



FINATEK

30 Montgomery St., suite 280
Jersey City, NJ 07302, USA

The Revolutionary MyForex™ Trading Platform

+1.201.644.4040; +1.866.222.8387 (fax)
info@finatek.com; <http://finatek.com>

User Manual

myForex Formulae Language

Contents

Contents	2
What is the myForex Formulae Language?	6
Glossary of Terms	7
Formula Building Blocks	9
Price Array Identifiers.....	9
Mathematical Operators.....	9
Operator Precedence.....	10
Formulae Functions	10
Function Parameters.....	11
Writing Comments	11
Nesting Functions	11
The if() function	12
Using "And" and "Or" Operators	12
Using Variables.....	13
Using Variables to Enhance and Simplify	13
Naming of Variables	13
Using Variables to Represent Numbers	13
Using Variables to Represent Mathematical Expressions	14
Self Referencing Formulae	15
Self Referencing Formulas Using PREV	15
Self-Referencing Formula Calculations.....	15
Functions	16
Absolute Value	16
Addition	16
Arc Tangent	16
Aroon	16
Aroon Down	17
Aroon UP	17

Average Directional Movement.....	18
Average True Range.....	18
Bars Since.....	19
Bollinger Band Bottom	19
Bollinger Band Top	19
Bollinger Bands.....	20
Ceiling.....	20
Cosine.....	21
Cross.....	21
Cumulate.....	21
Day Of Month.....	21
Day Of Week.....	22
Detrended Price Oscillator	22
Directional Movement Index	22
Directional Movement Rating.....	23
Divergence.....	23
Division	24
Exponent.....	24
Floor.....	24
Fraction	24
Gap Down	25
Gap Up.....	25
Highest.....	25
Highest Bars Ago	25
Highest High Value	25
Highest High Value Bars Ago	26
Highest Since.....	26
Highest Since Bars Ago	26
Hour	26
If.....	27
Inside	27
Integer.....	28
IsDefined.....	28
IsUndefined.....	28
Last Value in Data Array	28
Linear Regression Indicator	28

Linear Regression Slope.....	29
Logarithm	29
Lowest.....	29
Lowest Bars Ago	29
Lowest Low Value	29
Lowest Since.....	30
Lowest Since Bars Ago.....	30
MACD	30
Maximum	31
Median Price	31
Midpoint	31
Minimum	32
Minus Directional Movement.....	32
Minute	32
Modulus	32
Momentum	33
Month	33
Moving Average	33
Multiplication	33
Negative.....	34
Outside.....	34
Parabolic SAR.....	34
Peak Bars Ago	34
Peak Value.....	35
Performance	35
Plus Directional Movement	35
Power	36
Precision	36
Prev.....	36
Price Oscillator	37
Projection Band Bottom	38
Projection Band Top	38
Rally	38
Rally With Volume.....	38
Rate of Change.....	38
Reaction.....	39

Reaction With Volume	39
Reference	39
Relative Strength Index.....	39
Round	40
Sine.....	40
Square Root.....	40
Standard Deviation	40
Standard Error	41
Standard Error Band Bottom.....	41
Standard Error Band Top.....	42
Stochastic Momentum Index.....	42
Stochastic Oscillator	42
Subtraction.....	43
Summation.....	43
Tick	44
Trough Bars Ago.....	44
Trough Value	44
Typical Price	44
Ultimate Oscillator.....	45
Value When	45
Variance.....	45
Vertical Horizontal Filter	45
Volatility, Chaikin's	46
Weighted Close.....	46
Wilder's Smoothing	46
Williams' %R	47
Williams' A/D.....	47
Year	47
Zig Zag.....	48

What is the myForex Formulae Language?

The myForex Formulae Language is an easy to use programming language which allows you to define and create custom indicators, system tests, explorations, and experts. It is patterned after popular spreadsheet languages.

In its simplest form, the myForex Formulae Language is comprised of high-level functions (e.g., `mov()`, `rsi()`, `abs()`), mathematical operators (e.g., `+`, `-`, `/`, `*`), and parameters (open, high, low, close, etc.). Each of these basic components can be combined to create your own indicators with the Indicator Builder.

In order to effectively use the Indicator Builder you need to get familiar with the myForex Formulae Language.

The Indicator Builder will be used in this Manual to teach the myForex Formulae Language.

Glossary of Terms

This glossary defines the terms used with the myForex Formulae Language. Understanding (or remembering) these terms is not required, however, adding these terms to your vocabulary will make discussion easier with other myForex Trading analysts.

COMMENT: Text within a formula that is not part of the formula. A comment must be surrounded by the characters { and }.

CONSTANT: A specific type of parameter that is required by a function. Constants can be subdivided into the following groups:

CALCULATION METHOD CONSTANT: Used to define the mode of calculation. Defined as **PERCENT** or **POINTS**. (PERCENT and POINTS can be abbreviated to % or \$.)

COMPARISON CONSTANT: Used in the if() function to define the comparison operation. Defined by >, >=, <, <=, <>, or =.

FORMULA CONSTANT: Used within the fml() function to reference another formula. A formula constant is specified as the name of another formula enclosed in quotation marks (e.g., fml("My Formula")).

MOVING AVERAGE TYPE CONSTANT: Used to define the moving average calculation method. Defined as EXPONENTIAL, SIMPLE, TIME SERIES, TRIANGULAR, VARIABLE, or WEIGHTED. (These can be abbreviated as E, S, T, TRI, VAR, and W.)

NUMERIC CONSTANT: A single numeric value. A function requiring a numeric constant cannot accept a data array since a data array may contain multiple, rather than single, numeric values. An example of a numeric constant is the "10" in the formula "mov(C,10,E)."

DATA ARRAY: A data array defines a specific set of information (data) that is used within a formula. Data arrays can be subdivided into more specific definitions:

FUNCTION RESULT ARRAY: A data array that is created as the result of the execution of a function.

LITERAL ARRAY: A data array defined using a single numeric constant.

PRICE ARRAY: An array containing the information stored in the high, low, close, etc., data arrays.

FORMULA: A combination of comments, constants, functions, mathematical operators and/or price array identifiers.

FUNCTION: A pre-defined mathematical operation that can be performed on a set of parameters to produce a desired data array.

OPERATOR, MATHEMATICAL: The +, -, *, and / operators.

OPERATOR, LOGICAL: The <, >, <=, >=, =, <>, AND, and OR operators.

PARAMETER: An item contained within a function. When a function has multiple parameters, they are separated by commas

PRECEDENCE: The order in which a formula is evaluated (see Operator Precedence).

PRICE ARRAY IDENTIFIERS: The letters or words used to reference price arrays (Open, High, Low, Close, Volume, PREV).

Formula Building Blocks

Price Array Identifiers

One of the most basic building blocks of a formula is called a price array identifier. A price array identifier "identifies" specific price fields that the formula should operate on. The valid price array identifiers are open, high, low, close, volume, open interest, and indicator.

Price array identifiers can be abbreviated as shown in the following table. Note that these are not case-specific.

Long Name	Abbreviation
Open	O
High	H
Low	L
Close	C
Volume	V
Previous Value	PREV

Examples of the use of price array identifiers in formulas are shown below. The actual price array identifier component of the formulas are **in bold** for these examples.

```
mov( close,10, simple )
if ( h > ref(h,-1), mov(h,20,s), mov(s,20,s) )
stdev( volume, 20 )
```

Mathematical Operators

Mathematical operators are the "glue" that binds formulas. Formulas can contain the following mathematical operators. (They also can contain advanced operators such as square root, as explained later.)

- + Addition
- Subtraction (or negative)
- * Multiplication
- / Division

The following formulas illustrate the use of operators in a formula:

```
( H + L ) / 2
mov(c,10,s)- mov(c,20,s) / (h + l + c)
close + ((1.02 * high)- high)
```

Operator Precedence

Parentheses were used in many of the preceding formulas in this chapter to control the operation precedence (the order in which the operators are calculated). myForex Trading always does operations within the innermost parentheses first.

When parentheses are not used, the precedence is as follows:

- Negative values
- * Multiplication
- / Division
- + Addition
- Subtraction
- < Less than
- > Greater than
- <= Less than or equal to
- >= Greater than or equal to
- = Equal to
- <> Not equal to
- And Logical "And"
- Or Logical "Or"
- := Variable assignment operator

The expression "H + L / 2" (without parenthesis) would be calculated by myForex Trading as "L / 2" plus "H," since division has a higher precedence. This would result in a much different value than "(H + L) / 2."

For ease of reading, we recommend that you always use parenthesis to control precedence.

Formulae Functions

In addition to the four mathematical operators, myForex Trading contains over 100 "functions" that perform mathematical operations. See [Functions](#) for a complete listing of all built-in functions.

The following formula consists of a single function that plots the square roots of the closing prices. The names of the actual functions in the following examples are in **bold**.

sqrt (`CLOSE`)

The following formula consists of a single function that plots a 14-period [RSI](#) indicator.

rsi (14)

The following formula consists of a single function that plots a 10-period simple moving average.

mov (`c`, 10, `s`)

The following formula consists of two functions. The result is the difference between the [MACD](#) indicator and a 9-period exponential moving average of the MACD.

```
macd () - mov(macd(), 9, exponential)
```

As you've probably noticed, this syntax is very similar to the syntax used to enter formulas in spreadsheets.

All functions must be followed by a pair of parentheses. If an opening parenthesis is not the first character after a function name, an error message will be displayed.

Function Parameters

Parameters provide a function the necessary information required to calculate. For example, the `sqrt()` function requires a single "parameter" within the parentheses. Other functions, such as `macd()`, do not require any parameters. Others, like the Volume Oscillator require four parameters.

The following formula calculates a 14-period [Relative Strength Index](#). The actual parameters are in **bold** in the following examples.

```
rsi ( 14 )
```

Some functions require multiple parameters within the parentheses. For example, the moving average function (shown below) requires three parameters.

```
mov ( rsi (14), 30, simple )
```

In the above formula, the parameters instruct MetaStock to calculate a 30-period simple moving average of a 14-period RSI. Note that another function (i.e., `rsi(14)`) serves as one of the parameters.

If you forget to insert the proper parameter, MetaStock will display a window reminding you of the expected parameter.

Writing Comments

When writing long, complex formulas, it is helpful to make comments throughout the formula to describe what is going on. Experienced programmers will attest to the fact that the prudent use of comments makes programming code much easier to debug and work with.

Comments can be entered in a formula by surrounding them with "{" and "}" braces. The following formula contains two comments (shown in bold).

```
macd() {the MACD times} * ((H+L+C) / 3) {the average price}
```

Note that comments within comments will cause an error message to be displayed.

Nesting Functions

As has been eluded to in earlier examples, a function can be "nested" within a function. The nested function can serve as the main function's data array parameter. The following examples show functions nested within functions. The nested functions are in bold.

```
stdev( stoch(5,3), 10 )
```

The first example calculates a stochastic oscillator and then calculates a 10-period standard deviation of the stochastic oscillator.

```
mov( rsi(15), 10, SIMPLE)
```

The second example calculates a 10-period simple moving average of a 15-period Relative Strength Index (RSI).

```
mov( mov( rsi(15), 20, W), 10, SIMPLE)
```

The third example calculates a 20-period weighted moving average of a 15-period RSI, and then calculates a 10-period simple moving average of this moving average.

This technique (placing functions within functions) is referred to as the "nesting of functions."

The if() function

The [if\(\)](#) function is used to create conditional (i.e., "if-then") statements. It is perhaps the most used function in the myForex Formulae Language. It contains three parameters as shown in the following example.

```
if( close > mov(c,10,s), rsi(9), rsi(14) )
```

The above "if" statement reads (in English) as follows: If today's close is greater than today's 10-day simple moving average of the close, then plot a 9-day RSI, otherwise, plot a 14-day RSI.

The next formula plots "positive volume" if the close is greater than the median price. Otherwise, "negative volume" is plotted.

```
if( CLOSE > (HIGH+LOW)/2, +V, -V )
```

If you simply want an expression to be evaluated as either true or false, it can be done without the use of the if() function. The following formula will result in either a 1 (true) or a 0 (false).

```
rsi(14) > 70
```

If the 14-period RSI is greater than 70, then this formula will evaluate to "true" and return the number 1. If it is below 70, the formula will evaluate to "false" and return the number 0. This is done without the if() function being used. The formula below uses the if() function and will return the same results, but it is longer.

```
if(rsi(14) > 70, 1, 0 )
```

Using "And" and "Or" Operators

If a formula requires multiple conditions, you can combine the conditions with "and" and "or" operators. For example, maybe you'd like to plot a +1 when the [MACD](#) is greater than zero and the RSI is greater than 70. The formula could be written two ways:

```
macd() > 0 AND rsi(14) > 70
```

or

```
if(macd() > 0 and rsi(14) > 70, +1, 0)
```

You can add as many conditions within a formula as space allows. For example:

```
If(macd() > 0 AND rsi(14) > 70 AND CCI(14) > 100 AND close >
mov(close,10,e), +1, 0)
```

You can even combine AND and OR operators within the same formula as follows:

```
If((macd() > 0 OR close > mov(close,10,e)) AND rsi(14) > 70, +1, 0)
```

The formula above says to plot a "+1" if either the MACD is greater than zero or the close is above its moving average, and the RSI is greater than 70.

Note that parentheses were placed around the OR condition because precedence specifies that the AND condition be evaluated first (see [Operator Precedence](#)). If the parentheses were not placed around the OR condition, the moving average and the RSI would have been grouped together with the AND condition, which is not how we want the formula evaluated.

Using Variables

Using Variables to Enhance and Simplify

In order to shorten, simplify, enhance, and make the maintenance of complex formulas easier, you may want to use variables. A variable is an alphanumeric name (up to 20 characters) that is assigned to an expression or a single value. Up to 20 variables can be used in a formula. Variables must be assigned before the variable is used in the formula. A semi-colon must be used at the end of the variable assignment line. Variables cannot be assigned within a function.

Naming of Variables

The following rules apply to the naming of variables:

- * Variable names cannot contain commas, parenthesis, spaces, underscores, etc.
- * Variable names cannot duplicate names already used by functions (e.g., mov, rsi, cci, if, etc.).
- * A variable cannot be assigned a name that matches the parameters reserved for use in formulas (e.g., open, high, low, close, simple, o, c, l, h, s, e, w, etc.).

The following would produce an error, since the letter "s" is reserved for the moving average function, mov(), to mean "simple."

```
s:= (h+l+c)/3;
```

- * Variable names must contain at least one alpha letter (e.g., T1234).
- * Variable names are not case sensitive (e.g., "PERIODS" is the same as "periods").

Using Variables to Represent Numbers

Suppose you would like to be able to quickly adjust the time periods throughout a formula without having to edit each use of the time periods individually. This could be accomplished by assigning the time periods to a variable—in this case, a variable named "periods".

```
periods := 10;
```

```
out := c > mov(c,periods, s) and ref(c,-1) > ref(mov(c,periods,s),-1) and
h > mov(h,periods,s) and ref(h,-1) > ref(mov(h,periods,s),-1);
```

In the above formula, the number "10" will be substituted wherever the variable named "periods" appears in the formula. When you want to adjust the time periods in this formula, simply edit the number assigned to the "periods" variable. If you change "10" to "20," the number "20" will be substituted throughout the formula.

Of course, you could also assign multiple variables to represent multiple numbers as follows:

```
periods1 := 10;
periods2 := 20;
out := c > mov(c,periods1, s) and ref(c,-1) > ref(mov(c,periods1,s),-1)
and h > mov(h,periods2,s) and ref(h,-1) > ref(mov(h,periods2,s),-1);
```

In this case, the variables "periods1" and "periods2" represent two different values that can be used as many times as desired throughout the formula.

Using Variables to Represent Mathematical Expressions

Variables can be assigned to represent a mathematical expression (i.e., formula). Perhaps this is the most useful benefit of variables. Assigning variables to represent formulas (especially long, complex ones) makes formulas easier to read, easier to modify, and faster to calculate.

For example, the following formula (designed to spot securities locked between support and resistance levels), is quite complex and difficult to read. It can be simplified by using variables. It was originally written before variable support was added to myForex Formulae Language.

```
OUT := (If(Abs((Trough(1,L,1)-Trough(2,L,1))/Trough(2,L,1))<.015 AND
Abs((Trough(2,L,1)-Trough(3,L,1))/Trough(3,L,1))<.015, {then}
(Trough(1,L,1)+Trough(2,L,1)+Trough(3,L,1))/3,0))>0
and
(If(Abs((Peak(1,H,1)-Peak(2,H,1))/Peak(2,H,1))<.015 AND Abs((Peak(2,H,1)-
Peak(3,H,1))/Peak(3,H,1))<.015, {then}
(Peak(1,H,1)+Peak(2,H,1)+Peak(3,H,1))/3,0))>0
and
c>=(If(Abs((Trough(1,L,1)-Trough(2,L,1))/Trough(2,L,1))<.015 AND
Abs((Trough(2,L,1)-Trough(3,L,1))/Trough(3,L,1))<.015, {then}
(Trough(1,L,1)+Trough(2,L,1)+Trough(3,L,1))/3,0)) and
c<=(If(Abs((Peak(1,H,1)-Peak(2,H,1))/Peak(2,H,1))<.015 AND
Abs((Peak(2,H,1)-Peak(3,H,1))/Peak(3,H,1))<.015, {then}
(Peak(1,H,1)+Peak(2,H,1)+Peak(3,H,1))/3,0));
```

By defining two variables (in **bold**) to represent the two expressions that are used repeatedly, you can simplify the formula substantially. In addition to being easier to read, the simplified formula also calculates quicker since the support and resistance expressions now only need to calculate once.

```
Support:= (If(Abs((Trough(1,L,1)-Trough(2,L,1))/Trough(2,L,1))<.015 AND
Abs((Trough(2,L,1)-Trough(3,L,1))/Trough(3,L,1))<.015, {then}
(Trough(1,L,1)+Trough(2,L,1)+Trough(3,L,1))/3,0));
```

```
Resistance:= (If(Abs((Peak(1,H,1)-Peak(2,H,1))/Peak(2,H,1))<.015 AND
Abs((Peak(2,H,1)-Peak(3,H,1))/Peak(3,H,1))<.015,
{then}(Peak(1,H,1)+Peak(2,H,1)+Peak(3,H,1))/3,0));
```

```
OUT := Support <> 0 and Resistance <> 0 and close >= Support and close <= Resistance;
```

Self Referencing Formulae

Self Referencing Formulas Using PREV

The [PREV](#) constant allows you to create self-referencing formulas. A self-referencing formula is one that is able to reference the "previous" period's value of itself.

For example, the following is an example of a self-referencing formula:

```
OUT := ((H+L+C)/3) + PREV;
```

This simple formula divides the high, low, and closing prices by 3 and then adds this value to yesterday's value of the $((H+L+C)/3)$.

The calculation of the popular indicator On Balance Volume illustrates the use of the PREV function.

```
_OBV := (if(c>ref(c,-1),1,-1)*volume)+PREV;
```

Although On Balance Volume can be calculated without the use of the PREV function, an exponential moving average cannot (other than using the `mov()` function).

The following formula shows how a 18% exponential moving average (approximately 10-periods) is calculated using the PREV function.

```
MOV18 := (close*0.18)+(PREV*0.82);
```

Self-Referencing Formula Calculations

You are using the PREV variable in a location within a custom indicator that will cause the calculation to be very slow. You should normally try to avoid using the PREV variable as a DATA ARRAY when there is also a PERIODS specified (e.g., `mov(prev, 20, s)`, `hhv(prev, 30)`, etc.). Although use of the PREV function in this way is allowable, it will take a long time to calculate, particularly if a lot of data is loaded in the chart.

Functions

The following functions form the basis of myForex Formulae Language and they can be used for creation of proprietary indicators.

Absolute Value

abs (DATA ARRAY)

The function calculates the Absolute Value of the DATA ARRAY.

The formula `abs(-12)` will return +12;

The formula `abs(12)` also returns +12.

Addition

add (DATA ARRAY, DATA ARRAY)

The function adds the two parameters together.

The formula `add(H,12.7)` adds 12.7 to the High prices.

This formula also could be written as `H+12.7`.

Arc Tangent

atan (Y DATA ARRAY, X DATA ARRAY)

The function returns the Arc Tangent of Y/X.

The value is returned in degrees from 0 to 359.9.

The formula `atan(10,0)` returns 90.

Aroon

aroon (PERIODS, switch)

Calculates the Aroon Down and Aroon Up components of the Aroon Indicator.

The function `Aroon (period,0)` calculates the [Aroon Down](#) component of the Aroon Indicator.

{Aroon Down component of the Aroon Indicator}

```
_AroonDown := Aroon (PERIODS,0);
{end}
```

The function aroon (14,0) calculates the Aroon Down component of the Aroon Indicator over the period of 14 price values.

The function Aroon (period,1) calculates the [Aroon Up](#) component of the Aroon Indicator.

```
{Aroon Up component of the Aroon Indicator}
_AroonUp := Aroon (PERIODS,1);
{end}
```

The function aroon (14,1) calculates the Aroon Up component of the Aroon Indicator over the period of 14 price values.

Aroon Down

arooldown (PERIODS)

Calculates the Aroon Down component of the [Aroon Indicator](#).

The function AroonDown (period) calculates the Aroon Down component of the Aroon Indicator by formula:

```
{Aroon Down component of the Aroon Indicator}
_AroonDown := ((PERIODS - LLVBars (L,PERIODS +1)) / PERIODS) * 100;
{end}
```

or

```
{Aroon Down component of the Aroon Indicator}
_AroonDown := Aroon (PERIODS,0);
{end}
```

The function arooldown (14) calculates the Aroon Down component of the Aroon Indicator over the period of 14 price values.

Aroon UP

aroondup (PERIODS)

Calculates the Aroon Up component of the [Aroon Indicator](#).

The function AroonUp (period) calculates the Aroon Up component of the Aroon Indicator by formula:

```
{Aroon Up component of the Aroon Indicator}
_AroonUp := ((PERIODS - HHVBars (L,PERIODS +1)) / PERIODS) * 100;
{end}
```

or

```
{Aroon Up component of the Aroon Indicator}
_AroonUp := Aroon (PERIODS,1);
{end}
```

The function aroondup (14) calculates the Aroon Up component of the Aroon Indicator over the period of 14 price values.

Average Directional Movement

adx (PERIODS)

Calculates the Average Directional Movement Indicator.

The function adx (14) calculates the Average Directional Movement Indicator over the period of 14 price values.

The function adx (PERIODS) calculates the Average Directional Movement Indicator by formula:

```
{Average Directional Movement Indicator}
_ADX := Wilders ((Abs (pdi (PERIODS)- mdi (PERIODS)) / (pdi (PERIODS) +
mdi (PERIODS))) * 100, PERIODS);
{end}
```

or

```
{Average Directional Movement Indicator}
_ADX := Wilders ( dx (PERIODS), PERIODS);
{end}
```

or full formula:

```
{Average Directional Movement Indicator -> adx (periods)}
{Plus Directional Movement -> pdi (PERIODS)}
PlusDM := if (H > Ref (H,-1) AND L >= Ref (L,-1), H- Ref (H,-1), if (H >
Ref (H,-1)
AND L < Ref (L,-1) AND H-Ref (H,-1) > Ref (L,-1)-L, H- Ref (H,-1), 0));
PlusDI := 100 * Wilders (PlusDM, PERIODS) / Wilders ( ATR (1), PERIODS);
{Minus Directional Movement -> mdi (PERIODS)}
MinusDM := if (L < Ref (L,-1) AND H <= Ref (H,-1), Ref (L,-1)-L, if (H >
Ref (H,-1)
AND L < Ref (L,-1) AND H- Ref (H,-1) < Ref (L,-1)-L, Ref (L,-1) -L, 0));
MinusDI := 100 * Wilders (MinusDM, PERIODS) / Wilders ( ATR (1),
PERIODS);
{Directional Movement Index -> dx (PERIODS)}
DIDif := Abs (PlusDI - MinusDI);
DISum := PlusDI + MinusDI;
DMIndex := (DIDif / DISum) * 100;
{Average Directional Movement Indicator}
_ADX := Wilders ( DMIndex, PERIODS);
{end}
```

Average True Range

atr (PERIODS)

The function calculates the Average True Range (ATR).

The function ATR (PERIODS) calculates the Average True Range by formula:

```
{ATR -> Custom}
TH:=If (Ref (C,-1) > H, Ref (C,-1), H);
TL:=If (Ref (C,-1) < L, Ref (C,-1), L);
TR:=TH-TL;
_ATR:=Mov (TR, PERIODS, S);
```

{end}

The formula atr(20) calculates the Average True Range over the period of 20 price values.

The predefined Average True Range uses this method of calculation – Custom.

If the [Wilders](#) method of calculations is chosen, the following formula is used:

```
{ATR -> Wilders}
TH:=If (Ref (C,-1) > H, Ref (C,-1), H);
TL:=If (Ref (C,-1) < L, Ref (C,-1), L);
TR:=TH-TL;
_ATR:=Wilders (TR, PERIODS);
{end}
```

Bars Since

barssince (DATA ARRAY)

Calculates the number of bars (time periods) that have passed since DATA ARRAY was true.

The formula barssince (macd () > 0) will return the value equal to the number of bars when the [MACD](#) line was greater than 0. The value is calculated for the entire period of data loaded to the chart.

Bollinger Band Bottom

bbandbot (DATA ARRAY, PERIODS, METHOD, DEVIATIONS)

Calculates the Bottom Bollinger Band components of the [Bollinger Bands Indicator](#) of DATA ARRAY using METHOD calculation method and shifted downward DEVIATION standard deviations.

Valid methods are SIMPLE, EXPONENTIAL, WEIGHTED (these can be abbreviated as S, E, W).

The formula bbandbot (close, 20, S, 2) calculates the Bottom Bollinger Band of Close price over the period of 20 price values Simple by type and [Standard Deviation](#) equal to 2.

Bollinger Band Top

bbandtop (DATA ARRAY, PERIODS, METHOD, DEVIATIONS)

Calculates the Top Bollinger Band components of the [Bollinger Bands Indicator](#) of DATA ARRAY using METHOD calculation method and shifted downward DEVIATION standard deviations.

Valid methods are SIMPLE, EXPONENTIAL, WEIGHTED (these can be abbreviated as S, E, W).

The formula bbandtop (close, 20, S, 2) calculates the Top Bollinger Band of Close price over the period of 20 price values Simple by type and [Standard Deviation](#) equal to 2.

Bollinger Bands

bbands (DATA ARRAY, PERIODS, METHOD, DEVIATIONS, switch)

Calculates the [Bottom Bollinger Band](#) and [Top Bollinger Band](#) components of the Bollinger Bands Indicator of DATA ARRAY using METHOD calculation method and shifted downward DEVIATION [Standard Deviations](#).

Valid methods are SIMPLE, EXPONENTIAL, WEIGHTED (these can be abbreviated as S, E, W).

The function `bbands` (data array, period, metod, deviations, 0) calculates the [Top Bollinger Band](#) component of the Bollinger Bands Indicator.

```
TopBollingerBand := bbands (DATA ARRAY, PERIODS, METHOD, DEVIATIONS, 0);
```

The function `bbands` (data array, period, metod, deviations, 0) calculates the [Bottom Bollinger Band](#) component of the Bollinger Bands Indicator.

```
BottomBollingerBand := bbands (DATA ARRAY, PERIODS, METHOD, DEVIATIONS, 1);
```

The formula `bbands` (close, 20, S, 2, 0) calculates [Top Bollinger Band](#) the of Close price over the period of 20 price values Simple by type and [Standard Deviation](#) equal to 2.

The formula `bbands` (close, 20, S, 2, 1) calculates [Bottom Bollinger Band](#) the of Close price over the period of 20 price values Simple by type and [Standard Deviation](#) equal to 2.

So, the Bollinger Bands Indicator is calculated as follows:

```
{Bollinger Bands}
{The middle band is a n -period simple moving average}
MiddleBollingerBand:= mov (C, PERIODS, S);
{Calculates the Bottom Bollinger Band and Top Bollinger Band components}
TopBollingerBand := bbands (DATA ARRAY, PERIODS, METHOD, DEVIATIONS, 0);
BottomBollingerBand := bbands (DATA ARRAY, PERIODS, METHOD, DEVIATIONS, 1);
{end}
```

or the method of calculations using the [stdev\(\)](#) function to calculate the upper and lower bands:

```
{Bollinger Bands}
{The middle band is a n -period simple moving average}
MiddleBollingerBand:= mov (C, PERIODS, S);
{Calculates the Bottom Bollinger Band and Top Bollinger Band components}
TopBollingerBand := MiddleBollingerBand + ( 2 * stdev( C, PERIODS ));
BottomBollingerBand := MiddleBollingerBand - ( 2 * stdev( C, PERIODS ));
{end}
```

Ceiling

ceiling (DATA ARRAY)

Calculates the lowest integer that is greater than DATA ARRAY.

The formula `ceiling` (9.2) returns 10.

The formula `ceiling` (-9.2) returns -9.

Cosine

cos (DATA ARRAY)

Returns the Cosine of DATA ARRAY.

Assumes that the DATA ARRAY values are in degrees.

The formula `cos (0)` returns 1.

The formula `cos (Close)` returns Cosine of price Close in degrees.

Cross

cross (DATA ARRAY 1, DATA ARRAY 2)

Plots a +1 when DATA ARRAY 1 crosses above DATA ARRAY 2.

Otherwise, 0 is plotted.

If it is required to know when DATA ARRAY 1 crosses below DATA ARRAY 2, the formula `cross (DATA ARRAY 2, DATA ARRAY 1)` is used.

The formula `cross (Close, mov (Close,9,S))` returns the value equal to 1, only once and only when the Close price crosses the [Moving Average](#) plotted from Close prices with the interval at 9 and Simple by the type of averaging.

The rest of the time – if and until the next crossover in this direction (above) the formula `cross (Close, mov (Close,9,S))` returns 0.

To monitor a situation when the crossover occurs below, the formula `cross (mov (Close,9,S), Close)` should be used.

The main difference of the cross function from the comparison functions (`>`, `<`, etc.) is that the comparison functions will return 1 as long as the condition is true.

The formula `Close > mov (Close,9,S)` is true and returns 1 all the time when this condition is met. In other cases it returns 0.

Cumulate

cum (DATA ARRAY)

Calculates a Cumulative Sum of the DATA ARRAY from the first period in the chart.

The formula `cum (1)` calculates an indicator that rises one point for each day since the beginning of the chart. The formula `cum (Close)` calculates the cumulative total of all closing prices from the beginning of the chart.

Day Of Month

dayofmonth ()

Plots the day of the month.

If today was July 15th, 15 would be plotted.

Day Of Week

dayofweek ()

Plots the day of the week.

1=Monday, 2=Tuesday, 3=Wednesday,
4=Thursday, 5=Friday, 6=Saturday, 7=Sunday.

Detrended Price Oscillator

dpo (PERIODS)

Calculates the predefined Detrended Price Oscillator.

The function dpo (PERIODS) calculates the Detrended Price Oscillator by formula:

```
{Detrended Price Oscillator}
_DPO := Close - Ref ( Mov (Close, PERIODS, S), -(PERIODS / 2 + 1));
{end}
```

Directional Movement Index

dx (PERIODS)

Calculates the predefined Directional Movement Index.

The function dx (14) calculates the Directional Movement Index component of the [Average Directional Movement Indicator](#) over the period of 14 price values.

The function dx (PERIODS) calculates by formula:

```
{Directional Movement Index - > dx (PERIODS)}
DMIndex := (Abs (pdi (PERIODS) - mdi (PERIODS)) / (pdi (PERIODS) + mdi
(PERIODS)) * 100;
{ DMIndex - > dx (PERIODS) }
{end}
```

Or full formula:

```
{Directional Movement Index - > dx (PERIODS)}
{Plus Directional Movement -> pdi (PERIODS)}
PlusDM:=if (H > Ref (H,-1) AND L >= Ref (L,-1), H- Ref (H,-1), if (H >
Ref (H,-1)
AND L < Ref (L,-1) AND H-Ref (H,-1) > Ref (L,-1)-L, H- Ref (H,-1), 0));
PlusDI:=100 * Wilders (PlusDM, PERIODS) / Wilders ( ATR (1), PERIODS);
{Minus Directional Movement -> mdi (PERIODS)}
MinusDM:=if (L < Ref (L,-1) AND H <= Ref (H,-1), Ref (L,-1)-L, if (H >
Ref (H,-1)
AND L < Ref (L,-1) AND H- Ref (H,-1) < Ref (L,-1)-L, Ref (L,-1) -L, 0));
MinusDI:=100 * Wilders (MinusDM, PERIODS) / Wilders ( ATR (1), PERIODS);
```

```
{Directional Movement Index -> dx (PERIODS)}
DIDif:=Abs (PlusDI - MinusDI);
DISum:=PlusDI + MinusDI;
DMIndex := (DIDif / DISum) * 100; { DMIndex - > dx (PERIODS) }
{end}
```

Directional Movement Rating

adxr (PERIODS)

Calculates the predefined Directional Movement Rating.

The function adxr (14) calculates the Directional Movement Rating over the period of 14 price values.

The function adxr (PERIODS) calculates the Directional Movement Rating by formula:

```
{Directional Movement Rating}
_ADXR:= ( ADX (PERIODS) + Ref (ADX (PERIODS),1 - PERIODS))/2;
{end}
```

or full formula:

```
{Directional Movement Rating - > adxr (periods)}
{Plus Directional Movement -> pdi (PERIODS)}
PlusDM := if (H > Ref (H,-1) AND L >= Ref (L,-1), H- Ref (H,-1), if (H >
Ref (H,-1)
AND L < Ref (L,-1) AND H-Ref (H,-1) > Ref (L,-1)-L, H- Ref (H,-1), 0));
PlusDI := 100 * Wilders (PlusDM, PERIODS) / Wilders ( ATR (1), PERIODS);
{Minus Directional Movement -> mdi (PERIODS)}
MinusDM := if (L < Ref (L,-1) AND H <= Ref (H,-1), Ref (L,-1)-L, if (H >
Ref (H,-1)
AND L < Ref (L,-1) AND H- Ref (H,-1) < Ref (L,-1)-L, Ref (L,-1) -L, 0));
MinusDI := 100 * Wilders (MinusDM, PERIODS) / Wilders ( ATR (1),
PERIODS);
{Directional Movement Index -> dx (PERIODS)}
DIDif := Abs (PlusDI - MinusDI);
DISum := PlusDI + MinusDI;
DMIndex := (DIDif / DISum) * 100;
{Average Directional Movement Indicator}
_ADX := Wilders ( DMIndex, PERIODS);
{Directional Movement Rating}
_ADXR:= (_ADX + Ref (_ADX ,1 - PERIODS)) / 2;
{end}
```

Divergence

divergence(DATA ARRAY 1, DATA ARRAY 2, % MINIMUM CHANGE)

Plots a +1 if DATA ARRAY 1 diverges from DATA ARRAY 2 (i.e., DATA ARRAY 1 is increasing and DATA ARRAY 2 is decreasing). Plots a -1 if DATA ARRAY 1 converges from DATA ARRAY 2 (i.e., DATA ARRAY 1 is decreasing and DATA ARRAY 2 is increasing). A zero is plotted if they are

moving in the same direction. Movements in DATA ARRAY 1 less than % MINIMUM CHANGE are ignored.

The Divergence function is based on the [Zig Zag](#) formula. First, a % MINIMUM CHANGE Zig Zag is calculated for DATA ARRAY 1. Next, a Zig Zag is calculated for DATA ARRAY 2 using the % MINIMUM CHANGE required to match the number of Zig Zag segments in DATA ARRAY 1 over the data range loaded. The two Zig Zags are then compared for divergence and convergence.

The formula "divergence(close, rsi (21), 3)" looks for divergences between the close and a 21-period RSI. Movements in the close less than 3% are ignored.

Division

div (DATA ARRAY, DATA ARRAY)

Divides the first parameter by the second.

Division by zero produces zero.

The formula div (10,2) returns 5.

This formula also could be written as 10 / 2.

Exponent

exp (DATA ARRAY)

Calculates e raised to the DATA ARRAY power.

Exponent - is the same as the Exponential Function e^x .

$E = 2.7183$ and is the basis of a natural logarithm.

The formula exp (1) returns 2.7183.

This formula exp (5) returns 148.4132.

Floor

floor (DATA ARRAY)

Calculates the highest integer that is less than DATA ARRAY.

The function floor (17.8) returns 17.

The formula floor(-17.8) returns -18.

Fraction

frac (DATA ARRAY)

Eliminates the integer portion of DATA ARRAY and returns the fractional part.

The formula `frac (12.7)` returns 0.7.

The formula `frac (-12.7)` returns -0.7.

Gap Down

`gapdown ()`

Plots a +1 on the current value of the instruments's prices Gap Down.

Otherwise a 0 is plotted.

A gap down occurs if previous value Low is greater than current High.

Gap Up

`gapup ()`

Plots a +1 on the current value of the instrument's prices Gap Up.

Otherwise a 0 is plotted.

A gap up occurs if previous value High is greater than current Low.

Highest

`highest (DATA ARRAY)`

Calculates the highest value in the DATA ARRAY since the first value loaded in the chart.

The formula `highest (Close)` returns the highest closing price value since the first value loaded in the chart.

Highest Bars Ago

`highestbars (DATA ARRAY)`

Calculates the number of periods that have passed since the DATA ARRAY's highest value.

This includes all data loaded in the chart.

The formula `highestbars (close)` returns the number of periods that have passed since the closing price reached its highest peak.

Highest High Value

`hhv (DATA ARRAY, PERIODS)`

Calculates the highest value in the DATA ARRAY over the preceding PERIODS

(PERIODS includes the current value).

The formula `hhv(CLOSE, 5)` returns the highest Closing price over the preceding five periods.

The formula `hhv(HIGH, 7)` returns the highest High price over the preceding seven periods.

Highest High Value Bars Ago

`hhvbars (DATA ARRAY, PERIODS)`

Calculates the number of periods that have passed since the DATA ARRAY reached its PERIODS period peak.

The formula `hhvbars(close,55)` returns the number of periods that have passed since the closing price reached its 55-period peak.

Highest Since

`highestsince (Nth, EXPRESSION, DATA ARRAY)`

Returns the highest value of DATA ARRAY since the Nth most recent occurrence of EXPRESSION was true.

This includes all data loaded in the chart.

The formula `highestsince(2,cross(c,mov(c,13,s),close)` returns highest value of the Close since the second most recent occurrence of the Close crossing above its 13 -period Moving Average.

Highest Since Bars Ago

`highestsincebars (Nth, EXPRESSION, DATA ARRAY)`

Calculates the number of periods that have passed from the highest value of DATA ARRAY (after the Nth most recent occurrence of EXPRESSION was true). This includes all data loaded in the chart.

Put another way, this function returns the number of periods that have passed since the [highestsince\(\)](#) function returned its value.

The formula `highestsincebars(2,cross(c,mov(c,13,s),close)` returns the number of periods that have passed since the highest value of the Close (after the second most recent occurrence of the Close crossing above its 13 -period Moving Average).

Hour

`hour ()`

On an intraday chart, plots the number of hours that have passed during the day using a 24 hour clock. If the current time is recorded as 10:17:27, the `hour()` function will return 10 .

If

`if (DATA ARRAY > >= < <= <> = DATA ARRAY, THEN DATA ARRAY, ELSE DATA ARRAY)`

A conditional function that returns the second parameter (THEN) if the conditional expression defined by the first parameter is true. Otherwise, the third parameter is returned (ELSE).

The formula `if (1<2,3,5)` will always return the value 5. The formula `if (Close > mov (Close,9,S),1,0)` will be true and return 1 all of the time when the given condition is met. Otherwise it will return 0.

If the logical variable is implied that can assume only the value of 0 (False) or 1 (True), then the formula:

```
A:= if (Close > mov (Close,9,S),1,0);
```

may be written as

```
A:= Close > mov (Close,9,S);
```

where variable A may be equal only to 0 or 1.

The `if()` function is used to create conditional (i.e., "if-then") statements. It is perhaps the most used function in the formula language. It contains three parameters as shown in the following example.

```
if( close > mov(c,10,s), rsi(9), rsi(14) )
```

The above "if" statement reads (in English) as follows: If today's close is greater than today's 10-day simple moving average of the close, then plot a 9-day RSI, otherwise, plot a 14-day RSI.

The next formula plots "positive volume" if the close is greater than the median price. Otherwise, "negative volume" is plotted.

```
if( CLOSE > (HIGH+LOW)/2, +V, -V )
```

A good example of the `if()` function can be found in the On Balance Volume example.

If you simply want an expression to be evaluated as either true or false, it can be done without the use of the `if()` function. The following formula will result in either a 1 (true) or a 0 (false).

```
rsi(14) > 70
```

If the 14-period RSI is greater than 70, then this formula will evaluate to "true" and return the number 1. If it is below 70, the formula will evaluate to "false" and return the number 0. This is done without the `if()` function being used. The formula below uses the `if()` function and will return the same results, but it is longer.

```
if(rsi(14) > 70, 1, 0 )
```

Inside

`inside ()`

Plots a +1 when an Inside Value occurs.

An Inside Value occurs when current High is less than or equal to the High for the previous [Rally](#) or [Reaction](#) Value and current Low is greater than or equal to the previous Rally or Reaction Value's Low.

A range is determined by the first Inside Value and is only broken by a [Rally](#), [Reaction](#), or [Outside Value](#).

Integer

`int (DATA ARRAY)`

Removes the fractional portion of DATA ARRAY and returns the integer part.

The formula `int(12.7)` returns 12.

The formula `int(-17.8)` returns -17.

IsDefined

`isdefined (DATA ARRAY)`

Returns 1 if all data necessary to calculate the formula is available, 0 if not.

The formula `isdefined (mov (Close, 13, S))` will return a 0 if there are less than 13 periods of data loaded in the chart.

IsUndefined

`isundefined (DATA ARRAY)`

Returns 0 if all data necessary to calculate the formula is available, 1 if not.

The formula `isundefined (mov (Close, 13, S))` will return a 1 if there are less than 13 periods of data loaded in the chart.

Last Value in Data Array

`lastvalue (DATA ARRAY)`

This function loads an entire data array with the last calculated value of the specified DATA ARRAY. The result of this function can be used in place of a constant in any function argument.

If DATA ARRAY is undefined (e.g., only 100-periods loaded and the last value of a 200-period [Moving Average](#) is requested) then the lastvalue function returns zero.

Since this function loads an entire data array with the last value of another array, it allows the formula to look into the future.

This is unacceptable for most indicators, but is very beneficial for such methods as pattern recognition.

The formula `Mov (Close, lastvalue(fml("Determine Periods")), S)` calculates a Moving Average using the number of periods returned by the indicator named "Determine Periods".

Linear Regression Indicator

`linearreg (DATA ARRAY, PERIODS, SMOOTH METHOD, SMOOTH PERIODS)`

Calculates the predefined Linear Regression Indicator.

The formula `linearreg (Close,50,Simple,10)` calculates the predefined Linear Regression indicator. The Smooth Method and Smooth Periods are optional, therefore the following example is also valid `linearreg (C, 50)`.

Linear Regression Slope

`linregslope (DATA ARRAY, PERIODS)`

Calculates the predefined Linear Regression Indicator.

The formula `linregslope (c, 50)` calculates the predefined Linear Regression Indicator.

Logarithm

`log (DATA ARRAY)`

Calculates the natural logarithm of DATA ARRAY.

The formula `log (1)` will return 0.

The formula `log (12)` will return 2.4849.

Lowest

`lowest (DATA ARRAY)`

Calculates the lowest value in the DATA ARRAY since the first value loaded in the chart.

The formula `lowest (Close)` returns the lowest closing price value since the first value loaded in the chart.

Lowest Bars Ago

`lowestbars (DATA ARRAY)`

Calculates the number of periods that have passed since the DATA ARRAY's lowest value.

This includes all data loaded in the chart.

The formula `lowestbars (Close)` returns the number of periods that have passed since the closing price reached its lowest peak.

Lowest Low Value

`llv (DATA ARRAY, PERIODS)`

Calculates the lowest value in the DATA ARRAY over the preceding PERIODS (PERIODS includes the current value).

The formula llv (CLOSE, 5) returns the lowest Closing price over the preceding five periods.

The formula llv (LOW, 7) returns the lowest LOW price over the preceding seven periods.

Lowest Since

lowestsince (Nth, EXPRESSION, DATA ARRAY)

Returns the lowest value of DATA ARRAY since the Nth most recent occurrence of EXPRESSION was true. This includes all data loaded in the chart.

The formula lowestsince (2, cross(c,mov(c,13,s), close) returns lowest value of the Close since the second most recent occurrence of the Close crossing above its 13 -period Moving Average.

Lowest Since Bars Ago

lowestsincebars (Nth, EXPRESSION, DATA ARRAY)

Calculates the number of periods that have passed from the lowest value of DATA ARRAY (after the Nth most recent occurrence of EXPRESSION was true). This includes all data loaded in the chart. Put another way, this function returns the number of periods that have passed since the [lowestsince\(\)](#) function returned its value.

The formula lowestsincebars (2, cross(c,mov(c,13,s), close) returns the number of periods that have passed since the lowest value of the Close (after the second most recent occurrence of the Close crossing above its 13 -period Moving Average).

MACD

macd ()

Calculates the predefined MACD (Moving Average Convergence Divergence) Indicator.

The formula macd () returns the value of the MACD indicator (the value of the MACD's fast line).

The formula mov (macd (),9,E) returns the value of the MACD's signal line.

The function macd () calculates the MACD indicator by formula:

```
{ macd ( ) }
_MACDFastLine := Mov (Close,12,E) - Mov (Close,26,E);
```

The function mov (macd (),9,E) calculates the MACD's signal line by formula:

```
{ mov (macd ( ),9,E) }
_MACDSignalLine := Mov ((Mov (Close,12,E) - Mov (Close,26,E)),9,E);
```

In order to use other parameters for calculations of an indicator it is recommended to follow the template:

```
{my MACD}
{MACD - Fast Line}
```

```

_MACDFastLine := Mov (DATA ARRAY, PERIODS_1, TYPE Moving)
- Mov (DATA ARRAY, PERIODS_2, TYPE Moving);
{MACD - Signal Line}
_MACDSignalLine := Mov (_MACDFastLine, PERIODS_3, TYPE Moving);
{end}

```

Maximum

max (DATA ARRAY, DATA ARRAY)

Returns the largest of the two parameters.

The formula `max (CLOSE, Mov (Close,13,E))` returns either the Closing Price or `Mov (Close,13,E)`, whichever is greater.

The formula `max(-17, 13)` always returns 13.

Median Price

mp()

Calculates the predefined Median Price indicator.

The Median Price Indicator is calculated by adding the high price and the low price together, and then dividing by two. The result is the average, or median, price.

The function `mp()` calculates the Median Price Indicator by formula:

```

{MP() - Median Price Indicator}
_MedianPrice:= (HIGH + LOW) / 2;
{end}

```

Midpoint

mid (DATA ARRAY, PERIODS)

Returns the midpoint of the DATA ARRAY over the specified time PERIOD. The midpoint is the value halfway between the highest and lowest DATA ARRAY values during the specified PERIOD.

The formula `mid (CLOSE, 7)` is equivalent to `llv (C,7) + ((hmv (C,7) - llv (C,7)) / 2)`.

The function `mid (DATA ARRAY, PERIODS)` calculates by formula:

```

{Midpoint}
_Midpoint := llv (DATA ARRAY, PERIODS) + ((hmv (DATA ARRAY, PERIODS)
- llv (DATA ARRAY, PERIODS)) / 2);
{end}

```

Minimum

`min (DATA ARRAY, DATA ARRAY)`

Returns the smallest of the two parameters.

The formula `min (CLOSE, Mov (Close,13,E))` returns either the Closing Price or `Mov (Close,13,E)`, whichever is less.

The formula `max(-17, 13)` always returns -17.

Minus Directional Movement

`mdi (PERIODS)`

Calculates the predefined Minus Directional Movement Indicator component of the [Average Directional Movement Indicator](#).

The function `mdi (14)` calculates the Minus Directional Movement Indicator component of the Average Directional Movement Indicator over the period of the 14 price values.

The function `mdi (PERIODS)` calculates by formula:

```
{Minus Directional Movement Indicator -> mdi (PERIODS)}
{Minus Directional Movement}
MinusDM := if (L < Ref (L,-1) AND H <= Ref (H,-1), Ref (L,-1)-L, if (H >
Ref (H,-1)
AND L < Ref (L,-1) AND H- Ref (H,-1) < Ref (L,-1)-L, Ref (L,-1) -L, 0));
{Minus Directional Movement Indicator}
MinusDI := 100 * Wilders (MinusDM, PERIODS) / Wilders (ATR (1), PERIODS);
{end}
```

Minute

`minute ()`

On an intraday chart, plots the number of minutes that have passed during the current hour. If the current time is recorded as 10:17:22, the `minute()` function will return 17.

Modulus

`mod (DATA ARRAY_1, DATA ARRAY_2)`

Calculates the remainder (i.e., the fractional portion) of DATA ARRAY divided by DATA ARRAY. A division by zero produces a zero result.

The formula `mod (10, 3)` returns 1.0.

The formula `mod (-10.7, 3)` returns -1.7.

The same can be written as `-10.7 - (int (-10.7 / 3) * 3)`.

The function `mod (DATA ARRAY_1, DATA ARRAY_2)` calculates by formula:

```
{Modulus}
_mod := DATA ARRAY_1 - (int (DATA ARRAY_1 / DATA ARRAY_2) * DATA
ARRAY_2);
{end}
```

Momentum

mo (DATA ARRAY, PERIODS)

Calculates the predefined Momentum Indicator.

The DATA ARRAY argument is optional and will use CLOSE as the default if it is not included.

The function mo (Close,12) calculates the Momentum Indicator over the period of the 12 price values.

The function mo (Close,12) is equivalent to (Close / ref (Close, -12)) * 100.

The function mo (DATA ARRAY, PERIODS) calculates the Momentum Indicator by formula:

```
{Momentum Indicator -> mo (DATA ARRAY, PERIODS)}
_mo := (DATA ARRAY/ ref( DATA ARRAY, - PERIODS )) * 100;
{end}
```

Month

month ()

Plots the month of the year for the price.

If a bar was plotted on 04/01/2003 (01 April 2003), 04 would be plotted.

Moving Average

mov (DATA ARRAY, PERIODS, METHOD)

Calculates a PERIODS moving average of DATA ARRAY using METHOD calculation method.

Valid methods are EXPONENTIAL, SIMPLE, WEIGHTED (these can be abbreviated as E, S, W).

The formula mov(CLOSE, 25, EXPONENTIAL) returns the value of a 25-period Exponential Moving Average of the Closing prices.

The formula mov(CLOSE, 25, EXPONENTIAL) is equivalent to mov(C, 25, E).

Multiplication

mul (DATA ARRAY, DATA ARRAY)

Calculates DATA ARRAY multiplied by DATA ARRAY.

The function mul (CLOSE, 2) returns the closing price multiplied by two.

This function also could be written as $C * 2$.

Negative

neg (DATA ARRAY)

Calculates the negative of DATA ARRAY.

The formula `neg(10)` returns -10.

The formula `neg(-17)` returns +17.

This formula also could be written `-(-12)`.

Outside

outside ()

Plots a +1 when an Outside Value occurs.

An Outside Value occurs when current High is greater than the High for the previous Rally or Reaction Value and current Low is less than the previous Rally or Reaction Value's Low.

A range is determined by the first Outside Value and is only broken by a Rally, Reaction, or [Inside Value](#).

Parabolic SAR

sar (STEP, MAXIMUM)

Calculates the predefined Parabolic SAR Indicator.

STEP

As the price makes new Highs/Lows, the Parabolic SAR will rise/fall according to the size.

If the trade makes new Highs for three bars, then the SAR Step increases by 0.02 each bar (i.e., 0.02 to 0.04 to 0.06, etc.).

The SAR Step size of 0.02 is recommended in most cases.

MAXIMUM

The Maximum value the SAR Step can obtain.

The SAR Maximum of 0.20 is recommended in most cases.

The function `SAR (0.02, 0.20)` calculates the Parabolic SAR Indicator with `STEP=0.02` and `MAXIMUM=0.20`.

Peak Bars Ago

peakbars(Nth, DATA ARRAY, % MINIMUM CHANGE)

Plots the number of bars that have passed from the Nth peak. This uses the [Zig Zag](#) function (see Zig Zag) to determine the peaks. N=1 would return the number of bars that have passed since the most recent peak. N=2 would return the number of bars that have passed since the 2nd most recent peak. Etc.

```
peakbars(1,close,5)
```

Peak Value

peak(Nth, DATA ARRAY, % MINIMUM CHANGE)

Plots the value of DATA ARRAY Nth peak(s) ago. This uses the Zig Zag function (see Zig Zag) to determine the peaks. N=1 would return the value of the most recent peak. N=2 would return the value of the 2nd most recent peak. Etc.

```
peak(1,close,5)
```

Performance

per (DATA ARRAY)

Returns the predefined Performance Indicator.

The DATA ARRAY argument is optional and will use CLOSE as the default if it is not included.

The numeric value of the Performance Indicator is the percentage that the price has changed since the first period loaded. A value of 12 would mean that the price has increased 12% since the first value loaded in the chart.

Similarly, a value of -12% would mean that the price has fallen by 12% since the first value.

It recommends to use this pattern to calculate the percentage that the price has change since the certain period but not the first one loaded :

```
{the percentage that the chart value has changed since certain period}
_per := (DATA ARRAY/ ref( DATA ARRAY, - PERIODS )) * 100 - 100;
{end}
```

is equivalent to

```
{the percentage that the chart value has changed since certain period}
_per := mo (DATA ARRAY, PERIODS) - 100;
{end}
```

Or

```
{the percentage that the chart value has changed since certain period}
_per := (DATA ARRAY - ref( DATA ARRAY, - PERIODS )) / DATA ARRAY * 100;
{end}
```

Plus Directional Movement

pdi (PERIODS)

Calculates the predefined Plus Directional Movement Indicator component of the [Average Directional Movement Indicator](#).

The function pdi (14) calculates the Plus Directional Movement Indicator component of the Average Directional Movement Indicator over the period of the 14 price values.

The function pdi (PERIODS) calculates by formula:

```
{Plus Directional Movement Indicator -> pdi (PERIODS)}
{Plus Directional Movement}
PlusDM := if (H > Ref (H,-1) AND L >= Ref (L,-1), H- Ref (H,-1), if (H >
Ref (H,-1)
AND L < Ref (L,-1) AND H-Ref (H,-1) > Ref (L,-1)-L, H- Ref (H,-1), 0));
{Plus Directional Movement Indicator}
PlusDI := 100 * Wilders ( PlusDM, PERIODS) / Wilders ( ATR (1), PERIODS);
{end}
```

Power

power (DATA ARRAY, POWER)

Calculates DATA ARRAY raised to the POWER power.

A negative DATA ARRAY value raised to a non-integer POWER causes an error.

The formula power (10, 3) returns 1000.

Precision

prec (DATA ARRAY, PRECISION)

Truncates DATA ARRAY to PRECISION decimal places.

The formula prec (10.56789, 2) returns 10.560.

The formula prec (10.56789, 3) returns 10.5670.

The formula prec (10.56789, 4) returns 10.56780.

Small binary rounding errors may cause some minor distortion in the decimal portion of any number stored in a computer.

Prev

Prev

The PREV constant allows you to create self-referencing formulas.

A self referencing formula is one that is able to reference the "previous" period's value of itself.

For example, the following is an example of a self referencing formula:

```
((H+L+C)/3) + PREV
```

This simple formula divides the high, low, and closing prices by 3 and then adds this value to yesterday's value of the $((H+L+C)/3)$.

The calculation of the popular indicator On Balance Volume illustrates the use of the PREV function.

```
(if(c>ref(c,-1),1,-1)*volume)+PREV
```

Although On Balance Volume can be calculated without the use of the PREV function, an exponential moving average cannot (other than using the mov() function).

The following formula shows how a 18% exponential moving average (approximately 10-periods) is calculated using the PREV function.

```
(close*0.18)+(PREV*0.82)
```

You are using the PREV variable in a location within a custom indicator that will cause the calculation to be very slow.

You should normally try to avoid using the PREV variable as a DATA ARRAY when there is also a PERIODS specified (e.g., mov(prev, 20, s), hhv(prev, 30), etc.).

Although use of the PREV function in this way is allowable, it will take a long time to calculate, particularly if a lot of data is loaded in the chart.

Price Oscillator

oscp (PERIODS, PERIODS, MA_METHOD, DIFF_METHOD)

Calculates the PERIODS/PERIODS predefined Price Oscillator indicator calculated using the MA_METHOD [Moving Average](#) method expressed in DIFF_METHOD.

Valid MA_METHODs are SIMPLE, EXPONENTIAL, WEIGHTED (these can be abbreviated as S, E, W).

Valid DIFF_METHODs are PERCENT and POINTS (these can be abbreviated as % and \$).

The formula oscp(12, 26, E, \$) returns a 12-period/26-period exponential price oscillator expressed in points.

The following formula calculates a 12-period / 26-period exponential Price Oscillator expressed in points:

```
{12-period / 26-period exponential Price Oscillator expressed in points}
_OSCP := mov( close, 12, E) - mov( close, 26, E);
{end}
```

The following formula calculates a 12-period/26-period exponential Price Oscillator expressed in percent:

```
{12-period / 26-period exponential Price Oscillator expressed in percent}
_OSCP := ((mov( close, 12, E) - mov( close, 26, E)) / mov( close, 26, E))
* 100;
{end}
```

Projection Band Bottom

`Projbandbot(PERIODS)`

Calculates the bottom Projection Band.

`projbandbot(21)`

Projection Band Top

`Projbandtop(PERIODS)`

Calculates the top Projection Band.

`projbandtop(21)`

Rally

`rally ()`

Plots a +1 when a Rally value occurs. Otherwise, a 0 is plotted.

A Rally value occurs when current High is greater than the previous Rally or [Reaction](#) value's High and current Low is greater than or equal to the previous Rally or Reaction value's Low.

Rally With Volume

`rallywithvol()`

Plots a "+1" when a rally with volume day occurs. Otherwise, a "0" is plotted. A rally with volume occurs when today's high is greater than the previous rally or reaction day's high and today's low is greater than or equal to the previous rally or reaction day's low. Today's volume must be greater than the previous rally or reaction day's volume.

Rate of Change

`roc (DATA ARRAY, PERIODS, DIFF_METHOD)`

Calculates the PERIODS Rate-Of-Change of DATA ARRAY expressed as DIFF_METHOD.

Valid DIFF_METHODs are PERCENT and POINTS (these can be abbreviated as % and \$).

The formula `roc (CLOSE, 12, PERCENT)` returns the 12-period percent Rate-Of-Change of the Closing prices.

The function `roc (DATA ARRAY, PERIODS, PERCENT)` calculates the Rate-Of-Change Indicator by formula:

```
{Rate-Of-Change -> roc (DATA ARRAY, PERIODS, PERCENT)}
_ROC := ((DATA ARRAY - ref( DATA ARRAY, - PERIODS )) / ref( DATA ARRAY, -
PERIODS )) * 100;
{end}
```

The function roc (DATA ARRAY, PERIODS, POINTS) calculates the Rate-Of-Change Indicator by formula:

```
{Rate-Of-Change -> roc (DATA ARRAY, PERIODS, POINTS)}  
_ROC := DATA ARRAY - ref (DATA ARRAY, - PERIODS);  
{end}
```

Reaction

reaction ()

Plots a +1 when a Reaction value occurs. Otherwise, a 0 is plotted.

A Reaction value occurs when current high is less than or equal to the previous [Rally](#) or Reaction values's High and current Low is less than the previous [Rally](#) or Reaction value's Low.

Reaction With Volume

reactionwithvol()

Plots a "+1" when a reaction day occurs. Otherwise, a "0" is plotted. A reaction day occurs when today's high is less than or equal to the previous rally or reaction day's high and today's low is less than the previous rally or reaction day's low. Today's volume must be greater than the previous rally or reaction day's volume.

Reference

ref (DATA ARRAY, PERIODS)

References a previous or subsequent element in a DATA ARRAY.

A positive PERIOD references N periods in the future.

A negative PERIOD references N periods ago.

The formula ref (Close, -12) returns the Close price 12 periods ago.

Thus, the 12-period price [Rate-Of-Change](#) (expressed in points) can be written as Close - ref (Close, -12).

The formula ref (C, +12) returns the Close price 12 periods ahead.

Relative Strength Index

rsi (DATA ARRAY, PERIODS)

Calculates the predefined RSI Indicator.

The formula RSI (Close, 14) calculates the RSI Indicator of the 14-periods Close price .

The formula RSI(14) calculates the RSI Indicator of the 14-periods Close price.

The DATA ARRAY argument is optional and will use Close as the default if it is not included.

The formula RSI (Low, 14) calculates the RSI Indicator of the 14-periods Low price .

The function RSI (DATA ARRAY, PERIODS) calculates the RSI Indicator by formula:

```
{Relative Strength Index -> RSI (DATA ARRAY, PERIODS)}
rsi_r := (DATA ARRAY - Ref (DATA ARRAY,-1));
rsi_rs := Wilders ( if (rsi_r > 0, rsi_r, 0), PERIODS) / Wilders ( if
(rsi_r < 0, Abs(rsi_r), 0), PERIODS);
_RSI:= 100 - (100 / (1+rsi_rs));
{end}
```

Round

round (DATA ARRAY)

Rounds DATA ARRAY to the nearest integer.

The formula round (+12.5) returns +13.

The formula round (-12.5) returns -13.

The formula round (-12.4) returns -12.

Sine

sin (DATA ARRAY)

Returns the sine of DATA ARRAY.

This function assumes that the DATA ARRAY values are in degrees.

To plot a sine wave use the formula sin (cum(7)).

Increasing the value in this formula (i.e., 7) will increase the frequency of the sine wave.

Square Root

sqrt (DATA ARRAY)

Calculates the Square Root of DATA ARRAY.

The Square Root of a negative number always returns a zero result.

The formula sqrt (16) returns 4.

Standard Deviation

stdev (DATA ARRAY, PERIODS)

Calculates the predefined Standard Deviation.

The formula `stdev (Close, 21)` calculates Standard Deviation over the period 21 of the Close prices. A 4-period Standard Deviation Indicator can be written as follows.

The first statement assigns a 4-period [Simple Moving Average](#) to the variable named `4PeriodMA`.

The second statement sums the squares of the differences between the Moving Average and the closing prices on each of the preceding four periods.

It then calculates the square root of this total.

```
{A 4-period Standard Deviation Indicator}
4PeriodMA:= Mov( CLOSE, 4, S );
_StDev := Sqrt (( Power (4PeriodMA - C, 2) +Power ((4PeriodMA)- Ref (C,-
1), 2)
+ Power ((4PeriodMA)- Ref (C,-2), 2) + Power ((4PeriodMA)- Ref (C,-3),
2) ) / 4 );
{and}
```

An easier method is to take the square root of the variance function as shown below:

```
{A 4-period Standard Deviation Indicator}
_StDev := sqrt (var( close, 4 ));
{and}
```

The easiest way is to write the Standard Deviation formula using the predefined `stdev()` function

Standard Error

ste (DATA ARRAY, PERIODS)

Calculates the predefined Standard Error Indicator.

The formula `ste (Close, 21)` calculates Standard Error Indicator over the period 21 of the Close prices.

Standard Error Band Bottom

stebandbot (DATA ARRAY, PERIODS, ERRORS, SMOOTH METHOD, SMOOTH PERIODS)

Calculates the bottom Standard Error Band of DATA ARRAY shifted downward ERRORS standard errors.

The formula `stebandbot (Close, 21, 2, Simple, 10)` calculates the bottom Standard Error Band over the period 21 of Close shifted downward 2 standard errors.

The Smooth Method and Smooth Periods are optional, therefore the following example is also valid `stebandbot(close, 21, 2)`.

The Smooth Method may be any of the Moving Average types used in the [Moving Average](#) function.

Standard Error Band Top

stebandtop (DATA ARRAY, PERIODS, ERRORS, SMOOTH METHOD, SMOOTH PERIODS)

Calculates the bottom Standard Error Band of DATA ARRAY shifted upward ERRORS standard errors.

The formula stebandtop (Close, 21, 2, Simple, 10) calculates the bottom Standard Error Band over the period 21 of Close shifted upward 2 standard errors.

The Smooth Method and Smooth Periods are optional, therefore the following example is also valid stebandtop (close, 21, 2).

The Smooth Method may be any of the Moving Average types used in the [Moving Average](#) function.

Stochastic Momentum Index

stochmomentum (PERIODS, SMOOTHING, DOUBLE SMOOTHING)

Calculates the predefined Stochastic Momentum Index.

The formula stochmomentum (PERIODS, SMOOTHING, DOUBLE SMOOTHING) calculates the Stochastic Momentum Index by formula:

```
{Stochastic Momentum Index}
_StochMomentum := 100 * (Mov ( Mov (Close - (0.5 * ( hhv (High, PERIODS)
+ llv (Low, PERIODS))), SMOOTHING, E), DOUBLE SMOOTHING,E) / (0.5*Mov
(Mov (hhv (High, PERIODS) - llv (Low, PERIODS), SMOOTHING, E), DOUBLE
SMOOTHING, E)));
{end}
```

The following formula calculates a stochmomentum (13,25,2) Stochastic Momentum Index:

```
{Stochastic Momentum Index -> stochmomentum (13,25,2)}
_StochMomentum := 100 * ( Mov (Mov (C - (0.5 * (hhv (H,13) + llv
(L,13))),25,E), 2, E) / (0.5 * Mov (Mov (hhv (H,13) - llv (L,13), 25, E),
2, E)));
{end}
```

Stochastic Oscillator

stoch (%K PERIODS, %K SLOWING)

Calculates the predefined Stochastic Oscillator.

The formula stoch (5, 3) returns the value of a 5-period %K slowed 3-periods.

The following formula calculates a 5-period %K Stochastic Oscillator with 3-period slowing:

```
{%K Stochastic Oscillator}
_K_Stoch := (sum (C - llv (L,5), 3 ) / sum (hhv(H,5) - llv (L,5), 3) ) *
100;
{end}
```

This next formula calculates a 3-period %D of the %K in the preceding formula.

```
{%D Stochastic Oscillator}
_D_Stoch := Mov ((sum (C - llv (L, 5), 3 ) / sum(hhv (H, 5) - llv (L, 5),
3) ) * 100), 3, S);
{end}
```

Or

```
{%D Stochastic Oscillator}
_D_Stoch := Mov (stoch (5, 3), 3, S);
{end}
```

In order to use other parameters for calculations of an indicator it is recommended to follow the template.

The function `stoch (%K PERIODS, %K SLOWING)` calculates the %K Stochastic Oscillator by formula:

```
{%K Stochastic Oscillator}
_K_Stoch := (sum (Close - llv (L,%K PERIODS), %K SLOWING)
/ sum (hhv(H,%K PERIODS) - llv (L,%K PERIODS), %K SLOWING)) * 100;
{end}
```

This next formula calculates a N-period %D of the %K in the preceding formula `stoch (%K PERIODS, %K SLOWING)`.

```
{%D Stochastic Oscillator}
_D_Stoch := Mov (sum (Close - llv (L,%K PERIODS), %K SLOWING)
/ sum (hhv(H,%K PERIODS) - llv (L,%K PERIODS), %K SLOWING)) * 100), %D
PERIODS, S);
{end}
```

Or

```
{%D Stochastic Oscillator}
_D_Stoch := Mov (stoch (%K PERIODS, %K SLOWING),%D PERIODS,S);
{end}
```

Subtraction

sub (DATA ARRAY, DATA ARRAY)

Calculates DATA ARRAY minus DATA ARRAY.

The formula `sub (10, 2)` returns eight.

This formula also could be written as `10 - 2`.

Summation

sum (DATA ARRAY, PERIODS)

Calculates a Cumulative Sum of the DATA ARRAY for the specified number of lookback PERIODs (including today).

The formula `sum (CLOSE, 12)` returns the sum of the preceding 12 closing prices.

A 12-period [Simple Moving Average](#) could be written `sum (C,12) / 12`.

Tick

`tick ()`

Plots the number of ticks that have come in during the current minute.

If the current tick is recorded as 12:17:37, 37 represents the tick count in the 17th minute of the 12th hour.

At the start of the 18th minute, the tick count will reset to 0.

Note that this function only works on charts with an intraday interval set to 0 (i.e., tick charts).

When plotted on tick charts, the value will range from 0 to 999—meaning up to 999 ticks can be recorded in one minute.

Using this function on bar charts (e.g., 1-minute, 5-minute, etc) will result in a value of zero.

Trough Bars Ago

`troughbars(Nth, DATA ARRAY, % MINIMUM CHANGE)`

Plots the number of bars that have passed from the Nth trough. This uses the Zig Zag function (see Zig Zag) to determine the troughs. If Nth is 1, then this will return the number of bars that have passed since the most recent trough. If Nth is 2, this will return the number of bars that have passed since the 2nd most recent trough. Etc.

```
troughbars(1,close,5)
```

Trough Value

`trough(Nth, DATA ARRAY, % MINIMUM CHANGE)`

Plots the value of DATA ARRAY Nth trough(s) ago. This uses the Zig Zag function (see Zig Zag) to determine the troughs. N=1 would return the value of the most recent trough. N=2 would return the value of the 2nd most recent trough. Etc.

```
trough( 1,close,5 )
```

Typical Price

`typical ()`

Calculates the predefined Typical Price Indicator.

The formula `typical ()` is defined as the $(High + Low + Close) / 3$.

```
{Typical Price Indicator}
_typical := (High + Low + Close) / 3;
{end}
```

Ultimate Oscillator

ult (CYCLE1, CYCLE2, CYCLE3)

Calculates the predefined Ultimate Oscillator Indicator using the three cycle lengths supplied as parameters. Note that each of the three parameters must be greater than the preceding parameter or an error message will be displayed (e.g., ult(5, 5, 5) is not valid).

The formula ult(7, 14, 21) returns the default Ultimate Oscillator.

The function ult (CYCLE1, CYCLE2, CYCLE3) calculates the Ultimate Oscillator Indicator by formula:

```
{Ultimate Oscillator Indicator}
_Ult :=((((Sum ((Max((C - L) , (C - Ref (L,-1))))),CYCLE1)) / (Sum
((Max(Max((H - L) ,H - Ref(L ,-1)) ,Max ((Ref(H ,-1) - L) , (Ref (H ,-1)
- Ref (L,-1)))))) ,CYCLE1))) *4)
+ (((Sum ((Max((C - L) , (C - Ref (L ,-1) ))),CYCLE2)) / (Sum ((Max (Max
((H - L) ,H - Ref (L ,-1)) ,Max ((Ref (H ,-1) - L) , (Ref (H ,-1) - Ref
(L,-1)))))) ,CYCLE2))) *2)
+ ((Sum ((Max (( C - L) , (C - Ref (L ,-1) ))),CYCLE3)) / (Sum ((Max
(Max(( H - L) , H - Ref ( L ,-1)) ,Max ((Ref (H ,-1) - L) , (Ref (H ,-1)
- Ref (L,-1)))))) ,CYCLE3))))/ CYCLE1) * 100;
{and}
```

Value When

valuewhen (Nth, EXPRESSION, DATA ARRAY)

Returns the value of the *DATA ARRAY* when the *EXPRESSION* was true on the *Nth* most recent occurrence.

This includes all data loaded in the chart.

The formula *valuewhen (7, cross (c,mov (c,13,s), rsi (14))* returns the value of the [RSI](#) on the 2nd most recent occurrence of the closing price crossing above its 13-period [Moving Average](#).

Variance

var (DATA ARRAY, PERIODS)

Calculates the statistical variance of DATA ARRAY over the specified time PERIOD.

The formula var (CLOSE, 20) calculates the statistical variance of Close over the specified time 20-period.

Vertical Horizontal Filter

vhf (DATA ARRAY, PERIODS)

Calculates the predefined Vertical Horizontal Filter of DATA ARRAY over the specified time PERIOD.

The formula vhf (CLOSE, 28) calculates Vertical Horizontal Filter of Close over the specified time 28 -period.

Volatility, Chaikin's

vol ([MA PERIODS](#), [ROC PERIODS](#))

Calculates the predefined Chaikin's Volatility Indicator.

The function vol ([MA PERIODS](#), [ROC PERIODS](#)) calculates the Chaikin's Volatility Indicator by formula:

```
{Chaikin's Volatility Indicator}
_Vol := ((Mov (High-Low, MA PERIODS, E) - REF (Mov (High-Low, MA PERIODS, E), ROC PERIODS)) / REF (Mov (High-Low, MA PERIODS, E), ROC PERIODS))
*100;
{end}
```

Or

```
{Chaikin's Volatility Indicator}
_MAPeriod := Mov (High-Low, MA PERIODS, E);
_Vol := ROC (_MAPeriod, ROC PERIODS, %);
{end}
```

Weighted Close

wc ()

Calculates the predefined Weighted Close Indicator.

The Weighted Close Indicator is calculated by multiplying the Close by two, adding the High and the Low, and dividing by four.

The result is the average price for the value with extra weight given to the closing price.

The function wc () calculates the Weighted Close Indicator by formula:

```
{Weighted Close Indicator}
_WC := ((Close * 2) + High + Low) / 4;
{end}
```

Wilder's Smoothing

wilders ([DATA ARRAY](#), [PERIODS](#))

Calculates the predefined Wilder's Smoothing Indicator.

The function Wilders ([DATA ARRAY](#), [PERIODS](#)) calculates the Wilder's Smoothing Indicator by formula:

```
{Wilder's Smoothing Indicator}
_Wilders := Mov ( DATA ARRAY, (PERIODS * 2) - 1, E);
{end}
```

The formula Wilders (Close, 8) is equivalent to Mov (Close, 8 * 2 - 1, E) or Mov (Close, 15, E).

Williams' %R

willr (%R PERIODS)

Calculates the predefined Williams' %R Indicator.

The formula used to calculate Williams' %R is similar to the [Stochastic Oscillator](#):

The formula willr (%R PERIODS) calculates the Williams' %R Indicator by formula:

```
{Williams' %R Indicator}
_Willr := ((hhv (H,%R PERIODS)- Close ) / (hhv (H,%R PERIODS) - llv (L,%R
PERIODS))) * - 100;
{end}
```

Notice that the formula is inverted by multiplying it by -100.

Williams' %R is plotted on an upside down scale with 0 at the top and 100 at the bottom.

To show the indicator in this upside down fashion.

Williams' A/D

willa ()

Calculates the predefined Williams A/D Indicator.

The formula willa () calculates the Williams A/D Indicator by formula:

```
{Williams A/D Indicator}
{true range of the high to a variable}
TrueRangeHigh := max (ref (Close,-1), High);
{true range of the low to a variable}
TrueRangeLow := min (ref (Close,-1), Low);
{calculates Williams' A/D Indicator}
_WillAD := cum (if (Close > ref (Close,-1), Close - TrueRangeLow,
if (Close < ref(Close,-1), Close - TrueRangeHigh,0)));
{end}
```

Year

year ()

Plots the year.

If a bar was plotted on 04/01/2003 (01 April 2003), the function will return 2003.

Zig Zag

`zig(DATA ARRAY, MINIMUM CHANGE, DIFF_METHOD)`

Calculates the MINIMUM CHANGE predefined Zig Zag indicator of DATA ARRAY using the DIFF_METHOD method of calculation.

Valid DIFF_METHODs are PERCENT and POINTS (these can be abbreviated as % and \$).

`zig(CLOSE, 5, PERCENT)`