

TradeScript™  

# Contents

## Chapter 1

### Introducing TradeScript™

Prerequisites .....	4
How This Guide Is Organized .....	4
The TradeScript™ Programming Language .....	5
Introduction: Important Concepts .....	5
Boolean Logic .....	6
Program Structure .....	6
Functions .....	7
Vector Programming .....	8
The REF Function .....	9
The TREND Function .....	10
Price Gaps and Volatility .....	11
Fundamental Analysis .....	12
Technical Analysis .....	12
Crossovers .....	13
Sectors and Industries .....	14
Key Reversal Script .....	14
Primitives .....	16
Conditional “IF” Function .....	17
LOOP Function .....	17
COUNTIF .....	19
LASTIF .....	19
SUMIF .....	19
SUM .....	20
AVG .....	20
MAX .....	20
MIN .....	21
MAXOF .....	21
MINOF .....	21
REF .....	22
TREND .....	22
CROSSOVER .....	22
Math Functions .....	23
ABS .....	23
SIN .....	23
COS .....	24
TAN .....	24
ATN .....	24
EXP .....	24
LOG .....	25
LOG10 .....	25

RND .....	25
Operators .....	26
Equal (=) .....	26
Greater Than (>) .....	26
Less Than (<) .....	26
Greater Than Or Equal To (>=) .....	27
Less Than Or Equal To (<=) .....	27
Not Equal (<> or !=) .....	27
AND .....	28
OR .....	28
XOR .....	29
NOT .....	29
EQV .....	29
MOD .....	30
Moving Averages .....	32
Simple Moving Average .....	32
Exponential Moving Average .....	33
Time Series Moving Average .....	33
Variable Moving Average .....	34
Triangular Moving Average .....	34
Weighted Moving Average .....	35
Welles Wilder Smoothing (Moving Average) .....	35
Volatility Index Dynamic Average - VIDYA (Moving Average) .....	36
Linear Regression Functions .....	37
R <sup>2</sup> (R-Squared) .....	37
Slope .....	37
Forecast .....	38
Intercept .....	38
Band Functions .....	39
Bollinger Bands .....	39
Keltner Channels .....	40
Moving Average Envelope .....	41
Prime Number Bands .....	41
Oscillator Functions .....	42
Momentum Oscillator .....	42
Chande Momentum Oscillator .....	43
Volume Oscillator .....	43
Price Oscillator .....	44
Detrended Price Oscillator .....	44
Prime Number Oscillator .....	45
Fractal Chaos Oscillator .....	45
Rainbow Oscillator .....	46
TRIX .....	46
Vertical Horizontal Filter .....	47
Ease Of Movement .....	47
Wilder's Directional Movement System .....	48

True Range .....	48
Williams %R.....	49
Williams' Accumulation / Distribution.....	49
Chaikin Volatility .....	50
Aroon.....	50
Moving Average Convergence / Divergence (MACD).....	51
High Minus Low .....	51
Stochastic Oscillator .....	52
Index Functions .....	53
Relative Strength Index.....	53
Mass Index.....	53
Historical Volatility Index.....	54
Money Flow Index.....	54
Chaikin Money Flow Index .....	55
Comparative Relative Strength Index .....	55
Price Volume Trend.....	56
Positive Volume Index.....	56
Negative Volume Index.....	57
On Balance Volume.....	57
Performance Index.....	58
Trade Volume Index .....	58
Swing Index .....	59
Accumulative Swing Index.....	59
Commodity Channel Index (CCI) .....	60
Parabolic Stop and Reversal (Parabolic SAR) .....	60
Stochastic Momentum Index.....	61
General Indicator Functions.....	62
Median Price .....	62
Typical Price.....	62
Weighted Close.....	63
Price Rate of Change.....	63
Volume Rate of Change .....	64
Highest High Value.....	64
Lowest Low Value.....	65
Standard Deviations .....	65
Correlation Analysis.....	66
Japanese Candlestick Patterns .....	67
Trading Systems .....	69
Moving Average Crossover System .....	70
Moving Average Crossover System Script.....	71
Price Gap System.....	72
Price Gap Script.....	73
Bollinger Bands System.....	74
Bollinger Bands Script .....	75
Historical Volatility and Trend .....	76
Historical Volatility and Trend Script.....	77

Parabolic SAR / MA System.....	78
Parabolic SAR / MA Script.....	79
MACD Momentum System.....	80
MACD Momentum Script.....	81
Narrow Trading Range Breakout .....	82
Narrow Trading Range Script.....	82
Fundamental Trading System.....	83
Fundamental Trading System Script .....	84
Outside Day System.....	85
Outside Day Script.....	86
Japanese Candlestick Engulfing Line System.....	87
Primitive Types .....	88
Price Vectors.....	88
Fundamental Variables .....	88
Basic Constants.....	89
Back Testing Flags.....	89
Moving Average Constants .....	90
Trend Constants (used by TREND function) .....	90
Points or Percent Constants (used by indicators) .....	90
Candlestick Pattern Constants .....	90
Sector and Industry Constants.....	94
Sector Constants.....	94
Industry Constants.....	96
Troubleshooting.....	99
<b>Index.....</b>	<b>101</b>

## ***Prerequisites***

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

Please note that TradeScript™ may be bundled or implemented into 3<sup>rd</sup> party software. Please refer to your trading software vendor for technical support.

## ***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 TradeScript™ programming area in your software.

The last section of this guide contains a reference of functions, properties, and constants supported by the TradeScript™ 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 TradeScript™ Programming Language***

TradeScript™ is the engine that drives the scripting language in your trading software. It is a non-procedural scientific vector programming language that was designed specifically for developing trading systems. A *script* is simply a set of instructions that tell the TradeScript™ 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. There are many uses.



### ***Introduction: Important Concepts***

TradeScript™ 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 TradeScript™ 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.

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

### ***Boolean Logic***

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

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

**LAST > OPEN**

*Script 2* evaluates to true when volume is two times the previous day's volume:

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

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

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

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

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

Once again, the instructions are nearly as plain as the English language. The use of Boolean logic with the **AND** and **OR** keywords is a very important concept that is used extensively by the TradeScript™ 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 X's the previous 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 TradeScript™ 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 TradeScript™ to identify trades where the closing price is in a 30-day uptrend.

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

Only one of two things will occur if you use a function incorrectly... TradeScript™ will automatically fix the problem and the script will still run, or TradeScript™ will report an error, tell you what’s wrong with the script, and then allow you to fix the problem and try again.

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

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

**TREND(30) = UP**

The following error occurs:

Error: argument of 'TREND' function not optional. [Click here for help.](#)

We are given the option to fix the script and try again. The TREND hyperlink provides help for the TREND function by listing the required arguments.

### **Vector Programming**

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

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

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

```

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

```

Nine to ten lines of code would be required to create the “MedianAverages” vector. But with TradeScript™, you can effectively accomplish the same thing using only one line:

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

And now MedianAverage is actually a new vector that contains the 30-period simple moving average of the median price of the stock at each point.

It is not uncommon to find array programming language “one-liners” that require more than a couple of pages of BASIC, Java or C++ code.

## **The REF Function**

At this point you may be wondering what “REF” and “TREND” are. These are two of the very useful primitives that are built into the TradeScript™ 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 TradeScript™ 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. TradeScript™ on the other hand references values backwards, from the end. This is most convenient since the purpose of TradeScript™ 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)
```

## The TREND Function

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.

TradeScript™ provides a primitive function aptly named TREND especially for detecting trends in stock price, volume, or indicators:

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

This tells TradeScript™ 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) = DOWN**

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

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

## Price Gaps and Volatility

Although the TREND function can be used for identifying trends and the REF function can be used for determining the degree 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)
```

Or:

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

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

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



And with a slight variation we can also 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.

## ***Fundamental Analysis***

Many investors and traders rely upon fundamental information such as the price-to-earnings (PE) ratio, dividend, and yield. This information can be used as part of your trading system simply by including the option in your script:

```
# the PE ratio is between 10 and 15  
PE_RATIO >= 10 AND PE_RATIO <= 15
```

Valid options are:

```
PE_RATIO  
DIVIDEND  
YIELD  
52_WEEK_HIGH  
52_WEEK_LOW  
ALL_TIME_LOW  
ALL_TIME_HIGH
```

These and other primitive variables are covered in chapter 5.

## ***Technical Analysis***

TradeScript™ 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 3.

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.

### **Crossovers**

You may be familiar with the term “crossover”, which is what happens when one series crosses over the top of another series as depicted in the image 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 one series has crossed over another.

For example, we can find the exact point in time when one moving average crossed over another by using the CROSSOVER function:

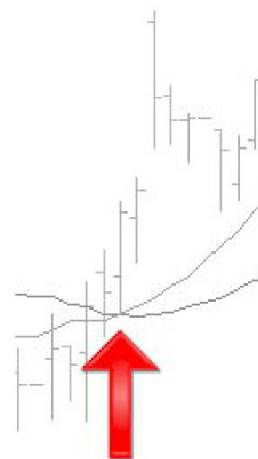
```
SET MA1 = SimpleMovingAverage(CLOSE, 28)
```

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

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

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

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



## ***Sectors and Industries***

Systems can be narrowed down to a specific sector or industry by setting the global SECTOR and/or INDUSTRY flags. To filter for securities in a specific industry and sector, you can write:

```
SET SECTOR = TECHNOLOGY  
SET INDUSTRY = SEMICONDUCTORS  
LAST > 0
```

A complete list of all available sectors & industries can be found in the Sectors & Industries section of chapter 5.

## ***Key Reversal Script***

Finally, before we move into the technical reference section of this guide let's create a script that finds Key Reversals, so that you can see firsthand how TradeScript™ can be used to create trading systems based upon complex rules.

The definition of a Key Reversal is that after an uptrend, the open must be above the previous close, the most current bar must make a new high, and the last price must be below the previous low. Let's translate that into script form:

```
# First make sure that the stock is in an uptrend  
TREND(CLOSE, 30) = UP  
  
# The open must be above yesterday's close  
AND OPEN > REF(CLOSE, 1)  
  
# Today must be making a new high  
AND HIGH >= ALL_TIME_HIGH  
  
# And the last price must be below yesterday's low  
AND LAST < REF(LOW, 1)
```

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

```
# First make sure that the stock is in a downtrend  
TREND(CLOSE, 30) = DOWN  
  
# The open must be below yesterday's close  
AND OPEN < REF(CLOSE, 1)  
  
# Today' must be making a new low  
AND LOW <= ALL_TIME_LOW
```



# 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.

## ***Primitives***

This chapter covers the built-in functions of TradeScript™, also known as *primitives*. These important functions define the TradeScript™ 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 TradeScript™ 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 TradeScript™ 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 TradeScript™ can be used to perform these same calculations in a much simpler way by means of vector operations and simulated control structure.

## **Conditional “IF” Function**

### **IF(Condition, True part, False part)**

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.

## **LOOP Function**

### **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 adds 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 stepwise 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:

<b>X</b>	<b>LOOP</b>
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:

In the case of **SET X = REF(X, 1)**, **Vector1** is **X** and **Vector2** is 1. Since we're adding the value of 1 (not a vector) to **X** in the following example, **Offset2** is set to zero:

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

And now **X** contains the series 1,2,3,4,5,6,...n

Example 3:

**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**

## **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.

## **LASTIF**

### **LASTIF(Condition)**

Similar to COUNTIF, except LASTIF returns a vector containing the number of days 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 < REF(CLOSE, 1))**

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

## **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:

### **SUMIF(CLOSE > REF(CLOSE,1) \* 1.05, 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%.

## **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.

## **AVG**

### **AVERAGE(Vector, Periods)**

### **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:

### **AVERAGE(CLOSE, 10)**

### **AVG(CLOSE, 10)**

Both scripts return a vector of averages based on a 10- period window.

## **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.

**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.

**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.

**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.

## **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. The valid range for the Periods argument is 1 - 250 unless otherwise noted.

## **TREND**

### **TREND(Vector)**

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

```
TREND(CLOSE) = UP AND TREND(VOLUME) = DOWN
```

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

## **CROSSOVER**

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

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

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



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

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

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

## ***Math 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***

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

Example:

**ABS(CLOSE - OPEN)**

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

### ***SIN***

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

Example:

**SIN(45)**

The script outputs 0.851

## ***COS***

COS returns the cosine for a number (angle).

Example:

***COS(45)***

The script outputs 0.525

## ***TAN***

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

Example:

***TAN(45)***

The script outputs 1.619

## ***ATN***

Returns the arctangent for a number.

Example:

***ATN(45)***

The script outputs 1.548

## ***EXP***

EXP raises  $e$  to the power of a number. The LOG function is the reverse of this function.

Example:

***EXP(3.26)***

The script outputs 26.28

## **LOG**

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

Example:

**LOG(26.28)**

The script outputs 3.26

## **LOG10**

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

Example:

**LOG10(26.28)**

The script outputs 1.42

## **RND**

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

Example:

**RND(100)**

Outputs a random number from 0 to 100.

## ***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. Also, the **SET** keyword must precede the variable name when the = operator is used for an assignment:

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

### ***Greater Than (>)***

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

Example:

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

### ***Less Than (<)***

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

Example:

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

### ***Greater Than Or Equal To (>=)***

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**

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 a "bitwise operator" to make a bit-by-bit comparison of two integers. If both bits in the comparison are 1, then a 1 is returned. Otherwise, a 0 is returned.

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

Example:

**(TRUE = TRUE AND FALSE = FALSE) = TRUE**

And:

**(TRUE = TRUE AND FALSE = TRUE) = FALSE**

**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**

### ***MOD***

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

Example:

**21 MOD 5 = 1**

And:

**22 MOD 5 = 2**



Stocks 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 TradeScript™ programming language as primitive functions, as outlined in this chapter.

TradeScript™ also allows you to program additional technical indicators by using a combination of primitive functions listed in Chapter 2 and in this chapter. Chapter 4 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 TradeScript™ programming language.

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

Long name:

**SimpleMovingAverage(CLOSE, 30)**

Abbreviated name:

**SMA(CLOSE, 30)**

SMA is the same function as SimpleMovingAverage and both methods work the same way. Each function provides an abbreviated name if available.

## ***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.

For example, the Moving Average Convergence / Divergence (MACD) indicator in TradeScript™ allows you to specify the moving average type used within the indicator's "Signal Line" 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.

### ***Simple Moving Average***

**SimpleMovingAverage(Vector, Periods)**

**SMA(Vector, Periods)**

MA Type Argument ID: **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-day SMA.

## ***Exponential Moving Average***

**ExponentialMovingAverage(Vector, Periods)**

**EMA(Vector, Periods)**

MA Type Argument ID: **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**

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

### **Example**

**CLOSE > EMA(CLOSE, 30)**

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

## ***Time Series Moving Average***

**TimeSeriesMovingAverage(Vector, Periods)**

**TSMA(Vector, Periods)**

MA Type Argument ID: **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 Moving Average is most often used to average values for a smoother representation of the underlying price or indicator.

### **Example**

**CLOSE > TSMA(CLOSE, 30)**

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

## ***Variable Moving Average***

**VariableMovingAverage(Vector, Periods)**

**VMA(Vector, Periods)**

MA Type Argument ID: **VARIABLE**

### **Overview**

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

### **Interpretation**

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

### **Example**

**CLOSE > VMA(CLOSE, 30)**

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

## ***Triangular Moving Average***

**TriangularMovingAverage(Vector, Periods)**

**TMA(Vector, Periods)**

MA Type Argument ID: **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 Moving Average is most often used to average values for a smoother representation of the underlying price or indicator.

### **Example**

**CLOSE > TMA(CLOSE, 30)**

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

## ***Weighted Moving Average***

**WeightedMovingAverage(Vector, Periods)**

**WMA(Vector, Periods)**

MA Type Argument ID: **WEIGHTED**

### **Overview**

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

### **Interpretation**

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

### **Example**

**CLOSE > WMA(CLOSE, 30)**

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

## ***Welles Wilder Smoothing (Moving Average)***

**WellesWilderSmoothing(Vector, Periods)**

**WWS(Vector, Periods)**

MA Type Argument ID: **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**

This indicator is used in the manner that any other moving average would be used.

### **Example**

**CLOSE > WWS(CLOSE, 30)**

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

## ***Volatility Index Dynamic Average - VIDYA (Moving Average)***

**VIDYA(Vector, Periods, R2Scale)**

MA Type Argument ID: **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.

### **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-day VIDYA with an  $R^2$  of 0.65.

## ***Linear Regression Functions***

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 by the condition that the sum of the square residuals is as small as possible. The indicators in this section are based upon this model.

### ***R<sup>2</sup> (R-Squared)***

**RSquared(Vector, Periods)**

**R2(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)**

#### **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.

## ***Forecast***

**Forecast(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)**

### **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, 30) > REF(CLOSE,1)**

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



## ***Band 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. The most popular band indicator is the Bollinger Bands, developed by stock trader John Bollinger in the early 1980's.

### ***Bollinger Bands***

**BollingerBandsTop(Vector, Periods, Standard Deviations, MA Type)**

**BBT(Vector, Periods, Standard Deviations, MA Type)**

**BollingerBandsMiddle(Vector, Periods, Standard Deviations, MA Type) **BBM(Vector, Periods, Standard Deviations, MA Type)****

**BollingerBandsBottom(Vector, Periods, Standard Deviations, MA Type) **BBB(Vector, Periods, Standard Deviations, MA Type)****

### **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 as an 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-day Bollinger Band Top calculated by 2 standard deviations, using an exponential moving average.

## ***Keltner Channels***

**KeltnerChannelTop(Periods, MA Type, Multiplier)**

**KCT(Periods, MA Type, Multiplier)**

**KeltnerChannelMedian(Periods, MA Type, Multiplier)**

**KCM(Periods, MA Type, Multiplier)**

**KeltnerChannelBottom(Periods, MA Type, Multiplier)**

**KCB(Periods, MA Type, 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**

Shift: **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, MA Type, Shift)**

**MAET(Periods, MA Type, Shift)**

**MovingAverageEnvelopeBottom(Periods, MA Type, Shift)**

**MAEB(Periods, MA Type, Shift)**

### **Overview**

Moving Average Envelopes consist of moving averages calculated from the underlying 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: **5**

### **Example**

**CLOSE > MAET(20, SIMPLE, 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**

**CLOSE > PNBT()**

Evaluates to true when the close is greater than the Prime Number Bands Top.

## ***Oscillator Functions***

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.

### ***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

## ***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.

## ***Volume Oscillator***

**VolumeOscillator(Short Term Periods, Long Term Periods, MA Type, Points or Percent) VO(Short Term Periods, Long Term Periods, MA Type, Points or Percent)**

### **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**

### ***Price Oscillator***

**PriceOscillator(Vector, Short Term Periods, Long Term Periods, MA Type)**  
**PO(Vector, Short Term Periods, Long Term Periods, MA Type)**

#### **Overview**

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

#### **Interpretation**

The Price Oscillator is basically 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.

### ***Detrended Price Oscillator***

**DetrendedPriceOscillator(Vector, Periods, MA Type)**  
**DPO(Vector, Periods, MA Type)**

#### **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: **20**

MA Type: **SIMPLE**

#### **Example**

**DPO(CLOSE, 20, SIMPLE) > 0**

Evaluates to true when the Detrended 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)**

### ***Fractal Chaos Oscillator***

**FractalChaosOscillator(Periods)**  
**FCO(Periods)**

#### **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.

#### **Recommended Parameters**

Periods: **21**

#### **Example**

**FCO(21) > REF(FCO(21),1)**

## ***Rainbow Oscillator***

**RainbowOscillator(Vector, Levels, MA Type)**

**RBO(Vector, Levels, MA Type)**

### **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: **3**

MA Type: **SIMPLE**

### **Example**

**SET R = RBO(CLOSE, 3, SIMPLE)**

**R > 0.8 AND REF(R, 1) > 0.8**

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

## ***TRIX***

**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.



### ***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**

Vector: **CLOSE**

Periods: **21**

#### **Example**

**VHF(CLOSE, 21) < 0.2**

### ***Ease Of Movement***

**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.

## ***Wilder's Directional Movement System***

**ADX(Periods), ADXR(Periods), DIP(Periods), DIN(Periods), TRSUM(Periods), DX(Periods)**

### **Overview**

The Welles Wilder's Directional Movement System contains five indicators; ADX, DI+, DI-, DX, and ADXR.

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.

Detailed information about this indicator and formulas can be found in Welles Wilder's book, "New Concepts in Technical Trading Systems". The standard Directional Movement System draws a 14 period DI+ and a 14 period DI- in the same chart panel. ADX is also sometimes shown in the same chart panel.

### **Interpretation**

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

### **Recommended Parameters**

Periods: 21

### **Example**

**DIP(14) > 60**

## ***True Range***

**TrueRange()**

**TR()**

### **Overview**

The True Range is a component of Wilder's Directional Movement System.

### **Example**

**TR() > 1.95**

## **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**

Periods: **14**

### **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 be distributing. Prices may reverse when the indicator converges with price.

### **Example**

**WAD() < 1**

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

## ***Chaikin Volatility***

**ChaikinVolatility(Periods, Rate of Change, MA Type)**

**CV(Periods, Rate of Change, MA Type)**

### **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) < -25**

## ***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: **25**

### **Example**

**AroonUp(25) > 80 AND AroonDown(25) > 80**

## ***Moving Average Convergence / Divergence (MACD)***

**MACD(Short Cycle, Long Cycle, Signal Periods, MA Type)**

**MACDSignal(Short Cycle, Long Cycle, Signal Periods, MA Type)**

### **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.

## ***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.

## **Stochastic Oscillator**

**SOPK(%K Periods, %K Slowing Periods, %D Periods, MA Type)**

**SOPD(%K Periods, %K Slowing Periods, %D Periods, MA Type)**

### **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**

% K Periods: **9**

% K Slowing Periods: **3**

% D Periods: **9**

MA Type: **SIMPLE**

### **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.

## ***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.

### ***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 price / RSI divergence, support / resistance levels and RSI chart formations.

#### **Recommended Parameters**

Vector: **CLOSE**

Periods: **14**

#### **Example**

**RSI(CLOSE, 14) > 55**

### ***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 27 or falls below 26.5.

#### **Recommended Parameters**

Periods: **25**

#### **Example**

**MI(25) > 27**

## ***Historical Volatility Index***

**HistoricalVolatilityIndex(Vector, Periods, Bar History, Standard Deviations) HVI(Vector, Periods, Bar History, Standard Deviations)**

### **Overview**

Historical volatility is the log-normal standard deviation. The Historical Volatility Index is based on the book by Don Fishback, "Odds: The Key to 90% Winners".

The formula for a 30-day historical volatility index between 1 and 0 is:  
 $\text{Stdev}(\text{Log}(\text{Close} / \text{Close Yesterday}), 30) * \text{Sqrt}(365)$

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: **15**

Bar History: **30**

Standard Deviations: **2**

### **Example**

**HVI(CLOSE, 15, 30, 2) < 0.01**

## ***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**

**MFI(15) < 20**



## ***Chaikin Money Flow Index***

**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,  $(\text{high} + \text{low}) / 2$ , 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: 15

### **Example**

**CMF(15) > 20 AND REF(CMF(15), 1) > 20**

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

## ***Comparative Relative Strength Index***

**ComparativeRelativeStrength(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.

## ***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)) = UP**

Evaluates to true when PVT is trending upwards.

## ***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)) = UP**

Evaluates to true when PVI is trending upwards.

## ***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)) = UP**

Evaluates to true when NVI is trending upwards.

## ***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)) = UP**

Evaluates to true when OBV is trending upwards.

## ***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 indicator 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%

## ***Trade Volume Index***

**TradeVolumeIndex(Vector, Minimum Tick Value)**

**TVI(Vector, Minimum Tick Value)**

### **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 be 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.

## ***Swing Index***

**SwingIndex(Limit Move Value)**

**SI(Limit Move Value)**

### **Overview**

The Swing Index (Wilder) 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.

## ***Accumulative Swing Index***

**AccumulativeSwingIndex(Limit Move Value)**

**ASI(Limit Move Value)**

### **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)) > UP**

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

## ***Commodity Channel Index (CCI)***

**CommodityChannelIndex(Periods, MA Type)**

**CCI(Periods, MA Type)**

### **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**

Periods: **21**

MA Type: **SIMPLE**

### **Example**

**CCI(12, SIMPLE) > 0 AND REF(CCI(12, SIMPLE), 1) < 0**

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

## ***Parabolic Stop and Reversal (Parabolic SAR)***

**ParabolicSAR(Min AF, Max AF)**

**PSAR(Min AF, Max AF)**

### **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.

## ***Stochastic Momentum Index***

**SMIK(%K Periods, %K Smooth, %K Double Periods, %D Periods, MA Type, %D MA Type)**

**SMID(%K Periods, %K Smooth, %K Double Periods, %D Periods, MA Type, %D MA Type)**

### **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**

%K Periods: **14**

%K Smoothing: **2**

%K Double Periods: **3**

%D Periods: **9**

MA Type: **SIMPLE**

%D MA Type: **SIMPLE**

### **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.

## **General Indicator Functions**

### **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.

### **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.



## ***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.

## ***Price Rate of Change***

**PriceRateOfChange(Vector, Periods)**  
**PROC(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**

**PROC(CLOSE, 12) > 0 AND REF(PROC(CLOSE, 12),1) < 0**

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

## ***Volume Rate of Change***

**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.

## ***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.

## ***Standard Deviations***

**StandardDeviations(Vector, Periods, Standard Deviations, MA Type)**  
**SDV(Vector, Periods, Standard Deviations, MA Type)**

### **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.

## **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.

## Japanese Candlestick Patterns

Just about every trader is familiar with Japanese Candlestick charting, which was popularized by Steve Nison, author of the book "Japanese Candlestick Charting Techniques". Many traders have been using a form of Japanese candlestick charting for decades, even before it was named "candlestick charting".

### What are Candlesticks?

The main feature of a candlestick is that the area between the open and close price is filled in, with emphasis on the direction. Typically you will see bars represented as dark candles on days where the price closed lower than the open, or white candles on days where the price closed higher than the open. The actual high and low prices are called "wicks". Candlesticks don't involve calculations, rather they simply offer a different perspective for viewing price action. The interpretation of candlesticks is based primarily on patterns that are formed from period to period. For example, you may have heard of terms like "Three Black Crows", "Morning Star", or "Dark Cloud Cover". These are all candlestick patterns, which are formed by two or more candlesticks.

### A Graphical Representation

A Japanese Candlestick pattern is a group of price bars as shown in figure 1. Traditionally you will see dark candles on days where the price closed lower than the open, or white candles on days where the price closed higher than the open. Sometimes in instructional material you will see gray bars, which means that the bar may be either white or black.

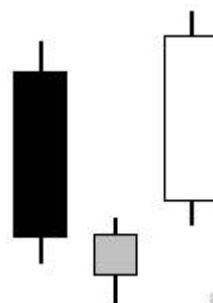


Figure 1

## Identifying Candlestick Patterns

Although you could very well write your own scripts to search for candlestick patterns, TradeScript™ provides a simple, built-in function that can identify up to two-dozen predefined patterns:

**CandlestickPattern()  
CSP()**

### Overview

The CandlestickPattern() 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 function returns the following constants. Also see chapter 5 for visual representations of these patterns:

**LONG\_BODY  
DOJI  
HAMMER  
HARAMI  
STAR  
DOJI\_STAR  
MORNING\_STAR  
EVENING\_STAR  
PIERCING\_LINE  
ENGULFING\_LINE  
HANGING\_MAN  
DARK\_CLOUD\_COVER  
BEARISH\_ENGULFING\_LINE  
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**

## ***Trading Systems***

A trading system is basically a set of rules that determine entry and exit points for any given stock. Traders often refer to these points as *trade signals*.

A trading system is objective and mechanical. The purpose is to provide a strategy to produce profits greater than losses by controlling your trades for you.

This chapter provides hands-on learning by teaching the trader how to translate trading system rules into script form using real trading systems as examples.

Trading systems usually include one or more technical indicators in their implementation. For example, a *Moving Average Crossover* system would buy when a short-term moving average crosses above a long-term moving average and sell when a short-term moving average crosses below a long-term moving average.

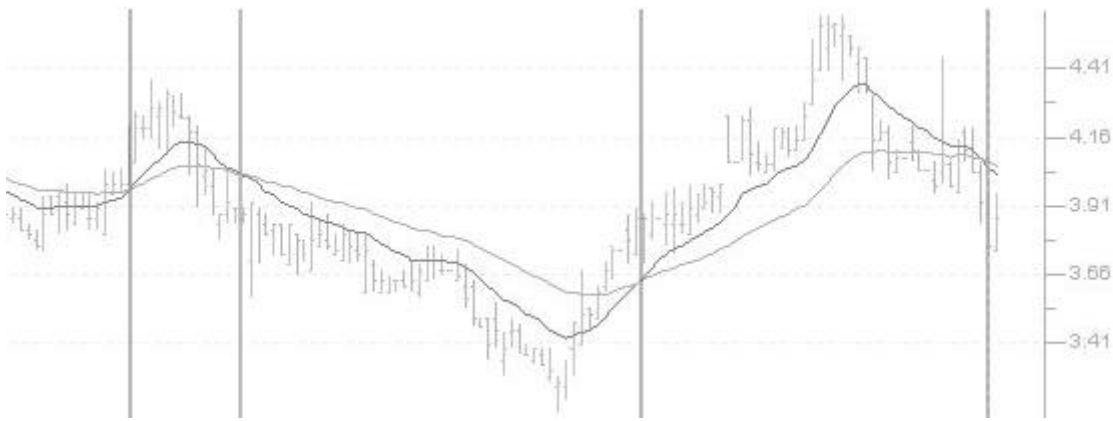
Trading systems may have any number of rules, such as “don’t buy unless volume is trending upwards”, or “exit if Parabolic SAR crosses the close”, etc.

The actual profitability of a trading system depends on how well the trading system’s rules perform on a trade-by-trade basis. Traders spend much of their time optimizing their trading systems in order to increase profits and reduce risks. In the case of a basic *Moving Average Crossover* system, this is accomplished by modifying the parameters of the moving averages themselves.

A trader may optimize a trading system by means of *back testing*. The back testing feature of TradeScript™ allows you to back test your trading systems and modify parameters to achieve the maximum amount of profit and minimum amount of risk. Refer to your trading software documentation for details.

## ***Moving Average Crossover System***

The Moving Average Crossover System is perhaps the simplest of all trading systems. This system uses two moving averages to generate signals. A buy signal is generated when a short-term moving average crosses over a longer-term moving average, and sells when a short-term moving average crosses below a long-term moving average.



The number of signals generated by this trading system is proportional to the length *and type* of moving averages used. Short-term moving averages generate more signals and enter into trades sooner than longer-term moving averages.

Unfortunately, a very short-term moving average crossover system will also generate more false signals than a longer-term system, while a very long-term system will generate fewer false signals, but will also miss a larger proportion of profits. This difficult balance applies to nearly every trading system and is the core subject of numerous books on technical analysis.

One solution to this problem is to use a secondary technical indicator to confirm entry and/or exit signals. A popular technical indicator used primarily for exit signals is the Parabolic SAR. The following script uses a 20/60 EMA for entries and a Parabolic SAR for exits.



## ***Moving Average Crossover System Script***

### **Buy Signals**

# 20-period EMA crosses over the 60-period EMA  
**CROSSOVER(EMA(CLOSE, 20), EMA(CLOSE, 60))**

### **Sell Signals**

# 20-period EMA crosses under the 60-period EMA  
**CROSSOVER(EMA(CLOSE, 60), EMA(CLOSE, 20))**

### **Exit Long**

# The close crosses above the Parabolic SAR  
**CROSSOVER(CLOSE, PSAR(CLOSE, 0.02, 0.2))**

### **Exit Short**

# The close crosses below the Parabolic SAR  
**CROSSOVER(PSAR(CLOSE, 0.02, 0.2), CLOSE)**

## Price Gap System

An upward price gap occurs when a stock opens substantially higher than the previous day's high price. This often occurs after an unexpected announcement, much better than expected earnings report, and so forth.

A large number of buy orders are executed when the market opens. During this time the price may be exaggerated as investors may be buying the stock simply because it shows strength at the opening.

The price often retreats to fill the gap once orders stop coming in and the demand for the stock subsides. The key to this trading system is that reversals usually occur during the *first hour* of trading. In other words, if the gap is not filled during the first hour then we may assume that buying will continue.



This trading system is often more successful if volume is around twice the five-day average of volume.

Example: The script returns securities that have gapped up by 2% and closed near the high. When the market opens on the following day, the strategy would be to buy stock after the first hour of trading if the strength sustained.

A stop-loss order would be set at the day's low. A conservative profit objective would normally be half the percentage of the gap, or 1% in this case.

## ***Price Gap Script***

### **Buy Signals**

# A 2% gap up in price over the previous day on high volume

**LOW > REF(HIGH,1) \* 1.02 AND  
VOLUME > SMA(VOLUME, 5) \* 2**

### **Sell Signals**

# A 2% gap down in price over the previous day on high volume

**HIGH < REF(LOW,1) \* 0.98 AND  
VOLUME > SMA(VOLUME, 5) \* 2**

### **Exit Long**

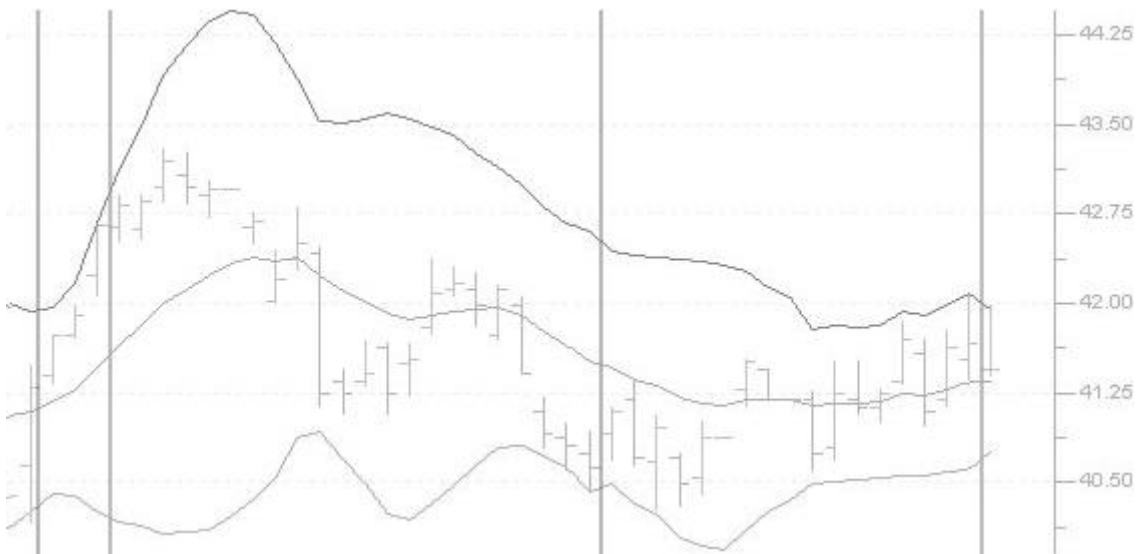
Use a profit objective roughly  $\frac{1}{2}$  the size of the gap with a stop-loss.

### **Exit Short**

Use a profit objective roughly  $\frac{1}{2}$  the size of the gap with a stop-loss.

## ***Bollinger Bands System***

Bollinger bands are similar to moving averages except they are shifted above and below the price by a certain number of standard deviations to form an envelope around the price. And unlike a moving average or a moving average envelope, Bollinger bands are calculated in such a way that allows them to widen and contract based on market volatility.



Prices usually stay contained within the bands. One strategy is to buy or sell after the price touches and then retreats from one of the bands. A move that originates at one band usually tends to move all the way to the other band.

Another strategy is to buy or sell if the price goes outside the bands. If this occurs, the market is likely to continue in that direction for some length of time.

The Bollinger band trading system outlined in this example uses a combination of both trading strategies. The system buys if a recent bar touched the bottom band and the current bar is within the bands, and also buys if the current high has exceeded the top band by a certain percentage. The system sells based on the opposite form of this strategy.

## ***Bollinger Bands Script***

### **Buy Signals**

```
# Buy if a previous value was below the low band and is now 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  
# Also buy if the close is above the top band plus 2%  
CLOSE > Top * 1.02
```

### **Sell Signals**

```
# Sell if a previous value was above the high band and is now 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  
# Also sell if the close is below the bottom band minus 2%  
CLOSE < Bottom * 0.98
```

## ***Historical Volatility and Trend***

This trading system buys or sells on increasing volume and lessening volatility. The concept is that trends are more stable if volatility has been decreasing and volume has been increasing over many days.



Volume is an important component to this trading system since almost every important turning point in a stock is accompanied by an increase in volume.

The key element in this trading system is the length of the primary price trend. The longer the price trend is, the more reliable the signal.

Also try experimenting with this trading system by substituting the TREND function for volume with the Volume Oscillator function, or the Volume Rate of Change function.

## ***Historical Volatility and Trend Script***

### **Buy Signals**

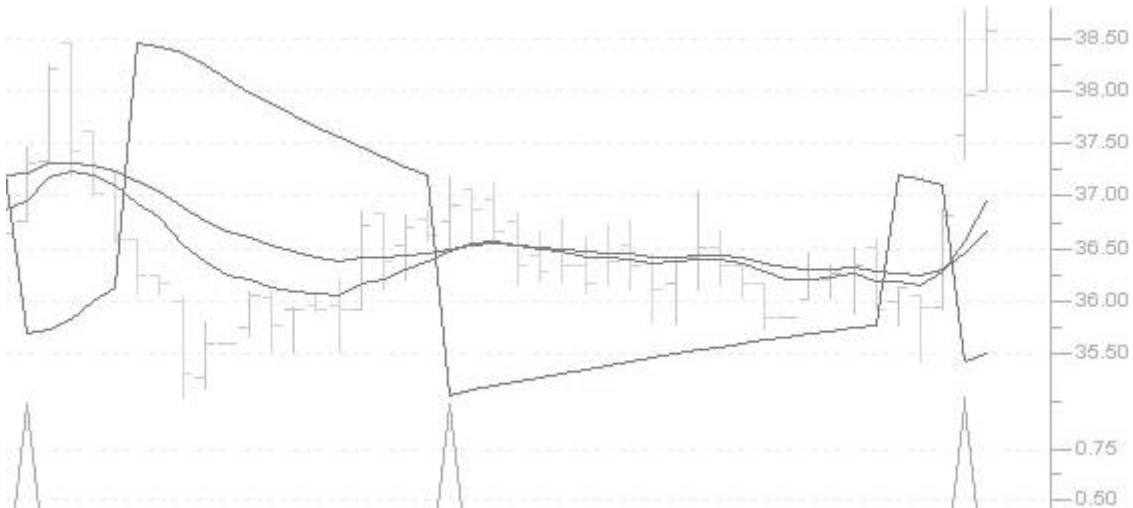
# Buy if volatility is decreasing and volume is increasing with price in an uptrend  
**HistoricalVolatility(CLOSE, 15, 252, 2) < REF(HistoricalVolatility(CLOSE, 15, 365, 2), 15)**  
**AND**  
**TREND(VOLUME, 5) = UP AND TREND(CLOSE, 40) = UP**

### **Sell Signals**

# Sell if volatility is decreasing and volume is increasing with price in a downtrend  
**HistoricalVolatility(CLOSE, 15, 252, 2) < REF(HistoricalVolatility(CLOSE, 15, 365, 2), 15)**  
**AND**  
**TREND(VOLUME, 5) = UP AND TREND(CLOSE, 40) = DOWN**

## ***Parabolic SAR / MA System***

This system is a variation of a standard moving average crossover system. Normally a Parabolic SAR is used only as a signal for exit points, however in this trading system we use the crossover of two exponential moving averages to decide if we should buy or sell whenever the Parabolic SAR indicator crosses over the close.



The Parabolic SAR can be used in the normal way after the trade has been opened. Profits should be taken when the close crosses the Parabolic SAR.

This example shows how to use Boolean logic to find securities that match the condition either for the current trading session or the previous trading day.



## ***Parabolic SAR / MA Script***

### **Buy Signals**

# Buy if the MAs crossed today or yesterday and

# if the PSAR crossed today or yesterday

**FIND STOCKS WHERE**

**(CROSSOVER(CLOSE, PSAR(0.02, 0.2)) OR  
CROSSOVER(REF(CLOSE,1), PSAR(0.02, 0.2)))**

**AND**

**(CROSSOVER(EMA(CLOSE, 10), EMA(CLOSE, 20)) OR  
CROSSOVER(REF(EMA(CLOSE, 10),1), REF(EMA(CLOSE, 20),1)))**

### **Sell Signals**

# Sell if the MAs crossed today or yesterday and

# if the PSAR crossed today or yesterday

**FIND STOCKS WHERE**

**(CROSSOVER(PSAR(0.02, 0.2), CLOSE) OR  
CROSSOVER(PSAR(0.02, 0.2), REF(CLOSE,1)))**

**AND**

**(CROSSOVER(EMA(CLOSE, 20), EMA(CLOSE, 10)) OR  
CROSSOVER(REF(EMA(CLOSE, 20),1), REF(EMA(CLOSE, 10),1)))**

## **MACD Momentum System**

In this trading system we use an exponential moving average and the TREND function to identify market inertia, and we use the Moving Average Convergence / Divergence (MACD) indicator to detect market momentum.



As you may know, the MACD indicator reflects the change of power between traders who are on the long side and traders who are on the short side. When the trend of the MACD indicator goes up, it indicates that the market is predominated by bulls, and when it falls, it indicates that bears have more influence. This is known as market momentum.

This system buys when both inertia (a TREND of the EMA) and momentum (the MACD) are both in favor of rising prices. The system sells when the reverse is true.

Exit signals are generated whenever either signal turns to the opposite direction.

## **MACD Momentum Script**

### **Buy Signals**

# Buy if both momentum and inertia are favorable

TREND(EMA(CLOSE, 20), 15) = UP AND  
TREND(MACD(13, 26, 9, SIMPLE), 5) = UP

### **Sell Signals**

# Sell if both momentum and inertia are favorable

TREND(EMA(CLOSE, 20), 15) = DOWN AND  
TREND(MACD(13, 26, 9, SIMPLE), 5) = DOWN

### **Exit Long Signal**

# Exit if either momentum or inertia become unfavorable

TREND(EMA(CLOSE, 20), 15) = DOWN OR  
TREND(MACD(13, 26, 9, SIMPLE), 5) = DOWN

### **Exit Short Signal**

# Exit if either momentum or inertia become unfavorable

TREND(EMA(CLOSE, 20), 15) = UP OR  
TREND(MACD(13, 26, 9, SIMPLE), 5) = UP

## ***Narrow Trading Range Breakout***

Stocks that remain bound by narrow trading ranges often tend to continue in the direction of their breakout. That is to say, if a stock remains in a narrow range between \$40 and \$45 for an extended period then breaks above \$50, it is likely to continue rising for the foreseeable future. The premise being that the longer a stock remains in a tight range, the more difficult it becomes to breakout of the trading range. Therefore when the breakout occurs, the uptrend should continue.



## ***Narrow Trading Range Script***

# Define a 2% trading range over 50 days

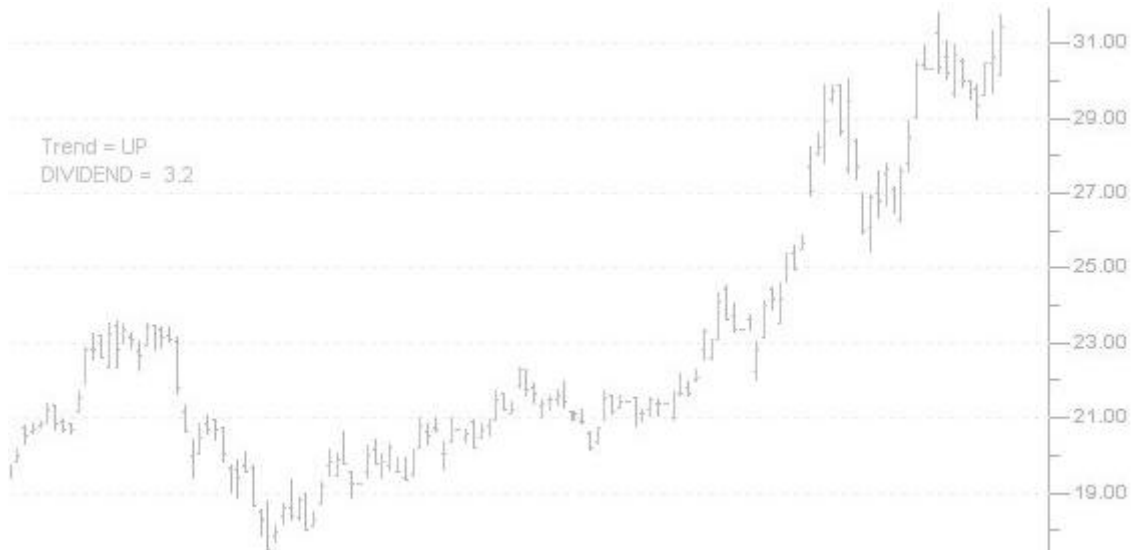
```
FIND STOCKS WHERE  
MAX(CLOSE, 50) < CLOSE * 1.01 AND  
MIN(CLOSE, 50) > CLOSE * 0.98 AND
```

# Filter out inactive securities

```
CLOSE != REF(CLOSE, 1) AND  
REF(CLOSE,1) != REF(CLOSE, 2) AND  
REF(CLOSE,2) != REF(CLOSE, 3)
```

## ***Fundamental Trading System***

The fundamental system evaluates to true when a security is in an uptrend and has a high dividend payout. The concept of this medium to long term trading system is that securities paying significant dividends tend to have less downside risk than other securities.



One of the biggest advantages of dividend paying securities is that you get paid just to hold them, which is in sharp contrast to the usual situation where one must sell a stock at a higher price to make a profit. This doesn't mean that dividend stocks won't go up however. Studies have shown that high dividend paying stocks may actually outperform non-dividend stocks in terms of total return.

## ***Fundamental Trading System Script***

### **Buy Signals**

# Buy in an uptrend with a high dividend  
**DIVIDEND > 3 AND TREND(CLOSE, 50) = UP**

### **Sell Signals**

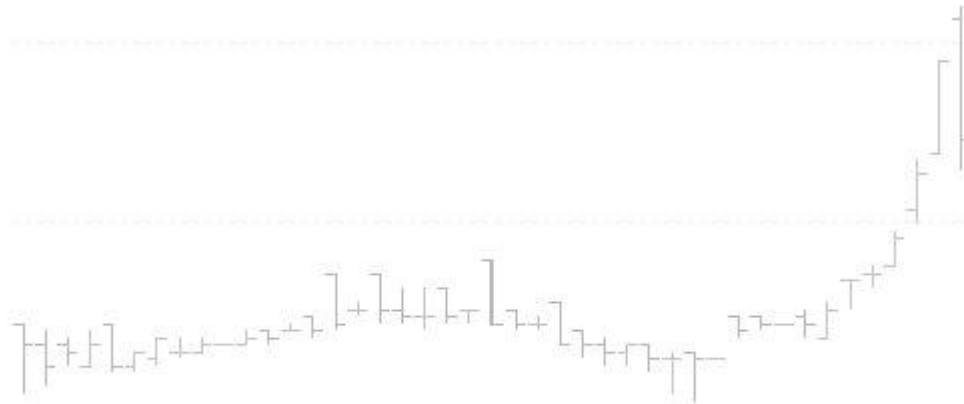
NA

### **Exit Signals**

Exit if and when dividend decreases or TREND changes to DOWN.

## ***Outside Day System***

An Outside Day occurs when the current bar's high price is higher than the previous bar's high price, and the current bar's low price is lower than the previous bar's low price. The close must be opposite of the trend (if the trend is up, the close must be lower than the open). Outside days occur frequently and may be used as part of a short term trading strategy.



Outside days that occur after a strong uptrend as shown in this chart indicate market indecision, and may signal a reversal or temporary correction in price. Depending on market direction, outside days can be either bullish or extremely bearish. If the reversal occurs at the stock's resistance level, it is interpreted as bearish. If it occurs at the stock's support level, it is interpreted as bullish.

## ***Outside Day Script***

### **Buy Signals**

# Find outside days

**LOW < REF(LOW, 1) AND  
HIGH > REF(HIGH, 1) AND  
HIGH > REF(HIGH, 1) AND  
CLOSE < OPEN AND**

# Outside days are more significant if the

# previous bar is shorter in height

**HIGH - LOW > (REF(HIGH, 1) - REF(LOW, 1)) \* 1.5 AND**

# The trend should be up

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

### **Sell Signals**

# Find outside days

**LOW < REF(LOW, 1) AND  
HIGH > REF(HIGH, 1) AND  
HIGH > REF(HIGH, 1) AND  
CLOSE < OPEN AND**

**HIGH - LOW > (REF(HIGH, 1) - REF(LOW, 1)) \* 1.5 AND**

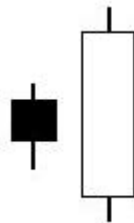
# The trend should be down for a sell signal

**TREND(CLOSE, 30) = DOWN**

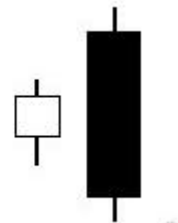


## Japanese Candlestick Engulfing Line System

Bullish and Bearish Engulfing Lines may be part of an effective short term trading strategy when used with volume. These patterns seem to be more predictive when the volume has increased sharply. Engulfing Lines often signal a trend reversal. The candlestick pattern consists of a short body, followed by a day with a taller body that completely engulfs the previous day's body.



Bullish Pattern



Bearish Pattern

A Bullish Engulfing Line indicates a potential reversal of a downtrend, whereas a Bearish Engulfing line indicates a potential reversal of an uptrend.

The patterns have more predictive power if they occur after a significant trend.

### Signals

#### # Bullish Pattern

```
CANDLESTICKPATTERN() = BULLISH_ENGULFING_LINE AND TREND(CLOSE, 30) =  
DOWN AND VOLUME > REF(VOLUME, 1)
```

#### # Bearish Pattern

```
CANDLESTICKPATTERN() = BEARISH_ENGULFING_LINE AND TREND(CLOSE, 30) =  
UP AND VOLUME > REF(VOLUME, 1)
```

Also see [chapter 5](#) for a complete list of other Japanese candlestick patterns.

## ***Primitive Types***

This chapter serves as a look-up reference for primitive variables, constants, and flags, such as definitions for fundamental data, candlestick patterns, technical indicator flags, and sector & industry constants.

### ***Price Vectors***

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

<b>OPEN</b>	Opening price of the day
<b>HIGH</b>	High price of the day (updates throughout the trading session)
<b>LOW</b>	Low price of the day (updates throughout the trading session)
<b>CLOSE</b>	Closing price of the day
<b>LAST</b>	The last price (updates throughout the trading session)
	Same value as <b>CLOSE</b> if after trading hours.
<b>VOLUME</b>	Trading volume for the day (updates throughout the session)

### ***Fundamental Variables***

Fundamental variables are updated throughout the trading session.

<b>PE_RATIO</b>	Price to Earnings Ratio
<b>DIVIDEND</b>	Annual Dividend
<b>YIELD</b>	Yield Value
<b>52_WEEK_HIGH</b>	The 52-week high
<b>52_WEEK_LOW</b>	The 52-week low
<b>ALL_TIME_HIGH</b>	The all time high since stock inception
<b>ALL_TIME_LOW</b>	The all time low since stock inception

**Basic Constants**

<b>TRUE</b>	1
<b>FALSE</b>	0
<b>PI</b>	3.1415926535897932384626433832795
<b>NULL</b>	0 (empty vector)
<b>ADD</b>	1 (used by <b>LOOP</b> function)
<b>SUBTRACT</b>	2 (used by <b>LOOP</b> function)
<b>MULTIPLY</b>	3 (used by <b>LOOP</b> function)
<b>DIVIDE</b>	4 (used by <b>LOOP</b> function)

**Back Testing Flags**

**MAX\_POSITION\_OPEN** Flag used for back testing. When this value is set via the **SET** keyword, all positions will be closed out after the specified number of days regardless of if the buy, sell, or exit script has been evaluated to **TRUE**. Example:

```
# Never allow a position stay open longer than 20 days  
SET MAX_POSITION_OPEN = 20
```

### ***Moving Average Constants***

<b>SIMPLE</b>	1
<b>EXPONENTIAL</b>	2
<b>TIME_SERIES</b>	3
<b>VARIABLE</b>	4
<b>TRIANGULAR</b>	5
<b>WEIGHTED</b>	6
<b>VOLATILITY</b>	7
<b>WILDER</b>	8

### ***Trend Constants (used by TREND function)***

<b>UP</b>	1
<b>DOWN</b>	2
<b>SIDEWAYS</b>	3

### ***Points or Percent Constants (used by indicators)***

<b>POINTS</b>	1 (specifies output to be measured in points)
<b>PERCENT</b>	2 (specifies output to be measured in percent)

### ***Candlestick Pattern Constants***

**LONG\_BODY**



**DOJI**



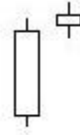
**HAMMER**



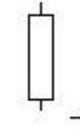
**HARAMI**



**STAR**



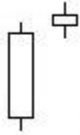
**DOJI\_STAR**



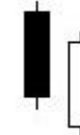
**MORNING\_STAR**



**EVENING\_STAR**



**PIERCING\_LINE**



**BULLISH\_ENGULFING\_LINE**



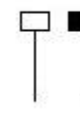
**BEARISH\_ENGULFING\_LINE**

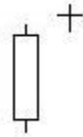
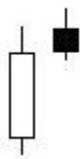
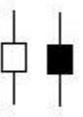
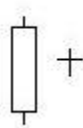
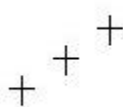
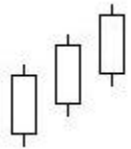
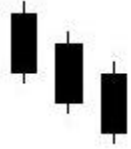
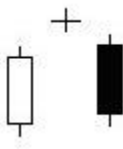
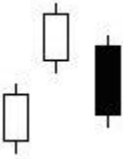
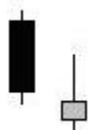


**DARK\_CLOUD\_COVER**

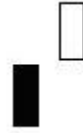


**HANGING\_MAN**



<b>BEARISH_DOJI_STAR</b>	
<b>BEARISH_SHOOTING_STAR</b>	
<b>SPINNING_TOPS</b>	
<b>HARAMI_CROSS</b>	
<b>BULLISH_TRISTAR</b>	
<b>THREE_WHITE_SOLDIERS</b>	
<b>THREE_BLACK_CROWS</b>	
<b>ABANDONED_BABY</b>	
<b>BULLISH_UPSIDE_GAP</b>	
<b>BULLISH_HAMMER</b>	

**BULLISH\_KICKING**



**BEARISH\_KICKING**



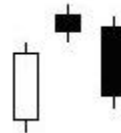
**BEARISH\_BELT\_HOLD**



**BULLISH\_BELT\_HOLD**



**BEARISH\_TWO\_CROWS**



**BULLISH\_MATCHING\_LOW**



## **Sector and Industry Constants**

*(these constants may not be supported in your software application)*

The following sector and industry constants may be used as a filter to limit your trading system to a specific sector or industry.

Sector and Industry constants can be set via the **SET** keyword, e.g.:

**SET Sector = Basic Materials**

Or:

**SET Industry = Personal & Household Products**

### **Sector Constants**

Services  
Capital Goods  
Transportation  
Consumer Cyclical  
Consumer Non-Cyclical  
Healthcare  
Services  
Basic Materials  
Energy  
Technology  
Services  
Technology  
Conglomerates  
Capital Goods  
Financial  
Basic Materials  
Consumer Non-Cyclical  
Basic Materials  
Consumer Non-Cyclical  
Consumer Cyclical  
Basic Materials  
Consumer Cyclical  
Basic Materials  
Healthcare  
Services  
Financial  
Basic Materials  
Consumer Cyclical  
Healthcare



Basic Materials  
Capital Goods  
Basic Materials  
Financial  
Transportation  
Capital Goods  
Financial  
Services  
Utilities  
Basic Materials  
Technology  
Consumer Non-Cyclical  
Energy  
Basic Materials  
Consumer Non-Cyclical  
Services  
Consumer Cyclical  
Services  
Transportation  
Services  
Consumer Cyclical  
Financial  
Services  
Financial  
Services  
Technology  
Services  
Technology  
Consumer Cyclical  
Consumer Non-Cyclical  
Transportation  
Services  
Transportation  
Utilities

### ***Industry Constants***

Advertising  
Aerospace & Defense  
Air Courier  
Airline  
Apparel/Accessories  
Appliance & Tool  
Audio & Video Equipment  
Auto & Truck Manufacturers  
Auto & Truck Parts  
Beverages (Alcoholic)  
Beverages (Non-Alcoholic)  
Biotechnology & Drugs  
Broadcasting & Cable TV  
Business Services  
Casinos & Gaming  
Chemical Manufacturing  
Chemicals - Plastics & Rubber  
Coal  
Communications Equipment  
Communications Services  
Computer Hardware  
Computer Networks  
Computer Peripherals  
Computer Services  
Computer Storage Devices  
Conglomerates  
Construction & Agricultural Machinery  
Construction - Supplies & Fixtures  
Construction - Raw Materials  
Construction Services  
Consumer Financial Services  
Containers & Packaging  
Crops  
Fabricated Plastic & Rubber  
Fish/Livestock  
Food Processing  
Footwear  
Forestry & Wood Products  
Furniture & Fixtures  
Gold & Silver  
Healthcare Facilities  
Hotels & Motels  
Insurance (Accident & Health)  
Insurance (Life)

Insurance (Miscellaneous)  
Insurance (Prop. & Casualty)  
Investment Services  
Iron & Steel  
Jewelry & Silverware  
Major Drugs  
Medical Equipment & Supplies  
Metal Mining  
Misc. Capital Goods  
Misc. Fabricated Products  
Misc. Financial Services  
Misc. Transportation  
Mobile Homes & RVs  
Money Center Banks  
Motion Pictures  
Natural Gas Utilities  
Non-Metallic Mining  
Office Equipment  
Office Supplies  
Oil & Gas - Integrated  
Oil & Gas Operations  
Oil Well Services & Equipment  
Paper & Paper Products  
Personal & Household Products  
Personal Services  
Photography  
Printing & Publishing  
Printing Services  
Railroads  
Real Estate Operations  
Recreational Activities  
Recreational Products  
Regional Banks  
Rental & Leasing  
Restaurants  
Retail (Apparel)  
Retail (Catalog & Mail Order)  
Retail (Department & Discount)  
Retail (Drugs)  
Retail (Grocery)  
Retail (Home Improvement)  
Retail (Specialty)  
Retail (Technology)  
S&Ls/Savings Banks  
Schools  
Scientific & Technical Instr.

Security Systems & Services  
Semiconductors  
Software & Programming  
Textiles - Non Apparel  
Tires  
Tobacco  
Trucking  
Waste Management Services  
Water Transportation  
Water Utilities

## ***Troubleshooting***

### **Problem**

TradeScript™ reports a script error such as: “The variable name ---- is not recognized” or “The argument of ---- function is not optional”.

### **Solution**

Follow the instructions on your screen to correct the script error.

---

### **Problem**

TradeScript™ reports a script timeout.

### **Solution**

Either shorten your script or contact your software vendor for support.



# Index

## A

Abandoned Baby, Candlesticks 92  
Abbreviations, Functions 31  
Abs 23  
Accumulative Swing Index 59  
ADD, Operator 89  
ADX, DI+, DI-, Directional System 48  
AND, Keyword 28  
Aron 50  
Arrays 9  
Atn 24  
Average, Avg 20

## B

Band Indicators 39  
Bands, Prime Number 41  
BASIC Programming Language 8  
Belt Hold, Bearish Candlesticks 92  
Belt Hold, Bullish Candlesticks 92  
Bollinger Bands 39  
Bollinger Bands Trading System 74  
Boolean Logic 6

## C

Chaikin Money Flow Index 55  
Chaikin Volatility Index 50  
Chande Momentum Oscillator 43  
CLOSE 88  
Commodity Channel Index 60  
Comparative RSI 55  
Conditional IF Function 17  
Constants 89  
Control Structure 16  
Correlation Analysis 66  
Cos 24  
COUNTIF 19  
C++ Programming Language 8  
Crossovers 13

## D

Dark Cloud Cover, Candlesticks 91  
Detrended Price Oscillator 44  
DIVIDE, Operator 89  
Dividend 12  
Doji, Candlesticks 90  
Doji Star, Bullish Candlesticks 90  
Doji Star, Bearish Candlesticks 91

## E

Ease Of Movement 47  
Engulfing Line System 91  
Engulfing Lines, Bearish Candles 91  
Engulfing Lines, Bullish Candles 91  
Equals, Operator 26  
Eqv 29  
Erroneous Results, Prevention 7  
Evening Star, Candlesticks 91  
Exp 24  
Exponential Moving Average 32

## F

FALSE 89  
Forecast 38  
Fractal Chaos Oscillator 45  
Functions, Built-In 7  
Fundamental Analysis 12  
Fundamental Trading System 83

## G

Gaps, Price 11  
General Indicators 62  
Greater Than, Operator 26  
Greater Than Or Equal To, Operator 27  
GUID, Globally Unique Identifier 15

## H

Hammer, Bullish Candlesticks 92  
Hanging Man, Candlesticks 91  
Harami, Candlesticks 90  
Harami Cross, Candlesticks 91  
HIGH 88  
High, 52 Week 12  
High, All Time 12  
High Minus Low 51  
Highest High Value 64  
Historical Volatility System 76  
Historical Volatility Index 54

## I

Index Functions 53  
Industries, Filter 14  
Inertia 80  
Intercept, Linear Regression 38

## J

Japanese Candlesticks 67  
Java Programming Language 8

## K

Keltner Channels 40  
Key Reversals 14  
Kicking, Bearish Candlesticks 92  
Kicking, Bullish Candlesticks 92

## L

LAST 88  
LASTIF 19  
Less Than 26  
Less Than Or Equal To, Operator 27  
Linear Regression 37

Linking Scripts, LINK 15  
Log 25  
Log10 25  
Long Body, Candlesticks 90  
LOOP Function 17  
LOW 88  
Low, 52 Week 12  
Low, All Time 12  
Lowest Low Value 65

## M

Moving Average Crossover Sys 70  
MACD 51  
MACD System 80  
Mass Index 53  
Matching Low, Bullish Candlesticks 93  
Math Functions 23  
Max 20  
MAX\_OPEN\_POSITION 89  
MAXOF 21  
Median Price 62  
Min 21  
MINOF 21  
Mod 30  
Momentum Oscillator 42  
Money Flow Index 54  
More Help and Support 100  
Morning Star, Candlesticks 90  
Moving Averages 32  
Moving Average Envelope 41  
Multidimensional 8  
MULTIPLY, Operator 89

## N

Negative Volume Index 57  
NOT 29  
Not Equal To 27  
Null 89



## O

On Balance Volume 57  
OPEN 88  
Operators 26  
OR Keyword 28  
Oscillator Indicators 42  
Outside Day System 85  
Overbought 42  
Oversold 42

## P

Parabolic SAR 60  
Parabolic SAR System 78  
PE Ratio 12  
Performance Index 58  
PI 89  
Piercing Line, Candlesticks 91  
Positive Volume Index 56  
Price 88  
Price Gap System 72  
Price Oscillator 44  
Price Rate of Change 63  
Price Volume Trend 56  
Prime Number Oscillator 45  
Primitives 16  
Program Structure 6

## R

R Squared Linear Regression 37  
Rainbow Oscillator 46  
REF 22  
Relative Strength Index 53  
Random, Rnd 25

## S

Scalar Operations 8  
Sector & Industry Constants 94  
Sectors Filter 14

Simple Moving Average 99  
Sin 23  
Slope, Linear Regression 37  
Standard Deviations 65  
Spinning Tops, Candlesticks 91  
Star, Candlesticks 90  
Stochastic Momentum Index 61  
Stochastic Oscillator 52  
SUBTRACT, Operator 89  
Sum 20  
SUMIF 19  
Swing Index 59

## T

Tan 24  
Technical Analysis 31  
Three Black Crows, Candlesticks 92  
Three White Soldiers, Candlesticks 92  
Time Series Moving Average 32  
Timeout, Script 100  
Trade Volume Index 58  
Trading Range Breakout System 82  
Trading System 69  
TREND 22  
Triangular Moving Average 34  
Tri Star, Bullish Candlesticks 92  
TRIX 46  
Troubleshooting 99  
TRUE 89  
True Range 48  
Two Crows, Bearish Candlesticks 93  
Typical Price 62

## U

Upside Gap, Bullish Candlesticks 92

Shooting Star, Bearish Candlesticks 91

## V

Variable Moving Average 34

Variable Missing, Error 99

Vector Programming 8

Vertical Horizontal Filter 47

VIDYA 36

Volatility Measure 11

Volume 88

Volume Oscillator 43

Volume Rate of Change 64

## W

Weighted Close 63

Weighted Moving Average 35

Welles Wilder Smoothing 35

Williams Accumulation / Distribution 49

Williams %R 49

## X

XOR Keyword 29

## Y

Yield 12

## Z

Zero Based Array 9