

# **Developer KIT**



**ProBuilder**  
2004 edition

**1.03e**



# Synopsis

|  |           |
|--|-----------|
| <b>ProBuilder Presentation.....</b>          | <b>2</b>  |
| <b>Terms of the language.....</b>            | <b>3</b>  |
| Price Terms.....                             | 3         |
| Intraday.....                                | 4         |
| Derivated Terms of Prices.....               | 5         |
| Temporal Terms.....                          | 6         |
| Undefined Term.....                          | 9         |
| <b>Expressions and operators.....</b>        | <b>10</b> |
| Access to bars of the past.....              | 10        |
| Binary arithmetic operators.....             | 13        |
| Unary arithmetic operators.....              | 15        |
| Comparison operators.....                    | 20        |
| Logical operators.....                       | 23        |
| <b>Language structure, instructions.....</b> | <b>25</b> |
| Brief description.....                       | 25        |
| Insertion of comments.....                   | 26        |
| Variables, parameters and definitions.....   | 27        |
| Control structures.....                      | 28        |
| Call of a user function.....                 | 34        |
| Return of a user function.....               | 35        |
| <b>Functions Library .....</b>               | <b>36</b> |
| Moving averages.....                         | 36        |
| ProBuilder Functions.....                    | 45        |
| ProRealTime technical indicators list.....   | 49        |

## ProBuilder Presentation

ProBuilder is the programming language of ProRealTime. It allows you to create customized indicators and will help you to create trading strategies or to customize your own alerts.

ProBuilder is very similar to BASIC programming language, very easy to handle and offers quotes provided by ProRealTime. The available quotes are:

- ▶ Opening price of each bar
- ▶ Closing price of each bar
- ▶ Highest price of each bar
- ▶ Lowest price of each bar
- ▶ Volume of each bar

Bars are the same that are displayed on your workstation.

Customized indicators are displayed as those provided by ProRealTime and can be modified easily. You can create as many indicators as needed.

Once you created a code the ProBuilder parser takes every bar in account ( from the 1st one to the latest one) and displays the value of your indicator for each bar. The indicator will be updated in real time.

## Terms of the language

### Price Terms

#### *Open, High, Low, Close, Volume*

Represents respectively :

- ▶ **Open**                      Opening price of the current bar
- ▶ **High**                      Highest price of the current bar
- ▶ **Low**                        Lowest price of the current bar
- ▶ **Close**                      Closing price of the current bar
- ▶ **OpenOfNextBar**        Opening price of the next bar
- ▶ **Volume**                    Volume of the current bar

#### *Description*

This is the basic information of technical analysis.

You will be able to combine them in order to emphasize on certain aspects of the information provided by financial markets.

#### **Example open, high, low, close, volume**

REM Easy arithmetic calculation

myIndicator = **open** + **high** + **low**

myIndicator = **volume** \* (myIndicator - **close**)

REM returns 'myIndicator' as output of the function

RETURN myIndicator

## Intraday

### *Dopen(), Dhigh(), Dlow(), Dclose()*

These codes allow you to access daily prices of the current bar.

This is only useful when using intraday bars or to define trading systems (see ProBackTest).

### *Description*

- ▶ **Dopen(n)**      open of the n<sup>th</sup> previous day from the current bar
- ▶ **Dhigh(n)**      high of the n<sup>th</sup> previous day from the current bar
- ▶ **Dlow(n)**        low of the n<sup>th</sup> previous day from the current bar
- ▶ **Dclose(n)**     close of the n<sup>th</sup> previous day from the current bar

Day-traders know the importance of the close of the day before and of the open that is a moment of emotion when the novices enter or exit the market.

The high and low of the previous days can indicate the price changes of the next day.

### **Example Dopen(), Dhigh(), Dlow(), Dclose()**

```
REM Intraday oscillator
```

```
REM Possible range of the current bar determined by daily considerations
```

```
IDmediane = (Dhigh(1) + Dlow(1)) / 2
```

```
IDrange = (Dhigh(1) - Dlow(1)) / 2
```

```
REM Build an indicator based on this possible range
```

```
IDoscillateur = (close - IDmediane) *100 / IDrange
```

```
RETURN IDoscillateur
```

## Derivated Terms of Prices

### *Range, TypicalPrice, WeightedClose, MedianPrice, TotalPrice*

- ▶ **Range**                      difference between **high** and **low**
- ▶ **TypicalPrice**              average between **high**, **low** and **close**
- ▶ **WeigthedClose**            weigthed average between **high** (weight 1) **low** (weight 1) and **close**(weight 2)
- ▶ **MedianPrice**              average between **high** and **low**
- ▶ **TotalPrice**                average between **open**, **high**, **low** and **close**

### *Description*

These terms emphasize on some aspects of financial market psychology shown on the current bar.

The *Range* shows the volatility of the current bar which is an estimation of how nervous investors are.

The *WeightedClose* focuses on the importance of the closing price bar (even more important when applied on daily bars or weekly bars).

The terms *TypicalPrice* and *TotalPrice* emphasize on intraday financial market psychology since they take 3 or 4 predominant prices of the current bar into account.

*MedianPrice* is using the Medians concept instead of the Averages concept which is quite useful when trying to receive theoretical models that don't take investors psychology into account.

### **Example Range, TypicalPrice, WeightedClose,MedianPrice,TotalPrice**

REM Indicator that emphasizes on investors emotivity

InvestorsEmotivity = **Range** \* Abs(**WeightedClose** - **MedianPrice**)

REM returns 'InvestorsEmotivity' as output of the function

RETURN InvestorsEmotivity

## Temporal Terms

### *Time, Date*

Hour and date of the close of the current bar.

### *Description*

- ▶ **Time**            Hour HHMMSS
- ▶ **Date**            Date YYYYMMDD

These functions allow you to detect the changes between to bars, or to test a particular date. In the following you will learn how to extract some parts of date and time (hour, minutes, day of week...).

### **Example Time**

REM Detecting 14 o'clock (local hour)

RETURN **Time** = 140000

### **Example 1 Date**

REM Détecting the 11.09.2001

RETURN **Date** = 20010911

### **Example 2 Date**

REM Detecting the changes of day

RETURN **Date** > **Date**[1]

*Minute, Hour,*

*DayOfWeek, Day, Month, Year*

These functions help you to extract some parts of date and time.

**Description**

- ▶ **Minute**                      Minute of the close of the current bar
- ▶ **Hour**                         Hour of the close of the current bar
- ▶ **Day**                            Day of the close of the current bar
- ▶ **Month**                        Month of the close of the current bar
- ▶ **Year**                         Year of the close of the current bar
- ▶ **DayOfWeek**                 Day of week of the close of the current bar  
(1=monday, 2=tuesday, 3=wednesday, 4=thursday, 5=wednesday)

Temporal terms are rarely used for the technical analysis even though traders know that some intraday periods of the day or some days of the year are more important than others.

**Exemple Minute, Hour, DayOfWeek, Month, BarIndex**

```
REM A measurement of the professionals power (on daily bars)
REM 1st supposition: There are twice more professionals on Wednesday, Thursday and Friday
REM 2nd supposition: They control closing price
IF DayOfWeek <= 2 THEN
    Weight = 1
ELSE
    Weight = 2
ENDIF

REM Builds a cumulative indicator based on "professionals power"
ProPower = ProPower + Weight * (Close - MedianPrice)
REM returns 'ProPower' as output of the function
RETURN ProPower
```

### ***BarIndex, IntradayBarIndex, Days***

Counts bars and days.

#### ***Description***

- ▶ **BarIndex**                    the index of the current bar among the bars available
- ▶ **IntradayBarIndex**        the index of the current bar among the bars of the day
- ▶ **Days**                        the index of the day from 1900

These variables allow you to write codes that take only certain bars in account and not a specific date.

#### **Example IntradayBarIndex**

```
REM This indicator is not null when we change of day (en intraday)
```

```
RETURN IntradayBarIndex = 0
```

## Undefined Term

### *Undefined*

Represents an undefined value.

### *Description*

When we build a technical indicator, we often use past bars. In this case, it is impossible to calculate the first values of the indicator.

number 0 that is displayed by default.

### **Example UNDEFINED**

```
REM A non optimized calculation of a 20 bars moving average
IF BarIndex < 19 THEN
  myMA = undefined

ELSE

  myMA = 0

  FOR i = 0 TO 19 DO
    myMA = myMA + close[i] // reading the closing prices of previous bars
  NEXT

  myMA = myMA / 20

ENDIF

REM Returns 'myMA' as output of the function

RETURN myMA
```

## Expressions and operators

### Access to bars of the past

[.]

Access to values or variables of previous bars.

#### *Description*

▶ expression[*count*]

The values calculated by ProBuilder for each bar are saved. This way you can access them anytime. This is very important since the technical analysis is based on the fact that financial markets have a memory.

#### **Example1** [.]

REM Calculation of the moving average value of the current bar, using closing prices:

```
myMA = average[30](close)
```

REM Difference between myMA current value and myMA previous value:

```
myGrowth = myMA - myMA[1]
```

REM Returns 'myGrowth' as output of the function. You may display it with a histogram view

```
RETURN myGrowth
```

**Example2 [.]**

REM Example of multiple calculation

```
RETURN (open+close[1])[3]
```

REM Such function returns for each bar, the calculated value of the third previous bar when  
REM adding its opening price and its previous closing price

REM We thus get the addition of the opening price of the third previous bar  
REM and the closing price of the fourth previous bar

**Warning**

Previous bars are only in read mode available.

The calculation of a variable only modifies its current value without modifying its previous values. On the above example, the past values of the variable x are equal to 0.

**Example3 [.]**

```
REM Example of a bad use of the [.]
```

```
x = open
```

```
REM A common mistake is to think that we get the opening price value of previous bars
```

```
myIndicator = x[3]
```

```
REM Actually, we get the x value calculated in the past
```

```
REM and such value is equal to 0 since of the below instruction:
```

```
x = 0
```

```
REM Returns 'myIndicator' as output of the function
```

```
RETURN myIndicator
```

**Note:**

ProBuilder uses values of every bar, from the first one to the last one and executes the function in order to calculate the current bar value of the indicator.

## Binary arithmetic operators

### + - \* / MOD

Basic arithmetic calculations.

#### *Description*

- ▶ a + b
- ▶ a – b
- ▶ a \* b
- ▶ a / b
- ▶ a MOD b

Such operators allows you to make all the basic arithmetic calculations :

- ▶ addition (+)
- ▶ subtraction (–)
- ▶ multiplication (\*)
- ▶ division (/)
- ▶ modulus or remainder of the division (**MOD**)

#### **Example1** + - \* /

```
REM Such program returns an arithmetic expression using price terms  
myIndicator = open * volume + (close - high) / low  
REM Returns 'myIndicator' as output of the function  
RETURN myIndicator
```

#### **Example2** MOD

```
REM Creation of a binary oscillator  
myOscillator = BarIndex MOD 2  
REM Returns 'myOscillator' as output of the function  
RETURN myOscillator
```

## **MIN MAX**

Traditional arithmetic calculations.

### **Description**

- ▶ **MIN**(a, b)
- ▶ **MAX**(a, b)

Such operators return the lowest and the highest of two elements.

### **Example1 MIN**

```
REM Such program returns the lowest price between the opening and closing price
```

```
myLowest = MIN(open, close)
```

```
REM Returns 'myLowest' as output of the function
```

```
RETURN myLowest
```

### **Example2 MAX**

```
REM Such program returns the highest price between the opening and closing price
```

```
myHighest = MAX(open, close)
```

```
REM Returns 'myHighest' as output of the function
```

```
RETURN myHighest
```

## Unary arithmetic operators

### *ROUND ABS SGN (-)*

Traditional arithmetic calculations.

#### *Description*

- ▶ **ROUND**(a)
- ▶ **ABS**(a)
- ▶ **SGN**(a)
- ▶ -a

Such operators execute traditional arithmetic calculations :

- ▶ round the value (**ROUND**)
- ▶ absolute value (**ABS**)
- ▶ sign of the value : 1 if a>0, -1 if a <0, 0 if a=0 (**SGN**)
- ▶ opposite of the value (-)

#### **Example1 ROUND**

```
REM Such program indicates the psychological crossings each 10 euros
REM  1: up crossing
REM -1: low crossing
my10 = ROUND(close / 10)
myCross = SGN(my10 - my10[1])
REM Returns 'myCross' as output of the function
RETURN myCross
```

#### **Example2 ABS**

```
REM A volatility indicator
myVolatility = ABS(close - close[1])
REM Returns 'myVolatility' as output of the function
RETURN myVolatility
```

**Example3 SGN**

REM Such indicator emphasizes on the trend of prices and not on its power  
REM Divergences with prices may be of an interest for technical analysis...

REM Indicates the variation trend (+1 = moves up, -1 = moves down)

```
myVariation = SGN(close - close[1])
```

REM Returns the cumulative sum of the variation trend

```
RETURN CUMSUM(myVariation)
```

## **SQUARE SQRT**

Square and square root.

### **Description**

- ▶ **SQUARE(a)**
- ▶ **SQRT(a)**

Such operators calculate the square and the square root of a number.

### **Example1 SQUARE, SQRT**

```
REM Such program calculates the standard deviation between latest 20 closing prices
sumY = 0
sumY2 = 0

FOR i = 0 TO 19
  sumY = sumY + close[i]
  sumY2 = sumY2 + SQUARE(close[i])
NEXT

myVolatility = sumY2/20 - SQUARE(sumY/ 20)
myStandardDeviation = SQRT(myVolatility)

REM Returns 'myStandardDeviation' as output of the function

RETURN myStandardDeviation
```

## **LOG EXP**

Logarithm and exponential functions.

### **Description**

- ▶ **LOG(a)**
- ▶ **EXP(a)**

These operators calculate the logarithm and the exponential of a number.

### **Example LOG, EXP**

```
REM Such program calculates the geometric moving average between latest 20 closing prices
```

```
sum = 0
```

```
FOR i = 0 TO 19 // Calculate the average between latest 20 logarithm of closing prices
```

```
  sum = sum + LOG(close[i])
```

```
NEXT
```

```
sum = sum / 20
```

```
REM Returns the geometric moving average
```

```
RETURN EXP(sum)
```

## **COS SIN TAN ATAN**

Tigonometical functions.

### **Description**

- ▶ **COS(a)**
- ▶ **SIN(a)**
- ▶ **TAN(a)**
- ▶ **ATAN(a)**

These operators calculate the cosine, the sine, the tangent and the cotangent of of a number.

### **Example ATAN**

```
REM Such program calculates the angle of the talus of the geometric moving average
REM between latest 30 closing prices

// Build the moving average
mm = AVERAGE[30](close)

// Calculate the talus
dy = (mm / mm[1] - 1)*100 // 1 unit = 1% variation of prices
dx = 1 // 1 unit = duration of one bar

// Convert the talus to an angle
myAngle = ATAN(dy/dx)

REM Returns 'myAngle' as output of the function

RETURN myAngle
```

## Comparison operators

< <= > >= <> =

Testing numbers.

### Description

- ▶ a < b
- ▶ a <= b or a=< b
- ▶ a > b
- ▶ a >= b or a=> b
- ▶ a = b
- ▶ a <> b

These operators compare elements. They are mainly used for comparing numbers. However, they also can be used as arithmetic operators with the following convention:

- ▶ when the result is TRUE, the comparison returns +1
- ▶ when the result is FALSE, the comparison returns 0

### Example1 < >

REM Such indicator emphasizes on the prices trend and not on its strength

```
myPositiveVariation = close > close[1]           // +1 when prices go up  
myNegativeVariation = close < close[1]          // 1 when prices go down
```

```
myVariation = myPositiveVariation - myNegativeVariation // +1=bull, -1=bear
```

REM Returns cumulative values of 'myVariation'

```
RETURN CUMSUM(myVariation)
```

**Example2 >=**

```
REM Such indicator emphasizes on the prices trend when volume increases
```

```
IF Volume >= Volume[1] THEN  
  myIndicator = myIndicator + (close - close[1])  
ENDIF
```

```
REM Returns 'myIndicator' as output of the function
```

```
RETURN myIndicator
```

**Warning**

In the above example, you must keep in mind that the expression « **close** - **close**[1] » is not defined for the first bar. There is a risk that « *myIndicator* » begins with an undefined value, and as a consequence all the values of « *myIndicator* » are wrong.

**However**, such risk doesn't exist since the result of the test « **volume** >= **volume**[1] » is also not defined, so it is considered as wrong.

## **CROSSES OVER - CROSSES UNDER**

Testing the crosses of two charts

### **Description**

- ▶ a Crosses Over b
- ▶ a Crosses Under b

This operators test the upward crossing and the downward crossing of two charts.

### **Example Crosses over, Crosses under**

```
REM Creating an indicator that counts the support and resistance breaks.  
REM We first have to define what a support or resistance break is.  
REM In our example, we use the opening price and closing price of the day before  
REM defining such support or resistance levels.
```

```
Resistance = MAX(open[1], close[1])
```

```
Support = MIN(open[1], close[1])
```

```
REM We now count the number of resistance breaks.
```

```
countResistanceBreak = CUMSUM(close CROSSES OVER Resistance)
```

```
REM We then count the number of resistance breaks.
```

```
countSupportBreak = CUMSUM (close CROSSES UNDER Support)
```

```
REM Returns the difference between both numbers
```

```
RETURN countResistanceBreak - countSupportBreak
```

## Logical operators

### *NOT OR AND XOR*

Traditional Boolean logic

#### *Description*

- ▶ **NOT**(a)
- ▶ a **OR** b
- ▶ a **AND** b
- ▶ a **XOR** b

These operators carry out combinations of comparisons, which are:

- ▶ Logical NOT (**NOT**)
- ▶ Logical OR (**OR**)
- ▶ Logical AND (**AND**)
- ▶ Logical Exclusive OR (**XOR**)

These operators can be used as a condition of the instructions controlling the datastream, or as arithmetic operators.

#### **Example1 XOR**

REM Indicator emphasizing the appearance of bearish strength  
REM Assumption 1: Decrease on volume may induce a bearish trend on prices  
REM Assumption 2: Bearish trend on prices creates new bearish trend  
REM Assumption 3: If the above assumptions are TRUE at the same time, a bullish trend  
REM can be expected

```
myVolumeCriteria = volume < volume[1]
```

```
myTrendCriteria = close < close[1]
```

```
myCriteria = myVolumeCriteria XOR myTrendCriteria
```

REM Returns the sum of the bearish strength of the latest 10 bars

```
RETURN SUMMATION[10](myCriteria)
```

**Example2 NOT**

```
REM This indicator emphasizes the trend of the prices when volume don't decrease
```

```
IF NOT (Volume < Volume[1]) THEN
```

```
    myIndicator = myIndicator + (close - close[1])
```

```
ENDIF
```

```
REM Returns 'myIndicator' as output of the function
```

```
RETURN myIndicator
```

## Language structure, instructions

### Brief description

The code of any function written with ProBuilder is composed of several instructions followed by *RETURN* to valid it.

The syntax is similar as BASIC : there is no need to define variables. The common structures (*IF THEN ELSE ENDIF*, *FOR TO NEXT*, *WHILE WEND*) can be used. Instructions are separated by a Return to another line.

With ProBuilder you do not need to define your variable. ProBuilder will do it for you ! A variable may represent a number, a boolean or a vector (table of numbers). It only depends on the instructions that use the variable. If you you access past bars of a variable, you know that this bar is a vector.

### **Example Variables don't need to be declared**

```
REM This program doesn't make anything at all.  
Variable = 10+8*7           // Variable is an integer  
Variable = 3.14159         // Variable is a decimal  
Variable = 5 > 2           // Variable is a boolean  
Variable = open           // Variable is a vector  
REM Returns 'variable' as output of the function  
RETURN variable
```

It is however possible (and often advised) to initialise your variables with the keyword **ONCE**. The instruction be then be executed exclusively on the first bar.

### **Example ONCE**

```
REM Calculation of the On Balance Volume  
ONCE myOBV = 10000 // Same instruction as IF BarIndex = 0 THEN myOBV = 10000  
IF BarIndex > 0 THEN  
    myOBV = myOBV + SGN(close - close[1]) * volume  
ENDIF  
REM Returns 'myOBV' as output of the function  
RETURN myOBV
```

## Insertion of comments

### *REM*

Insert comments in the program.

### *Description*

- ▶ **REM** comment

This keyword allows you to insert comments in your program. Each comment should start with REM. ProBuilder will then ignore this line.

### **Example1 REM**

```
REM Such program returns the closing price
```

```
RETURN close
```

It is also possible to start a comment on the middle of a line using `//`.

### **Example2 //**

```
RETURN close // Such program returns the closing price
```

## Variables, parameters and definitions

Parameters are displayed within the ProRealTime Software, on the right side of the window that allows you to edit your program. By going to “Properties” in your menu you can give a special type to each of your parameters and optimize the calculation of your indicator:

- ▶ integer
- ▶ decimal
- ▶ boolean
- ▶ type of moving average

Local variables do not need to be declared. Their default value is 0. It is possible to initialise them with the *ONCE* instruction.

### **[ONCE]**

Initialisation of local variables.

#### **Description**

- ▶ **[ONCE]** *variable = expression*

The keyword *ONCE* allows to initialize the variable to a given expression (value of first bar).

#### **ExampleONCE**

```
REM Calculation of the On Balance Volume

ONCE myOBV = 10000 // Same instruction as IF BarIndex = 0 THEN myOBV = 10000

IF BarIndex > 0 THEN
  myOBV = myOBV + SGN(close - close[1]) * volume
ENDIF

REM Returns 'myOBV' as output of the function

RETURN myOBV
```

## Control structures

### ***IF THEN ELSIF ELSE ENDIF***

Conditional instruction.

#### ***Description***

```
IF test1 THEN  
    Instructions1  
ELSIF test2 THEN  
    Instructions2  
ELSE  
    Instructions3  
    ...  
ENDIF
```

Instructions are executed depending on the test results. If *test1* is TRUE, the *Instructions1* are executed. Otherwise, if *test2* is TRUE, *Instructions2* are executed. If both *test1* and *test2* are FALSE, the *Instruction3* is executed.

#### **Example 1 IF THEN ELSE ENDIF**

```
REM Calculation of the On Balance Volume  
  
IF BarIndex > 0 THEN  
    myOBV = myOBV + SGN(close - close[1]) * volume  
  
ELSE  
  
    myOBV = 10000  
  
ENDIF  
  
REM Returns 'myOBV' as output of the function  
  
RETURN myOBV
```

**Example 2 IF THEN ELSIF ELSE ENDIF**

```
REM "States" indicator

IF close > average[20](close) AND volume > volume[1] THEN

    myState = 1

ELSIF close > average[20](close) AND volume < volume[1] THEN

    myState = 2

ELSIF close < average[20](close) AND volume > volume[1] THEN

    myState = 3

ELSE

    myState = 4

ENDIF

REM Returns 'myState' as output of the function

RETURN myState
```

## **WHILE DO WEND**

Conditional instruction.

### **Description**

```
WHILE test [DO]  
    instructions  
WEND
```

Instructions are executed if *test* is true. The keyword **DO** is optional.

### **Example** **WHILE DO WEND**

```
REM Calculation of the number of consecutive bullish days
```

```
bullish = close > close[1]
```

```
count = 0
```

```
WHILE bullish[count] AND count < BarIndex DO
```

```
    count = count + 1
```

```
WEND
```

```
REM Returns 'count' as output of the function
```

```
RETURN count
```

## **FOR TO|DOWNTO DO NEXT**

Iterative instruction.

### **Description**

```
FOR variable = expression1 TO|DOWNTO expression [DO]  
    instructions  
NEXT
```

Instructions are executed a given number of times. *variable* is initialised with the *expression1* and then is incremented (use of **TO**) or is decremented (use of **DOWNTO**) after each iteration. It ends when *variable* crosses *expression2*.

We note that instructions may be executed only once. (*FOR i=1 TO 0 as an instance*).

### **Example FOR TO|DOWNTO DO NEXT**

```
REM A non optimized calculation of a 20 bars moving average  
  
IF BarIndex < 19 THEN  
    myAverage = undefined  
ELSE  
    myAverage = 0  
    FOR i = 0 TO 19 DO                // calculation of SUMMATION[20](close)  
        myAverage = myAverage + close[i]  
    NEXT  
    myAverage = myAverage / 20  
ENDIF                                // equal to AVERAGE[20](close)  
REM Returns 'myAverage' as output of the function  
RETURN myAverage
```

### **Remark**

To get faster indicators, you should use the ProBuilder Functions when possible. In this case, the function **AVERAGE**.

## **BREAK**

End of a conditional or an iterative instruction.

### **Description**

#### ▶ **BREAK**

The instruction **BREAK** ends a **WHILE...WEND** or a **FOR...NEXT** instruction.

### **ExampleBREAK**

```
REM A calculation of the consecutive bullish days
```

```
bullish = close > close[1]
```

```
count = 0
```

```
WHILE count < BarIndex DO
```

```
    IF NOT bullish[count] THEN
```

```
        BREAK
```

```
    ENDIF
```

```
    count = count + 1
```

```
WEND
```

```
REM Returns 'count' as output of the function
```

```
RETURN count
```

## **CONTINUE**

Continuation of a conditional or an iterative instruction

### **Description**

#### ▶ **CONTINUE**

The instruction **CONTINUE** allows you to return to the initial test of a **WHILE...WEND** or **FOR...NEXT** instruction.

### **ExampleCONTINUE**

```
REM A calculation of the consecutive bullish days
```

```
bullish = close > close[1]
```

```
count = 0
```

```
WHILE count < BarIndex DO
```

```
    IF bullish[count] THEN
```

```
        count = count + 1
```

```
        CONTINUE
```

```
    ENDIF
```

```
BREAK
```

```
WEND
```

```
REM Returns 'count' as output of the function
```

```
RETURN count
```

## Call of a user function

### CALL | GOSUB

Call a User Function.

#### Description

▶  $var1, var2, \dots = [\text{CALL}|\text{GOSUB}] \text{function}[p1,p2,\dots](price)$

Parameters are transmitted between square brackets, the prices between brackets.

Transmitted parameters have to be in the same order as the order of declaration in the called function.

The transmitted price between brackets is only authorized if the called function uses *customClose*.

The calculated values become variables and are in the same order as assigned in the *RETURN* instruction.

#### ExampleCALL | GOSUB

##### FUNCTION «myFunction» - Parameter «p» integer

REM Calculation of the value of the variable price on the previous bar

RETURN **customClose**[p]

##### FUNCTION «myMainFunction »

REM Calculation of myFunction (parameter p = 10) on the values «open+close»

**myIndicator** = CALL myFunction[10](**open** + **close**)

REM Calculation of my smoothed function with a moving average

**mySmoothedIndicator** = average[10](myIndicator)

RETURN mySmoothedIndicator

## Return of a user function

### **RETURN**

Publication of the results of a user function.

### **Description**

- ▶ **RETURN** expression [**COLOURED BY** indicateur] [**AS** «libellé»],

The returned values can be named. The names will appear in the indicator window. You can choose on which bars the “up color” and the “down color” apply.

### **ExampleRETURN, AS**

REM Calculation of Bollinger Bands

```
mm = AVERAGE[20](close)
```

```
StandardDeviation = STD[20](close)
```

```
bsup = mm + 2 * StandardDeviation
```

```
binf = mm - 2 * StandardDeviation
```

REM Returns 2 curves with customized names

```
RETURN bsup AS «Bollinger Sup», binf AS «Bollinger Inf»
```

### **ExampleRETURN, COLOURED BY**

REM Calculation of the moving average of Bollinger

```
mm = AVERAGE[20](close)
```

REM Returns a curve without name, coloured by «up color» when the price is upper than

REM the curve and coloured by «down color» when the price is inferior to the curve

```
RETURN mm COLOURED BY (close - mm)
```

## Functions Library

### Moving averages

#### *Average*

Calculation of moving averages.

#### *Description*

- ▶ **Average**[count](price)
- ▶ **Average**[count, type](price)

The function *Average* calculates a moving average of a price.

By default, an arithmetical moving average is used. It is possible to choose a moving average of the indicator window by selecting a «MATYPE» parameter and specify the parameter of the function as shown below.

#### **Example1 Arithmetical Moving Average**

REM Calculation of a simple moving average on 20 bars

mm = **Average**[20](close)

RETURN mm

#### **Example2 Customized Moving Average**

REM Calculation of a customizable moving average on 20 bars

REM A parameter named «choice» of type «MATYPE» has to be created

mm = **Average**[20, choice](close)

RETURN mm

## ***ExponentialAverage***

Calculation of moving averages.

### ***Description***

▶ **ExponentialAverage**[count](price)

The function *ExponentialAverage* calculates the exponential moving average of the prices.

### **ExampleExponentialAverage**

REM Calculation of an exponential moving average on 20 bars

mm = **ExponentialAverage**[20](close)

RETURN mm

## ***WeightedAverage***

Calculation of moving averages.

### ***Description***

- ▶ **WeightedAverage**[count](price)

The function *WeightedAverage* calculates the Weighted moving average of the prices.

### **ExampleWeightedAverage**

REM Calculation of a weighted moving average on 20 bars

mm = **WeightedAverage**[20](close)

RETURN mm

## ***WilderAverage***

Calculation of moving averages.

### ***Description***

▶ **WilderAverage**[count](price)

The function *WilderAverage* calculates a Wilder moving average of the prices.

### **ExampleWilderAverage**

REM Calculation of a Wilder moving average on 20 bars

mm = **WilderAverage**[20](close)

RETURN mm

## ***TriangularAverage***

Calculation of moving averages.

### ***Description***

- ▶ **TriangularAverage**[count](price)

The function *TriangularAverage* calculates a Triangular moving average of the prices.

### **ExampleTriangularAverage**

REM Calculation of a Triangular moving average on 20 bars

mm = **TriangularAverage**[20](close)

RETURN mm

## ***EndPointAverage***

Calculation of moving averages.

### ***Description***

▶ **EndPointAverage**[count](price)

The function *EndPointAverage* calculates the End Point moving average of the prices.

### **ExampleEndPointAverage**

REM Calculation of an End point moving average on 20 bars

mm = **EndPointAverage**[20](close)

RETURN mm

## ***TimeSeriesAverage***

Calculation of moving averages.

### ***Description***

▶ **TimeSeriesAverage**[count](price)

The function *TimeSeriesAverage* calculates the Time Series moving average of the prices.

### **ExampleTimeSeriesAverage**

REM Calculation of a Time Series moving average on 20 bars

mm = **TimeSeriesAverage**[20](close)

RETURN mm

## **DEMA**

Calculation of moving averages.

### **Description**

▶ **DEMA**[count](price)

The function *DEMA* calculates the double exponential moving average of the prices.

### **ExampleDEMA**

REM Calculation of the double exponential moving average on 21 bars

mm = **DEMA**[21](close)

RETURN mm

## **TEMA**

Calculation of moving averages.

### **Description**

▶ **TEMA**[count](price)

The function *TEMA* calculates the triple exponential moving average of the prices.

### **ExampleTEMA**

REM Calculation of the triple exponential moving average on 21 bars

mm = **TEMA**[21](close)

RETURN mm

## ProBuilder Functions

### *CUMSUM*

Calculation of cumulative sum.

#### *Description*

- ▶ **CUMSUM**(price)

The function *CUMSUM* calculates the cumulative sum of an indicator.

#### **ExampleCUMSUM**

REM Calculation of the On Balance Volume

```
deltaPrice = SGN(close - close[1])
```

```
deltaVolume = deltaPrice * Volume
```

```
myOBV = CUMSUM(deltaVolume)
```

```
RETURN myOBV
```

## ***SUMMATION***

Calculation of a specific sum

### ***Description***

▶ **SUMMATION**[count](price)

The function *SUMMATION* calculates the sum of the *count* previous values of an indicator.

### **ExampleSUMMATION**

REM Calculation of a simple moving average

SpecificSum10 = SUMMATION[10](close)

myMM10 = SpecificSum10 / 10

RETURN myMM10

## **LOWEST**

Calculation of the lowest value on a given duration

### **Description**

▶ **LOWEST**[count](price)

The function *LOWEST* calculates the lowest value of the *count* previous values of an indicator.

### **ExampleLOWEST**

REM Calcul of the Williams %R

CustomHigh = HIGHEST[14](high)

CustomLow = LOWEST[14](low)

myWilliams = (close - CustomLow) / (CustomHigh - CustomLow) \* 100

RETURN myWilliams

## **HIGHEST**

Calculation of the highest value on a given duration

### **Description**

▶ **HIGHEST**[count](price)

The function *HIGHEST* calculates the highest value of the *count* previous values of an indicator.

### **ExampleLOWEST**

REM Calcul of the Williams %R

CustomHigh = HIGHEST[14](high)

CustomLow = LOWEST[14](low)

myWilliams = (close - CustomLow) / (CustomHigh - CustomLow) \* 100

RETURN myWilliams

## ProRealTime technical indicators list

### *AccumDistr*

Calculation of the «Accumulation Distribution» indicator

#### *Description*

- ▶ **AccumDistr**(price)

The function *AccumDistr* calculates the «Accumulation Distribution» indicator.

#### **Example AccumDistr**

REM Calculation of Accumulation Distribution indicator using opening prices

```
accumOuv = AccumDistr(open)
```

```
RETURN accumOuv
```

## **ADX**

Calculation of the «ADX» indicator

### **Description**

▶ **ADX**[count]

The function *ADX* calculates the «ADX» indicator on the *count* previous bars.

### **Example ADX**

```
REM Calculation of the ADX14
```

```
myADX = ADX[14]
```

```
RETURN myADX
```

## **ADXR**

Calculation of the «ADXR» indicator

### **Description**

▶ **ADXR**[count]

The function *ADXR* calculates the «ADXR» line of the “ADX” indicator on the *count* previous bars.

### **Example ADXR**

```
REM Calculation of the ADXR14
```

```
myADXR = ADXR[14]
```

```
RETURN myADXR
```

### ***AroonUp, AroonDown***

Calculation of the «Aroon» indicator

#### ***Description***

- ▶ **AroonUp**[count]
- ▶ **AroonDown**[count]

The functions *AroonUp* and *AroonDown* calculate the «Aroon» indicator on the *count* previous bars.

#### **Example AroonUp, AroonDown**

```
REM Calculation of the difference between both lines of Aroon14
```

```
HighLine = AroonUp[14]
```

```
LowLine = AroonDown[14]
```

```
myIndicator = HighLine - LowLine
```

```
RETURN myIndicator
```

### ***AverageTrueRange***

Calculation of the True Range moving average

#### ***Description***

▶ **AverageTrueRange**[count](price)

The function *AverageTrueRange* calculates the True Range moving average calculated using the *prices* of the *count* previous bars.

#### **Example AverageTrueRange**

REM Calculation of the AVT14 using closing prices

```
monAveragetrueRange = AveragetrueRange[14](close)
```

```
RETURN monAveragetrueRange
```

## ***BollingerUp, BollingerDown***

Calculation of the Bollinger Bands

### ***Description***

- ▶ **BollingerUp**[count](price)
- ▶ **BollingerDown**[count](price)

The functions *BollingerUp* and *BollingerDown* calculate the Bollinger Bands using the *prices* of the *count* previous bars.

### **Example1 BollingerUp, BollingerDown**

```
REM Calculation of the Bollinger Bands
```

```
HighLine = BollingerUp[20](close)  
LowLine = BollingerDown[20](close)  
RETURN HighLine, LowLine
```

### **Example2 BollingerUp, BollingerDown**

```
REM Calculation of the average standard deviation of the 20 latest opening prices
```

```
HighLine = BollingerUp[20](open) // 2 Standard Deviation above the average  
LowLine = BollingerDown[20](open) // 2 Standard Deviation below the average  
myStandardDeviation = (HighLine - LowLine)/4  
  
RETURN myStandardDeviation
```

### **Warning**

It is faster in the example above to use the function *STD* directly to calculate the standard deviation.

### ***BollingerBandWidth***

Calculation of the BollingerBandWidth indicator.

#### ***Description***

▶ **BollingerBandWidth**[count](price)

The function *BollingerBandWidth* calculates the ratio between the gap of the BollingerBands and their average. This function is calculated using the *prices* of the *count* previous bars.

#### **Example BollingerBandWidth**

```
REM Calculation of the ratio between the gap of the BollingerBands and their average
```

```
myGap = BollingerBandWidth[20](close)
```

```
RETURN myGap
```

## CCI

Calculation of the “Commodity Channel Index” indicator

### Description

- ▶ **CCI**[count]
- ▶ **CCI**[count](price)

The function *CCI* calculates the «Commodity Channel Index» indicator on the *count* previous bars. By default, such indicator uses **typicalPrice**. However, it is possible to use whatever prices you want.

### Example CCI

```
REM Calculation of the Commodity Channel Index indicator
```

```
myCCI = CCI[20](typicalPrice)
```

```
RETURN myCCI
```

## **ChaikinOsc**

Calculation of the “Chaikin Oscillator” indicator

### **Description**

▶ **ChaikinOsc**[p,q](price)

The function *ChaikinOsc* calculates the «Chaikin Oscillator» using *price* with *p* and *q* as parameters.

### **Example ChaikinOsc**

```
REM Calculation of the Chaikin Oscillator (MA3 short term, MA10 long term)
```

```
myChaikinOsc = ChaikinOsc[3,10](close)
```

```
RETURN myChaikinOsc
```

### ***ChandeKrollStopUp, ChandeKrollStopDown***

Calculation of the “Chande & Kroll’s Volatility Stop” indicator

#### ***Description***

- ▶ **ChandeKrollStopUp**[p, q, x]
- ▶ **ChandeKrollStopDown**[p, q, x]

These functions calculate the stop levels of the «Chande & Kroll’s Volatility Stop» indicator.

*P: Average True Range parameter*

*Q: «Highest» function parameter*

*X: True Range coefficient*

#### **Example ChandeKrollStopUp, ChandeKrollStopDown**

REM Calculation of the Chande & Kroll’s Volatility Stop indicator

myUpStop = ChandeKrollStopUp[10,20,30](close)

myDownStop = ChandeKrollStopDown[10,20,30](close)

RETURN myUpStop AS «Stop +», myDownStop AS “Stop -”

## **Chandle**

Calculation of the “Chandle Momentum Oscillator” indicator

### **Description**

▶ **Chandle**[count](price)

The function *Chandle* calculates the « Chandle Momentum Oscillator » using the *prices* of the *count* previous bars.

### **Example Chandle**

```
REM Calculation of the Chandle Momentum Oscillator indicator
```

```
myChandle = Chandle[20](close)
```

```
RETURN myChandle
```

## ***Cycle***

Calculation of the “Cycle” indicator

### ***Description***

▶ **Cycle**(price)

The function *Cycle* calculates the «Cycle » on the *price*.

### **Example Cycle**

```
REM Calculation of the «Cycle»
```

```
myCycle = Cycle(close)
```

```
RETURN myCycle
```

## *DI*

Calculation of the DI indicator (directional system)

### **Description**

▶ **DI**[count](price)

The function *DI* calculates the DI on the count previous bars of the *price*.

### **Example DI**

REM Calculation of the DI14 (using closing prices)

```
myDI = DI[14](close)
```

```
RETURN myDI
```

## ***Diplus, DIminus***

Calculation of the lines of the DI indicator (directional system)

### ***Description***

- ▶ **Diplus**[count](price)
- ▶ **DIminus**[count](price)

The function *Diplus* and *DIminus* calculate the DI+ and DI- lines on the count previous bars of the *price*.

### **Example DIplus, DImoins**

REM Calculation of the DI14 (using closing prices)

```
myDIplus = DIplus[14](close)
```

```
myDIminus = DIminus[14](close)
```

```
RETURN myDIplus AS "DI+", myDIminus AS "DI-"
```

## ***DPO***

Calculation of the «Detrended Price Oscillator» indicator

### ***Description***

▶ **DPO**[count](price)

The function *DPO* calculates the «Detrended Price Oscillator» on the count previous bars of *price*.

### **Example DPO**

```
REM Calculation of the Detrended Price Oscillator
```

```
myDPO = DPO[21](close)
```

```
RETURN myDPO
```

### ***EaseOfMovement***

Calculation of the «Ease Of Movement» indicator

### ***Description***

▶ **EaseOfMovement**[count]

The function *EaseOfMovement* calculates the «Ease Of Movement» on the count previous bars.

### **Example EaseOfMovement**

```
REM Calculation of the Ease Of Movement 14
```

```
myEase = EaseOfMovement[14]
```

```
RETURN myEase
```

### ***EaseOfMovementValue***

Calculation of the “Ease Of Movement Value indicator”.

### ***Description***

▶ **EMV**

The function *EaseOfMovementValue* calculates the «Ease Of Movement Value».

### **Exemple EMV**

```
REM Calculation of the Ease Of Movement Value
```

```
myEMV = EMV
```

```
RETURN myEMV
```

## ***ForceIndex***

Calculation of the «Force Index» indicator

### ***Description***

▶ **ForceIndex**(price)

The function *ForceIndex* calculates the «Force Index» on *price*.

### **Example Force Index**

```
REM Calculation of the Force Index (using closing prices)
```

```
myFI = ForceIndex(close)
```

```
RETURN myFI
```

### ***HistoricVolatility***

Calculation of the historic volatility.

#### ***Description***

▶ **HistoricVolatility**[count](price)

The function *HistoricVolatility* calculates then historic volatility on the *count* previous bars of *price*.

#### **Example HistoricVolatility**

```
REM Calculation of the historic volatility of the 20 previous bars
```

```
myVolatility = HistoricVolatility[20] (close)
```

```
RETURN myVolatility
```

## ***LinearRegression***

Calculation of the linear regression

### ***Description***

▶ **LinearRegression**[count](price)

The function *LinearRegression* calculates the linear regression using *price* of the *count* previous bars.

### **Example LinearRegression**

REM Calculation of the linear regression using closing prices of the 10 latest bars

```
myRegression = LinearRegression[10](close)
```

```
RETURN myRegression
```

### ***LinearRegressionSlope***

Calculation of the linear regression slope

#### ***Description***

- ▶ **LinearRegressionSlope**[count](price)

The function *LinearRegressionSlope* calculates the linear regression slope using *price* of the *count* previous bars.

#### **ExampleLinearRegressionSlope**

REM Calculation of the linear regression slope of the 10 latest bars

```
mySlope = LinearRegressionSlope[10](close)
```

```
RETURN mySlope
```

## **MACD**

Calculation of the «Moving Average Convergence Divergence» indicator

### **Description**

▶ **MACD**[p,q,r](price)

The function *MACD* calculates the «Moving Average Convergence Divergence» indicator on *price*.

### **Example MACD**

```
REM Calculation of the Moving Average Convergence Divergence
```

```
myMACD = MACD[12,26,9](close)
```

```
RETURN myMACD
```

### **MACDline**

Calculation of the MACD line of the «Moving Average Convergence Divergence» indicator

### **Description**

▶ **MACDline**[p,q,r](price)

The function *MACDline* calculates the MACD line of the «Moving Average Convergence Divergence» indicator on *price*.

### **Example MACDline**

REM Calculation of the Moving Average Convergence Divergence

```
myMACD = MACDline[12,26,9](close)
```

```
mySignal = ExponentialAverage[9](myMACD)
```

```
RETURN myMACD AS "MACD", mySignal AS "Signal"
```

### ***MassIndex***

Calculation of the «Mass Index» indicator

### ***Description***

▶ **MassIndex**[count]

The function *MassIndex* calculates the «Mass Index» indicator on *count* bars.

### **Example MassIndex**

```
REM Calculation of the Mass Index
```

```
myMass = MassIndex[25]
```

```
RETURN myMass
```

## ***Momentum***

Calculation of the momentum

### ***Description***

▶ **Momentum**[count](price)

The function *Momentum* calculates the momentum on the count previous bars of *price*.

### **Example Momentum**

```
REM Calculation of the Momentum
```

```
myMomentum = Momentum[12](close)
```

```
RETURN myMomentum
```

## ***MoneyFlow***

Calculation of the «Money Flow» indicator

### ***Description***

▶ **MoneyFlow**[count](price)

The function *MoneyFlow* calculates the «Money Flow» indicator on the count previous bars of *price*.

### **Example MoneyFlow**

```
REM Calculation of the Money Flow
```

```
myMoneyFlow = MoneyFlow[14](close)
```

```
RETURN myMoneyFlow
```

## ***MoneyFlowIndex***

Calculation of the «Money Flow Index» indicator

### ***Description***

▶ **MoneyFlowIndex**[count]

The function *MoneyFlowIndex* calculates the «Money Flow Index» indicator on *count* bars.

### **Example MoneyFlowIndex**

```
REM Calculation of the Money Flow Index
```

```
myMoneyFlow = MoneyFlowIndex[14]
```

```
RETURN myMoneyFlow
```

### ***NegativeVolumeIndex***

Calculation of the «Negative Volume Index» indicator

#### ***Description***

- ▶ **NegativeVolumeIndex(price)**

The function *NegativeVolumeIndex* calculates the «Negative Volume Index» on *price*.

#### **Example NegativeVolumeIndex**

```
REM Calculation of the Negative Volume Index
```

```
myNegativeVolumeIndex = NegativeVolumeIndex(close)
```

```
RETURN myNegativeVolumeIndex
```

## **OBV**

Calculation of the «On Balance Volume» indicator

### **Description**

▶ **OBV(price)**

The function *OBV* calculates the «On Balance Volume» on *price*.

### **Example OBV**

```
REM Calculation of the On Balance Volume
```

```
myOBV = OBV(close)
```

```
RETURN myOBV
```

### ***PositiveVolumeIndex***

Calculation of the «Positive Volume Index » indicator

#### ***Description***

▶ **PositiveVolumeIndex**(price)

The function *PositiveVolumeIndex* calculates the «Positive Volume Index» on *price*.

#### **Example PositiveVolumeIndex**

```
REM Calculation of the Positive Volume Index
```

```
myPositiveVolumeIndex = PositiveVolumeIndex(close)
```

```
RETURN myPositiveVolumeIndex
```

### ***PriceOscillator***

Calculation of the Price Oscillator

### ***Description***

▶ **PriceOscillator** [p,q](price)

The function *PriceOscillator* calculates the Price Oscillator on *price*.

### **Example PriceOscillaor**

REM Calculation of the Price Oscillator (MA5 short term, MA25 long term)

```
myOscillator = PriceOscillator[5,25](close)
```

```
RETURN myOscillator
```

## ***PVT***

Calculation of the «Price Volume Trend» indicator

### ***Description***

▶ **PVT(price)**

The function *PVT* calculates the «Price Volume Trend» indicator on *price*.

### **Example PVT**

```
REM Calculation of the Price Volume Trend
```

```
myPriceVolumandrend = PVT(close)
```

```
RETURN myPriceVolumandrend
```

## **R2**

Calculation of the R<sup>2</sup>

### **Description**

▶ **R2**[count](price)

The function *R2* calculates the R<sup>2</sup> using *price* on the *count* previous bars.

### **Example R2**

REM Calculation of the R<sup>2</sup>

```
myR2 = R2[10](close)
```

```
RETURN myR2
```

## **Repulse**

Calculation of the Repulse

### **Description**

▶ **Repulse**[count](price)

The function *Repulse* calculates the Repulse of *price* on the *count* previous bars.

### **Example Repulse**

```
REM Calculation of the Repulse
```

```
myRepulse = Repulse[5](close)
```

```
RETURN myRepulse
```

## ROC

Calculation of the Rate Of Change

### Description

▶ **ROC**[count](price)

The function *ROC* calculates the Rate Of Change of *price* on the *count* previous bars.

### Example ROC

```
REM Calculation of the Rate Of Change
```

```
myROC = ROC[12](close)
```

```
RETURN myROC
```

## **RSI**

Calculation of the Relative Strength Index

### **Description**

▶ **RSI**[count](price)

The function *RSI* calculates the Relative Strength Index of *price* on the *count* previous bars.

### **Example RSI**

REM Calculation of the Relative Strength Index

```
myRSI = RSI[14](close)
```

```
RETURN myRSI
```

## **SAR**

Calculation of the Parabolic SAR

### **Description**

#### ▶ **SAR**

The function *SAR* calculates the Parabolic SAR.

### **Example SAR**

```
REM Calculation of the Parabolic SAR
```

```
mySAR = SAR
```

```
RETURN mySAR
```

## ***SMI***

Calculation of the Stochastic Momentum Index indicator

### ***Description***

▶ **SMI**[p,q,r](price)

The function *SMI* calculates the Stochastic Momentum Index on *price*.

### **Example SMI**

```
REM Calculation of the Stochastic Momentum Index[14,3,5]
```

```
mySMI = SMI[14,3,5](close)
```

```
RETURN mySMI
```

### ***SmoothedStochastic***

Calculation of the smoothed Stochastic

#### ***Description***

▶ **SmoothedStochastic**[p,q](price)

The function *SmoothedStochastic* calculates the smoothed Stochastic on *price*.

#### **Example SmoothedStochastic**

```
REM Calculation of the smoothed Stochastic 14, 3
```

```
mySmoothedStochastic = SmoothedStochastic[14,3](close)
```

```
RETURN mySmoothedStochastic
```

## **STD**

Calculation of the standard deviation

### **Description**

▶ **STD**[count](price)

The function *STD* calculates the standard deviation on *price* on the *count* previous bars.

### **Example STD**

REM Calculation of the standard deviation

```
mySTD = STD[20](close)
```

```
RETURN mySTD
```

## **STE**

Calculation of the error deviation

### **Description**

▶ **STE**[count](price)

The function *STE* calculates the error deviation using *price* on the *count* previous bars.

### **Example STE**

REM Calculation of the error deviation indicator

```
mySTE = STE[10](close)
```

```
RETURN mySTE
```

## ***Stochastic***

Calculation of the Stochastic %K indicator

### ***Description***

▶ **Stochastic**[p,q](price)

The function *Stochastic* calculates the %K of *price*.

### **Example Stochastic**

```
REM Calculation of the Stochastic 14,3
```

```
myStochastic = Stochastic[14,3](close)
```

```
RETURN myStochastic
```

## ***SuperTrend***

Calculation of the O.Seban's SuperTrend indicator

### ***Description***

- ▶ **SuperTrend**[x: coef, y: count of bars](price)

The function *SuperTrend* calculates the O.Seban's trend following indicator.

### **Example SuperTrend**

```
REM Calculation of the O.Seban's Super Trend indicator
```

```
myTrend = SuperTrend[2, 10](close)
```

```
RETURN myTrend
```

## **TR**

Calculation of the True Range indicator

### **Description**

▶ **TR(price)**

The function *TR* calculates the «True Range» indicator calculated on the *count* previous bars of *price*.

### **Example TR**

```
REM Calculation of the True Range
```

```
myTrueRange = TR(close)
```

```
RETURN myTrueRange
```

## **TRIX**

Calculation of the TRader Index indicator

### **Description**

▶ **TRIX**[count](price)

The function *TRIX* calculates the TRader Index indicator calculated on the *count* previous bars of *price*.

### **Example TRIX**

```
REM Calculation of the TRader Index
```

```
myTRIX = TRIX[15](close)
```

```
RETURN myTRIX
```

### ***Variation***

Calculation of the price variation (in %)

### ***Description***

▶ **Variation**(price)

The function *Variation* calculates the *price* variation.

### **Example Variation**

```
REM Calculation of the price variation in %
```

```
myVariation = Variation(close)
```

```
RETURN myVariation
```

## ***Volatility***

Calculation of the Chaikin Volatility indicator

### ***Description***

▶ **Volatility**[p,q]

The function *Volatility* calculates the Chaikin Volatility indicator using the length of the moving average and the number of bars used as parameters.

### **Example Volatility**

```
REM Calculation of the Chaikin Volatility (MA10, and on 10 bars)
```

```
myVolatility = Volatility[10,10]
```

```
RETURN myVolatility
```

## ***VolumeOscillator***

Calculation of the Volume Oscillator indicator

### ***Description***

▶ **VolumeOscillator**[p,q]

The function *VolumeOscillator* calculates the Volume Oscillator indicator using the length of the short and the long moving average as parameters.

### **Example VolumeOscillator**

```
REM Calculation of the Volume Oscillator (MA5 short term, MA25 long term)
```

```
myOscillator = VolumeOscillator[5,25]
```

```
RETURN myOscillator
```

## **VolumeROC**

Calculation of the Volume Rate Of Change indicator

### **Description**

▶ **VolumeROC**[count]

The function *VolumeROC* calculates the Volume Rate Of Change indicator on the *count* previous bars.

### **Example VolumeROC**

```
REM Calculation of the Volume Rate Of Change
```

```
myRate = VolumeROC[12]
```

```
RETURN myRate
```

## ***Williams***

Calculation of the Williams %R indicator

### ***Description***

▶ **Williams**[count](price)

The function *Williams* calculates Williams %R indicator using the *count* latest bars of *price*.

### **Example Williams**

```
REM Calculation of the Williams%R
```

```
myWilliams = Williams[14](close)
```

```
RETURN myWilliams
```

### ***WilliamsAccumDistr***

Calculation of the Williams Accumulation Distribution indicator

#### ***Description***

▶ **WilliamsAccumDistr**(price)

The function *WilliamsAccumDistr* calculates the Williams Accumulation Distribution indicator on *price*.

#### **Example WilliamsAccumDistr**

```
REM Calculation of the Williams Accumulation Distribution
```

```
myWilliams = WilliamsAccumDistr(close)
```

```
RETURN myWilliams
```

## ***ZigZag, ZigZagPoint***

Calculation of the ZigZag indicator

### ***Description***

- ▶ **ZigZag**[var](price)
- ▶ **ZigZagPoint**[var](price)

The function *ZigZag* calculates the ZigZag indicator on *price* (parameter: **var%**).

### **Example ZigZag**

REM Calculation of the ZigZag 10% indicator

```
myZigzag = ZigZag[10](close)
```

```
RETURN myZigzag
```

The function *ZigZagPoint* calculates the ZigZag indicator on *price* (parameter: **var pts**).

### **Example ZigZagPoint**

REM Calculation of the ZigZag 10pts indicator

```
myZigzag = ZigZagPoint[10](close)
```

```
RETURN myZigzag
```