated on all available bars.

ABSOLUTEMINIMUM

ABSOLUTEMINIMUM(@vector)

The Absolute Minimum function returns the lowest value in the entire vector.

Example:

ABSOLUTEMINIMUM(LOW)

The script returns the lowest low calculated on all available bars.

ARCCOS

ARCCOS (@vector)

The function ARCCOS returns the arccosine of a number.

Example:

ARCCOS (0.707)

The script returns the value PI/4 (0.785).

ARCSIN

ARCSIN (@vector)

The function ARCSIN returns the arcsine of a number.

Example:

ARCSIN (0.707)

The script returns the value PI/4 (0.785).

ARCTAN

```
ARCTAN (@vector)
```

The function ARCTAN returns the arctangent of a number.

Example:

```
ARCTAN (1)
```

The script returns the value PI/4 (0.785).

CEIL

```
CEIL (@vector)
```

The CEIL function returns the input value rounded-up to the next higher integer.

Example:

```
CEIL (13.2)
```

The script returns 14.

COS

```
COS (@vector)
```

The function COS returns the cosine of an angle. The angle must be expressed in radians.

Example:

```
COS (PI/4)
```

The script returns 0.707.

DEGTORAD

DEGTORAD (@degrees)

The DEGTORAD function converts the value of an angle from degrees to radians.

Example:

```
DEGTORAD(45)
```

The script returns PI/4 (0.785).

DIV

DIV (@numerator, @denominator)

The DIV function performs the integer division. The DIV function requires two parameters, Numerator and Denominator.

Example:

```
DIV (12, 5)
```

The script returns 2, the integer part of the result of 12 divided by 5.

EXP

EXP (@vector)

The EXP function raises e to the power of a number (e^X). The LOG function is the reverse of this function.

Example:

```
EXP (3.26)
```

The script returns 26.28

FIRSTVALIDVALUE

FirstValidValue (@vector)

The First Valid Value functions evaluates the entire input vector looking-up for the first non null value, starting from the first bar.

Recommended Parameters
Vector: [Any Vector]

Example:

FirstValidValue(SMA(CLOSE, 14))

The above script returns the first calculated 14-period Simple Moving Average value.

FLOOR

FLOOR (@vector)

The FLOOR function returns the input value rounded-down to the next lowest integer number.

Example:

FLOOR (13.8)

The script returns 13.

FRACPORTION

FRACPORTION (@vector)

The FRACPORTION function returns the fractional part of a number.

Example:

FRACPORTION (16.789)

The script returns 0.789.

INTERPOLATE

INTERPOLATE (@vector, @condition, @interpolationType)

The Interpolate function calculates an interpolation in the input vector point where the corresponding condition evaluates to true. The interpolation can be calculated with the following interpolation type constants:
LINEAR: 1 (Linear Interpolation)
SPLINE: 2 (Bicubic SPLine Interpolation)

Example:

INTERPOLATE (CLOSE, InterpolateCondition, LINEAR)

Returns a linear-interpolated vector of the close prices where the InterpolateCondition is true.

INTPORTION

INTPORTION (@vector)

The INTPORTION function returns the integer part of a number.

Example:

INTPORTION (16.789)

The script returns 16.

ISLASTBAR

ISLASTBAR()

The ISLASTBAR function can be used to identify the last bar in a series of historical data.

Example:

SET C = ISLASTBAR() AND CLOSE > OPEN

The variable C will be TRUE only if the last bar of the historical data have a CLOSE price higher then the OPEN price, and FALSE for any other bar.

LOG

LOG (@vector)

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 returns 3.26

LOG10

LOG10 (@vector)

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

Example:

LOG10 (26.28)

The script returns 1.42

MOD

MOD (@numerator, @denominator)

The MOD function calculates the modulus of an integer division. The MOD function requires two parameters, the Numerator and the Denominator.

Example:

MOD (12, 5)

The script returns 2, the modulus of 12 divided by 5.

NEG

NEG (@vector)

The NEG function return the absolute value of a number multiplied by -1. Positive values will be transformed in negative values, while negative values will remain unaltered.

Example:

NEG (CLOSE – OPEN)

The script will always return a negative value, even if the OPEN price is lower then the CLOSE price.

POW

POW (@base, @exponent)

The POW function performs the power of a number. The function requires two parameters, the Base and the Exponent.

Example:

POW (10, 2)

The script returns 100, 10 to the power of 2.

RADTODEG

RADTODEG (@radiants)

The RADTODEG function converts the value of an angle from radians to degrees.

Example:

RADTODEG (PI/4)

The script returns 45.

RND or RANDOM

```
RND (@vector)
RANDOM (@vector)
```

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

Example:

```
RND (100)
RANDOM (100)
```

The script returns a random number between 0 and 100.

ROUND

```
ROUND(@vector, @digits)
```

The ROUND function rounds a number to the specified number of decimal digits.

Example:

```
ROUND (45.6789, 1)
```

The script returns 45.7

ROUNDTICK

```
ROUNDTICK(@vector, @tick)
```

The ROUNDTICK function rounds the input value to the nearest valid value.

Example:

```
ROUNDTICK (45.6789, 0.2)
```

The script returns 45.6

SIGN

```
SIGN (@vector)
```

The SIGN function returns the sign of a number. The return value can be:

@vector	**SIGN**
A positive value	+1
Zero	0
A negative value	-1

Example:

```
SIGN (100)
SIGN (0)
SIGN (-100)
```

The scripts above return +1 in the first case, 0 in the second case, and -1 in the last case.

SIN

```
SIN (@vector)
```

The function SIN returns the sine of an angle. The angle must be expressed in radians.

Example:

```
SIN (PI/4)
```

The script returns 0.707

SQRT - SQUAREROOT

```
SQRT (@vector)
SQUAREROOT (@vector)
```

The SQRT function returns the square root of a number.

Example:

```
SQUAREROOT (100)
```

The script returns 10

SQUARE

```
SQUARE (@vector)
```

The SQUARE function elevates a number to the power of 2.

Example:

```
SQUARE (5)
```

The script returns 25

TAN

```
TAN (@vector)
```

The function TAN returns the tangent of an angle. The angle must be expressed in radians.

Example:

```
TAN (PI/4)
```

The script returns 1

TODAYHIGH

```
TODAYHIGH ()
```

The TODAYHIGH function returns the HIGH price of the day, regardless of the timeframe.

Example:

```
HIGH = TODAYHIGH()
```

The script returns TRUE if the current bar HIGH is equal to the HIGH price of the day.

TODAYLOW

```
TODAYLOW ()
```

The TODAYLOW function returns the LOW price of the day, regardless ot the timeframe.

Example:

```
LOW = TODAYLOW()
```

The script returns TRUE if the current bar LOW is equal to the LOW price of the day.

TODAYOPEN

```
TODAYOPEN ()
```

The function TODAYOPEN returns the OPEN price of the day, regardless of the timeframe.

Example:

```
OPEN > TODAYOPEN()
```

The script returns TRUE if the current bar OPEN price is higher then the day’s OPEN price.

YESTERDAYCLOSE

```
YESTERDAYCLOSE ()
```

The YESTERDAYCLOSE function returns the CLOSE price of the previous trading day, regardless of the timeframe.

Example:

```
LAST > YESTARDAYCLOSE()
```

The script return TRUE if the current price is higher than the previous trading day’s CLOSE price.

YESTERDAYHIGH

```
YESTERDAYHIGH ()
```

The YESTERDAYHIGH function returns the HIGH price of the previous trading day, regardless of the timeframe.

Example:

```
HIGH > YESTARDAYHIGH()
```

The script returns TRUE if the current bar HIGH price is higher than the previous trading day's HIGH price.

YESTERDAYLOW

```
YESTERDAYLOW ()
```

The YESTERDAYLOW function returns the LOW price of the previous trading day, regardless of the timeframe.

Example:

```
LOW < YESTARDAYLOW()
```

The script returns TRUE if the current bar LOW price is lower than the previous trading day's LOW price.

YESTERDAYOPEN

```
YESTERDAYOPEN ()
```

The YESTERDAYOPEN function returns the OPEN price of the previous trading day, regardless of the timeframe.

Example:

```
OPEN > YESTARDAYOPEN()
```

The script returns TRUE if the current bar OPEN price is higher than the previous trading day's OPEN price.

Chapter 5

Technical Analysis Functions and Indicators

5

Securities can be analyzed by means of either fundamental analysis or technical analysis. Those who analyze securities using fundamental analysis rely on data such as the profits-to-earnings ratio, yield, and dividend whereas those who analyze securities using technical analysis look for technical patterns on stock charts by using calculations referred to as "technical indicators".

Technical analysis is a form of market analysis that studies the demand and supply for securities based on volume and price studies. Technicians attempt to identify price trends in a market using one or more technical indicators.

There are many different types of technical indicators and most are built into the EasyScript® programming language as primitive functions, as outlined in this chapter.

EasyScript® also allows you to program additional technical indicators by using a combination of primitive functions listed in Chapter 4 and in this chapter. Chapter 6 provides examples and techniques for building custom indicators and trading systems.

This chapter provides a comprehensive list of the primitive technical analysis functions that are supported by the EasyScript® programming language.

Please note that many technical indicator names are quite long, therefore function abbreviations have been conveniently provided wherever possible, as in the following example:

Long name:

```
SimpleMovingAverage (CLOSE, 30)
```

Abbreviated name:

```
SMA (CLOSE, 30)
```

SMA is the same function as SimpleMovingAverage and both methods work the same way.

Whenever available, the lowMark and highMark items are used to identify significative values/levels in the indicators.

Moving Averages

Moving averages are the foundation of 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.

This section covers the Simple moving average, which is simply an average price over time, the exponential moving average, which is more complex and places extra weight on prior values, plus several other types of moving averages like weighted averages, triangular averages, time series calculations, and so forth.

Each moving average in this section has an associated constant identifier that can be used as a function argument to specify the type of moving average to use by any given technical indicator that requires a moving average type.
For example, the Moving Average Convergence / Divergence (MACD) indicator in EasyScript® allows you to specify the moving average type used within the indicator's "Signal Line" calculation.

Simple Moving Average

```
SimpleMovingAverage (@vector, @periods)
SMA (@vector, @periods)
```

MA Type: **SIMPLE**

Overview:
The Simple Moving Average is simply an average of values over a specified period of time.

Interpretation:
A Moving Average is most often used to average values for a smoother representation of the underlying price or indicator.

Example:

```
CLOSE > SMA (CLOSE, 30)
```

Evaluates to true when the close is greater than a 30-period SMA.

Exponential Moving Average

```
ExponentialMovingAverage (@vector, @periods)
EMA (@vector, @periods)
```

MA Type: **EXPONENTIAL**

Overview:
An Exponential Moving Average is similar to a Simple Moving Average. An EMA is calculated by applying a small percentage of the current value to the previous value, therefore an EMA applies more weight to recent values.

Interpretation:
An Exponential Moving Average is most often used to average values for a smoother representation of the underlying price or indicator, exactly as an SMA, but is more reactive. On the other side, being less "elastic", it may generate more "false positives" signals.

Example:

```
CLOSE > EMA (CLOSE, 30)
```

Evaluates to true when the close is greater than a 30-period EMA.

Time Series Moving Average

```
TimeSeriesMovingAverage (@vector, @periods)
TSMA (@vector, @periods)
```

MA Type: **TIME_SERIES**

Overview:
A Time Series Moving Average is similar to a Simple Moving Average, except that values are derived from linear regression forecast values instead of regular values.

Interpretation:
A Time Series Moving Average is less influenced by spikes in input data than other moving averages, greatly reducing the noise in output.

Example:

```
CLOSE > TSMA (CLOSE, 30)
```

Evaluates to true when the close is greater than a 30-period TSMA.

Variable Moving Average

```
VariableMovingAverage (@vector, @periods)
VMA (@vector, @periods)
```

MA Type: **VARIABLE**

Overview:
A Variable Moving Average is similar to an exponential moving average except that it adjusts to volatility.

Interpretation:
A Variable Moving Average is used when historical volatility has to be considered in the analysis. This moving average is influenced by the size of variations in input data.

Example:

```
CLOSE > VMA (CLOSE, 30)
```

Evaluates to true when the close is greater than a 30-period VMA.

Triangular Moving Average

```
TriangularMovingAverage (@vector, @periods)
TMA (@vector, @periods)
```

MA Type: **TRIANGULAR**

Overview:
The Triangular Moving Average is similar to a Simple Moving Average, except that more weight is given to the price in the middle of the moving average periods.

Interpretation:
A Triangular Moving Average is generally used with a period equal to 2 times the dominant cycle: with this configuration, compared to a Simple Moving Average, is more reactive in identifying trends, but is much more smoothed.

Example:

```
CLOSE > TMA (CLOSE, 30)
```

Evaluates to true when the close is greater than a 30-period TMA.

Weighted Moving Average

```
WeightedMovingAverage (@vector, @periods)
WMA (@vector, @periods)
```

MA Type: **WEIGHTED**

Overview:
A Weighted Moving Average places more weight on recent values and less weight on older values.

Interpretation:
A Weighted Moving Average is generally used to achieve less lag and less “false positive” signals.

Example:

```
CLOSE > WMA (CLOSE, 30)
```

Evaluates to true when the close is greater than a 30-period WMA.

Welles Wilder Smoothing Moving Average

```
WellesWilderSmoothing (@vector, @periods)
WWS (@vector, @periods)
```

MA Type: **WILDER**

Overview:
The Welles Wilder's Smoothing indicator is similar to an exponential moving average. The indicator does not use the standard exponential moving average formula. Welles Wilder described 1/14 of today's value + 13/14 of yesterday's average as a 14-day exponential moving average.

Interpretation:
The Welles Wilder's Smoothing is used to identify important price levels in both directions, that can be used as supports and resistances.

Example:

```
CLOSE > WWS (CLOSE, 30)
```

Evaluates to true when the close is greater than a 30-period WWS.

VIDYA Moving Average

VIDYA (V@vector, @periods, @r2Scale)

MA Type: **VIDYA**

Overview:
VIDYA (Volatility Index Dynamic Average), developed by Mr. Tuschar Chande, is a moving average derived from linear regression R^2.

Interpretation:
A Moving Average is most often used to average values for a smoother representation of the underlying price or indicator. Because VIDYA is a derivative of linear regression, it quickly adapts to volatility.

Recommended Parameters:
R2Scale is a value specifying the R-Squared scale to use in the linear regression calculations. Mr. Chande recommends a value between 0.5 and 0.8 (default value is 0.65).

Example:

CLOSE > VIDYA (CLOSE, 30, 0.65)

Evaluates to true when the close is greater than a 30-period VIDYA with an R^2 of 0.65.

Double Exponential Moving Average

DoubleExponentialMovingAverage (@vector, @periods)
DEMA (@vector, @periods)

MA Type: **DOUBLE**

Overview:
The Double Exponential Moving Average is based on the Exponential Moving Average.

Interpretation:
A Double Exponential Moving Average is very reactive and adaptive with the intent to reduce lag compared to traditional Moving Averages.

Example:

CLOSE > DEMA (CLOSE, 30)

Evaluates to true when the close is greater than a 30-period DEMA.

Triple Exponential Moving Average

```
TripleExponentialMovingAverage (@vector, @periods)
TEMA (@vector, @periods)
```

MA Type: **TRIPLE**

Overview:
The Triple Exponential Moving Average is based on the Exponential Moving Average, like the DEMA, with an additional final calculation step.

Interpretation:
A Triple Exponential Moving Average is very reactive and adaptive with the intent to reduce lag compared to traditional Moving Averages.

Example:

```
CLOSE > TEMA (CLOSE, 30)
```

Evaluates to true when the close is greater than a 30-period TEMA.

Generic Moving Average

MovingAverage (@vector, @periods, @matype)

Overview:
The Moving Average function can be used to calculate all different types of moving averages, selecting the desired type with the MA Type parameter.

Interpretation:
A Moving Average is most often used to average values for a smoother representation of the underlying price or indicator.

Example:

MovingAverage (CLOSE, 21, EXPONENTIAL) > MovingAverage (CLOSE, 21, SIMPLE)

Evaluates to true when the EXPONENTIAL Moving Average is greater than the SIMPLE Moving Average.

The MA Type parameter can assume any of the following values:

SIMPLE: 1 (Simple Moving Average)
EXPONENTIAL: 2 (Exponential Moving Average)
TIME_SERIES: 3 (Time Series Moving Average)
VARIABLE: 4 (Variable Moving Average)
TRIANGULAR: 5 (Triangular Moving Average)
WEIGHTED: 6 (Weighted Moving Average)
VOLATILITY: 7 (VIDYA Moving Average)
WILDER: 8 (Welles Wilder Smoothing)
DOUBLE: 9 (Double Exponential Moving Average)
TRIPLE: 10 (Triple Exponential Moving Average)

Linear Regression

A classic statistical problem is to try to determine the relationship between two random variables X and Y such as the closing price of a stock over time. Linear regression attempts to explain the relationship with a straight line fit to the data. The linear regression model postulates that

$$Y = A + BX + E$$

where the "residual" e is a random variable with mean zero. The coefficients A and B are determined such as the sum of the square residuals is as small as possible. The indicators in this section are based upon this model.

Forecast

Forecast (@vector, @periods)
LR (@vector, @periods)
LinearReg (@vector, @periods)
LinearRegressioForecast (@vector, @periods)

Overview:
Returns the linear regression forecast for the next period based on the linear regression calculation over the specified number of periods.

Example:

Forecast (CLOSE, 30) > REF (CLOSE,1)

Evaluates to true when the forecast is higher than the previous closing price.

Intercept

Intercept (@vector, @periods)
LinearRegressionIntercept (@vector, @periods)

Overview:
Returns the linear regression intercept for the last period’s Y value, based on the linear regression calculation over the specified number of periods.

Example:

Intercept (CLOSE, 21) > REF (CLOSE,1)

Evaluates to true when the intercept is higher than the previous closing price.

R^2 (R-Squared)

RSquared (@vector, @periods)
R2 (@vector, @periods)
LinearRegressionRSquared (@vector, @periods)

Overview:
R-Squared is the coefficient of determination for the supplied vector over the specified periods. The values oscillate between 0 and 1.

Example:

R2 (CLOSE, 30) < 0.1

Evaluates to true when the coefficient of determination is less than 0.1.

Slope

Slope (@vector, @periods)
LinearRegressionSlope (@vector, @periods)

Overview:
Returns the linear regression slope value for the data being analyzed over the specified number of periods. Values oscillate from negative to positive numbers.

Example:

SLOPE (CLOSE, 30) > 0.3

Evaluates to true when the slope is greater than 0.3.

Bands Functions

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.

Bollinger Bands

BollingerBandsTop (@vector, @periods, @standardDeviations, @maType)
BBT (@vector, @periods, @standardDeviations, @maType)

BollingerBandsMiddle (@vector, @periods, StandardDeviations, MAType)
BBM (@vector, @periods, @standardDeviations, @maType)

BollingerBandsBottom (@vector, @periods, StandardDeviations, MAType)
BBB (@vector, @periods, @standardDeviations, @maType)

Overview:
Bollinger bands rely on standard deviations in order to adjust to changing market conditions. When a stock becomes volatile the bands widen (move further away from the average). Conversely, when the market becomes less volatile the bands contract (move closer to the average). Tightening of the bands is often used asan early indication that the stock's volatility is about to increase.

Interpretation:
Bollinger Bands (as with most bands) can be imposed over an actual price or another indicator. When prices rise above the upper band or fall below the lower band, a change in direction may occur when the price penetrates the band after a small reversal from the opposite direction.

Recommended Parameters:
Vector: **CLOSE**
Periods: **20**
Standard Deviations: **2**
MA Type: **EXPONENTIAL**

Example:

CLOSE > BBT (CLOSE, 20, 2, EXPONENTIAL)

Evaluates to true when the close is greater than a 20-periods Bollinger Band Top calculated by 2 standard deviations, using an exponential moving average.

Fractal Chaos Bands

FractalChaosBandsTop (@periods)
FCBT (@periods)

FractalChaosBandsBottom (@periods)
FCBB (@periods)

Overview:
The Fractal Chaos Bands is based on the assumption that prices changes in a chaotic manner, and uses the chaos theory formulas to calculate two bands.

Interpretation:
This indicator can be used as any other band indicator, to identify the current trend. Higher band slope indicates more strength in the upside or downside trend, lower slope and flatness indicates a sideways trend or a relatively low volatility period.

Recommended Parameters:
Periods: **10**

Example:

CLOSE > FractalChaosBandsTop (10)

Evaluates to true when the close is greater than a 10-periods Fractal Chaos Bands Top.

High Low Bands

HighLowBandsTop (@periods)
HLT (@periods)

HighLowBandsMedian (@periods)
HLM (@periods)

HighLowBandsBottom (@periods)
HLB (@periods)

Overview:
The High Low Bands indicators is based on two moving averages, calculated on high and low prices.

Interpretation:
As any other band indicator, and increase in market volatility and the beginning of strong trends is signaled by price crossing the bands in either direction.

Recommended Parameters:
Periods: **10**

Example:

```
CLOSE > HLT (10) OR CLOSE < HLB (10)
```

Evaluates to true when the close is outside of the 10-periods indicator bands.

Keltner Channel

KeltnerChannelTop (@periods, @maType, @multipler)
KCT (@periods, @maType, @multiplier)

KeltnerChannelMedian (@periods, @maType, @multipler)
KCM (@periods, @maType, @multiplier)

KeltnerChannelBottom (@periods, @maType, @multipler)
KCB (@periods, @maType, @multiplier)

Overview:
Keltner channels are calculated from the Average True Range and shifted up and down from the median based on the multiplier.

Interpretation:
Like other bands, Keltner channels can be imposed over an actual price or another indicator. Keltner bought when prices closed above the upper band and sold when prices closed below the lower band. Keltner channels can also be interpreted the same way as Bollinger bands are interpreted.

Recommended Parameters:
Periods: **15**
MA Type: **EXPONENTIAL**
Multiplier: **1.3**

Example:

CLOSE > KCT (15, EXPONENTIAL, 1.3)

Evaluates to true when the close closes above the Keltner channel top.

Moving Average Envelope

MovingAverageEnvelopeTop (@periods, @maType, @shift)
MAET (@periods, @maType, @shift)

MovingAverageEnvelopeBottom (@periods, @maType, @shift)
MAEB (@periods, @maType, @shift)

Overview:
Moving Average Envelopes consist of moving averages calculated from the underling price, shifted up and down by a fixed percentage.

Interpretation:
Moving Average Envelopes (or trading bands) can be imposed over an actual price or another indicator. When prices rise above the upper band or fall below the lower band, a change in direction may occur when the price penetrates the band after a small reversal from the opposite direction.

Recommended Parameters:
Periods: **20**
MA Type: **SIMPLE**
Shift: **0,05**

Example:

```
CLOSE > MAET (20, SIMPLE, 0,5)
```

Evaluates to true when the close is greater than a 20-day Moving Average Envelope Top calculated by 5% using a simple moving average.

Prime Number Bands

PrimeNumberBandsTop()
PNBT()

PrimeNumberBandsBottom()
PNBB()

Overview:
This novel indicator identifies the nearest prime number for the high and low and plots the two series as bands.

Example:

```
PNBT <> REF (PNBT,1) OR PNBB <> REF (PNBB,1)
```

Evaluates to true when any of Prime Number Bands changes from the previous period.

Starc

StarcTop (@periods, @maType, @multplier)
StarcMedian (@periods, @maType, @multplier)
StarcBottom (@periods, @maType, @multplier)

Overview:
The Starc Channel is calculated by adding and substracting to/from the Moving Average the Average True Range multiplied by a scale factor.

Interpretation:
The Starc Channel can be used exactly as the Bollinger Bands. A long signal is identified by a price above the top band, a short signal is identified by a price below the bottom band.

Recommended Parameters:
Periods: **15**
MA Type: **EXPONENTIAL**
Multiplier: **1.3**

Example:

```
CLOSE > StarcTop (15, EXPONENTIAL, 1.3)
```

Evaluates to true when the close is higher then the 15-periods Starc Top band with scale factor of 1.3.

Oscillators

This section covers technical indicators that oscillate from one value to another. Most oscillators measure the velocity of directional price or volume movement. These indicators often go into overbought and oversold zones, at which time a reaction or reversal is possible. 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. A large percentage of technical indicators oscillate, so this section covers quite a few functions.

Aroon

```
AroonUp (@periods)
AroonDown (@periods)
```

Overview:
The Aroon indicator was developed by Tushar Chande in the mid 1990's. This indicator is often used to determine whether a stock is trending or not and how stable the trend is.

Interpretation:
Trends are determined by extreme values (above 80) of both lines (Aroon up and Aroon down), whereas unstable prices are determined when both lines are low (less than 20).

Recommended Parameters:
Periods: **10**

Example:

```
AroonUp (10) > 80 AND AroonDown (10) < 20
```

Evaluates to true when the 10-periods AroonUp is greater then 80 and the 10-periods AroonDown is less than 20.

Aroon Oscillator

AroonOsc (@periods)

Overview:
The Aroon Oscillator indicator is directly derived from the Aroon Indicator, it measures the difference between Aroon Up and Aroon Down, thus it ranges from -100 to +100.

Interpretation:
The common interpretation is to consider the beginning of an upside trend when the indicator crosses above +50, and the beginning of a downside trend when the indicator crosses below -50.

Recommended Parametes:

Function		BeeTrader Indicator	
Periods	**10**	Periods	**10**
		lowMark	**-50**
		highMark	**50**

Example:

CROSSOVER (AroonOsc(10), 50)

Evaluates to true when the 10-periods Aroon Oscillator crosses above the +50 line, indicating the beginning of an upside trend.

Average True Range

AverageTrueRange (@periods, @maType)
ATR (@periods, @maType)

Overview:
The average true range (ATR) is a technical analysis indicator that measures market volatility by decomposing the entire range of an asset price for that period. Specifically, ATR is a measure of volatility introduced by market technician J. Welles Wilder Jr. The true range indicator is taken as the greatest of the following: current high less the current low; the absolute value of the current high less the previous close; and the absolute value of the current low less the previous close. The average true range is then a moving average, generally using 14 days, of the true ranges.

Interpretation:
The ATR is commonly used as an exit method that can be applied no matter how the entry decision is made. Average true range can also give a trader an indication of what size trade to put on in derivatives markets. It is possible to use the ATR approach to position sizing that accounts for an individual trader's own willingness to accept risk as well as the volatility of the underlying market.

Recommended Parameters:
Periods: **14**
MA Type: **SIMPLE**

Example:

ATR(21, SIMPLE) > 2

Evaluates to true when the ATR is greater than 2.

Chaikin Money Flow

```
ChaikinMoneyFlow (@periods)
CMF (@periods)
```

Overview:
The Chaikin Money Flow oscillator is a momentum indicator that spots buying and selling by calculating price and volume together. This indicator is based upon Accumulation / Distribution, which is in turn based upon the premise that if a stock closes above its midpoint, (high + low) / 2, for the day then there was accumulation that day, and if it closes below its midpoint, then there was distribution that day.

Interpretation:
A buy signal is generated when the indicator is rising and is in positive territory.
A sell signal is generated when the indicator is falling and is in negative territory.

Recommended Parameters:
Periods: **21**

Example:

```
CMF (21) > REF (CMF (21), 1)
```

Evaluates to true when the Chaikin Money Flow Index is bullish.

Chaikin Volatility

```
ChaikinVolatility (@periods, @rateOfChange, @maType)
CV (@periods, @rateOfChange, @maType)
```

Overview:
The Chaikin Volatility Oscillator is a moving average derivative of the Accumulation / Distribution index. This indicator quantifies volatility as a widening of the range between the high and the low price.

Interpretation:
The Chaikin Volatility Oscillator adjusts with respect to volatility, independent of long-term price action. The most popular interpretation is to sell when the indicator tops out, and to buy when the indicator bottoms out.

Recommended Parameters:

Periods:	**10**
Rate of Change:	**10**
MA Type:	**SIMPLE**

Example:

```
CV (10, 10, SIMPLE) > 0
```

Evaluates to true when the Chaikin Volatility Oscillator is in positive territory.

Chande Forecast Oscillator

```
ChandeForecastOscillator (@vector, @periods)
CFO (@vector, @periods)
```

Overview:
The Chande Forecast Oscillator plots the percentage difference between the closing price and the n-period linear regression forecasted price.

Interpretation:
The oscillator is above zero when the forecast price is greater than the closing price (an upside trend is forecasted) and less than zero if it is below (a downside trend is forecasted).

Recommended Parameters:
Vector: **CLOSE**
Periods: **21**

Example:

```
CFO (CLOSE, 21) > 0
```

Evaluates to true if the Chande Forecast Oscillator is in positive territory.

Chande Momentum Oscillator

```
ChandeMomentumOscillator (@vector, @periods)
CMO (@vector, @periods)
```

Overview:
The Chande Momentum Oscillator (CMO), developed by Tushar Chande, is an advanced momentum oscillator derived from linear regression. This indicator was published in his book titled "New Concepts in Technical Trading" in the mid 90's.

Interpretation:
The CMO enters into overbought territory at +50, and oversold territory at -50. You can also create buy/sell triggers based on a moving average of the CMO.Also, increasingly high values of CMO may indicate that prices are trending strongly upwards. Conversely, increasingly low values of CMO may indicate that prices are trending strongly downwards.

Recommended Parameters:
Vector: **CLOSE**
Periods: **14**

Example:

```
CMO (CLOSE, 14) > 48
```

Evaluates to true when the CMO of the close is overbought.

Coppock Curve

CoppockCurve (@vector)
CPKC (@vector)

Overview:
The Coppock Curve (CC) was introduced by economist Edwin Coppock in October 1962. Coppock initially developed the indicator for long-term monthly charts; this will appeal to long-term investors, as signals are quite infrequent on this timeframe. It is a momentum indicator that oscillates above and below zero.

Interpretation:
The zero line of the Coppock Curve acts as a trade trigger; buy when the CC moves above zero, and sell when the CC moves below zero.

Recommended Parameters:
Vector: **CLOSE**

Example:

```
SET COP = CPKC (CLOSE)
COP > 0 AND COP > REF (COP, 1)
```

Evaluates to true when the Coppock Curve is raising and in positive territory.

Detrended Price Oscillator

DetrendedPriceOscillator (@vector, @periods, @maType)
DPO (@vector, @periods, @maType)

Overview:
Similar to the Price Oscillator except DPO is used when long-term trends or outliers make the underlying price difficult to analyze.

Interpretation:
Buying occurs when the oscillator rises. Selling occurs when the oscillator falls.

Recommended Parameters:
Vector: **CLOSE**
Periods: **14**
MA Type: **SIMPLE**

Example:

```
DPO (CLOSE, 14, SIMPLE) > 0
```

Evaluates to true when the Detrended Price Oscillator is in positive territory.

Directional Movement Index

ADX (@periods)
DIP (@periods)
DIN (@periods)

Overview:
The Directional Movement Index is composed by 3 indicators, ADX, DIP (DI+) and DIM (DI-).
The ADX (Average Directional Movement Index) is an indicator of how much the market is trending, either up or down: the higher the ADX line, the more the market is trending and the more suitable it becomes for a trend-following system. This indicator consists of two lines: DI+ and DI-, the first one being a measure of uptrend and the second one a measure of downtrend.

Interpretation:
A buy signal is given when DI+ crosses over DI-, a sell signal is given when DI-crosses over DI+.

Recommended Parameters

Function		BeeTrader Indicator	
Periods	**14**	PeriodsADX	**30**
		PeriodsDI	**14**

Example:

CROSSOVER (DIN (14), DIP (14)) AND ADX > 18

Evaluates to true when the 14-periods DI- crosses over the 14-periods DI+ and ADX is greater than 18.

Ease of Movement Oscillator

EaseOfMovement (@vector, @periods)
EOM (@vector, @periods)

Overview:
The Ease of Movement oscillator displays a unique relationship between price change and volume.

Interpretation:
The Ease of Movement oscillator rises when prices are trending upwards under low volume, and likewise, the Ease of Movement oscillator falls when prices are trending downwards under low volume.

Recommended Parameters:
Vector: **CLOSE**
Periods: **21**

Example:

EOM (CLOSE, 21) > 0

Evaluates to true when the Ease of Movement is in positive territory.

Elder Force Index

ElderForceIndex ()
EFI ()

Overview:
The Elder Force Index is an oscillator that measures the force, or power, of bulls behind particular market rallies and of bears behind every decline. The three key components of the force index are the direction of price change, the extent of the price change, and the trading volume.

Interpretation:
In general, traders will want to buy when the two-day EMA of the force index is negative and sell when it is positive. These traders, however, should always keep in mind the overarching principle of trading in the direction of the 13-day EMA of prices. The 13-day EMA of the force index is a longer-term indicator, and, when it crosses above the centerline, the bulls are exerting the greater force. When it is negative, the bears have control of the market. Of particular importance are divergences between a 13-day EMA of force index and prices, which correspond with precise points, indicating crucial turning points of the market.

Example:

```
EFI() > 0
```

Evaluates to true when the Elder Force Index is in positive territory.

Fractal Chaos Oscillator

```
FractalChaosOscillator()
FCO()
```

Overview:
The chaotic nature of stock market movements explains why it is sometimes difficult to distinguish daily charts from monthly charts if the time scale is not given. The patterns are similar regardless of the time resolution. Like the chambers of the nautilus, each level is like the one before it, but the size is different. To determine what is happening in the current level of resolution, the fractal chaos oscillator can be used to examine these patterns.

Interpretation:
A buy signal is generated when the oscillator tops, and a sell signal is generated when the oscillator bottoms.

Example:

```
FCO() > REF (FCO(), 1)
```

Evaluates to true when the FCO is raising.

High Minus Low

```
HighMinusLow ()
HML ()
```

Overview:
This function returns the high price minus the low price for each bar.

Interpretation:
This indicator is often used as a component for other technical indicators but can be used with a moving average to show the change in price action over time.

Example:

```
SET A = SMA (HML (), 14)
A > REF (A, 10)
```

Evaluates to true when the height of each bar has been increasing over the past several bars.

Intraday Momentum Index

IntradayMomentumIndex ()
IMI ()

Overview:
The intraday indicator was developed by Tushar Chande to provide investors with a way to find optimal days to buy and sell. The Intraday Momentum Index looks at the relationship between a stock's open and close price over the course of the day, rather than how the open/close price varies between days. It combines some features of the relative strength index, namely the relationship between up closes and down closes and whether there is an indication that a stock is overbought or oversold, with candlestick charts.

Interpretation:
The IMI behaves like an oscillator, ranging from 0 to 100. Levels above 70 are considered as overbought territory, while levels below 30 are considerer as oversold territory.
Note: thresholds values of 30 and 70 must be adapted to the historical data on which the indicator is applied.

Recommended Parameters:

Function	BeeTrade Indicator	
	lowMark	**48**
	highMark	**52**

Example:

```
IMI() > 52
```

Evaluates to true when the IMI is greater than 52.

Klinger Volume Oscillator

KlingerVolumeOscillator (@shortCycle, @songCycle, @signalPeriods, @maType)
KVO (@shortCycle, @longCycle, @signalPeriods, @maType)

Overview:
The Klinger Volume Oscillator was developed by Stephen Klinger to determine the long-term trend of money flow while remaining sensitive enough to detect short-term fluctuations. The indicator compares the volume flowing through securities with the security's price movements and then converts the result into an oscillator. The Klinger oscillator shows the difference between two moving averages which are based on more than price. Traders watch for divergence on the indicator to signal potential price reversals. Like other oscillators, a signal line can be added to provide additional trade signals.

Interpretation:
The Klinger Oscillator is fairly complex to calculate, but it's based on the idea of force volume, which accounts for volume, price and trend. Using this data, the oscillator is created by looking at the difference between two exponential moving averages of force volume involving different time frames (typically 34 and 55). The idea is to show how the volume flowing through the securities is impacting its long-term and short-term price direction. A signal line (typically a 13-period moving average) is used to trigger buy or sell signals. This technique is very similar to signals that are created with other indicators such as the MACD. While these are the basic signals generated by these indicators, it's important to note that these techniques may generate a lot of trading signals that may not be as effective in sideways markets.

Recommended Parameters:
Short Cycle: **34**
Long Cycle: **55**
Signal Periods: **13**
MA Type: **SIMPLE**

Example:

```
SET K = KVO (34, 55, 13, SIMPLE)
K > REF (K, 1) AND TREND (CLOSE, 30) = DOWN
```

Evaluates to true when the KVO is raising in a downside trend, identifying a divergence.

Moving Average Convergence / Divergence (MACD)

```
MACD (@shortCycle, @longCycle, @signalPeriods, @maType)
MACDSignal (@shortCycle, @longCycle, @signalPeriods, @maType)
MACDS (@shortCycle, @longCycle, @signalPeriods, @maType)
```

```
MACDHistogram (@shortCycle, @longCycle, @signalPeriods, @maType)
MACDH (@shortCycle, @longCycle, @signalPeriods, @maType)
```

Overview:
The MACD is a moving average oscillator that shows potential overbought/oversold phases of market fluctuation. The MACD is a calculation of two moving averages of the underlying price/indicator.

Interpretation:
Buy and sell interpretations may be derived from crossovers (calculated by the MACDSignal function), overbought / oversold levels of the MACD and divergences between MACD and underlying price.

Recommended Parameters:

Long Cycle:	**26**
Short Cycle:	**13**
Signal Periods:	**9**
MA Type:	**SIMPLE**

Example:

```
SET A = MACDSignal (13, 26, 9, SIMPLE)
SET B = MACD (13, 26, 9, SIMPLE)
CROSSOVER(A, B) = TRUE
```

Evaluates to true when the MACD Signal line recently crossed over the MACD.

Momentum Oscillator

```
MomentumOscillator (@vector, @periods)
MO (@vector, @periods)
```

Overview:
The momentum oscillator calculates the change of price over a specified length of time as a ratio.

Interpretation:
Increasingly high values of the momentum oscillator may indicate that prices are trending strongly upwards. The momentum oscillator is closely related to MACD and Price Rate of Change (ROC).

Recommended Parameters:
Vector: **CLOSE**
Periods: **14**

Example:

```
MO (CLOSE, 14) > 90
```

Evaluates to true when the momentum oscillator of the close is over 90.

Oscillator (Generic Oscillator)

```
OSCILLATOR(@vector, @periods)
OSC(@vector, @periods)
```

Overview:
The Generic Oscillator function calculates an oscillator based on previous values of the input vector. The output values ranges from 0 to 100.

Interpretation:
High output values correspond historically high input vector values, while low output values correspond to historically low input vector values.

Example:

```
OSC(CLOSE, 21) > 70
```

Evaluates to true when the closing price is in overbought territory.

Pretty Good Oscillator

PrettyGoodOscillator (@periods)
PGO (@periods)

Overview:
The Pretty Good Oscillator (PGO) by Mark Johnson measures the distance of the current close from its N-day simple moving average, expressed in terms of an average true range over a similar period.

Interpretation:
Johnson’s approach was to use it as a breakout system for longer term trades. If the PGO rises above 3.0 then go long, or below -3.0 then go short, and in both cases exit on returning to zero (which is a close back at the SMA).

Recommended Parameters:
Periods: **14**

Example:

```
SET PG =  PGO (14)
PGO > 3
```

Evaluates to true when the PGO is higher than 3.

Price Oscillator

PriceOscillator (@vector, @shortTermPeriods, @longTermPeriods, @maType)
PO (@vector, @shortTermPeriods, @longTermPeriods, @maType)

Overview:
Similar to the Volume Oscillator, the Price Oscillator is calculated based on a spread of two moving averages.

Interpretation:
The Price Oscillator is a moving average spread. Buying usually occurs when the oscillator rises, and selling usually occurs when the oscillator falls.

Recommended Parameters:
Vector: **CLOSE**
Short Term Periods: **9**
Long Term Periods: **14**
MA Type: **SIMPLE**

Example:

PO (CLOSE, 9, 14, SIMPLE) > 0

Evaluates to true when the Price Oscillator is in positive territory.

Prime Number Oscillator

```
PrimeNumberOscillator (@vector)
PNO (@vector)
```

Overview:
Finds the nearest prime number from either the top or bottom of the series, and plots the difference between that prime number and the series itself.

Interpretation:
This indicator can be used to spot market turning points. When the oscillator remains at the same high point for two consecutive periods in the positive range, consider selling. Conversely, when the oscillator remains at a low point for two consecutive periods in the negative range, consider buying.

Recommended Parameters:
Vector: **CLOSE**

Example:

```
PNO (CLOSE) = REF (PNO (CLOSE), 1)
AND REF (PNO (CLOSE), 2) <>  PNO (CLOSE)
```

Evaluates to true when the last 2 bars have the same PNO value and the third last has a different value.

Rainbow Oscillator

```
RainbowOscillator (@vector, @levels, @maType)
RBO (@vector, @levels, @maType)
```

Overview:
The rainbow oscillator is calculated based upon multiple time frames of a moving average.

Interpretation:
The trend may reverse suddenly when values stay above 0.80 or below 0.20 for two consecutive days.

Recommended Parameters:
Vector: **CLOSE**
Levels: **21**
MA Type: **SIMPLE**

Example:

```
SET R = RBO (CLOSE, 21, SIMPLE)
R > 0  AND REF (R, 1)  < 0
```

Evaluates to true when the Rainbow Oscillator has been above 0.8 for at least two consecutive days.

Random Walk Index

RandomWalkIndexTop (@periods)
RWIT (@periods)

RandomWalkIndexBottom (@periods)
RWIB (@periods)

Overview:
The Random Walk Index is an oscillator that tries to determine whether the price movement of a financial instrument is random (random) or a consequence of a statistically significant and therefore more reliable TREND. The indicator consists of two functions, the RWIT and the RWIB, the first identifies the long trends, the second the short trends.
Note: it can also be displayed as a histogram.

Interpretation:
The interpretation is straightforward, the greater the width of the range between the two bands drawn, the greater the strength of the trend, long or short on the basis of which of the two is above the zero line. The input signal can also be generated by crossing the two functions and if RWIT crosses RWIB from bottom to top, there is a BUY signal. Combined use with a momentum indicator that eliminates false signals is appropriate.

Recommended Parameters:
Periods: **7**

Example:

```
SET rdTop = RWIT (7)
SET rdDown = RWIB (7)
CROSSOVER (rdTop, rdDown)
```

Evaluates to true when the RWIT crosses above the RWIB.

Relative Strength Index Percent (RSI Percent)

RSIPercent (@vector, @periods)

Overview:
The RSI Percent is an oscillator calculated over the standard RSI, with the same periods.

Interpretation:
The interpretation is the same as the standard RSI: consider an overbought territory all values above 70, and an oversold territory all values below 30.

Recommended Parameters:
Vector: CLOSE
Periods: 14

Example:

RSIPercent (CLOSE, 14) > 80

Evaluates to true when the oscillator of the RSI is in overbought territory.

Schaff Trend Cycle

```
SchaffTrendCycle (@vector, @periods, @shortCycle, @longCycle, @maType)
STC (@vector, @periods, ShortCycle, @longCycle, @maType)
```

Overview:
The Schaff Trend Cycle indicator by Doug Schaff, is based on the assumption that trends accelerate and slow down cyclically, thus reacting to abrupt price changes, while ignoring small variations. According to Doug Schaff's conception of the indicator, after a certain period of time, the trend returns to the original point of development on the markets, and the cycle of its subsequent movements begins to repeat itself.

Interpretation:
The common use of the Schaff Trend Cycle indicator is represented by the generation of a sell signal when the indicator line crosses below the 75% line, and the generation of a buy signal when the indicator line crosses above the 20% line.
Doug Schaff also proposes a filtering of the signals: if the bar that follows the signal bar closes beyond the previous high, this means a greater probability of the movement of the prices and, consequently, of the purchases. The sell signal is represented by the opposite situation, when the bar that follows the signal bar, closes below the low of the previous bar.

Recommended Parameters:
Vector: **CLOSE**
Periods: **10**
Short Cycle: **23**
Long Cycle: **50**
MA Type: **SIMPLE**

Example:

```
SET schaft = STC (CLOSE, 10, 23, 50, SIMPLE)
CROSSOVER (schaft, 20)
```

Evaluates to true when the Schaff Trend Cycle indicator crosses above the 20% percent line, generating a buy signal.

Sine Wave

SineWave(@vector, @periods)
SW(@vector, @periods)
LeadSineWave(@vector, @periods)
LSW(@vector, @periods)

Overview:
The Sine Wave is a technical analysis tool based on advanced mathematics that indicates if a market is trending or in a cycle mode. It helps traders identify the start and finish of a trending move as well as possible shifts in the trend. This leading indicator is also called the MESA indicator and was developed by John Ehlers based on an algorithm, that was originally applied to digital signal processing. It consists of 2 lines, called the Sine Wave and the Lead Wave.

Interpretation:
When the price is trending, the lines do not cross and usually run parallel and distant from each other. Crossovers between these lines could signal turning points and generate buy or sell signals under the right conditions. The indicator can also signal if a market is overbought or oversold (meaning the price is unjustifiably high or unjustifiably low), which could point to a stalling or reversal of the trend.

Recommended Parameters:
Vector: **CLOSE**
Periods: **14**

Example:

CROSSOVER (SW(CLOSE, 14), LSW(CLOSE, 14))

Evaluates to true when the Sine Wave crosses above the Lead Sine Wave.

Stochastic Oscillator

SOPK (@%K Periods, @%KSlowPeriods, @%DPeriods, @maType)
SOPD (@%K Periods, @%KSlowPeriods, @%DPeriods, @maType)

Overview:
The Stochastic Oscillator is a popular indicator that shows where a security's price has closed in proportion to its closing price range over a specified period of time.

Interpretation:
The Stochastic Oscillator has two components: %K (the SOPK function) and %D (the SOPD function). %K is most often displayed on a stock chart as a solid line and %D is often shown as a dotted line. The most widely used method for interpreting the Stochastic Oscillator is to buy when either component rises above 80 or sell when either component falls below 20. Another way to interpret the Stochastic Oscillator is to buy when %K rises above %D, and conversely, sell when %K falls below %D.

Recommended Parameters:

Function		BeeTrader Indicator	
%K Periods	**9**	%K Periods	**9**
%K SlowPeriods	**3**	%K SlowPeriods	**3**
%D Periods	**9**	%D Periods	**9**
MA Type	**SIMPLE**	MA Type	**SIMPLE**
		highMark	**80**
		lowMark	**20**

Example:

SOPK (9, 3, 9, SIMPLE) > 80 OR SOPD (9, 3, 9, SIMPLE) > 80

Evaluates to true when the Stochastic Oscillator is in oversold territory.

Trix Oscillator

```
TRIX (@vector, @periods)
```

Overview:
TRIX is a momentum oscillator that shows the rate of change of an exponentially averaged closing price.

Interpretation:
The most common usage of the TRIX oscillator is to buy when the oscillator rises and sell when the oscillator falls.

Recommended Parameters:
Vector: **CLOSE**
Periods: **9**

Example:

```
TRIX (CLOSE, 9) > 0.9
```

Evaluates to true when TRIX is in overbought territory.

True Range

```
TrueRange ()
TR ()
```

Overview:
The True Range measures the price movement range of the current bar, as the difference between the high and low prices. It’s an immediate volatility indicator.

Example:

```
TR () > 1.95
```

Evaluates to true when the True Range is greater than 1.95.

Ultimate Oscillator

UltimateOscillator (@cycle1, @cycle2, @cycle3)
ULTOSC (@cycle1, @cycle2, @cycle3)

Overview:
Larry Williams wanted to build an oscillator that didn't suffer much from different calculation periods, which often invalidates the results of many 7, 14, 21 periods indicators. In his algorithm he takes into account three different periods trying to get an oscillator with few false signals.
The concept starts from the observation that the pressure, the momentum moving the prices, generates a more developed, defined and lasting trend.

Interpretation:
As for other oscillators this also has values that vary in a range from 0 to 100. There are overbought and oversold zones, defined by the values over 70 and below 30. A common practice is to act in case of divergences between the graph of prices and that of the indicator. In this particular case, we have a buy signal in the case of a divergence so formed: the oscillator has a new peak or simply continues to rise while the price marks a new low. The sell signal is generated with a contrary conformation.

Recommended Parameters

Function		BeeTrader Indicator	
Cycle1	**7**	Cycle1	**7**
Cycle2	**14**	Cycle2	**14**
Cycle3	**28**	Cycle3	**28**
		lowMark	**30**
		highMark	**70**

Example:

```
SET U = ULTOSC (7,14, 28)
U > REF (U, 1) AND TREND (CLOSE, 30) = DOWN
```

Evaluates to true when the Ultimate Oscillator is raising and there is a downside trend.

Vertical Horizontal Filter

VerticalHorizontalFilter (@vector, @periods)
VHF (@vector, @periods)

Overview:
The Vertical Horizontal Filter (VHF) identifies whether a market is in a trending or a choppy movement phase.

Interpretation:
The VHF indicator is most commonly used as an indicator of market volatility. It is also frequently used as a component to other technical indicators.

Recommended Parameters:

Function:		BeeTrader Indicator:	
Vector	**CLOSE**	Vector	**CLOSE**
Periods	**21**	Periods	**21**
		lowMark	**0,2**
		highMark	**0,3**

Example:

VHF (CLOSE, 21) < 0.2

Evaluates to true when VHF is lower than 0,2.

Volume Oscillator

VolumeOscillator (@shortTermPeriods, @longTermPeriods, @maType, @pointsOrPercent)
VO (@shortTermPeriods, @longTermPeriods, @maType, @pointsOrPercent)

Overview:
The Volume Oscillator shows a spread of two different moving averages of volume over a specified period of time.

Interpretation:
Offers a clear view of whether or not volume is increasing or decreasing.

Recommended Parameters:
Short Term Periods: **9**
Long Term Periods: **21**
MA Type: **SIMPLE**
Points or Percent: **PERCENT**

Example:

VO (9, 21, SIMPLE, PERCENT) > 0

Evaluates to true when the Volume Oscillator is in positve territory.

Williams %R

WilliamsPctR (@periods)
WPR (@periods)

Overview:
Developed by trader Larry Williams, the Williams' %R indicator measures overbought/oversold levels. This indicator is similar to the Stochastic Oscillator. The outputs range from 0 to -100.

Interpretation:
The market is considered overbought when the %R is in a range of 0 to -20, and oversold when %R is in a range of -80 to -100.

Recommended Parameters:

Function		BeeTrader Indicator	
Periods	**14**	Periods	**14**
		lowMark	**-80**
		highMark	**-20**

Example:

WPR (14) < -80

Evaluates to true when Williams' %R is oversold.

Williams Accumulation Distribution

WilliamsAccumulationDistribution ()
WAD ()

Overview:
Another indicator developed by trader Larry Williams, the Accumulation / Distribution indicator shows a relationship of price and volume.

Interpretation:
When the indicator is rising, the security is said to be accumulating. Conversely, when the indicator is falling, the security is said to being distributing. Prices may reverse when the indicator converges with price.

Example:

WAD () < 1

Evaluates to true when Williams' Accumulation / Distribution is below 1.

Z-Score

ZScore (@vector, @periods)

Overview:
Z-Score is the number of standard deviations from the mean a data point is. More technically it's a measure of how many standard deviations below or above the population mean a raw value is.

Interpretation:
Values above +2 are an indication there is a 95% probability the values in the input vector will decline. Values below -2 are an indication there is a 95% probability the values in the input vector will increase.

Recommended Parameters:
Vector: CLOSE
Periods: 14

Example:

ZScore(CLOSE, 14) < -2

Evaluates to true when the Z-Score is below -2.

Index Functions

This section covers technical indicators that are known as indexes, such as the famous Relative Strength Index, Historical Volatility Index, and many others.

Accumulative Swing Index

```
AccumulativeSwingIndex (@limitMoveValue)
ASI (@limitMoveValue)
```

Overview:
The Accumulation Swing Index (Wilder) is a cumulative total of the Swing Index, which shows comparative price strength within a single security by comparing the current open, high, low, and close prices with previous prices.

Interpretation:
The Accumulation Swing Index may be analyzed using technical indicators, line studies, and chart patterns, as an alternative view of price action.

Recommended Parameters:
Limit Move Value: **1**

Example:

```
TREND (ASI (1), 30) > UP
```

Evaluates to true when the Accumulative Swing Index is trending upwards.

Commodity Channel Index

```
CommodityChannelIndex (@periods, @maType)
CCI (@periods, @maType)
```

Overview:
Donald Lambert developed the CCI indicator. Although the purpose of this indicator is to identify cyclical turns in commodities, it is often used for securities.

Interpretation:
This indicator oscillates between an overbought and oversold condition and works best in a sideways market.

Recommended Parameters

Function		BeeTrader Indicator	
Periods	**21**	Periods	**21**
MA Type	**SIMPLE**	MA Type	**SIMPLE**
		Signal Periods	**10**
		lowMark	**-80**
		highMark	**80**

Example:

```
CCI (21, SIMPLE) > 80 AND REF (CCI (21, SIMPLE), 1) < 80
```

Evaluates to true when the CCI has just moved into positive territory.

Comparative Relative Strength Index

ComparativeRelativeStrenghtIndex (@vector1, @vector2)
CRSI (@vector1, @vector2)

Overview:
The Comparative Relative Strength index compares one vector with another.

Interpretation:
The base vector is outperforming the other vector when the Comparative RSI is trending upwards.

Recommended Parameters:
Vector1: **CLOSE**
Vector2: **[Any]**

Example:

CRSI (CLOSE, VOLUME) > 1

Evaluates to true when the trend in price has outpaced the trend in volume.

Elder Thermometer Index

ElderThermometerIndex (@periods, @maType)
ETHERM (@periods, @maType)

ElderThermometerAverage (@periods, @maType)
ETHERMAVG (@periods, @maType)

Overview:
The Elder Thermometer Index is a volatility indicator, usually dubbed as "Market Temperature", which is used to differentiate between sleepy, quiet and hot market periods.

Interpretation:
Market Thermometer gives trading signals based on the relationship between its histogram and its moving average. The best time to enter new positions is when Market Thermometer falls below its moving average, while exits are signaled when Market Thermometer rises to triple the height of its moving average.

Recommended Parameters:
Periods: **22**
Ma Type: **SIMPLE**

Example:

```
SET T = ElderThermometerIndex (22, SIMPLE)
SET Trigger = SMA (T, 22)
CROSSUNDER (T, Trigger)
```

Evaluates to true when the Elder Thermometer Index crosses under its trigger line.

Gopalakrishnan Range Index

GopalakrishnanRangeIndex (@periods)
GRI (@periods)

Overview:
Gopalakrishnan Range Index, developed by Jayanthi Gopalakrishnan, attempts to determine the variability of price data based on the log of the price range over a number of periods.
Note: beeTrader® returns always an absolute value of the GRI.

Interpretation:
The objective of this index is not to generate buy or sell signals, but to provide an overall picture of price variability. Contiguous prices leads to lower values of the GRI. Prices spotting many gaps leads to higher values of the GRI.

Recommended Parameters:
Periods: **10**

Example:

GRI (10) > 0.5

Evaluates to true when the GRI is greater than 0.5.

Historical Volatility Index

HistoricalVolatilityIndex (@vector, @periods, @barHistory, @standardDeviations)
HistoricalVolatility (@vector, @periods, @barHistory, @standardDeviations)
HVI (@vector, @periods, @barHistory, @standardDeviations)

Overview:
Historical volatility is the log-normal standard deviation. The formula for a 30-periods historical volatility index between 1 and 0 is: Stdev(Log(Close / Close Yesterday), 30) * Sqrt(365). Some traders use 252 instead of 365 for the bar history that is used by the square root function. The Log value is a natural log (i.e. Log10).

Interpretation:
High values of HVI indicate that the stock is volatile, while low values of HVI indicate that the stock is either flat or trending steadily.

Recommended Parameters:

Vector:	**CLOSE**
Periods:	**30 o 60 o 90**
Bar History:	**365**
Standard Deviations:	**2**

Example:

HVI (CLOSE, 30, 365, 2) > 0.01

Evaluates to true when the HVI is greater than 0.01 (1%).

Market Facilitation Index

MarketFacilitationIndex ()
MKTFI ()

Overview:
The Market Facilitation Index (MFI) is the creation of Bill Williams. The indicator endeavors to establish the effectiveness of price movement by computing the price movement per volume unit. This is accomplished by subtracting the days low from the high and dividing the result by the total volume.

Interpretation:
As an indicator on its own the MFI is of no significant value. Nonetheless, if the current price candles, MFI and volume are compared to the previous candles, MFI and volume, the index starts to have some significant tradable data. The four possible groupings of MFI and volume were termed Green, Fade, Fake and Squat by Williams.
Green – The MFI increases and the volume increases: This means that the number of participants entering the market increases, therefore the volume increases and the fresh incoming players align their positions in the direction of candlestick growth. Notice the long solid candles in the candlestick chart which indicates that the trend has begun and is picking up speed.
Fade – The MFI falls and volume falls: It means that the market participants are indifferent and the price movement is small on small volumes. This usually happens at the end of a trend.
Fake – The MFI increases, but the volume falls: It is highly likely that the market is being supported by broker speculation and not any significant client volume.
Squat – The MFI falls, but the volume increases: In this particular situation bulls and bears are fighting between themselves to see who will dominate the next trend. These battles are noticeable by the large sell and buy volumes. However, the price does not change appreciably since the strengths are equal. One of the competing parties, either the buyers or the sellers, will ultimately triumph in the battle. Usually, the fracture of such a candle indicates if this particular candle determines the continuation of the trend, or terminates the trend.

Example:

```
SET M = MKTFI()
M > REF(M, 1)
```

Evaluates to true when the MFI is raising.

Mass Index

```
MassIndex (@periods)
MI (@periods)
```

Overview:
The Mass Index identifies price changes by indexing the narrowing and widening change between high and low prices.

Interpretation:
According to the inventor of the Mass Index, reversals may occur when a 25-period Mass Index rises above 26 or falls below 25.

Recommended Parameters:

Function		BeeTrader Indicator	
Periods	**25**	Periods	**25**
		lowMark	**25**
		highMark	**26**

Example:

```
SET M = MassIndex (25)
LASTIF (CROSSOVER (M, 26)) < 10
AND
CROSSOVER (25, M)
```

Evaluates to true when the 25-periods Mass Index just crossed below the low threshold (25) and had crossed above the high threshold (26) in the previous 10 bars.

Money Flow Index

```
MoneyFlowIndex (@periods)
MFI (@periods)
```

Overview:
The Money Flow Index measures money flow of a security, using volume and price for calculations.

Interpretation:
Market bottoms may be identified by values below 20 and tops may be identified by values above 80. Divergence of price and Money Flow Index may be watched.

Recommended Parameters:
Periods: **15**

Example:

```
SET M = MFI (15)
CROSSOVER (M, 20)
```

Evaluates to true when the indicator crosses above the low mark (20).

Negative Volume Index

```
NegativeVolumeIndex (@vector)
NVI (@vector)
```

Overview:
The Negative Volume Index is similar to the Positive Volume Index, except it puts focus on periods when volume decreases from the previous period.

Interpretation:
The interpretation of the Negative Volume Index is that well-informed investors are buying when the index falls and uninformed investors are buying when the index rises.

Recommended Parameters:
Vector: **CLOSE**

Example:

```
TREND (NVI (CLOSE), 30) = DOWN
```

Evaluates to true when NVI is trending downwards.

Performance Index

```
PerformanceIndex (@vector)
PFI (@vector)
```

Overview:
The Performance indicator calculates price performance as a normalized value or percentage.

Interpretation:
A Performance indicator shows the price of a security as a normalized value. If the Performance indicator shows 50, then the price of the underlying security has increased 50% since the start of the Performance indicator calculations. Conversely, if the indictor shows -50, then the price of the underlying security has decreased 50% since the start of the Performance indicator calculations.

Recommended Parameters:
Vector: **CLOSE**

Example:

```
PFI (CLOSE) > 45
```

Evaluates to true when the performance index is over 45%.

Positive Volume Index

```
PositiveVolumeIndex (@vector)
PVI (@vector)
```

Overview:
The Positive Volume Index puts focus on periods when volume increases from the previous period.

Interpretation:
The interpretation of the Positive Volume Index is that the majority of investors are buying when the index rises, and selling when the index falls.

Recommended Parameters:
Vector: **CLOSE**

Example:

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

Evaluates to true when PVI is trending upwards.

Range Action Verification Index (RAVI)

RAVI (@vector, @shortCycle, @longCycle)

Overview:
The RAVI indicator (Range Action Verification Index) is a TREND indicator on daily timeframe developed by Tushar Chande. The RAVI is calculated using Moving Average of different lengths.

Interpretation:
A value greater than 0.3 identifies an upside trend, a value lower than -0.3 identifies a downside trend.

Recommended Parameters:
Vector: **CLOSE**
Short Cycle: **7**
Long Cycle: **65**

Example:

```
RAVI (CLOSE, 7, 65) > 0,3
```

Evaluates to true when the RAVI is greater than 0.3.

Relative Strength Index

RelativeStrengthIndex (@vector, @periods)
RSI (@vector, @periods)

Overview:
The RSI is popular indicator developed by trader Welles Wilder. The RSI is a popular indicator that shows comparative price strength within a single security.

Interpretation:
The most widely used method for interpreting the RSI is to consider an overbought territory all values above 70, and an oversold territory all values below 30. In the past the values 80 and 20 have been used as thresholds instead of 70 and 30, to avoid false positives.

Recommended Parameters:

Function		BeeTrade Indicator	
Vector	**CLOSE**	Vector	**CLOSE**
Periods	**14**	Periods	**14**
		lowMark	**30**
		highMart	**70**

Example:

RSI (CLOSE, 14) > 80

Evaluates to true when the RSI is greater than 80.

Stochastic Momentum Index

SMIK(@%KPeriods, @%KSmoothPeriods, @%KDoublePeriods, @%DPeriods, @maType, @%DMAType)
SMID(@%KPeriods, @%KSmoothPeriods, @%KDoublePeriods, @%DPeriods, @maType, @%DMAType)

Overview:
The Stochastic Momentum Index, developed by William Blau, first appeared in the January 1993 issue of Stocks & Commodities magazine. This indicator plots the closeness of price relative to the midpoint of the recent high / low range.

Interpretation:
The Stochastic Momentum Index has two components: %K (SMIK) and %D (SMID). %K is most often displayed on a chart as a solid line and %D is often shown as a dotted line. The most widely used method for interpreting the Stochastic Momentum Index is to buy when either component rises above 40 or sell when either component falls below 40. Another way to interpret the Stochastic Momentum Index is to buy when %K rises above %D, or sell when %K falls below %D.

Recommended Parameters:

Function		BeeTrader Indicator	
%K Periods	**14**	%K Periods	**14**
%K SmoothPeriods	**2**	%K SmoothPeriods	**2**
%K Double Periods	**3**	%K Double Periods	**3**
%D Periods	**9**	%D Periods	**9**
MA Type	**SIMPLE**	MA Type	**SIMPLE**
%D MA Type	**SIMPLE**	%D MA Type	**SIMPLE**
		lowMark	**-40**
		highMark	**40**

Example:

SMID (14, 2, 3, 9, SIMPLE, SIMPLE) > 40 OR
SMIK (14, 2, 3, 9, SIMPLE, SIMPLE) > 40

Evaluates to true when the Stochastic Momentum Index is in oversold territory.

Swing Index

SwingIndex (@limitMoveValue)
SI (@limitMoveValue)

Overview:
The Swing Index is a popular indicator that shows comparative price strength within a single security by comparing the current open, high, low, and close prices with previous prices.

Interpretation:
The Swing Index is a component of the Accumulation Swing Index.

Recommended Parameters:
Limit Move Value: **1**

Example:

SI (1) > 0

Evaluates to true when the Swing Index is in positive territory.

Trade Volume Index

TradeVolumeIndex (@vector, @minimumTickValue)
TVI (@vector, @minimumTickValue)

Overview:
The Trade Volume index shows whether a security is being accumulated or distributed (similar to the Accumulation/Distribution index).

Interpretation:
When the indicator is rising, the security is said to be accumulating. Conversely, when the indicator is falling, the security is said to being distributing. Prices may reverse when the indicator converges with price.

Recommended Parameters:
Vector: **CLOSE**
Minimum Tick Value: **0.25**

Example:

TVI (CLOSE, 0.25) > 0

Evaluates to true when the Trade Volume Index is in positive territory.

Other Technical Indicators

This section highlights all indicators and functions not fitting in any of the previous sections.

Center of Gravity

```
CenterOfGravity (@vector, @periods)
COG (@vector, @periods)
```

Overview:
The Center of Gravity (COG) indicator is a technical indicator developed by John Ehlers in 2002, used to identify potential turning points in the price as early as possible. In fact, the creator claims zero lag to the price, and the smoothing effect of the indicator helps to spot turning points clearly and without distractions.

Interpretation:
A buy signal is triggered when the COG line crosses above the signal line, while a sell signal is triggered when the COG line crosses below the signal line. Just like other oscillators, the COG indicator returns the best results in range-bound markets and should be avoided when the price is trending.

Recommended Parameters:

Function		BeeTrader Indicator	
Vector	**CLOSE**	Vector	**CLOSE**
Periods	**21**	Periods	**21**
		Signal Periods	**3**

Example:

```
SET C = CenterOfGravity (CLOSE, 21)
SET SignalLine = SMA (C, 3)
CROSSOVER (C, SignalLine)
```

Evaluates to true when the COG line crosses above the signal line.

Correlation Analysis

CorrelationAnalysis (@vector1, @vector2)
CA (@vector1, @vector2)

Overview:
Correlation analysis is used to determine the relationship between two vectors.

Interpretation:
The function returns a value indicating the relationship between two vectors. The vectors may contain price, indicator values, or other values.

Recommended Parameters:
Vector1: **[Any Vector]**
Vector2: **[Any Vector]**

Example:

CA (CLOSE, SMA (CLOSE, 14)) > 0.99

Evaluates to true when the close price movement highly correlates with the 14-day SMA movement.

Ehler Fisher Transform

EhlerFisherTransform (@periods)
EFT (@periods)

EhlerFisherTrigger (@periods)
EFTT (@periods)

Overview:
The Fisher Transform is a technical indicator created by John Ehlers that converts prices into a Gaussian normal distribution. In this way, the indicator highlights when prices have moved to an extreme, based on recent prices. This may help in spotting turning points in the price of an asset. It also helps show the trend and isolate the price waves within a trend.

Interpretation:
The Fisher Transform indicator is unbounded, which means extremes can occur for a long time. An extreme is based on the historical readings for the asset in question and indicates the possibility of a reversal. This should be confirmed by the Fisher Transform changing direction. For example, following a strong price rise and the Fisher Transform reaching an extremely high level, when the Fisher Transform starts to head lower that could signal the price is going to drop, or has already started dropping. The Fisher Transform has a signal trigger line attached to it, a moving average of the Fisher Transform value, so it moves slightly slower than the Fisher Transform line. When the Fisher Transform crosses the trigger line it is used by some traders as a trade signal. For example, when the Fisher Transform drops below the signal line after hitting an extreme high, that could be used as a signal to sell a current long position. As with many indicators, the Fisher will provide many trade signals. Many of these will not be profitable signals. Therefore, some traders prefer to use the indicator in conjunction with trend analysis. For example, when the price is rising overall, use the Fisher Transform for buy and sell signals, but not for short-sell signals. During a downtrend, use it for short-sell signals and ideas on when to cover.

Recommended Parameters:
Periods: **21**

Example:

```
SET F = EFT (21)
SET T = EFTT(21)
CROSSOVER (F,  T)
```

Evaluates to true when the EFT crosses above its signal trigger line.

Elder Ray

ElderRayBearPower (@periods, @maType)
ERBear (@periods, @maType)

ElderRayBullPower (@periods, @maType)
ERBull (@periods, @maType)

Overview:
Elder Ray is based on oscillators, labeling the components as "bull power" or "bear power." These are combined with an exponential moving average (EMA), which is a trend-following indicator essential to the calculation. Bull power is a simple calculation, derived by subtracting an exponential moving average of closing prices from a high price of any given security. Bear power subtracts the EMA from the corresponding low price of that trading day.

Interpretation:
By measuring the distance from the bar's high to the EMA, bull power represents the capacity of bulls to push prices above the average consensus of value (price). Bull power rises when bulls are stronger and falls when they are weaker, even becoming negative when they are utterly weak. Bear power, by contrast, is the capacity of bears to push prices below the moving average. The distance between the low and the EMA, which widens when the bears are weaker and narrows when they are stronger, gives this figure. Bear power is typically negative, so if it turns positive, the bulls have taken complete control.

Recommended Parameters:
Periods: **13**
MA Type: **EXPONENTIAL**

Example:

```
SET Bear = ERBear (13, EXPONENTIAL)
(Bear < 0) AND (Bear > REF (Bear, 1))
```

Evaluates to true when the Elder Ray Bear Power is in negative territory but raising.

Gann Swing

GannSwing (@seriesHigh, @seriesLow)
GS (@seriesHigh, @seriesLow)

Overview:
The Gann Swing is an indicator developed to identify market swings.

Interpretation:
A raising value of the Gann Swing indicator identify an upside trend, while a declining value identify a downside trend. Typically a signal is generated when the slope of Gann Swing changes from positive to negative or viceversa.

Recommended Parameters:
seriesHigh: **HIGH**
seriesLow: **LOW**

Example:

```
SET GSwing = GannSwing (HIGH, LOW)
SET Slope = GSwing - REF (GSwing, 1)
CROSSOVER (Slope, ZeroLine())
```

Evaluates to true when the slope of the Gann Fann changes from negative to positive.

Gaussian Filter

GaussianFilter (@periods, @periods, @poles)
GAUSS (@periods, @periods, @poles)

Overview:
It’s a low-pass filter eliminating lag of classic smoothing indicators such as moving averages, while maintaining responsiveness and precision. The Poles parameters can be used to adjust responsiveness of the indicator.

Interpretation:
The Gaussian Filter indicator can be used exactly as a moving average, with the advantage of a much reduced lag on price changes.

Recommended Parameters:
Vector: **CLOSE**
Periods: **20**
Poles: **1**

Example:

```
SET GF = GaussianFilter (CLOSE, 20, 1)
(GF > REF (GF, 1)) AND (GF > CLOSE)
```

Evaluates to true when the Gaussian Filter is raising and is higher than the close price.

Highest High Value

HighestHighValue (@periods)
HHV (@periods)

Overview:
Returns the highest value of the high price over the specified number of periods.

Interpretation:
Used as a component for calculation by many other technical indicators.

Recommended Parameters:
Periods: **21**

Example:

```
HIGH = HHV (21)
```

Evaluates to true when the high is the highest high in the past 21 bars.

Lowest Low Value

LowestLowValue (@periods)
LLV (@periods)

Overview:
Returns the lowest value of the low price over the specified number of periods.

Interpretation:
Used as a component for calculation by many other technical indicators.

Recommended Parameters:
Periods: **21**

Example:

LOW = LLV (21)

Evaluates to true when the low is the lowest low in the past 21 bars.

Median Price

MEDIANPRICE ()
MP ()

Overview:
A Median Price is simply an average of one period's high and low values.

Interpretation:
A Median Price is often used as an alternative way of viewing price action and also as a component for calculating other technical indicators.

Example:

CROSSOVER (CLOSE, SMA (MP (), 14))

Evaluates to true when the close crossed over the 14-day SMA of the Median Price.

On Balance Volume

OnBalanceVolume (@vector)
OBV (@vector)

Overview:
The On Balance Volume index shows a relationship of price and volume in the form of a momentum index.

Interpretation:
On Balance Volume generally precedes actual price movements. The premise is that well-informed investors are buying when the index rises and uninformed investors are buying when the index falls.

Recommended Parameters:
Vector: **CLOSE**

Example:

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

Evaluates to true when OBV is trending upwards.

Parabolic Stop & Reversal (Parabolic SAR)

ParabolicSAR (@minAF, @maxAF)
PSAR (@minAF, @maxAF)

Overview:
Author Welles Wilder developed the Parabolic SAR. This indicator is always in the market (whenever a position is closed, an opposing position is taken). The Parabolic SAR indicator is most often used to set trailing price stops.

Interpretation:
A stop and reversal (SAR) occurs when the price penetrates a Parabolic SAR level.

Recommended Parameters:
Min AF (Accumulation Factor): **0.02**
Max AF (Accumulation Factor): **0.2**

Example:

```
CROSSOVER (CLOSE, PSAR (0.02, 0.2)) = TRUE
```

Evaluates to true when the close recently crossed over the Parabolic SAR.

Parabolic Stop & Reversal Extended (Parabolic SAR Extended)

ParabolicSARExtended (@startValue, @offsetOnReverse, @initialAccelerationLong, @accelerationLong, @maxAccelerationLong, @initialAccelerationShort, @accelerationShort, @maxAccelerationShort)
PSAREXT (@startValue, @offsetOnReverse, @initialAccelerationLong, @accelerationLong, @maxAccelerationLong, @initialAccelerationShort, @accelerationShort, @maxAccelerationShort)

Overview:
The Parabolic SAR Extended is derived from the standard Parabolic SAR. This indicator provides additional control over the extension of the output value compared to the standard Parabolic SAR.

Interpretation:
A stop and reversal (SAR) occurs when the price penetrates a Parabolic SAR level.

Recommended Parameters:
Start Value: 0
Offset On Reverse: 0
Initial Acceleration Long: 0.02
Acceleration Long: 0.02
Max Acceleration Long: 0.2
Initial Acceleration Short: 0.02
Acceleration Short: 0.02
Max Acceleration Short: 0.2
Note:
Start Value = 0 – initial indicator sign is automatically determined.
Start Value > 0 – initial indicator sign is always positive.
Start Value < 0 – initial indicator sign is always negative.

Example:

CROSSOVER (PSAREXT (0, 0, 0.02, 0.02, 0.2, 0.02, 0.02, 0.2), ZeroLine())

Evaluates to true when the Parabolic SAR Extended has entered the positive territory.

Price Rate of Change (ROC)

PriceRateOfChange (@vector, @periods)
PROC (@vector, @periods)
ROC (@vector, @periods)

Overview:
The Price ROC shows the difference between the current price and the price one or more periods in the past.

Interpretation:
A 12-day Price ROC is most often used as an overbought/oversold indicator.

Recommended Parameters:
Vector: **CLOSE**
Periods: **12**

Example:

ROC (CLOSE, 12) > 0 AND REF (ROC (CLOSE, 12),1) < 0

Evaluates to true when the Price ROC recently shifted into positive territory.

Price CCI Divergence

PriceCCIDivergence(@periods,@cciPeriods, @angle, @highThreshold, @lowThreshold)
PriceCCIDivergence(@periods,@cciPeriods, @angle, @highThreshold, @lowThreshold, @filterThreshold)
PCCIDIV (@periods, @cciPeriods, @angle, @highThreshold, @lowThreshold)
PCCIDIV(@periods,@cciPeriods,@angle,@highThreshold,@lowThreshold, @filterThreshold)

Overview:
This indicator shows bearish and bullish divergences between the CCI and prices. It contains two indicators: the detection of bearish and bullish divergences.

Interpretation:
1 indicates a possible upward trend;
0 indicates a trading range;
-1 indicates a possible bearish trend.

Recommended Parameters:
Periods: **30**
CCI Periods: **20**
Angle: **0**
High Threshold: **100**
Low Threshold: **-100**
Vector: **CLOSE**

Optional Parameters:
Filter Threshold: **0.5**

Example:

PCCIDIV (30, 20, 0, 100, -100) > 0

Evaluates to true when the Price CCI Divergence identifies a possible upward trend.

Price Volume Trend

PriceVolumeTrend (@vector)
PVT (@vector)

Overview:
Also known as Volume Price Trend. This indicator consists of a cumulative volume that adds or subtracts a multiple of the percentage change in price trend and current volume, depending upon their upward or downward movements. PVT is used to determine the balance between a stock's demand and supply.This indicator shares similarities with the On Balance Volume index.

Interpretation:
The Price and Volume Trend index generally precedes actual price movements. The premise is that well-informed investors are buying when the index rises and uninformed investors are buying when the index falls.

Recommended Parameters:
Vector: **CLOSE**

Example:

TREND (PVT (CLOSE), 20) = UP

Evaluates to true when PVT is trending upwards.

QStick

QSTICK (@periods, @maType)
QS (@periods, @maType)

Overview:
The QStick Indicator, designed by Tushar Chande, is used with intraday timeframe and very often as an aid to Candlestick analysis. The QStick Indicator is the Moving Average of the difference between close and open that measures and represents the strength and consistency of bearish or bullish. It is designed also calculating a Signal Line using a 15-period Simple Moving Average.

Interpretation:
Signals are generated by crossing the zero line. The buy signal is generated when the QStick Indicator passes through the zero line and then when passing from the negative quadrant to the positive one. The sell signal is triggered when the indicator drops back into negative territory below the zero line. The Signal Line will be an aid to identify the beginnings of the ascent or descent, anticipating the waiting for the indicator to cross the zero.

Recommended Parameters:

Function		BeeTrader Indicator	
Periods	**10**	Periods	**10**
MA Type	**SIMPLE**	MA Type	**SIMPLE**
		Signal Periods	**15**

Example:

```
SET QIndicator = QStick (10, SIMPLE)
SET SignalLine = SMA (QIndicator, 15)
CROSSOVER (QIndicator, 0) AND SignalLine < QIndicator
```

Evaluates to true when the Qstick passes from the negative to the positive territory, and is greater than it’s signal line.

Simple Super Trend

SimpleSuperTrend (@atrLength, @atrMultiplier)
SST (@atrLength, @atrMultiplier)

Overview:
The Simple SuperTrend is a simplified version, therefore with fewer parameters and a simpler algorithm, of the indicator devised by Oliver Seban applicable to almost all financial instruments. It is similar in its conception to the Parabolic SAR, with the advantage of being positioned horizontally in the moments of trading range, thus avoiding an early exit from the market. The Simple SuperTrend is an indicator that can be very useful therefore for the correct exit from a market position.

Interpretation:
It’s interpretation is straightforward because it consists of a series of points, one for each bar, drawn under the close for upward trends and above the close for bearish trends. The indicator has noticeable delay in trend swings, which could lead to substantial losses, so it is advisable to use it in combination with other indicators that identify the collapse of the TREND, for example the Price ROC.

Recommended Parameters:
Atr Length: **9**
Atr Multiplier: **1**

Example:

ST (9, 1) > CLOSE

Evaluates to true when Simple SuperTrend is greater than the close.

Standard Deviations

StandardDeviations (@vector, @periods, @standardDeviations, @maType)
SDV (@vector, @periods, @standardDeviations, @maType)
StdDev (@vector, @periods, @standardDeviations, @maType)

Overview:
Standard Deviation is a common statistical calculation that measures volatility. Many technical indicators rely on standard deviations as part of their calculation.

Interpretation:
Major highs and lows often accompany extreme volatility. High values of standard deviations indicate that the price or indicator is more volatile than usual.

Recommended Parameters:
Vector: **CLOSE**
Periods: **21**
Standard Deviations: **2**
MA Type: **SIMPLE**

Example:

SDV (CLOSE, 21, 2, SIMPLE) > REF (SDV (CLOSE, 21, 2, SIMPLE), 10)

Evaluates to true when 21 period Standard Deviations are greater than 10 days ago.

Standard Error

```
StandardError (@vector, @periods, @smoothPeriods)
StdErr (@vector, @periods, @smoothPeriods)
```

Overview:
Standard Error is a statistical calculation that measures the standard error in a data series.

Interpretation:
The greater the error the more the data will deviate from the Linear Regression.

Recommended Parameters:
Vector: **CLOSE**
Periods: **21**
Standard Deviations: **3**

Example:

```
STDERR (CLOSE, 21, 3) > REF (STDERR (CLOSE, 21, 3), 10)
```

Evaluates to true when the 21-periods Standard Error is greater than 10 periods ago.

Super Trend

SuperTrend (@periods, @strength, @atrMultiplier)
ST (@periods, @strength, @atrMultiplier)

Overview:
The Simple SuperTrend is an indicator devised by Oliver Seban applicable to almost all financial instruments. It is similar in its conception to the Parabolic SAR, with the advantage of being positioned horizontally in the moments of trading range, thus avoiding an early exit from the market. The Simple SuperTrend is an indicator that can be very useful therefore for the correct exit from a market position.

Interpretation:
It's interpretation is straightforward because it consists of a series of points, one for each bar, drawn under the close for upward trends and above the close for bearish trends. The indicator has noticeable delay in trend swings, which could lead to substantial losses, so it is advisable to use it in combination with other indicators that identify the collapse of the TREND, for example the Price ROC.

Recommended Parameters:
Periods: **9**
Strength: **5**
Atr Multiplier: **1**

Example:

ST (9, 5, 1) > CLOSE

Evaluates to true when the SuperTrend is greater than the close.

Swing High/Low

```
SwingHigh (@vector, @occurencies, @strength, @length)
SwingLow (@vector, @occurencies, @strength, @length)
```

Overview:
The Swing High and Swing Low functions can be used to detect swings in the market.

Interpretation:
A swing is signaled by a change in the output value.

Example:
```
SwingHigh (CLOSE, 1, 3, 5) <> REF (SwingHigh (CLOSE, 1, 3, 5), 1)
```

Evaluates to true when a swing is detected.

Twiggs Money Flow

```
TwiggsMoneyFlow (@periods)
TMF (@periods)
```

Overview:
The Twiggs Money Flow is a variant of the Chaikin Money Flow, by Colin Twiggs. Compared to the Chaikin Money Flow, it uses the True Range instead of the high-low range, thus limiting the peaks due to the gaps of the financial instruments. A Welles Wilder Smoothing on volumes is also used to prevent volume peaks from affecting the result.

Interpretation:
When the TMF is higher than 0, we are in a phase of accumulation, and therefore prices will tend to rise. When TMF is less than 0, we are in a distribution phase and prices will tend to move downwards.
Note: in some cases the values are close to zero, so the beeTrader® indicator presents a Multiplier parameter to display significative values.

Recommended Parameters:

Function		BeeTrader Indicator	
Periods	**21**	Periods	**21**
		Multiplier	**1000**

Example:

```
SET T = 1000 * TMF (21)
T > 0 AND REF (T, 1) < 0
```

Evaluates to true when the TMF crosses above the zero line.

Typical Price

TypicalPrice ()
TP ()

Overview:
A Typical Price is an average of one period's high, low, and close values.

Interpretation:
A Typical Price is used as a component for the calculation of several technical indicators.

Example:

CROSSOVER (CLOSE, SMA (TP (), 14))

Evaluates to true when the close crossed over the 14-day SMA of the Typical Price.

Variance

Variance (@vector, @periods, @maType)
VAR (@vector, @periods, @maType)

Overview:
Returns the statistical variance of the vector used.

Interpretation:
The higher the price deviation, the greater the volatility. High Variance values indicate that the price, or indicator, is more volatile than usual.

Recommended Parameters:
Vector: **CLOSE**
Periods: **21**
MAType: **SIMPLE**

Example:

VAR (CLOSE, 21, SIMPLE) > REF (VAR (CLOSE, 21, SIMPLE), 10)

Evaluates to true when the 21-period Variance is greater than 10 periods ago.

Volume Rate of Chance (Volume ROC)

```
VolumeRateOfChange (@vector, @periods)
VROC (@vector, @periods)
```

Overview:
The Volume Rate of Change indicator shows whether or not volume is trending in one direction or another.

Interpretation:
Sharp Volume ROC increases may signal price breakouts.

Recommended Parameters:
Vector: **VOLUME**
Periods: **12**

Example:

```
VROC (VOLUME, 12) > 0 AND REF (VROC VOLUME, 12), 1) < 0
```

Evaluates to true when the Volume ROC recently moved into positive territory.

Weighted Close

```
WeightedClose ()
WC ()
```

Overview:
Weighted Close is an average of each day's open, high, low, and close, where more weight is placed on the close.

Interpretation:
The Weighted Close indicator is a simple method that offers a simplistic view of market prices.

Example:

```
WC () > REF (WC (), 1)
```

Evaluates to true when the weighted close is higher than the previous value.

Wilder Volatility

WilderVolatility (@periods)
WV (@periods)

Overview:
For the calculation of the arithmetic, non-statistical volatility (Historical Volatility Index), Wilder created his formula starting from the True Range, calculating the difference between bars that have passed the previous bars upwards or downwards, creating the True Range.

Interpretation:
It can be used as research or evaluation of more or less volatile financial instruments.

Recommended Parameters:
Periods: **30 or 60 or 90**

Example:

```
WV (30) > WV (60)
```

Evaluates to true when the 30-period Wilder Volatility is greater than the 60-period Wilder Volatility.

Zero Line

ZeroLine ()
ZERO ()

Overview:
The Zero Line function is used in the indicators to draw the line at 0.

Example:

```
SET PLOT3 = ZERO()
```

Draw the 0 line as third indicator output (PLOT3).

Zig Zag

ZigZag (@vector, @pointsOrPercent, @retrace)

Overview:
This indicator aims to identify market swings, or trend reversals.

Interpretation:
If the indicator has an upward trend it identifies a bullish trend, on the contrary it identifies a bearish trend.

Recommended Parameters:
Vector: **CLOSE**
PointsOrPercent: **Percent**
Retrace: **10**

Example:

```
ZigZag (CLOSE, Percent, 10) > CLOSE
```

Evaluates to true when the 10% ZigZag indicator is greater than the close.

Strategy Position

This section contains the functions relating to the status of the active Strategy, which can be used to manage the Strategy as a whole in Money Management. The use of these functions is significant only in the Signal or User Functions used however in the Signal.
Note: These functions force the execution of bar-by-bar scripts and not all bars in a single pass, and thus are computationally heavy and may slow down Strategy execution.

Example:

```
INPUTS: @trailAmount(50), @trailPercent(10), @loss(-50)

# Trailing Stop simulation
SET ru = RunUp()
SET above = ru - @trailAmount
SET pcnt = @trailAmount - (above * @trailPercent / 100.0)
SET TRAIL_COND = (RunUp() > @trailAmount) AND (OpenPosition() < pcnt)

# Stop-Loss simulation
SET STOP_COND = DrawDown() < @loss

# Exit condition
TRAIL_COND OR STOP_COND
```

Average Entry Price

AverageEntryPrice()

Overview:
The Average Entry Price is the average filled price of the financial instrument to which the Strategy is applied.

Interpretation:
The Average Entry Price function can be used to set a maximum or minimum loading price in advance for the financial instrument to which the Strategy is applied.

Example:

```
AverageEntryPrice() > Last
```

Evaluates to true when the Strategy's Average Entry Price is greater than the LAST price.

Bars Since Entry

BarsSinceEntry()

Overview:
Bars Since Entry returns the number of bars since the last signal of entry (Buy or Sell).

Interpretation:
The Bars Since Entry function can be used to close positions opened for too long.

Example:

```
BarsSinceEntry() > 25
```

Evaluates to true when more than 25 bars have passed since entering the market.

Bars Since Exit

BarsSinceExit()

Overview:
Bars Since Exit returns the number of bars since the last executed output signal (Exit Long, Exit Short, Stop Loss, Take Profit, Trailing Stop or Max Position Open).

Interpretation:
The Bars Since Exit function can be used to avoid opening too close positions over time.

Example:

```
BarsSinceExit() < 25
```

Evaluates to true when less than 25 bars have passed since exiting market position.

Current Contracts

CurrentContracts()

Overview:
Current Contracts is the number of contracts currently used by the Strategy.

Interpretation:
The Current Contracts function can be used to set a maximum or minimum number of contracts in advance for the financial instrument to which the Strategy is applied.

Example:

```
CurrentContracts() > 2
```

Evaluates to true when the Strategy's Current Contracts is greater than 2.

DrawDown

DrawDown()

Overview:
The Draw Down of a Strategy is the maximum unrealized Loss obtained from the Strategy. The return value is always a non-negative number.

Interpretation:
The Draw Down function can be used to set in advance the maximum or minimum Draw Down thresholds for the entire Strategy. The value of Draw Down must be expressed in absolute value.

Example:

```
DrawDown() > OpenPositionProfit()
```

Evaluates to true when the Strategy's Draw Down is greater than the Open Position.

Last Entry Price

LastEntryPrice()

Overview:
Last Entry Price is the filled price of the last order executed on the financial instrument to which the Strategy is applied.

Interpretation:
The Last Entry Price function can be used to reduce the quantity of contracts in an Exit Signal if the exit itself does not generate a loss.

Example:

```
LastEntryPrice() > Last
```

Evaluates to true when the Strategy’s Last Entry Price is greater than the last price.

Open Position

OpenPosition()

Overview:
The Open Position of a Strategy is the Proft/Loss of any open trade, net of commissions or slippage, or 0 if there are no trades in place.

Interpretation:
The Open Position function can be used to set in advance the thresholds of monetary exits.

Example:

```
OpenPosition() > 500
```

Evaluates to true when the Strategy’s Open Position is greater than 500.

RunUp

RunUp()

Overview:
The Run Up of a Strategy is the maximum non-consolidated Profit obtained from the Strategy since the last entry.

Interpretation:
The Run Up function can be used to set in advance the maximum or minimum Run Up thresholds for the entire Strategy.

Example:

```
RunUp() > OpenPositionProfit()
```

Evaluates to true when the Strategy's Run Up is greater than the Open Position.

Total Net Profit

TotalNetProfit()

Overview:
The Total Net Profit of a Strategy is the sum of realized Profits and Losses by the Strategy trades.

Interpretation:
The Total Net Profit function can be used to set in advance the thresholds of monetary exits referring to the entire Strategy.

Example:

```
TotalNetProfit() > 1000
```

Evaluates to true when the Strategy's Total Net Profit is greater than 1000.

Inter-Script Functions

This section shows the functions for exchanging vectors between different scripts. These functions can be used to exchange data vectors between different Charts, whatever the securities or timeframes.

Note: these functions do not perform any data transformation between different timeframes. The vector saved by SetGlobalVar is transposed without changes when it is called by the GetGlobalVar function.

Example 1:

Chart 1: Future DAX - timeframe 5 minutes

Buy Script:
```
# Save data with ID 1
SET A = SetGlobalVar (1, (EMA(CLOSE,14) > EMA(CLOSE,21)))
```

Chart 2: Future DJ EURO STOXX 50 - timeframe 5 minutes

Buy Script:
```
# Sets the B variable with the data previously saved with ID 1
SET B = GetGlobalVar (1)
B
```

The DJ EURO STOXX 50 Buy Script uses the condition from the DAX chart.

Example 2:

Chart 1: Future DAX - timeframe 5 minutes

Buy Script:
```
# Save data with ID 1
SET A = SetGlobalVar (1, (EMA(CLOSE,14) > EMA(CLOSE,21)))
```

Chart 2: Future DAX - timeframe 1 hour

Buy Script:
```
# Sets the B variable with the data previously saved with ID 1
SET B = GetGlobalVar (1)
B
```

The 1-hour DAX Buy Script uses the condition from the 5-minutes chart.

GetGlobalVar

GetGlobalVar(@id)

Overview:
The GetGlobalVar function reads a vector previously assigned to a specific identification number (ID) on another script using the SetGlobalVar function.

Interpretation:
The GetGlobalVar and SetGlobalVar functions are useful for exchanging data vectors between different scripts, possibly also applied to different charts.

Example:

```
SET B = GetGlobalVar (1)
```

Assign the data vector with ID 1 previously assigned via the SetGlobalVar function in a different script to variable B of the current script.

SetGlobalVar

SetGlobalVar(@id, @vector)

Overview:
The SetGlobalVar function associates a data vector to a specific identification number (ID). The data can then be read via the GetGlobalVar function in a different script.

Interpretation:
The GetGlobalVar and SetGlobalVar functions are useful for exchanging data vectors between different scripts, possibly also applied to different charts.

Example:

```
SET Cond = EMA(CLOSE,14) > EMA(CLOSE,21)
SET A = SetGlobalVar (1, Cond)
```

Associates the data vector Cond to ID 1 in order to be used on another script.

Important note: always assign the return value of the SetGlobalVar function to a variable, otherwise EasyScript® will mark the script as not syntactically correct.

Colors Functions

This section highlights the functions available in EasyScript® to manipulate indicators drawing colors.

COLORIZE

COLORIZE (@vector, @minValue, @max Value, @colorOrder, @centerPoint)

Overview:
The Colorize function can be used to highlight the amplitude of movement in your own indicators. This function calculates a color in the gradient between green and red based on the input vector values.

Required Parameters:
Vector: input data vector on which to calculate colors
Min. Value: minimum value of input data to use when calculating colors
Max. Value: maximum value of input data to use when calculating colors

Note: if values in Vector exceeds Min. Value or Max. Value, the function will then use Min. Value and Max. Value respectively in the colors calculation.

Optional Parameters:
Color Order: ordering, COLORIZE_ASCENDING or COLORIZE_DESCENDING.
Center Point: when specified and not equal to NAN, indicates that values in input vector have to be interpreted as absolute difference with Center Point.

Recommended Parameters:
Color Order: **COLORIZE_ASCENDING**
Center Point: **NAN**

Example:

```
SET PLOTCOLOR1 = COLORIZE(PLOT1, -3.0, 3.0)
```

Sets the color of the first indicator output line with a gradient between red and green, based on the indicator value itself.

RGB

RGB (@red, @green, @blue)

Overview:
The RGB function can be used to specify the exact color to used when drawing an indicator line. Input values must be in the range between 0 and 255.

Example:

```
SET PLOTCOLOR1 = RGB (255, 255, 153)
```

Sets the color of the first indicator output line to a specific value, in this example a light yellow.

PlayOptions.it Functions

This section contains some functions designed and conceived by PlayOptions.it to provide support to beeTrader® users.

Fiuto Canale

FiutoCanaleLowestChannel()
FiutoCanaleHighestChannel()
FiutoCanaleLowChannel()
FiutoCanaleHighChannel()

Overview:
Fiuto Canale is an indicator designed for options trading, however it guarantees excellent results also for trading on the underlying futures. It consists of two channels, the Highest-Lowest channel and the High-Low channel. The Highest-Lowest channel represents the long-term trend, while the High-Low channel represents the shortest period trend, and therefore is more susceptible to change.

Interpretation:
A signal of a possible bullish trend is given when the Low channel detaches uphill from the Lowest channel. On the other hand, a signal of a possible bearish trend is given when the High channel comes down from the Highest channel.

Example:

FiutoCanaleHighChannel() < FiutoCanaleHighestChannel()

Evaluates to true when the High channel is detached from the Highest channel.

Fiuto Facile

FiutoFacileLowestChannel()
FiutoFacileHighestChannel()
FiutoFacileLowChannel()
FiutoFacileHighChannel()

Overview:
Fiuto Facile is an indicator derived from Fiuto Canale, maintaining all it's main features.

Interpretation:
Compared to Fiuto Canale, from which it derives, there is a Linear Regression that can provide an anticipation of the trend reversal when it cuts (CROSSUNDER for the long-short inversion, CROSSOVER for the short-long inversion) the Highest-Lowest channel.

Example:

FiutoFacileHighChannel() < FiutoFacileHighestChannel()

Evaluates to true when the High channel is detached from the Highest channel.

Trend Slope

TrendSlope(@price, @periods)

Overview:
The Trend Slope indicator is an overbought / oversold indicator that is effective in both short and long term timeframes.

Interpretation:
Positive Trend Slope values indicate rising trend, on the contrary negative Trend Slope values indicate a downward trend. It's recommended to use this indicator alongside a trigger line.

Recommended Parameters:
Price: **CLOSE**
Periods: **14**

Example:

CROSSOVER (TrendSlope (CLOSE, 14)), 0)

Evaluates to true when the Trend Slope crosses above 0.

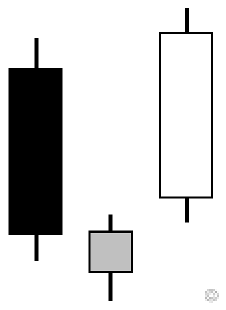
Japanese Candlestick Patterns

To draw Candlesticks, four prices are required: open, close, high and low. The figure obtained from these four data is precisely the Candlestick or Japanese Candle which is formed by a central body called a real-body and by two connected appendices called shadows (upper shadow and lower shadow). The interpretation of the candles is mainly based on patterns that are formed from period to period. The extremes of the figure are given by the low and high price, while the real-body is obtained from the difference between the close and the open price. If the close is higher than the open then we will have a white (or green) rectangle, while if the close is lower than the open the real-body will be black (or red).

Graphic representation example:

A pattern of Japanese candlesticks is a group of price bars, as shown in Figure 1. Traditionally you will see dark candles on days when the close price is lower than the open price, and clear on days when the close price is higher than that open. Sometimes you might find gray bars, which means the bar can be either white or black.

Figure 1



Identifying Candlestick Patterns

Although you could very well write your own scripts to search for candlestick patterns, EasyScript® provides a simple, built-in function that can identify up to two-dozen predefined patterns:

CandlestickPattern ()
CSP ()

Overview:
The Candlestick Pattern function identifies candlestick patterns automatically. The function takes no arguments and outputs a constant representing one of the two-dozen candlestick patterns as outlined below.

Example:

```
CSP () = MORNING_STAR
```

Evaluates to true when the candlestick pattern is a Morning Star.

Patterns:
The Candlestick Pattern function returns the following constants.

LONG_BODY	**BEARISH_ENGULFING_LINE**
DOJI	**BEARISH_DOJI_STAR**
HAMMER	**BEARISH_SHOOTING_STAR**
HARAMI	**SPINNING_TOPS**
STAR	**HARAMI_CROSS**
DOJI_STAR	**BULLISH_TRISTAR**
MORNING_STAR	**THREE_WHITE_SOLDIERS**
EVENING_STAR	**THREE_BLACK_CROWS**
BEARISH_TWO_CROWS	**ABANDONED_BABY**
PIERCING_LINE	**BULLISH_ENGULFING_LINE**
HANGING_MAN	**BULLISH_UPSIDE_GAP**
DARK_CLOUD_COVER	**BULLISH_HAMMER**
BULLISH_KICKING	**BULLISH_MATCHING_LOW**
BEARISH_KICKING	**BULLISH_BELT_HOLD**
BEARISH_BELT_HOLD	

Note: see chapter 2 for visual representations of these patterns.

Chapter 6

Build your own Scripts

6

Foreword

The EasyScript® programming language, as has been shown so far, has a wide range of Function and Indicator, which the user can use simply by interacting with the editor in beeTrader®.
However to satisfy even the most demanding users, EasyScript® allows the creation of Function that can be reused in other scripts. Taking advantage of the primitive variables and perhaps the Function already present the only limitation that EasyScript® has is the user's creativity. The scripts can be saved in encrypted form using a password, so that your work can be used by others, but not displayed and copied.

Build your own Functions

Often in your Signal or Indicator the same calculations are used several times, or sets of them. To simplify the writing of Signal, Indicator, Condition and Expert Advisor, EasyScript® allows you to group these sets of calculations in a single Function, which returns the numerical result of the calculations. In EasyScript® a Function must return only one result.

Example: suppose we want to build an Indicator and a Signal based on the difference of two Moving Averages. Since both the Indicator and the Signal use the same calculation, it is possible to realize a Function that performs this calculation. This Function is called "beeMomentum".

```
SET RETURN = EMA(@price, @shortPeriods) - EMA(@price, @longPeriods)
```

This Function has three parameters defined with the INPUTS line:
Price: price on which to calculate the Moving Averages
Short Periods: length of the short-term EMA
Long Periods: long-term EMA length

```
INPUTS: @price(CLOSE), @shortPeriods(7), @longPeriods(14)
```

The return value of the Function must be assigned to a variable whose name corresponds to that of the Function file itself or, in a much simpler way, to the predefined variable RETURN.

The Function beeMomentum we just created returns the difference between two Moving Averages. The result will be positive when the short-term EMA is greater than the long-term EMA (bullish trend); the result will be negative when the short-term EMA is lower than the long-term EMA (bearish trend).

Build your own Indicators

An Indicator is the graphic representation of the result of one or more Function, whether integrated in EasyScript® or created by the user. Each Indicator can define up to a maximum of eight output values. The colors with which it will be drawn in beeTrader® can be set directly on the text of the Indicator.

Example: suppose we want to create an Indicator that graphically represents the "beeMomentum" Function created in the previous paragraph. The Indicator will have a single output value that will be represented as a green or red histogram depending on whether the value is respectively positive or negative.

```
# Script input variables declaration
INPUTS: @price(CLOSE), @shortPeriods(7), @longPeriods(14)

# Assign the result of the Function beeMomentum to the first output value of the Indicator
SET PLOT1 = beeMomentum (@price, @shortPeriods, @longPeriods)

# Set the output color based on the sign of the value
SET PLOTCOLOR1 = IF (PLOT1 > 0, COLOR_GREEN, COLOR_RED)
```

The Indicator, exactly as the Function, has three input parameters to be declared using the INPUTS line:
Price: price on which to calculate the Moving Averages
Short Periods: length of the short-term EMA
Long Periods: long-term EMA length

The output values of the Indicators must be assigned to the predefined variables PLOT1, ..., PLOT8; the colors of the Indicator output values must be assigned to the predefined variables PLOTCOLOR1, ..., PLOTCOLOR8.
Each Indicator can therefore have up to a maximum of eight output values.

In Indicators, the way in which to assign drawing colors often follows common patterns, which can be summarized in EasyScript® in the following ways:

1. above or below a defined value (or zero)

```
SET PLOTCOLOR1 = IF(PLOT1 > @value, COLOR_GREEN, COLOR_RED)
```

2. value of the indicator increasing or decreasing

```
SET Increasing = PLOT1 > REF(PLOT1, 1)
SET PLOTCOLOR1 = IF(Increasing, COLOR_GREEN, COLOR_RED)
```

3. indicator increasing above a defined value (or zero) or decreasing below a defined value (or zero)

```
# Increase condition
SET Increasing = PLOT1 > @value AND PLOT1 > REF(PLOT1, 1)
# Decrease condition
SET Decreasing = PLOT1 < @value AND PLOT1 < REF(PLOT1, 1)
# Set color based on exceeding the defined value
```

```
SET PLOTCOLOR1 = IF(PLOT1 > @value, COLOR_GREEN, COLOR_RED)
# If Increasing condition is true, use a lighter color
SET PLOTCOLOR1 = IF(Increasing, COLOR_LIGHT_GREEN, PLOTCOLOR1)
# If Decreasing condition is true, use a lighter color
SET PLOTCOLOR1 = IF(Decreasing, COLOR_LIGHT_RED, PLOTCOLOR1)
```

Build your own Conditions

A Condition is simply an expression that represents a condition, whose value can be exclusively true or false.
The Conditions are used in beeTrader® to create Alerts and in Stock Scanner.

Example: continuing the example of the previous paragraphs, we create a Condition to verify the sign of the output value of the Function beeMomentum.

```
INPUTS: @price(CLOSE), @shortPeriods(7), @longPeriods(14)
beeMomentum (@price, @shortPeriods, @longPeriods) > 0
```

The Condition returns true when "beeMomentum" is positive, ie the instrument to which it has been applied is in a bullish trend.

Build your own Expert Advisors

An Expert Advisor allows the user to make "intelligent" comments automatically when certain conditions occur. Each Expert Advisor consists of two conditions, Buy and Sell, and a customized text message.

Example: continuing the example of the preceding paragraphs, we create an Expert Advisor that displays a complex text message based on the sign of the output value of the Function beeMomentum.

Buy Script
```
INPUTS: @price(CLOSE), @shortPeriods(7), @longPeriods(14)
beeMomentum (@price, @shortPeriods, @longPeriods) > 0
```

Sell Script
```
beeMomentum (@price, @shortPeriods, @longPeriods) < 0
```

Message
```
beeMomentum Function result: the trend is [bullish|bearish]
```

When the Buy script evaluates to true, the following message will be displayed:
beeMomentum Function result: the trend is bullish

When the Sell script evaluates to true, the following message will be displayed:
beeMomentum Function result: the trend is bearish

In the Message field of the Expert Advisor it’s possible to have beeTrader® display a different text message depending on which of the two conditions is verified. To do this the two alternative texts must be enclosed between square brackets [] and separated from each other by the pipe character |.

Build your own Signals

A Signal allows you to specify under which conditions to send orders, in paper trading or in real market. A Signal consists of at least one of the Buy Script or Sell Script condition, while the other two Exit Long and Exit Short conditions are optional. It is also possible to add Money Management features by setting the related variables.

Example: continuing the example of the previous paragraphs, we create a Signal that generates a Buy or Sell order based on the sign of the output value of the Function beeMomentum.

Buy Script

```
INPUTS: @price(CLOSE), @shortPeriods(7), @longPeriods(14)
beeMomentum (@price, @shortPeriods, @longPeriods) > 0
```

Sell Script

```
beeMomentum (@price, @shortPeriods, @longPeriods) < 0
```

This Signal generates a Buy signal when beeMomentum is positive, or when the trend is bullish; on the contrary it generates a Sell signal when beeMomentum is negative, or when the trend is bearish.

Chapter 7

Trading System Examples & Techniques

7

Trading Systems

A Trading System is nothing more than a set of rules that are applied to the performance of a particular financial instrument.
These rules include when it was decided to enter the market by buying the instrument, when to sell it, when to sell it Short (to earn a discount on the stock), when to exit the position that was previously assumed.
Leaving a position can be determined by rules based on indicators or on money management rules.
In any case, the event we have combined with the condition will take place in the future of the established condition.
The limit of the conditions for which one of the possible events must take place (Buy, Sell, ExitLong, ExitShort, TakeProfit, StopLoss, TrailingStop) is only the imagination and the experience of the trader.
To complete the project of a Trading System, in addition to writing the rules, you have to face three other steps:

Backtest

Once the system is built, it must be verified on a series of historical data to evaluate its multiple characteristics, for example it could have gained but in only one trade and this event could not be repeated, or have generated an equity line at a loss, etc.

Optimizer

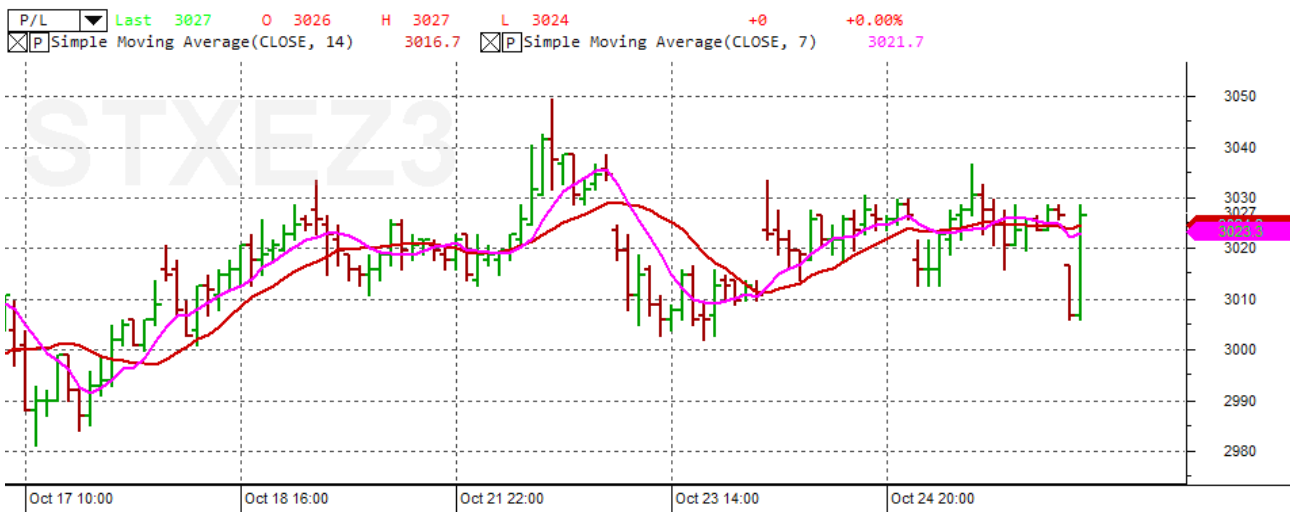
The second step concerns the optimization phase, that is to say we define ranges of values within which beeTrader® will calculate which of these would have generated the best performance.

Paper Trading

Once we have chosen the parameters we have to take one last step which is to send the system we have built ahead over time. With new data, in real time. To do this, choose the "Strategy" function and select "Paper Trading" which will execute system with real data but without using money.

bee Moving Average Crossover

The Moving Average Crossover is perhaps the simplest of all trading systems. This system uses two Moving Averages to generate signals. A Buy signal is generated when the short-term moving average crosses the long-term moving average upwards, a Sell signal is generated when the short-term moving average goes down the long-term moving average.



The number of signals generated by this trading system is proportional to the length and type of Moving Averages that are used. Short-term Moving Averages generate more signals than long-term Moving Averages.

Moving Averages calculated over a short period generate more signals than Moving Averages calculated over a long period.
Here the first real problem arises which will then be found in many systems, that is, to use short calculation periods generate many signals which, due to the frequency with which they are calculated, could also be wrong or, as they say in trading, “false ".

The most immediate solution to this sort of mediation between precision and frequency can be obtained by using different indicators in the same system.
By making some entries and exits run to those built just for this purpose.
In the example we see how to match the input signal generated by the crossing of two Exponential Moving Averages (EMAs) at 20 and 60 periods with the output signal from the trade, generated by the Parabolic SAR indicator.

Buy Script

```
# 20-period EMA crosses above 60-period EMA
CROSSOVER (EMA (CLOSE, 20), EMA (CLOSE, 60))
```

Sell Script

```
# 20-period EMA crosses below 60-period EMA
CROSSOVER (EMA (CLOSE, 60), EMA (CLOSE, 20))
```

Exit Long Script

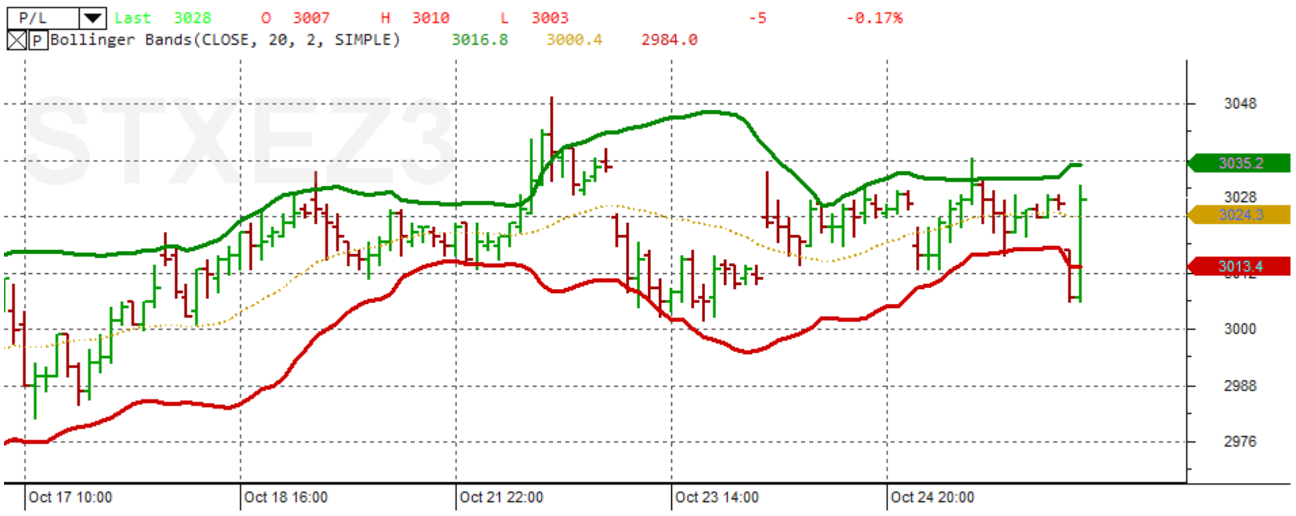
```
# Close crosses below Parabolic SAR
CROSSOVER (PSAR (0.02, 0.2), CLOSE)
```

Exit Short Script

```
# Close crosses above Parabolic SAR
CROSSOVER (CLOSE, PSAR (0.02, 0.2))
```

bee Bollinger Bands

The Bollinger Bands are similar to a Moving Average except that they are shifted above and below the price of a certain number of standard deviations to form a band around the price. And unlike a Simple Moving Average or a Moving Average Envelope, the Bollinger Bands are calculated in such a way as to allow expansions and contractions based on market volatility.



Prices generally remain contained within the bands. A strategy is to buy or sell when prices touch the bands and then retrace because a movement that originates in a band usually tends to move within the same band.
Another strategy is to buy or sell if the price goes outside the bands. In this case, the market is destined to continue in this direction for a certain period of time.
The Bollinger Bands trading system described in this example uses a combination of both strategies. The system generates a Buy if a recent bar has touched the lower band and the current bar is within bands, and generates a Buy even if the current high price has exceeded the upper band by a certain percentage. The system generates a Sell under opposite conditions.

Buy Script

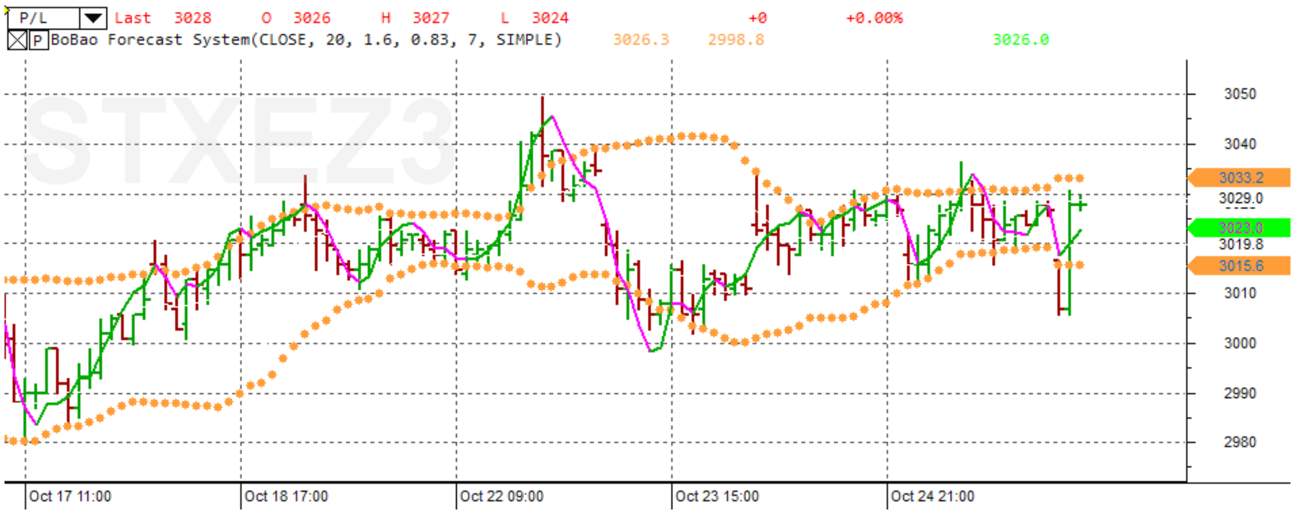
```
# Close was below bottom line, now it’s above
SET Bottom = BBB (CLOSE, 20, 2, EXPONENTIAL)
SET Top = BBT (CLOSE, 20, 2, EXPONENTIAL)
((REF (CLOSE, 1) < REF (Bottom, 1)) AND
CLOSE > Bottom) OR
# Close above top line by at least 2%
CLOSE > Top * 1.02
```

Sell Script

```
# Close was above top line, now it’s below
SET Bottom = BBB (CLOSE, 20, 2, EXPONENTIAL)
SET Top = BBT (CLOSE, 20, 2, EXPONENTIAL)
((REF (CLOSE, 1) > REF (Top, 1)) AND
CLOSE < Top) OR
# Close is below bottom line by at least 2%
CLOSE < Bottom * 0.98
```

Bobao Forecast System

This system, designed by Cagalli Tiziano, is based on the use of two pairs of Bollinger Bands with different settings and the use of a Linear Regression.
Linear Regression will generate a signal when crossing one of the two bands of the pair with the highest standard deviation.
The passage of the regression line on the second pair of bands with a lower standard deviation will serve as a possible exit point if money management systems are not used.



In the above figure the entry bands are drawn in points and when the Linear Regression crosses the upper band from top to bottom a Sell signal will be generated. Crossing from top to bottom will generate an ExitShort signal.
When the Linear Regression crosses the lower band from the bottom upwards, a Buy signal will be generated which will be stopped by an ExitLong signal in the case of opposite crossing or when the regression line crosses the lower band from top to bottom.

Buy Script

```
# Input variables
INPUTS: @price(CLOSE), @BandPeriods(20, 10, 30, 1), @BigDev(1.6, 1, 3, 0.1),
@SmalDev(0.83, 0.3, 0.9, 0.03), @matype(SIMPLE), @SLperiods(7)

SET BigTop = BollingerBandsTop(@price, @BandPeriods, @BigDev, @matype)
SET BigBottom = BollingerBandsBottom(@price, @BandPeriods, @BigDev, @matype)
SET SmallTop = BollingerBandsTop(@price, @BandPeriods, @SmalDev, @matype)
SET SmallBottom = BollingerBandsBottom(@price, @BandPeriods, @SmalDev,
@matype)
SET SignalLine = LR(@price, @SLperiods)

# Buy
CROSSOVER(SignalLine, BigBottom)
```

Sell Script

```
SET BigTop = BollingerBandsTop(@price, @BandPeriods, @BigDev, @matype)
SET BigBottom = BollingerBandsBottom(@price, @BandPeriods, @BigDev, @matype)
SET SmallTop = BollingerBandsTop(@price, @BandPeriods, @SmalDev, @matype)
SET SmallBottom = BollingerBandsBottom(@price, @BandPeriods, @SmalDev,
@matype)
SET SignalLine = LR(@price, @SLperiods)

# Sell
CROSSOVER(BigTop,SignalLine)
```

ExitLong Script

```
SET BigTop = BollingerBandsTop(@price, @BandPeriods, @BigDev, @matype)
SET BigBottom = BollingerBandsBottom(@price, @BandPeriods, @BigDev, @matype)
SET SmallTop = BollingerBandsTop(@price, @BandPeriods, @SmalDev, @matype)
SET SmallBottom = BollingerBandsBottom(@price, @BandPeriods, @SmalDev,
@matype)
SET SignalLine = LR(@price, @SLperiods)

# ExitLong
CROSSOVER(SmallBottom,SignalLine)
OR CROSSOVER(SmallTop,SignalLine)
OR CROSSOVER(BigBottom,SignalLine)
```

ExitShort Script

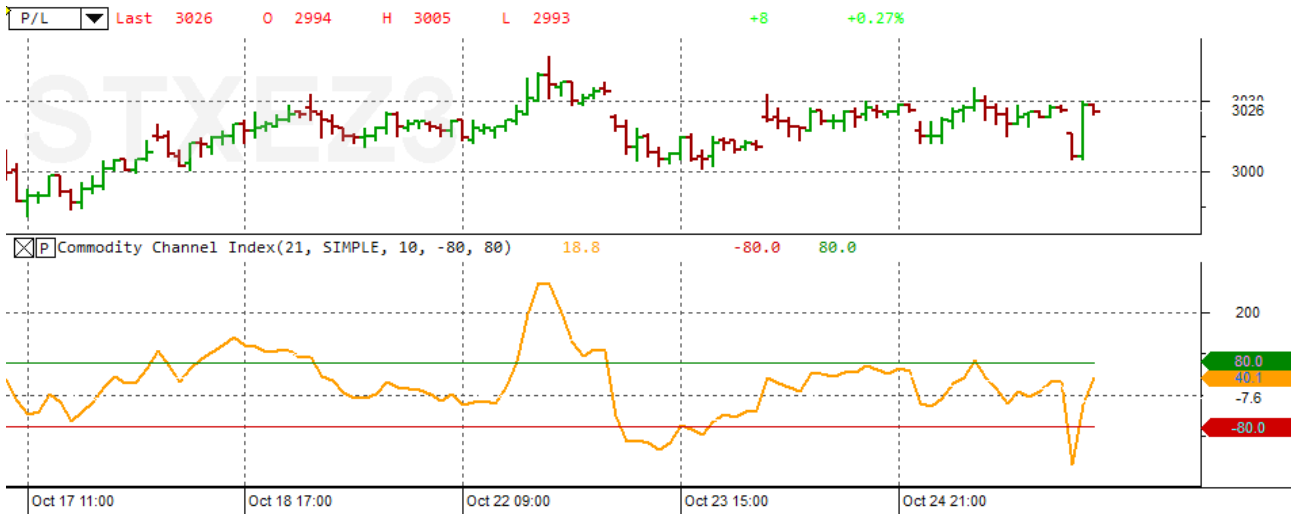
```
SET BigTop = BollingerBandsTop(@price, @BandPeriods, @BigDev, @matype)
SET BigBottom = BollingerBandsBottom(@price, @BandPeriods, @BigDev, @matype)
SET SmallTop = BollingerBandsTop(@price, @BandPeriods, @SmalDev, @matype)
SET SmallBottom = BollingerBandsBottom(@price, @BandPeriods, @SmalDev,
@matype)
SET SignalLine = LR(@price, @SLperiods)

# ExitShort
CROSSOVER(SignalLine, SmallTop)
OR CROSSOVER(SignalLine, SmallBottom)
OR CROSSOVER(SignalLine, BigTop)
```

You can see that in this way there are two conditions for the entry and four for the exit: the entrances only from crossing the upper bands, the exits from crossing the same bands but in the opposite direction or the different bands.

bee Commodity Channel Index

Donald Lambert developed the CCI indicator. Although the purpose of this indicator is to identify cyclic shifts of Commodities, it is often used for securities. It is an oscillator that develops between the values around 100 and -100 in 70/80% of cases. The shorter the period, the more the values greater than 100 will increase.



We build a simple but effective system based on crossing the marking lines of the overbought and oversold zones.
If the CCI crosses the level of oversold from the bottom up the system will generate a BUY. The SELL is generated by the opposite situation or from the CCI crossing from top to bottom of the overbought marking line.

Buy Script

```
# Input variables
INPUTS: @periods(21), @matype(SIMPLE), @lowMark(-80), @highMark(80)
# Calculations
SET C = CCI(@periods, @matype)
# Buy condition
SET cond = CROSSOVER(C, @lowMark)
cond
```

Sell Script

```
# Input variables already declared in the Buy Script, proceed directly to calculations
SET C = CCI(@periods, @matype)
# Sell condition
CROSSOVER(@highMark, C)
```

bee Swing Trailing

This system is based on swings, that is on the changes of trends to generate the input and output signals.

However, the ingredients of money management are added so that the user can appreciate the validity of managing the money during the duration of a trade.
We use the Trailing Stop which defines the objective of first gain, a point from which a part of the same defined by the Trailing percent will be put at risk.
In this way, if you enter a lasting trend, you will be able to stay in position longer, having secured a gain (Trailing Stop) and putting at risk a percentage part (Trailing Percent) that will allow you to stay in position in case of slight price fluctuations.

In case of wrong entries that do not immediately lead to the Trailing Stop but instead begin to generate losses here is that we use the Stop Loss that will make us exit the position with a loss equal to what we have set as a rule.

Important Notes on Money Management

MAX_POSITION_OPEN

Its use becomes efficient if combined with a price condition, that is: we leave the position if after X periods or the entry price corresponds approximately to the LAST price.

```
SET MAX_POSITION_OPEN = 20
(LASTENTRYPRICE < LAST * 1,05) OR (LASTENTRYPRICE > LAST * 0,95)
```

STOP_LOSS

It is the closing of the position but also the materialization of the loss, it must be used after a careful analysis of the MAX INTRADAY DRAWDOWN (see beeAnalyzer) and positioned at a value that cancels the fewest possible trades. The more the STOP_LOSS is tight, the more the system will lose. In a STOP & REVERSE Trading System, it will clearly have to be positioned far below the normal STOP & REVERSE entry because it must serve exclusively as an emergency if other things have not worked.

TAKE_PROFIT

It is a function to be used in multi trades or when the Trading System is applied to several securities. The sense is to enter an underlying for a short time, reach the goal and then move to another underlying.

TRAILING_STOP & TRAILING_PERCENT

This function is mainly used on Trading Systems that have been designed to generate signals not on signal crossings, but on conditions that are prolonged over time. If we write about buying and selling at the intersection of two Moving Averages it will mean that the Trading System will send a few orders, or at every intersection and therefore the TRAILING_STOP would make us go out and not return. Conversely, if we write about buying and selling if the Moving Average is above or below another Moving Average, the Trading System will generate continuous entry signals after each exit with TRAILING_STOP.
It is necessary to pay particular attention to the setting of the TRAILING_STOP and TRAILING_PERCENT values. The result of TRAILING_STOP * TRAILING_PERCENT / 100 must be greater than the minimum movement tick value.

Example of TRAILING_PERCENT calculation on DAX Futures: Big Point Value = 25 €, minimum movement tick value 12,5€.

Correct setting:
TRAILING_STOP = 250
TRAILING_PERCENT = 10
10% of 250 = € 25
€ 25 > € 12.5

Incorrect setting:
TRAILING_STOP = 100
TRAILING_PERCENT = 10
10% of 100 = 10 €
€ 10 < € 12.5