



PROGRAMMING GUIDE

Indicators & Basic Functions (ProBuilder)

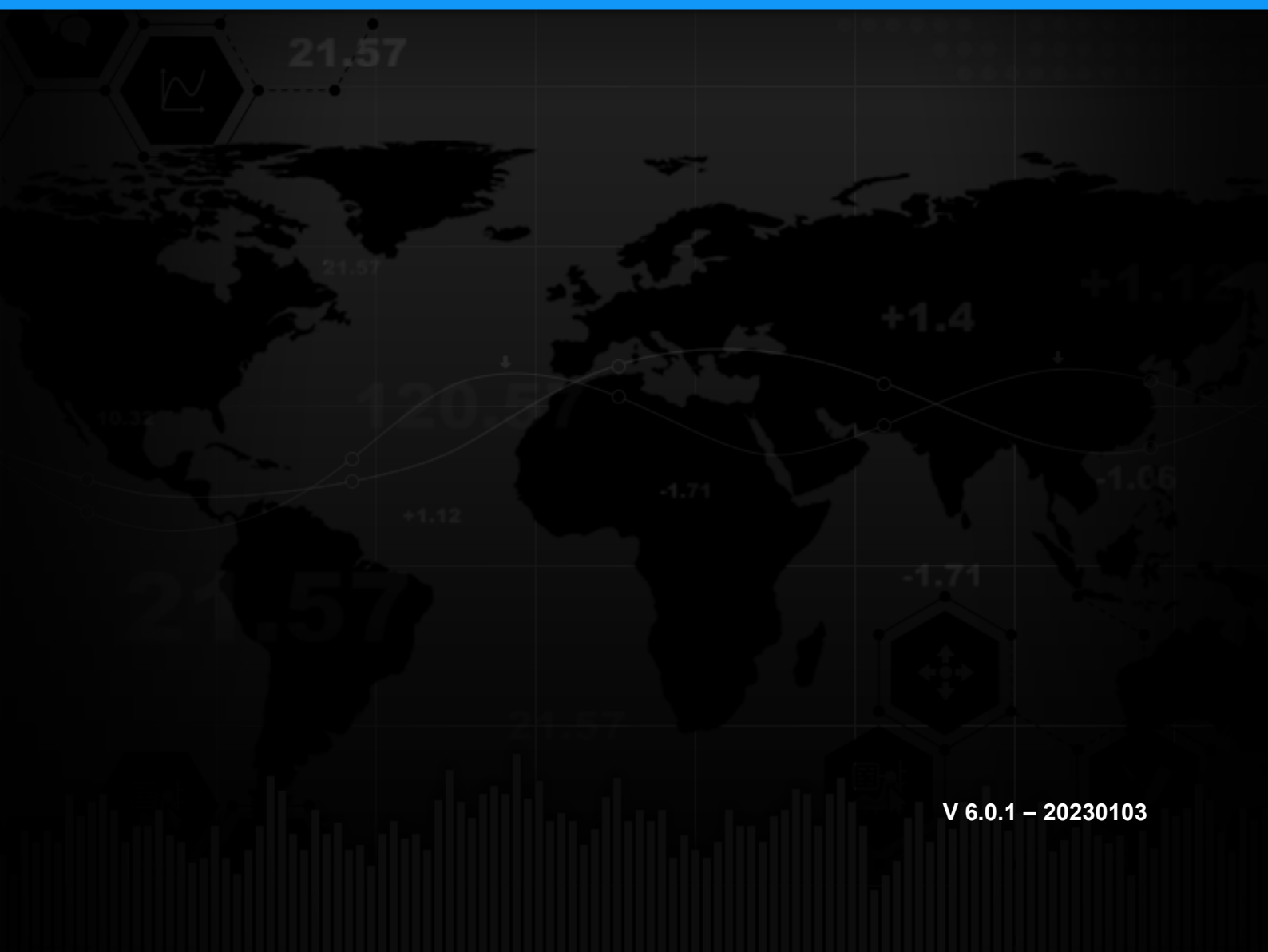





TABLE OF CONTENTS

 Introduction to ProBuilder	1
 Chapter I: Fundamentals	2
➡ Using ProBuilder.....	2
> Indicator creation quick tutorial.....	2
> Programming window keyboard shortcuts.....	5
➡ Specificities of ProBuilder programming language.....	6
➡ Financial constants.....	7
> Price and volume constants adapted to the timeframe of the chart.....	7
> Daily price constants.....	8
> Temporal constants.....	8
> Constants derived from price.....	12
> The Undefined constant.....	13
➡ How to use pre-existing indicators?.....	13
➡ Adding customizable variables.....	16
 Chapter II: Math Functions and ProBuilder instructions	18
➡ Control Structures.....	18
> Conditional IF instruction.....	18
• One condition, one result (IF THEN ENDIF).....	18
• One condition, two results (IF THEN ELSE ENDIF).....	18
• Sequential IF conditions.....	18
• Multiple conditions (IF THEN ELSE ELSIF ENDIF).....	19
> Iterative FOR Loop.....	21
• Ascending loop (FOR, TO, DO, NEXT).....	21
• Descending loop (FOR, DOWNTO, DO, NEXT).....	22
> Conditional WHILE Loop.....	23
> BREAK.....	24
• With WHILE.....	24
• With FOR.....	24
> CONTINUE.....	25
• With WHILE.....	25
• With FOR.....	25
> ONCE.....	26
➡ Mathematical Functions.....	27
> Common unary and binary Functions.....	27
> Common mathematical operators.....	27

> Charting comparison functions.....	27
> Summation functions.....	28
> Statistical functions.....	28
➡ Logical operators.....	28
➡ ProBuilder instructions.....	29
> RETURN.....	29
> Comments.....	29
> CustomClose.....	29
> CALCULATEONLASTBARS.....	29
> CALL.....	30
> AS.....	30
> COLOURED.....	30
➡ Drawing instructions.....	32
> Additional parameters.....	35
➡ Multi-period instructions.....	38
> List of available time frames.....	40
➡ Arrays (Data tables).....	41
> Specific functions.....	41

Chapter III: Practical aspects.....43

➡ Create a binary or ternary indicator : why and how ?.....	43
➡ Creating stop indicators to follow a position.....	44
> Static Take Profit STOP.....	45
> Static STOP loss.....	45
> Inactivity STOP.....	46
> Trailing Stop.....	47

Chapter IV: Exercises.....48

➡ Candlesticks patterns.....	48
➡ Indicators.....	49

Glossary.....51

Warning: ProRealTime does not provide investment advisory services. This document is not in any case personal or financial advice nor a solicitation to buy or sell any financial instrument. The example codes shown in this manual are for learning purposes only. You are free to determine all criteria for your own trading. Past performance is not indicative of future results. Trading systems may expose you to a risk of loss greater than your initial investment.

Introduction to ProBuilder

ProBuilder is ProrealTime's programming language. It allows you to create personalized technical indicators, trading strategies (ProBacktest) or screening programs (ProScreener). Two specific manuals exist for ProBacktest and ProScreener due to some specific characteristics of each of these modules.

ProBuilder is a BASIC-type programming language, very easy to handle and exhaustive in terms of available possibilities.

You will be able to create your own programs using the quotes from any instrument provided by ProRealTime. Some basic available elements include:

- Opening of each bar: Open
- Closing of each bar: Close
- Highest price of each bar: High
- Lowest price of each bar: Low
- Volume of each bar: Volume

Bars or candlesticks are the common charting representations of real time quotes. Of course, ProRealTime offers you the possibility of personalizing the style of the chart. You can use Renko, Kagi, Heikin-Ashi and many other styles.

ProBuilder evaluates the data of each price bar starting with the oldest one to the most recent one, and then executes the formula developed in the language in order to determine the value of the indicators on the current bar.

The indicators coded in ProBuilder can be displayed either in the price chart or in an individual one.

In this document, you will learn, step by step, how to use the available commands necessary to program in this language thanks to a clear theoretical overview and concrete examples.

In the end of the manual, you will find a Glossary which will give you an overall view of all the ProBuilder commands, pre-existing indicators and other functions completing what you would have learned after reading the previous parts.

Users more confident in their programming skills can skip directly to chapter II or just refer to the Glossary to quickly find the information they want.

For those who are less confident, we recommend watching our video tutorial entitled "[Programming simple and dynamic indicators](#)" and reading the whole manual.

If you have any questions about ProBuilder, you can ask them to our ProRealTime community on the [ProRealCode forum](#), where you will also find an [online documentation](#) with many examples.

We wish you success and hope you will enjoy the manual!

The ProRealTime team

Chapter I: Fundamentals

Using ProBuilder

Indicator creation quick tutorial

The indicator programming area is available from the "Indicator" button located in upper left corner of each chart in your ProRealTime platform or from the menu Display > Indicators/Backtest.

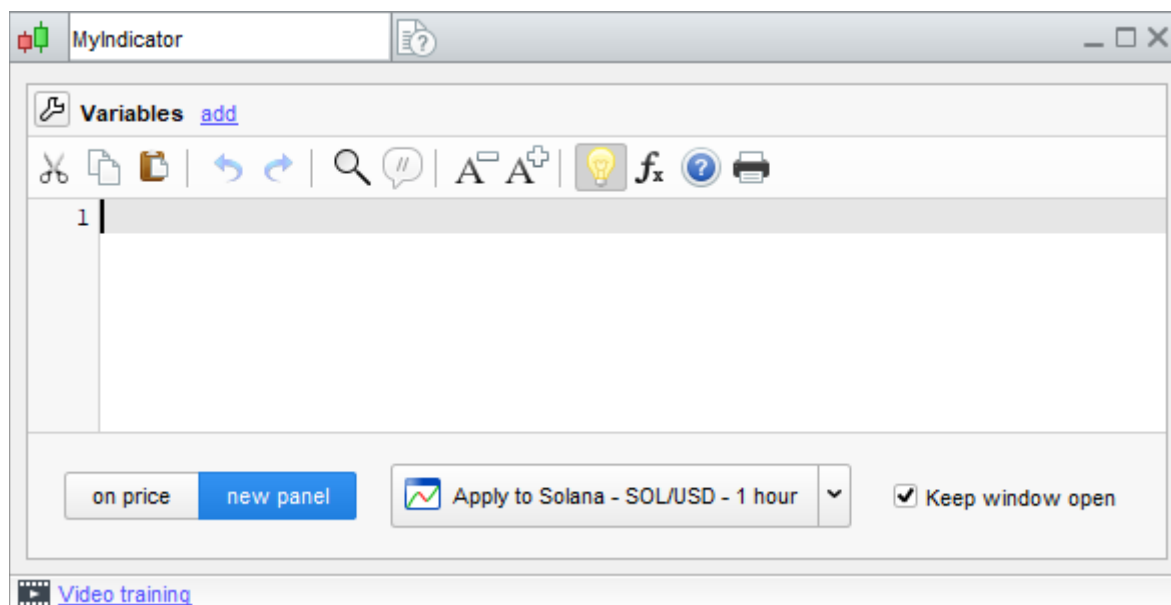
The indicators management window will be displayed. You will then be able to:

- Display a pre-existing indicator
- Create a personalized indicator, which can be used afterwards on any security

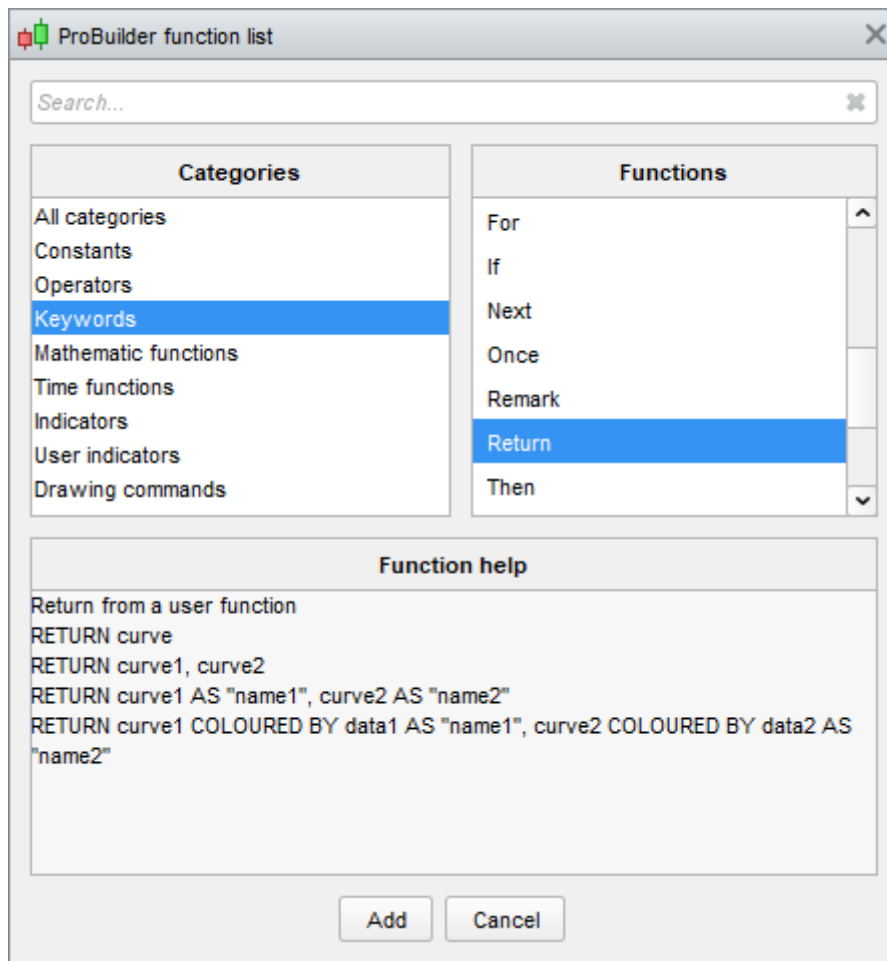
If you choose the second possibility, click on "Create" to access the programming window.

At that time, you will be able to choose between:

- Programming directly an indicator in the text zone designed for writing code or
- Using the help function by clicking on "Insert Function" (fx icon), this will open a new window in which you can find all the functions available. This library is separated in 8 categories, to give you constant assistance while programming.



Let's take for example the first ProBuilder key element: the **"RETURN"** function (available in the "ProBuilder function list" f_x (see the image below).

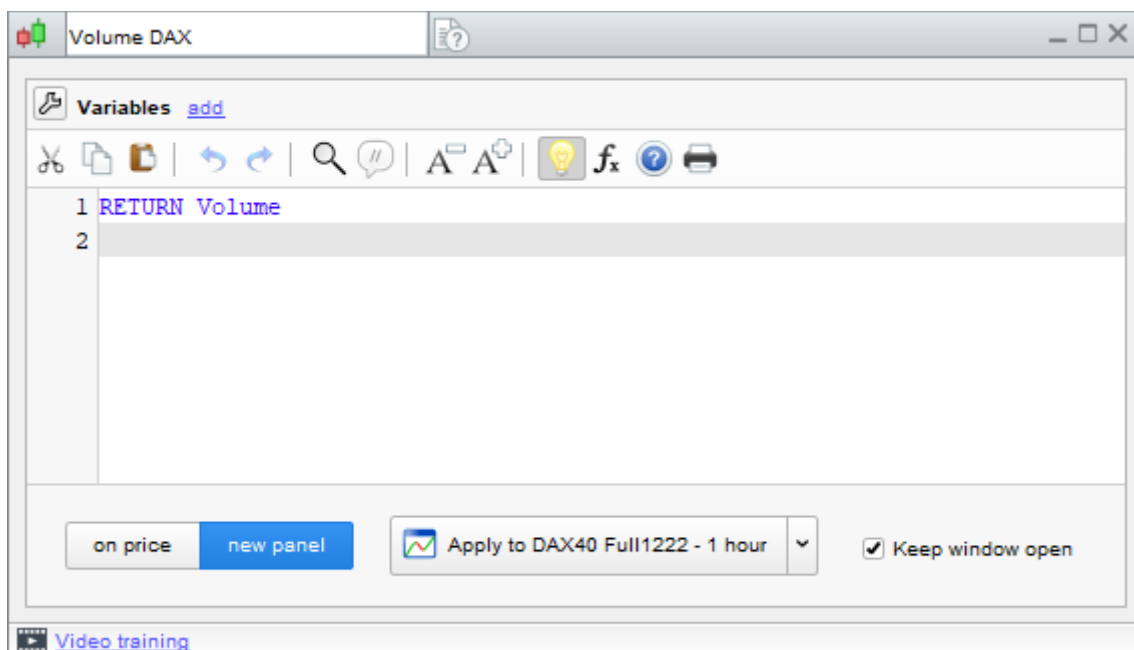


Select the word **"RETURN"** and click on "Add". The command will be added to the programming zone.



RETURN allows you to display the result

Suppose we want to create an indicator displaying the Volume. If you have already inserted the function "RETURN", then you just need to click one more time on "Insert function". Next, click on "Constants" in the "Categories" section, then in the right side of the window, in the section named "Functions", click on "Volume". Finally, click on "Add". Don't forget to add a space in between each instruction as shown below.



Before clicking on the "Apply to DAX40" button, specify at the top of the window the name of your indicator: here we have called it "DAX Volume". Finally, click on "Apply to DAX" and you will see the chart with your indicator.



Programming window keyboard shortcuts

The programming window has a number of useful features that can be accessed by keyboard shortcuts:

- Select all (Ctrl + A): Select all text in the programming window
- Copy (Ctrl + C): Copy the selected text
- Paste (Ctrl + X): Paste copied text
- Undo (Ctrl + Z): Undo the last action in the programming window
- Redo (Ctrl + Y): Redo the last action in the programming window
- Find / Replace (Ctrl + F): Find a text in the programming window / replace a text in the programming window
- Comment / Uncomment (Ctrl + R): Comment the selected code / Uncomment the selected code (commented code will be preceded by "//" and colored grey. It will not be taken into account when the code is executed).
- Auto-complete (Ctrl+Space): Allows you to display suggested instructions or keywords

For Mac users, the same keyboard shortcuts can be accessed with the "Command" key in place of the "Ctrl" key. Most of these features can also be accessed by right-clicking in the programming window.

Specificities of ProBuilder programming language

Specificities

The ProBuilder language allows you to use many classic commands as well as sophisticated tools which are specific to technical analysis. These commands will be used to program from simple to very complex indicators.

The main ideas to know in the ProBuilder language are:

- It is **not necessary to declare variables**
- It is **not necessary to type variables**
- There is **no difference between capital letters and small letters**
- **We use the same symbol "=" for mathematical equality and to attribute a value to a variable**

What does this mean?

- Declaring a variable X means indicating its existence. In ProBuilder, you can directly use X without having to declare it. Let's take an example:

With declaration: let be variable X, we attribute to X the value 5

Without declaration: We attribute to X the value 5 (therefore, implicitly, X exists and the value 5 is attributed to it)

In ProBuilder you just need to write: X=5

- Typing a variable means defining its nature. For example: is the variable an integer (ex: 3; 8; 21; 643; ...), a decimal number (ex: 1.76453534535...), a boolean (RIGHT=1, WRONG=0),...?
- In ProBuilder, you can write your command with capital letters or small letters. For example, the group of commands IF / THEN / ELSE / ENDIF can be written iF / tHeN / ELse / endIf (and many other possibilities!)
- Affect a value to a variable means give the variable a value. In order to understand this principle, you must assimilate a variable with an empty box which you can fill with an expression (ex: a number). The following diagram illustrate the Affectation Rule with the Volume value affected to the variable X:

X ← Volume

As you can see, we must read from right to left: Volume is affected to X.

If you want to write it under ProBuilder, you just need to replace the arrow with an equal sign:

X = Volume

The same = symbol is used:

- For the affectation of a variable (like the previous example)
- As the mathematical comparison operator (1+ 1= 2 is equivalent to 2 = 1 + 1).

Financial constants

Before coding your personal indicators, you must examine the elements you need to write your code such as the opening price, the closing price, etc.

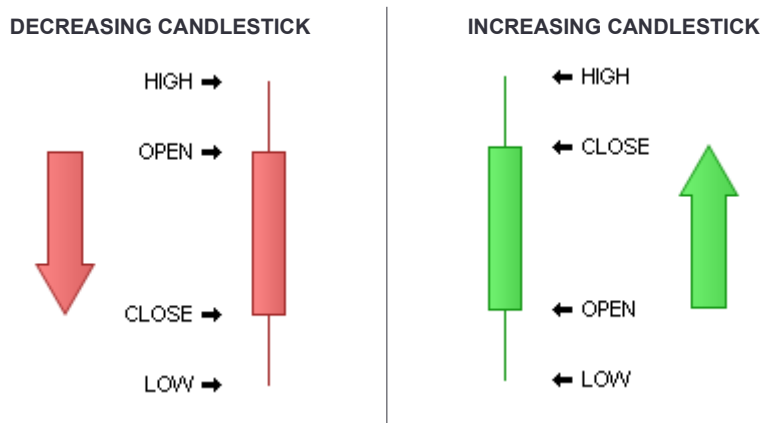
These are the "fundamentals" of technical analysis and the main things to know for coding indicators.

You will then be able to combine them in order to draw out some information provided by financial markets. We can group them together in 5 categories:

Price and volume constants adapted to the timeframe of the chart

These are the "classical" constants and also the ones used the most. They report by default the value of the current bar (whatever the timeframe used).

- **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
- **Volume:** The number of securities or contracts exchanged during the current bar



Example: Range of the current bar

```
a = High
b = Low
MyRange = a - b
RETURN MyRange
```

If you want to use the information of previous bars rather than the current bar, you just need to add between square brackets the number of bars that you want to go back into the past.

Let's take for example the closing price constant. Calling the the price is done in the following way:

Value of the closing price of the current bar: `Close`

Value of the closing price of the bar preceding the current bar: `Close[1]`

Value of the closing price of the nth bar preceding the current one: `Close [n]`

This rule is valid for any constant. For example, the opening price of the 2nd bar preceding the current can be expressed as: `Open[2]`.

The reported value will depend on the displayed timeframe of the chart.

Daily price constants

Contrary to the constants adapted to the timeframe of the chart, the daily price constants refer to the value of the day, regardless of the timeframe of the chart.

Another difference between Daily price constants and constants adapted to the timeframe of the chart is that the daily price constants use parentheses and not square brackets to call the values of previous bars.

- **DOpen(n)**: Opening price of the nth day before the one of the current bar
- **DHigh(n)**: Highest price of the nth day before the one of the current bar
- **DLow(n)**: Lowest price of the nth day before the one of the current bar
- **DClose(n)**: Closing price of the nth day before the one of the current bar

Note: if "n" is equal to 0, "n" refers to the current day. The maximum and minimum values are not yet definitive for n=0, we will obtain results which can change during the day depending on the minimum and maximum reached by the value.



The constants adapted to the timeframe of the chart use square brackets while the daily price constants use brackets.

`Close[3]` ➔ The closing price 3 periods ago

`Dclose(3)` ➔ The closing price 3 days ago

Temporal constants

Time is often a neglected component of technical analysis. However traders know very well the importance of some time periods in the day or dates in the year. It is possible in your programs to take into account time and date and improve the efficiency of your indicators. The Temporal constants are described hereafter:

- **Date**: indicates the date of the close of each bar in the format YearMonthDay (YYYYMMDD)

Temporal constants are considered by ProBuilder as whole numbers. The Date constant, for example, must be used as one number made up of 8 figures.

Let's write down the program:

```
RETURN Date
```

Suppose today is July 4th, 2020. The program above will return the result 20200704.

The date can be read in the following way:

20200704 = 2020 years 07 months and 04 days.

Note that when writing a date in the format YYYYMMDD, MM must be between 01 and 12 and DD must be between 01 and 31.

- **Time:** indicates the time of closing of each bar in the format HHMMSS (HourMinuteSecond)

Example:

```
RETURN Time
```

This indicator shows us the closing time of each bar in the format HHMMSS:



Time can be read as follows:

160000 = 16 hours 00 minutes and 00 seconds.

Note that when writing a time in the format HHMMSS, HH must be between 00 and 23, MM must be between 00 and 59 and SS must be also between 00 and 59.

It is also possible to use **Time** and **Date** in the same indicator to do analysis or display results at a precise moment. In the following example, we want to limit our indicator to the date of October 1st at precisely 9am and 1 second:

```
a = (Date = 20081001)
```

```
b = (Time = 090001)
```

```
RETURN (a AND b)
```

The following constants work the same way:

- **Timestamp:** [UNIX](#) date and time (number of seconds since January 1st, 1970) of the close of each bar.
- **Second:** Second of the close of each bar (between 0 and 59).
- **Minute:** Minute of the close of each bar (from 0 to 59): Only for intraday charts.
- **Hour:** Hour of the close of each bar (from 0 to 23): Only for intraday charts.
- **Day:** Day of the months of the closing price of each bar (from 1 to 28 or 29 or 30 or 31)
- **Month:** Month of the closing price of each bar (from 1 to 12)
- **Year:** Year of the closing price of each bar
- **DayOfWeek:** Day of the Week of the close of each bar (0=Sunday, 1=Monday, 2=Tuesday, 3=Wednesday, 4=Thursday, 5=Friday, 6=Saturday)

Derivative constants also exist for **Open** :

- **OpenTimestamp**: [UNIX](#) date and time of the open of each bar.
- **OpenSecond** : Second of the open of each bar (between 0 and 59).
- **OpenMinute** : Minute of the open of each bar (between 0 and 59).
- **OpenHour** : Time of the open of each bar (between 0 and 23).
- **OpenDay**: Day of the month of the open of each bar (between 1 and 28 or 29 or 30 or 31).
- **OpenMonth** : Month of the open of each bar (between 1 and 12).
- **OpenYear**: Year of the open of each bar.
- **OpenDayOfWeek**: Day of the week at the open of each bar (0=Sunday,1=Monday, 2=Tuesday, 3=Wednesday, 4=Thursday, 5=Friday,6=Saturday).
- **OpenTime**: HourMinuteSecond encoded as HHMMSS indicating the opening time of each bar.
- **OpenDate**: Date (YYYYMMDD) of the open of the current bar.

Example of the use of these constants:

```
a = (Hour > 17)
b = (Day = 30)
RETURN (a AND b)
```

- **CurrentHour** : Current time (market time).
- **CurrentMinute**: Current minute (of the local market).
- **CurrentMonth**: Current month (of the local market)
- **CurrentSecond**: Current second (of the local market)
- **CurrentTime**: Current HourMinuteSecond (of the local market)
- **CurrentYear**: Current Year (of the local market)
- **CurrentDayOfWeek**: Current Day of the week with the market time zone as a reference

The difference between the "Current" constants and the "non-Current" constants presented above is the "Current" aspect.

The following picture brings to light that difference (applied on the **CurrentTime** and **Time** constants). We can highlight the fact that for "Current" constants, we must set aside the time axis and only take in consideration the displayed value (the value of the current time is displayed over the whole history of the chart).



Time indicates the closing time of each bar.

CurrentTime indicates the current market time.

If you want to set up your indicators with counters (number of days passed, number of bars passed etc...), you can use the Days, BarIndex and IntradayBarIndex constants.

- **Days:** Counter of days since 1900

This constant is quite useful when you want to know the number of days that have passed. It is particularly relevant when you work with an (x) tick or (x) volume view.

The following example shows you the number of days passed since 1900.

`RETURN Days`

(Be careful not to confuse the constants "Day" and "Days").

- **BarIndex:** Counter of bars since the beginning of the displayed historical data

The counter starts from left to right and counts each bar, including the current bar. The first bar loaded is considered bar number 0. Most of the time, **BarIndex** is used with the **IF** instruction presented later in the manual.

- **IntradayBarIndex:** Counter of intraday bars

The counter displays the number of bars since the beginning of the day and then resets to zero at the beginning of every new day. The first bar of the counter is considered bar number 0.

Let's compare the two counter constants with two separated indicators:

```
RETURN BarIndex
```

and

```
RETURN IntradayBarIndex
```



We can clearly see the difference between them: **IntradayBarIndex** resets itself to zero at the beginning of every new day.

Constants derived from price

These constants allows you to get more complete information compared to **Open**, **High**, **Low** and **Close**, since they combine those prices so to emphasize some aspects of the financial market psychology shown on the current bar.

- **Range**: difference between High and Low.
- **TypicalPrice**: average between High, Low and Close
- **WeightedClose**: weighted average of High (weight 1), Low (weight 1) and Close (weight 2)
- **MedianPrice**: average between High and Low
- **TotalPrice**: average between Open, High, Low and Close

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

WeightedClose focuses on the importance of the closing price.

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

MedianPrice is the median price of the candlestick, calculated by computing the average of the High and Low.

Range in %:

```
MyRange = Range
```

```
Calcul = (MyRange / MyRange[1] - 1) * 100
```

```
RETURN Calcul
```


The Undefined constant

The keyword **Undefined** allows you to indicate to the software not to display the value of the indicator.

- **Undefined:** undefined data (equivalent to an empty box)

You can find an example later in the manual.

How to use pre-existing indicators?

Up until now, we have described you the possibilities offered by ProBuilder concerning constants and how to call values of bars of the past using these constants. Pre-existing indicators (the ones already programmed in ProRealTime) function the same way and so do the indicators you will code.

ProBuilder indicators are made up of three elements which syntax is:

```
NameOfFunction [calculated over n periods] (applied to which price or indicator)
```

When using the "Insert Function" button to look for a ProBuilder function and then enter it into your program, default values are given for both the period and the price or indicator argument. Example for a moving average of 20 periods :

```
Average[20] (Close)
```

The values can be modified. For example, we can replace the 20 bars defined by default with any number of bars (ex: Average[10], Average[15], Average[30], ..., Average[n]). In the same way, we can replace "Close" with "Open" or **RSI (Relative strength index)**. This would give us for example:

```
Average[20] (RSI[5] (Close) )
```

Here are some sample programs:

Program calculating the exponential moving average over 20 periods applied to the closing price:

```
RETURN ExponentialAverage[20] (Close)
```

Program calculating the weighted moving average over 20 bars applied to the typical price

```
mm = WeightedAverage[20] (TypicalPrice)
RETURN mm
```

Program calculating the Wilder average over 100 candlesticks applied to the Volume

```
mm = WilderAverage[100] (Volume)
RETURN mm
```

Program calculating the MACD (histogram) applied to the closing price.

The MACD is built with the difference between the 12-period exponential moving average (EMA) minus the 26-period EMA. Then, we make a smoothing with an exponential moving average over 9 periods and applied to the MACD line to get the Signal line. Finally, the MACD is the difference between the MACD line and the Signal line.

```
// Calculation of the MACD line
MACDLine = ExponentialAverage[12] (Close) - ExponentialAverage[26] (Close)
// Calculation of the MACD Signal line
MACDSignalLine = ExponentialAverage[9] (MACDLine)
// Calculation of the difference between the MACD line and its Signal
MACDHistogram = MACDLine - MACDSignalLine
RETURN MACDHistogram
```

Calculation of an average with two parameters

You also have the possibility to use a second parameter with the average function (which indicates the type of average to use). We obtain the following formula :

Average [Nbr. of periods, Type of average]

The parameter Type of average designates, as its name indicates, the type of average that will be used. There are 9 of them and they are indexed from 0 to 8 :

0=Single	4=Triangular	8=Zero delay
1=Exponential	5=Least Squares	
2=Weighted	6=Time series	
3=Wilder	7=Hull	

Calculation of Ichimoku lines




Since the Ichimoku indicator includes many lines, some of these lines have been introduced separately in the ProBuilder language to allow you to get the most out of this indicator.

The lines as follows:

```

SenkouSpanA[TenkanPeriod,KijunPeriod,Senkou-SpanBPeriod]
TenkanSen [TenkanPeriod,KijunPeriod,Senkou-SpanBPeriod]
KijunSen[TenkanPeriod,KijunPeriod,Senkou-SpanBPeriod]
SenkouSpanB[TenkanPeriod,KijunPeriod,Senkou-SpanBPeriod]
```

With for each line the usual Ichimoku parameters:

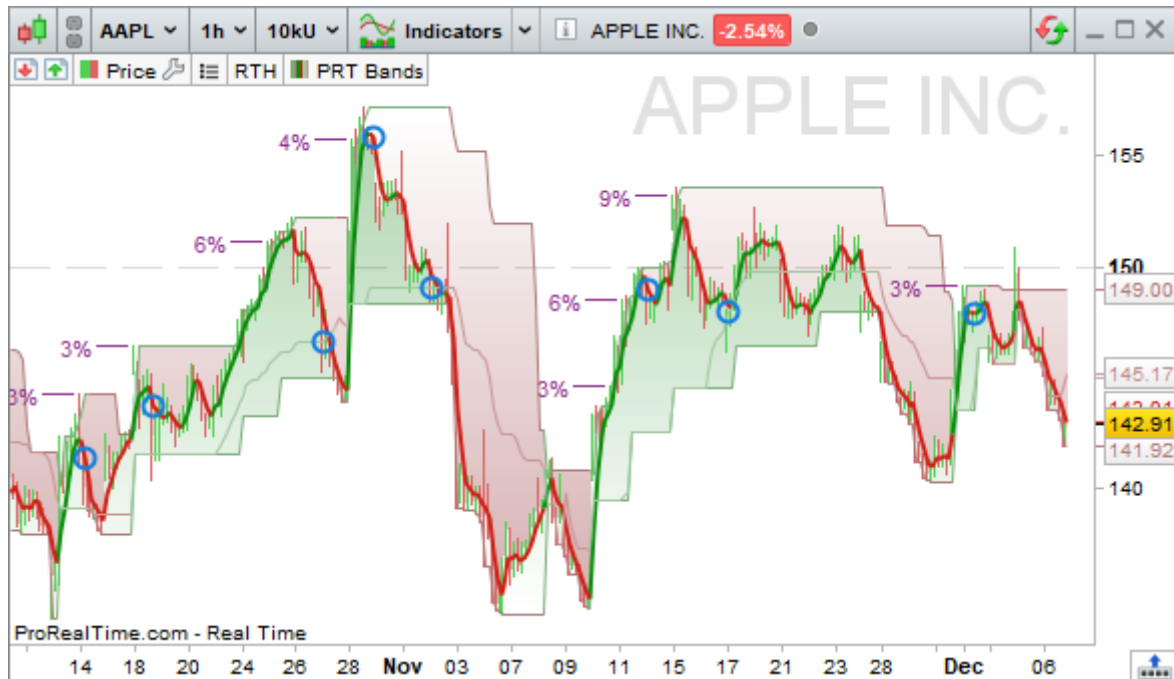
-  **Tenkan**: alert line, (high point + low point)/2 over the last *n* periods
-  **Kijun**: signal line, (high point + low point)/2 over the last *n* periods
-  **SenkouSpanB**: long term average point projection, (high point + low point)/2 over the last *n* periods

Calculation of PRT Bands

PRT Bands is a visual indicator that simplifies the detection and monitoring of trends. It is exclusive to the ProRealTime platform.

It can help you to:

- detect a reversal of trends
- identify and follow an upward trend
- measure the intensity of the trend
- find potential entry and exit points



Here are the different PRT Bands data available in the ProBuilder language:

- **PRTBANDSUP**: returns the value of the top line of the indicator
- **PRTBANDSDOWN**: returns the value of the bottom line of the indicator
- **PRTBANDSSHORTTERM**: returns the value of the short term (thick) line of the indicator
- **PRTBANDSMEDIUMTERM**: returns the value of the medium term (thin) line of the indicator

[Learn more about the PRT Bands indicator](#)

Adding customizable variables

When you code an indicator, you may want to use customizable variables. The variables option in the upper-left corner of the window allows you to assign a default value to an undefined variable in your program and change its value in the settings window of the indicator without modifying the code of your program.

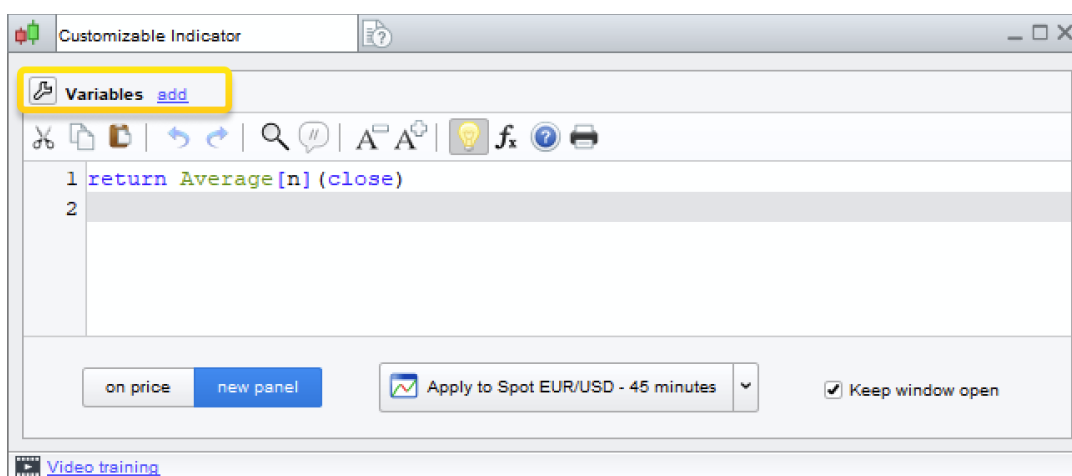
Let's calculate a simple moving average on 20 periods:

```
RETURN Average[20] (Close)
```

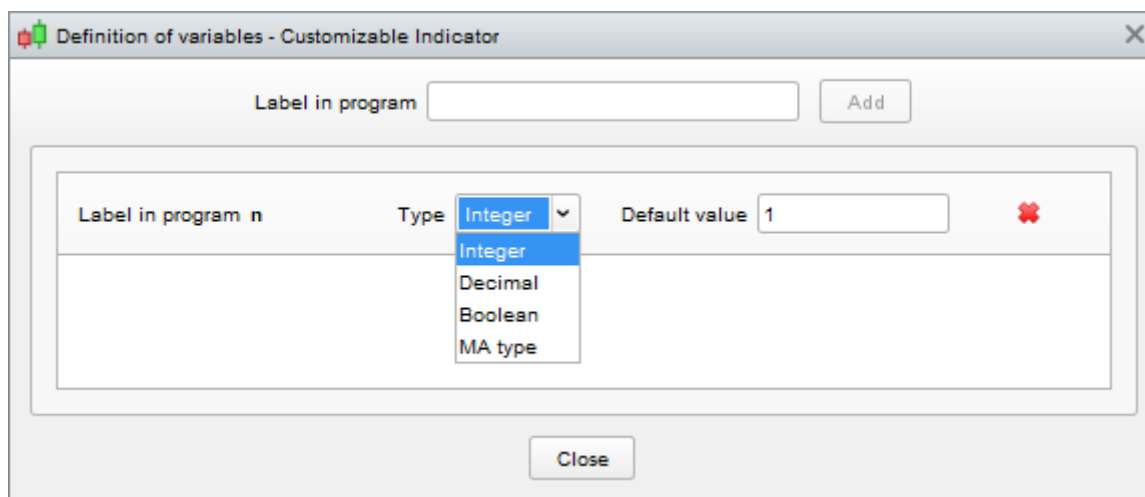
In order to modify the number of periods for the calculation directly from the indicator "Settings" interface, replace 20 with the variable "n":

```
RETURN Average[n] (Close)
```

Then, click on "Add" next to "Variables" and another window named "Variable definition" will be displayed.

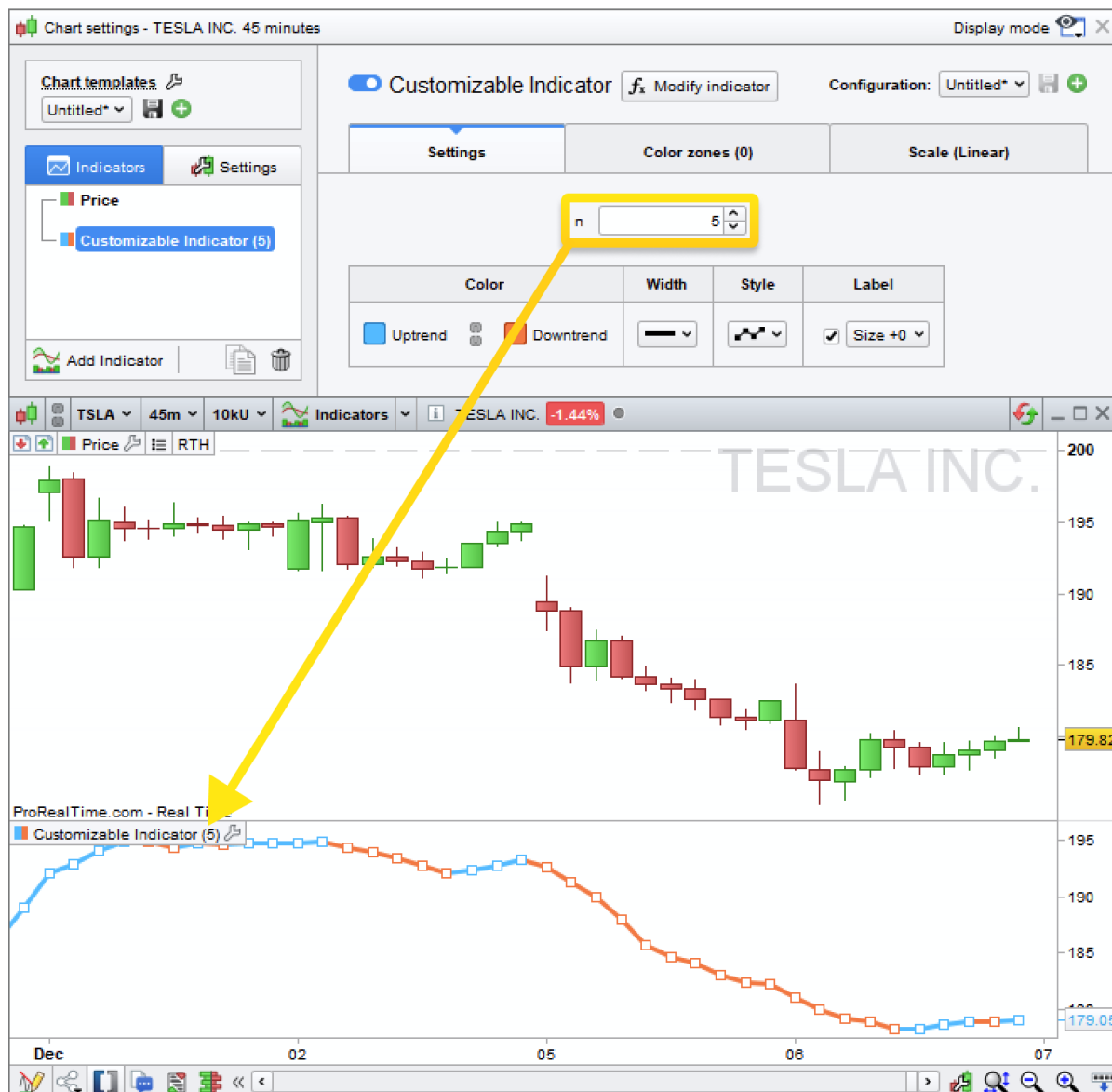


Enter the name of your variable, here " n " and click on " Add ", you can then fill in a Type and a Default Value as shown in the example below :







Click on the "Close" button.

In the "Settings" tab you will see a new parameter which will allow you to modify the number of periods used in the calculation of the moving average:



List of available types for variables:

-  **Integer:** integer between -2,000,000,000 and 2,000,000,000 (ex: 450)
-  **Decimal:** decimal number with a precision of 5 significant digits (ex: 1.03247)
-  **Boolean:** True (1) or False (0)
-  **Moving Average Type:** Allows you to set the value of the second parameter which defines the type of moving average used in the calculation of the **Average** indicator (see above).

Of course, it is possible to create many variables giving you the possibility to manipulate multiple parameters at the same time.

Chapter II: Math Functions and ProBuilder instructions

Control Structures

Conditional IF instruction

The **IF** instruction is used to make a choice of conditional actions, ex: to make a result dependent on the verification of one or more defined conditions.

The structure is made up of the instructions **IF**, **THEN**, **ELSE**, **ELSIF**, **ENDIF**, which are used depending on the complexity of the conditions you defined.

One condition, one result (IF THEN ENDIF)

We can look for a condition and define an action if that condition is true. On the other hand, if the condition is not valid, then nothing will happen (By default, Result = 0).

In this example, if current price is greater than the 20-period moving average, then we display: Result = 1 and display this on the chart.

<code>Result = 0</code>	Result is equal to 0.
<code>IF Close > Average[20](Close) THEN</code>	IF closing price > 20-period moving average
<code>Result = 1</code>	THEN Result = 1, otherwise Result is unchanged
<code>ENDIF</code>	END OF CONDITION
<code>RETURN Result</code>	



RETURN must always be followed with the storage variable containing the result in order to display the result on the chart (in the last example we use the variable "Result").

One condition, two results (IF THEN ELSE ENDIF)

We can also define a different result if the condition is not true. Let us go back to the previous example: if the price is greater than the moving average on 20 periods, then display 1, else, displays -1.

```
IF Close > Average[20](Close) THEN
    Result = 1
ELSE
    Result = -1
ENDIF
RETURN Result
```

NB: We have created a binary indicator. For more information, see the section on binary and ternary indicators later in this manual.

Sequential IF conditions

You can create sub-conditions after the validation of the main condition, meaning conditions which must be validated one after another. For that, you need to build a sequence of **IF** structures, one included in the other. You should be careful to insert in the code as many **ENDIF** as **IF**. Example:

Double conditions on moving averages:

```
IF (Average[12](Close) - Average[20](Close) > 0) THEN
    IF ExponentialAverage[12](Close) - ExponentialAverage[20](Close) > 0 THEN
        Result = 1
    ELSE
        Result = -1
    ENDIF
ENDIF
RETURN Result
```

Multiple conditions (IF THEN ELSE ELSIF ENDIF)

You can define a specific result for a specific condition. The indicator reports many states: if Condition 1 is valid then do Action1; else, if Condition 2 is valid, then do Action 2 ...if none of the previously mentioned conditions are valid then do Action n.

This structure uses the following instructions: **IF, THEN, ELSIF, THEN.... ELSE, ENDIF.**

The syntax is:

```
IF (Condition1) THEN
    (Action1)
ELSIF (Condition2) THEN
    (Action2)
ELSIF (Condition3) THEN
    (Action3)
...
...
...
ELSE
    (Action n)
ENDIF
```

You can also replace **ELSIF** with **ELSE IF** but your program will take longer to write. Of course, you will have to end the loop with as many instance of **ENDIF** as **IF**. If you want to make multiple conditions in your program, we advise you to use **ELSIF** rather than **ELSE IF** for this reason.

Example: detection of bearish and bullish engulfing lines using the Elsif instruction

This indicator displays 1 if a bullish engulfing line is detected, -1 if a bearish engulfing line is detected, and 0 if neither of them is detected.

```
// Detection of a bullish engulfing line
Condition1 = Close[1] < Open[1]
Condition2 = Open < Close[1]
Condition3 = Close > Open[1]
Condition4 = Open < Close
```

```
// Detection of a bearish engulfing line
Condition5 = Close[1] > Open[1]
Condition6 = Close < Open
Condition7 = Open > Close[1]
Condition8 = Close < Open[1]
```

```
IF Condition1 AND Condition2 AND Condition3 AND Condition4 THEN
    a = 1
ELSIF Condition5 AND Condition6 AND Condition7 AND Condition8 THEN
    a = -1
ELSE
    a = 0
ENDIF
RETURN a
```



Example: Resistance Demark pivot

```
IF DClose(1) > DOpen(1) THEN
    Phigh = DHigh(1) + (DClose(1) - DLow(1)) / 2
    Plow = (DClose(1) + DLow(1)) / 2
ELSIF DClose(1) < DOpen(1) THEN
    Phigh = (DHigh(1) + DClose(1)) / 2
    Plow = DLow(1) - (DHigh(1) - DClose(1)) / 2
ELSE
    Phigh = DClose(1) + (DHigh(1) - DLow(1)) / 2
    Plow = DClose(1) - (DHigh(1) - DLow(1)) / 2
ENDIF
RETURN Phigh , Plow
```

Example: BarIndex

In the chapter I of our manual, we presented **BarIndex** as a counter of bars loaded. **BarIndex** is often used with **IF**. For example, if we want to know if the number of bars in your chart exceeds 23 bars, then we will write:

```
IF BarIndex <= 23 THEN
    a = 0
ELSIF BarIndex > 23 THEN
    a = 1
ENDIF
RETURN a
```


Iterative FOR Loop

FOR is used when we want to exploit a finite series of elements. This series must be made up of whole numbers (ex: 1, 2, 3, ..., 6, 7 or 7, 6, ..., 3, 2, 1) and ordered.

Its structure is formed of **FOR**, **TO**, **DOWNTO**, **DO**, **NEXT**. **TO** and **DOWNTO** are used depending on the order of appearance in the series of the elements (ascending order or descending order). We also highlight the fact that what is between **FOR** and **DO** are the extremities of the interval to scan.

Ascending loop (FOR, TO, DO, NEXT)

```
FOR Variable = BeginningValueOfTheSeries TO EndingValueOfTheSeries DO
    (Action)
NEXT
```

Example: Smoothing of a 12-period moving average

Let's create a storage variable (Result) which will sum the 11, 12 and 13-period moving averages.

```
Result = 0
FOR Variable = 11 TO 13 DO
    Result = Result + Average[Variable](Close)
NEXT
// Let's create a storage variable (AverageResult) which will divide Result by 3 and
// display average result. Average result is a smoothing of the 12-period moving average.
AverageResult = Result / 3
RETURN AverageResult
```

Let's see what is happening step by step:

Mathematically, we want to calculate the average the arithmetic moving averages of periods 11, 12 and 13.

Variable will thus take successively the values 11, 12 then 13

```
Result = 0
Variable = 11
```

Result receives the value of the previous Result + MA11 i.e.: $(0) + MA11 = (0 + MA11)$

The **NEXT** instruction takes us to the next value of the counter

```
Variable = 12
```

Result receives the value of the previous Result + MA12 or : $(0 + MA11) + MA12 = (0 + MA11 + MA12)$

The **NEXT** instruction takes us to to the next value of the counter

```
Variable = 13
```

Result receives the value of the previous Result + MA13 or : $(0 + MA11 + MA12) + MA13 = (0 + MA11 + MA12 + MA13)$

The value 13 is the last value of the counter.

NEXT closes the **FOR** loop because there is no more next value.

Result is displayed

This code simply means that **Variable** will initially take the value of the beginning of the series, then **Variable** will take the next value (the previous one + 1) and so on until **Variable** exceeds or is equal to the value of the end of the series. Then the loop ends.

Example: Average of the highest value over the last 5 bars

SUMhigh = 0	
IF BarIndex < 5 THEN	If there are not yet 5 periods displayed
MAhigh = Undefined	Then we attribute to MAhigh value "Undefined" (not displayed)
ELSE	ELSE
FOR i = 0 TO 4 DO	FOR values of i between 0 to 4
SUMhigh = High[i]+SUMhigh	We sum the 5 last "High" values
NEXT	
ENDIF	
MAhigh = SUMhigh / 5	We calculate the average for the last 5 periods and store the result in MAhigh
RETURN MAhigh	We display MAhigh

Descending loop (FOR, DOWNTO, DO, NEXT)

The descending loop uses the following instructions: **FOR**, **DOWNTO**, **DO**, **NEXT**.

Its syntax is:

```
FOR Variable = EndingValueOfTheSeries DOWNTO BeginningValueOfTheSeries DO
    (Action)
NEXT
```

Let us go back to the previous example (the 5-period moving average of "High"):

Note that we have just reversed the limits of the scanned interval.

```
SUMhigh = 0
IF BarIndex < 5 THEN
    MAhigh = Undefined
ELSE
    FOR i = 4 DOWNTO 0 DO
        SUMhigh = High[i] + SUMhigh
    NEXT
ENDIF
MAhigh = SUMhigh / 5
RETURN MAhigh
```

Conditional WHILE Loop

WHILE is used to keep doing actions while a condition remains true. You will see that this instruction is very similar to the simple conditional instruction **IF/THEN/ENDIF**.

This structure uses the following instructions: **WHILE**, (**DO** optional), **WEND** (end **WHILE**), its syntax is:

```
WHILE (Condition) DO
    (Action 1)
    ...
    (Action n)
WEND
```

This code lets you show the number of bars separating the current candlestick from a previous higher candlestick within the limit of 30 periods.

```
i = 1
WHILE high > high [i] and i < 30 DO
    i = i + 1
WEND
RETURN i
```

Example: indicator calculating the number of consecutive increases

```
Increase = (Close > Close[1])
Count = 0
WHILE Increase[Count] DO
    Count = Count + 1
WEND
RETURN Count
```

*General comment on the conditional instruction **WHILE**:*

*Similar to **IF**, the program will automatically assign the value 0 when the validation condition is unknown.*

For example:

```
Count = 0
WHILE i <> 11 DO
    i = i + 1
    Count = Count + 1
WEND
RETURN Count
```

In the code above, the variable *i* is not defined, it will automatically take the value 0 during the first loop and starting from the first candlestick.

The loop will use its resources to define the variable *i* and give it the value 0 by default. Count will be processed and the return value 0 because its value is re-initialized at the beginning of each candlestick and *i* will be greater than 11 at the end of the first candlestick, thus preventing entering the loop for the next candlestick. By defining *i* from the beginning, we will have very different results:

```
i = 0
Count = 0
WHILE i <> 11 DO
    i = i + 1
    Count = Count + 1
WEND
RETURN Count
```

In this code, *i* is initialized to 0 at the beginning of each candlestick, so we pass each time in the loop and we have 11 and 11 as return values for *i* and count.

BREAK

The **BREAK** instruction allows you to make a forced exit out of a **WHILE** loop or a **FOR** loop. Combinations are possible with the **IF** command, inside a **WHILE** loop or a **FOR** loop.

With WHILE

When we want to exit a conditional **WHILE** loop, we use **BREAK** in the following way:

```
WHILE (Condition) DO
    (Action)
    IF (ConditionBreak)
        BREAK
    ENDIF
WEND
```

The use of **BREAK** in a **WHILE** loop is only interesting if we want to test an additional condition for which the value can not be known while in the **WHILE** loop. For example, let's look at a stochastic which is only calculated in a bullish trend:

```
line = 0
Increase = (Close - Close[1]) > 0
i = 0
WHILE Increase[i] DO
    i = i + 1
    // Si high - low, we exit the loop to avoid a division by zero.
    IF (high-low) = 0 then
        BREAK
    ENDIF
    osc = (close - low) / (high - low)
    line = AVERAGE [i] (osc)
WEND
RETURN line
```

With FOR

When we try to get out of an iterative **FOR** loop, without reaching the last (or first) value of the series, we use **BREAK**.

```
FOR Variable = SeriesStartValue TO SeriesEndValue DO
    (Action)
    BREAK
NEXT
```

Let's take for example an indicator cumulating increases of the volume of the last 19 periods. This indicator will be equal to 0 if the volume decreases.

```
Count = 0
FOR i = 0 TO 19 DO
    IF (Volume[i] > Volume[i + 1]) THEN
        Count = Count + 1
    ELSE
        BREAK
    ENDIF
NEXT
RETURN Count
```

In this code, if **BREAK** weren't used, the loop would have continued until 19 (last element of the series) even if the condition count is not valid.

With **BREAK**, on the other hand, as soon as the condition is no longer validated, it returns the result.

CONTINUE

The **CONTINUE** instruction is used to finish the current iteration of a **WHILE** or **FOR** loop. This command is often used with **BREAK**, either to leave the loop (**BREAK**) or to stay in the loop (**CONTINUE**).

With WHILE

Let's create a program counting the number of candlesticks whose close and open are greater than those of the candlestick preceding them. If the condition is not valid, then the counter will be reset to 0.

```
Increase = Close > Close[1]
condition = Open > Open[1]
Count = 0
WHILE condition[Count] DO
    IF Increase[Count] THEN
        Count = Count + 1
        CONTINUE
    ENDIF
    BREAK
WEND
RETURN Count
```

When using **CONTINUE**, if the **IF** condition is not valid, then the **WHILE** loop is not ended. This allows us to count the number of candlesticks detected with this condition verified. Without the **CONTINUE** instruction, the program would leave the loop, whether the **IF** condition is verified or not.. Then, we would not be able to continue counting the number of candlesticks detected and the result would be binary (1, 0).

With FOR

Let's create a program counting the number of candlesticks whose close and open are greater than those of the candlestick preceding them. If the condition is not valid, then the counter will be reset to 0.

```
Increase = Close > Close[1]
Count = 0
FOR i = 1 TO BarIndex DO
    IF Increase[Count] THEN
        Count = Count + 1
        CONTINUE
    ENDIF
    BREAK
NEXT
RETURN Count
```

FOR gives you the possibility to test the condition over all the data loaded. When used with **CONTINUE**, if the **IF** condition is validated, then we do not leave the **FOR** loop and resume it with the next value of i. This is how we count the number of patterns detected by this condition.

Without **CONTINUE**, the program would leave the loop, even if the **IF** condition is validated. Then, we would not be able to count the number of patterns detected and the result would be binary (1, 0).



It is important that you make sure that you always have a valid exit condition for **FOR** and **WHILE** loops to ensure that your code works properly.

ONCE

The **ONCE** instruction is used to initialize a variable at a certain value "**only once**".

Knowing that for the whole program, the language will read the code for each bar displayed on the chart before returning the result, you must then keep in mind that **ONCE**:

- Is processed only one time by the program including the second reading.
- During the second reading of the program, it will stock the values calculated in the previous reading.

To fully understand how this command works, you need to perceive how the language processes the code, hence the usefulness of the next example.

These are two programs returning respectively 0 and 15 and which only difference is the ONCE command added:

Program 1

```

1 Count = 0
2 i = 0
3 IF i <= 5 THEN
4     Count = Count + i
5     i = i + 1
6 ENDIF
7 RETURN Count

```

Program 2

```

1 ONCE Count = 0
2 ONCE i = 0
3 IF i <= 5 THEN
4     Count = Count + i
5     i = i + 1
6 ENDIF
7 RETURN Count

```

Let's see how the language read the code.

Program 1:

The language will read L1 (Count = 0; i = 0), then L2, L3, L4, L5 and L6 (Count = 0; i = 1), then return to L1 and reread everything exactly the same way. The result displayed is 0 (zero), as after the first reading.

Program 2:

For the first bar, the language will read L1 (Count = 0; i = 0), then L2, L3, L4, L5, L6 (Count = 0; i = 1). When it arrives at the line "**RETURN**", it restarts the loop to calculate the value of the next bar starting from L3 (**the lines with ONCE are processed only one time**), L4, L5, L6 (Count = 1; i = 2), then go back again (Count = 3; i = 3) and so forth to (Count = 15; i = 6). Arrived at this result, the **IF** loop is not processed anymore because the condition is not valid anymore; the only line left to read is L7, hence the result is 15 for the remaining bars loaded.

Mathematical Functions

Common unary and binary Functions

Let's focus now on the Mathematical Functions. You will find in ProBuilder the main functions known in mathematics. Please note that *a* and *b* are examples and can be numbers or any other variable in your program.

- **MIN**(*a*, *b*): calculate the minimum of *a* and *b*
- **MAX**(*a*, *b*): calculate the maximum of *a* and *b*
- **ROUND**(*a*, *n*): round *a* to the nearest whole number, with a precision of *n* digits after the decimal point
- **ABS**(*a*): calculate the absolute value of *a*
- **SGN**(*a*): shows the sign of *a* (1 if positive, -1 if negative)
- **SQUARE**(*a*): calculate *a* squared
- **SQRT**(*a*): calculate the square root of *a*
- **LOG**(*a*): calculate the Neperian logarithm of *a*
- **POW**(*a*,*b*) : calculate *a* raised to the power of *b*
- **EXP**(*a*): calculate the exponent of *a*
- **COS**(*a*) / **SIN**(*a*) / **TAN**(*a*): calculate the cosine/sine/tangent of *a* (in degrees)
- **ACOS**(*a*) / **ASIN**(*a*) / **ATAN**(*a*): calculates the arc-cosine/arc-sine/arc-tangent of *a* (in degrees)
- **FLOOR**(*a*, *n*) : returns the largest integer less than *a* with a precision of *n*
- **CEIL**(*a*, *n*) : returns the smallest integer greater than *a* with a precision of *n*
- **RANDOM**(*a*,*b*) : generates a random integer between *a* and *b* (included)

Let's code the example of the normal distribution in mathematics. It's interesting because it use the square function, the square root function and the exponential function:

```
// Normal Law applied to x = 10, StandardDeviation = 6 and MathExpectation = 8
// Let's define the following variables in the variable option:
StandardDeviation = 6
MathExpectation = 8
x = 10
Indicator = EXP((1 / 2) * (SQUARE(x - MathExpectation) / StandardDeviation)) /
(StandardDeviation * SQRT(2 / 3.14))
RETURN Indicator
```

Common mathematical operators

- **a < b**: *a* is strictly less than *b*
- **a <= b** or **a <= b**: *a* is less than or equal to *b*
- **a > b**: *a* is strictly greater than *b*
- **a >= b** or **a >= b**: *a* is greater than or equal to *b*
- **a = b**: *a* is equal to *b* (or *b* is attributed to *a*)
- **a <> b**: *a* is different from *b*

Charting comparison functions

- **a CROSSES OVER b**: the *a* curve crosses over the *b* curve
- **a CROSSES UNDER b**: the *a* curve crosses under the *b* curve

Summation functions

- **CUMSUM:** Calculates the sum of a price or indicator over all bars loaded on the chart

The syntax of cumsum is:

```
CUMSUM (price or indicator)
```

Ex: `CUMSUM(Close)` calculates the sum of the close of all the bars loaded on the chart.

- **SUMMATION:** Calculates the sum of a price or indicator over the last n bars

The sum is calculated starting from the most recent value (from right to left)

The syntax of **SUMMATION** is:

```
SUMMATION[number of bars] ((price or indicator)
```

Ex: `SUMMATION[20] (Open)` calculates the sum of the open of the last 20 bars.

Statistical functions

The syntax of all these functions is the same as the syntax for the Summation function, that is:

```
LOWEST[number of bars] (price or indicator)
```

- **LOWEST:** displays the lowest value of the price or indicator written between brackets, over the number of periods defined
- **HIGHEST:** displays the highest value of the price or indicator written between brackets, over the number of periods defined
- **STD:** displays the standard deviation of a price or indicator, over the number of periods defined
- **STE:** displays the standard error of a price or indicator, over the number of periods defined

Logical operators

As any programming language, it is necessary to have at our disposal some Logical Operators to create relevant indicators. These are the 4 Logical Operators of ProBuilder:

- **NOT(a):** logical NO
- **a OR b:** logical OR
- **a AND b:** logical AND
- **a XOR b:** exclusive OR (a OR b but not a AND b)

Calculation of the trend indicator: On Balance Volume (OBV):

```
IF NOT ((Close > Close[1]) OR (Close = Close[1])) THEN
    MyOBV = MyOBV - Volume
ELSE
    MyOBV = MyOBV + Volume
ENDIF
RETURN MyOBV
```


ProBuilder instructions

- **RETURN:** displays the result of your indicator
- **CALL:** calls another ProBuilder indicator to use in your current program
- **AS:** names the result displayed
- **COLOURED:** colors the displayed curve in with the color of your choice

RETURN

We have already seen in chapter I how important the **RETURN** instruction was. It has some specific properties we need to know to avoid programming errors.

The main points to keep in mind when using **RETURN** in order to write a program correctly are that Return is used:

- One and only one time in each ProBuilder program
- Always at the last line of code
- Optionally with other functions such as **AS** and **COLOURED** and **STYLE**
- To display many results; we write **RETURN** followed with what we want to display and separated with a comma (example: **RETURN** a,b)

Comments

// or **/**/** allows you to write comments inside the code. They are mainly useful to remember how a function you coded works. These remarks will be read but of course not processed by the program. Let's illustrate the concept with the following example:

```
// This program returns the simple moving average over 20 periods applied to the closing price
RETURN Average[20] (Close)
```



Don't use special characters (examples: é,ù,ç,ê...) in ProBuilder code. Special characters may be used only within comments.

CustomClose

CustomClose is a variable allowing you to display the **Close**, **Open**, **High**, **Low** constants and many others, which can be customized in the Settings window of the indicator.

Its syntax is the same as the one of the constants adapted to the timeframe of the chart:

```
CustomClose[n]
```

Example:

```
RETURN CustomClose[2]
```

By clicking on the wrench in the upper left corner of the chart, you will see that it is possible to customize the prices used in the calculation.

CALCULATEONLASTBARS

CALCULATEONLASTBARS: This parameter allows you to increase the speed at which an indicator will be calculated by defining the number of bars that can be used to calculate the indicator (less bars used in the calculation = faster calculation speed).

Example: **DEFPARAM CALCULATEONLASTBARS = 200**

Warning: the use of the **DEFPARAM** instruction must be done at the beginning of the code.

CALL

CALL allows you to use a personal indicator you have coded before in the platform.

The quickest method is to click "Insert Function" then select the "User Indicators" category and then select the name of the indicator you want to use and click "Add".

For example, imagine you have coded the Histogram MACD and named it HistoMACD.

Select your indicator and click on "Add". You will see in the programming zone:

```
myHistoMACD = CALL "HistoMACD"
```

The software gave the name "myHistoMACD" to the indicator "HistoMACD".

This means that for the rest of your program, if you want to use the HistoMACD indicator, you will have to call it "myHistoMACD".

An example when several variables are returned by your CALL:

```
myExponentialMovingAverage, mySimpleMovingAverage = CALL "Averages"
```

AS

The keyword **AS** allows you to name the different results displayed. This instruction is used with **RETURN** and its syntax is:

```
RETURN Result1 AS "Curve Name1", Result2 AS "Curve Name2", ...
```

This keyword makes it easier to identify the different curves on your chart.

Example:

```
a = ExponentialAverage[200] (Close)
```

```
b = WeightedAverage[200] (Close)
```

```
c = Average[200] (Close)
```

```
RETURN a AS "Exponential Average", b AS "Weighted Average", c AS "Arithmetic Average"
```









COLOURED

COLOURED is used after the **RETURN** command to color the value displayed with the color of your choice, defined with the RGB norm (Red, Green, Blue) or by using predefined colors.

The 140 predefined colors can be found in the following documentation:

[W3 School : HTML Color Names](#)

Here are the main colors of the RGB standard as well as their predefined HTML name:

COLOR	RGB VALUE (between 0 and 255) (RED, GREEN, BLUE)	HTML Color name
	(0, 0, 0)	Black
	(255, 255, 255)	White
	(255, 0, 0)	Red
	(0, 255, 0)	Green
	(0, 0, 255)	Blue
	(255, 255, 0)	Yellow
	(0, 255, 255)	Cyan
	(255, 0, 255)	Magenta

The syntax for using the Coloured command is as follows:

```
RETURN Indicator COLOURED(RedValue, GreenValue, BlueValue)
```

Or alternatively:

```
RETURN Indicator COLOURED("cyan")
```

Optionally, you can control the opacity of your curve with the alpha parameter (between 0 and 255):

```
RETURN Indicator COLOURED(Red Value, Green Value, BlueValue, AlphaValue)
```

The **AS** command can be associated with the **COLOURED**(. , . , .) command:

```
RETURN Indicator COLOURED(RedValue, GreenValue, BlueValue) AS "Name of the curve"
```

Let's go back to the previous example and insert **COLOURED** in the "RETURN" line.

```
a = ExponentialAverage[200] (Close)
```

```
b = WeightedAverage[200] (Close)
```

```
c = Average[200] (Close)
```

```
RETURN a COLOURED("red") AS "Exponential Moving Average", b COLOURED("green") AS "Weighted Moving Average", c COLOURED("blue") AS "Simple Moving Average"
```

This picture shows you the color customization of the result.



Drawing instructions

These commands allow you to draw objects on the charts but also to customize your candles, the bars of your charts as well as the colors of all these elements.

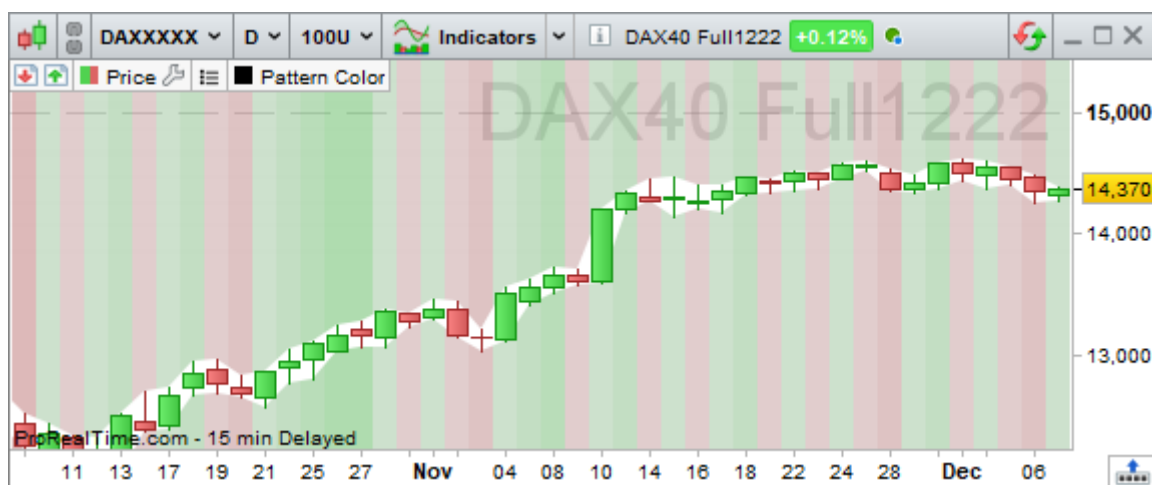
For each instruction below, the color can be defined in a similar way to the color of your curve (COLOURED instruction above) with either a predefined color (HTML Color Name) in quotes, or an RGB value (R,G,B) on which you can apply an alpha opacity parameter: (HTML Color Name,alpha) or (R,G,B,alpha)

- **BACKGROUNDCOLOR** (R, G, B, a) : Lets you color the background of the chart or specific bars (such as odd/even days). The colored zone starts halfway between the previous bar and the next bar

Example: **BACKGROUNDCOLOR** (0, 127, 255, 25)

Its possible to use a variable for the colors if you want the background color to change based on your conditions.

Example: **BACKGROUNDCOLOR** (0, color, 255, 25)



- **COLORBETWEEN**: Allows you to fill the space between two values with a certain color.

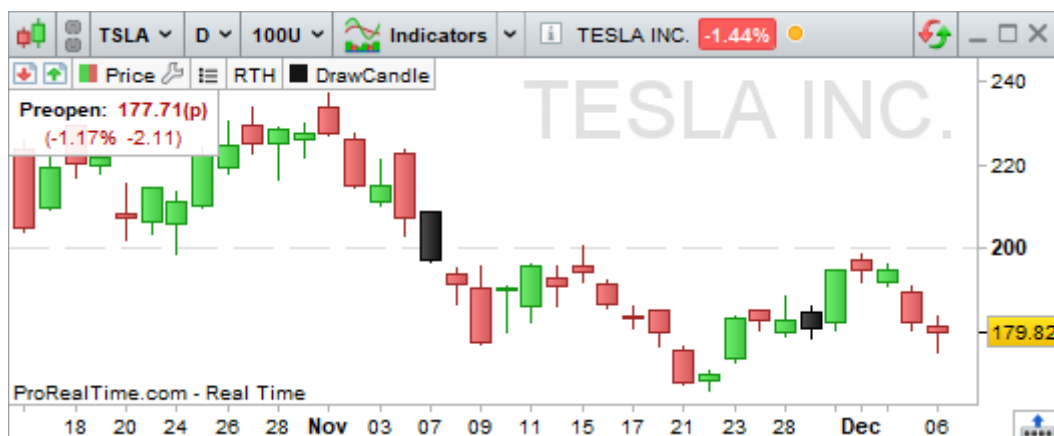
Example : **COLORBETWEEN** (open, close, "white")

- **DRAWBARCHART** : Draws a custom bar on the chart. Open, high, low and close can be constants or variables.

Example: **DRAWBARCHART** (open, high, low, close) **COLOURED** (0, 255, 0)

- **DRAWCANDLE** : Draws a custom candlestick on the chart. Open, high, low and close can be constants or variables.

Example: **DRAWCANDLE** (open, high, low, close) **COLOURED** ("black")



For all the drawing instructions below, the x-axis is expressed by default as a bar number ([BARINDEX](#)) and the y-axis corresponds to the vertical scale of the values in your graph. However, you can change this behavior with the [ANCHOR](#) command described later.

- **DRAWARROW**: Draws an arrow pointing right. You need to define a point for the arrow (x and y axis). You can also choose a color.

Example: **DRAWARROW** (x1, y1) **COLOURED** (R, G, B, a)

- **DRAWARROWUP**: Draws an arrow pointing up. You need to define a point for the arrow (x and y axis). You can also choose a color.

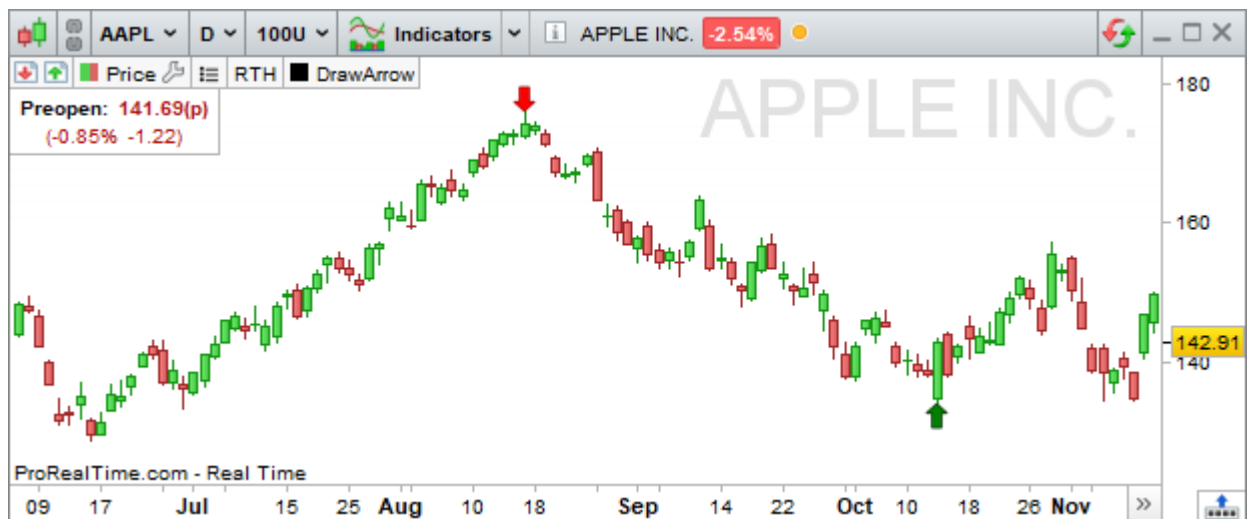
Example: **DRAWARROWUP** (x1, y1) **COLOURED** (R, G, B, a)

This is useful to add visual buy signals.

- **DRAWARROWDOWN**: Draws an arrow pointing down. You need to define a point for the arrow (x and y axis). You can also choose a color.

Example: **DRAWARROWDOWN** (x1, y1) **COLOURED** (R, G, B, a)

- This is useful to add visual sell or buy signals.



- **DRAWRECTANGLE**: Draws a rectangle on the chart.

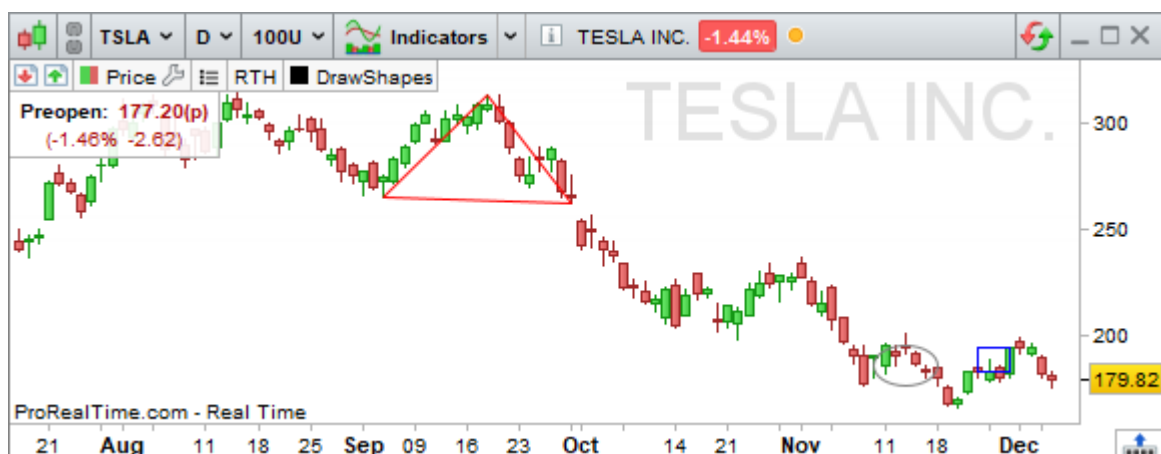
Example: **DRAWRECTANGLE** (x1, y1, x2, y2) **COLOURED** (R, G, B, a)

- **DRAWTRIANGLE**: Draws a triangle on the chart.

Example : **DRAWTRIANGLE** (x1, y1, x2, y2, x3, y3) **COLOURED** (R, G, B, a)

- **DRAWELLIPSE**: Draws an ellipse on the chart.

Example: **DRAWELLIPSE** (x1, y1, x2, y2) **COLOURED** (R, G, B, a)



- **DRAWPOINT**: Draws a point on the chart.

Example : **DRAWPOINT** (x1, y1, pointSize) **COLOURED** (R, G, B, a)

- **DRAWLINE** : Draws a line on the chart.

Example: **DRAWLINE** (x1, y1, x2, y2) **COLOURED** (R, G, B, a)

- **DRAWHLINE** : Draws a horizontal line on the chart.

Example: **DRAWHLINE** (y1) **COLOURED** (R, G, B, a)

- **DRAWVLINE** : Draws a vertical line on the chart.

Example: **DRAWVLINE** (x1) **COLOURED** (R, G, B, a)

- **DRAWSEGMENT** : Draws a segment on the chart.

Example: **DRAWSEGMENT** (x1, y1, x2, y2) **COLOURED** (R, G, B, a)

Example: **DRAWSEGMENT** (barindex, close, barindex[5], close[5])



- **DRAWRAY** : Draws a ray on the graph

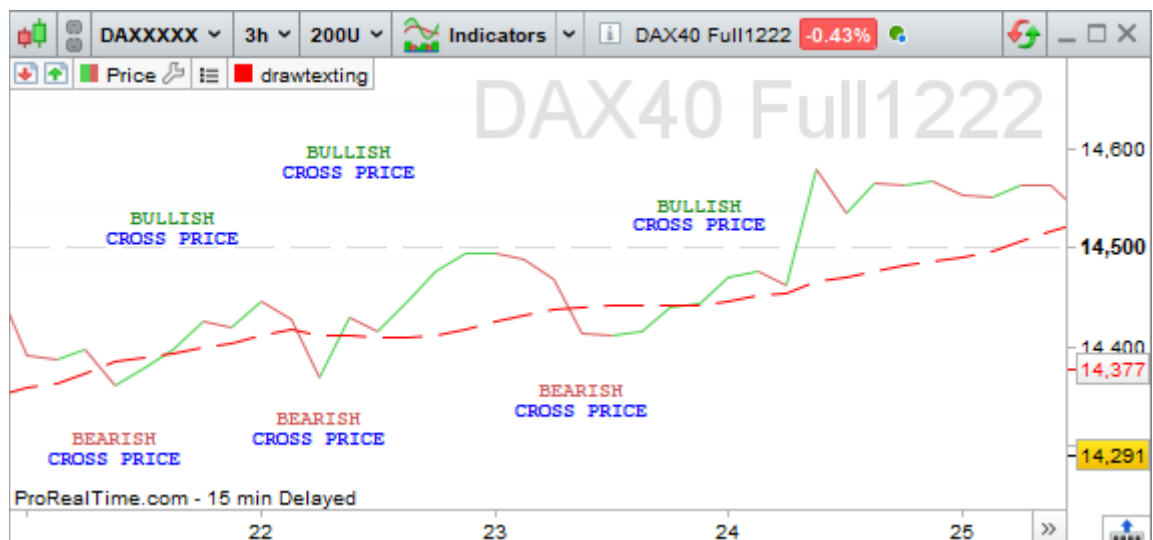
Example : **DRAWRAY** (x1, y1, x2, y2)

- **DRAWTEXT**: Adds a text field to the chart with text of your choice at a specified location. This text can be configured using different style settings.

Example: **DRAWTEXT** ("your text", x1, y1, **SERIF**, **BOLD**, 10) **COLOURED** (R, G, B, a)

Example: **DRAWTEXT** (value, x1, y1, font, fontStyle, fontSize) **COLOURED** (R,G,B,a)

Example: **DRAWTEXT** (value, x1, y1) **COLOURED** ("green")



- Here are the different possible values for the **font** and **font style** parameters, the **font size** is between 1 and 30:

Font	Font style
DIALOG	STANDARD
MONOSPACED	BOLD
SANSERIF	BOLDITALIC
SERIF	ITALIC

- DRAWONLASTBARONLY** : Parameter that lets you draw drawn objects on the last bar only. This parameter should always be used with "CALCULATEONLASTBARS" to optimize calculations.

Example: `DEFPARAM DRAWONLASTBARONLY = true`

Additional parameters

For some of these design commands, various additional instructions can be applied in no particular order:

BORDERCOLOR

This instruction allows you to define the color of the border of a drawn object (excluding lines and arrows).

Example 1: `DRAWRECTANGLE(barindex, close, barindex[5], close[5]) BORDERCOLOR(color)`

Example 2: `DRAWRECTANGLE(barindex, close, barindex[5], close[5]) BORDERCOLOR("red")`

ANCHOR

This instruction allows you to define the anchor point of the object when you want to draw it from a starting point other than the candlesticks.

`DRAWTEXT(close, n, p) ANCHOR(referencePoint, horizontalShift, verticalShift)`

It can take several values as parameters:

- Parameter 1:** the position of the anchor

Value	Description
TOPLEFT	Fixed at the top left of the chart
TOP	Fixed at the top of the chart (middle)
TOPRIGHT	Fixed at the top right of the chart
RIGHT	Fixed to the right of the graph (middle)
BOTTOMRIGHT	Fixed at the bottom right of the chart
BOTTOM	Fixed at the bottom of the graph (middle)
BOTTOMLEFT	Fixed at the bottom left of the chart
LEFT	Fixed to the left of the graph (middle)
MIDDLE	Fixed in the center of the graph

- Parameter 2:** the type of value to set the positioning on the horizontal axis
 - INDEX:** The values entered in the drawing of the object for the horizontal axis will refer to the barindex of the candlesticks
 - XSHIFT:** The values entered in the drawing of the object for the horizontal axis will refer to an offset value in pixels (positive or negative with respect to an orthonormal reference frame)
- Parameter 3:** the type of value to set the positioning on the vertical axis
 - VALUE:** The values entered in the drawing of the object for the vertical axis will refer to a price
 - YSHIFT:** The values entered in the drawing of the object for the vertical axis will refer to an offset value in pixels (positive or negative with respect to an orthonormal reference frame)

Examples:

`DRAWTEXT(previousClose, -20, -50) ANCHOR(TOPRIGHT, XSHIFT, YSHIFT)`

Displays the `previousClose` variable value at the top right of the graph with an offset of `-20` on the horizontal axis and `-50` on the vertical axis.

`DRAWTEXT("Top", barindex-10, -20) ANCHOR(TOP, INDEX, YSHIFT)`

Draws the text "Top" at the top of the chart with an offset of `-20` on the vertical axis and positioned in the continuity of the 10^{ème} `barindex` before the last one.

STYLE

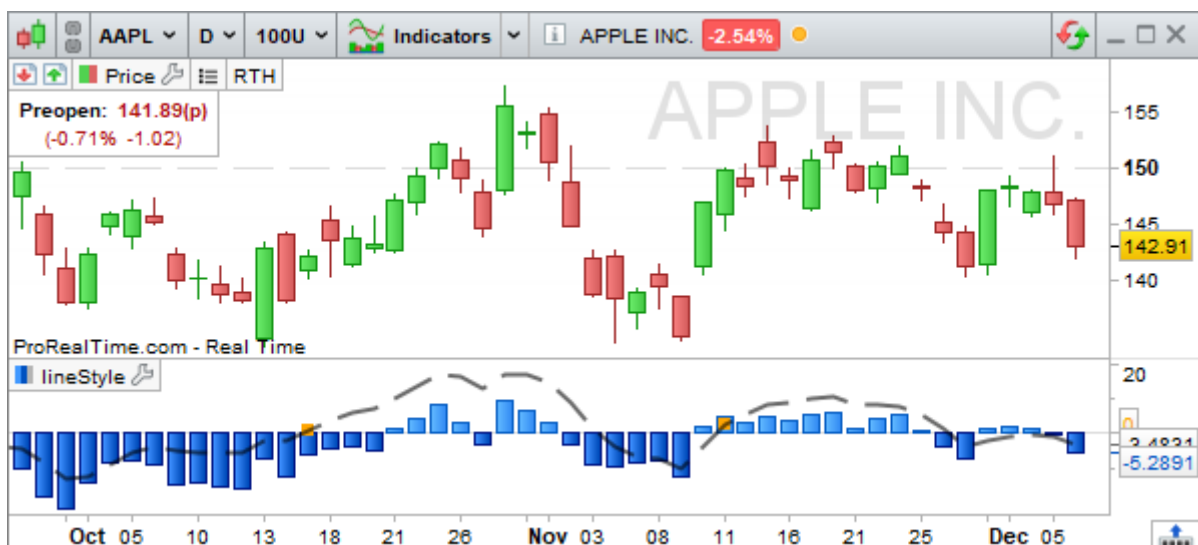
This instruction allows you to define a style for objects (except arrows) or for returned values.

`DRAWRECTANGLE(x1, y1, x2, y2) STYLE(style, lineWidth)`

There are different styles:

- DOTTEDLINE:** this style transforms the line into a dotted line, there are 5 different configurations that represent 5 different dotted line lengths: `DOTTEDLINE`, `DOTTEDLINE1`, `DOTTEDLINE2`, `DOTTEDLINE3`, `DOTTEDLINE4`
- LINE:** this style restores the default line style (full line)
- HISTOGRAM:** this style, only applicable in the `RETURN` instruction of an indicator, displays the returned values as a histogram.
- POINT:** this style, only applicable in the `RETURN` instruction of an indicator, displays the returned values as a point.

`lineWidth` which defines the thickness of the line, will take a value between `1` (the thinnest) and `5` (the thickest).









Note: for the drawing functions it is possible to specify a date rather than a candlestick index thanks to the [DateToBarIndex](#) function which allows to transform a date to the nearest associated bar index.

The instruction is written in the following form:

[DateToBarIndex](#) (date)

Expected date formats:

-  YYYY / Example: 2022
-  YYYYMM / Example: 202208
-  YYYYMMDD / Example: 20220815
-  YYYYMMDDHH / Example: 2022081517
-  YYYYMMDDHHMM / Example: 202208151730
-  YYYYMMDDHHMMSS / Example: 20220815173020




Multi-period instructions

ProBuilder allows you to work on different time periods in your Backtests, Indicators and Screeners, giving you access to more complete data when designing your codes.

The instruction is structured as follows:

`TIMEFRAME (X TimeUnit , Mode)`

With the following parameters:

-  `TimeUnit`: The type of period chosen (see [List of available time frames](#))
-  `X`: The value associated with the selected period
-  `Mode`: The selected calculation mode (optional)

Example: `TIMEFRAME (1 Hour)`

You can use multi-timeframe instructions only to call time units greater than your base time unit (time unit of the chart).

The secondary time units called must also be a multiple of the base time unit .

Thus on a 10 minutes chart:

We can call the following time frames: 20 minutes, 1 hour, 1 day.

We can't call the 5 minutes or 17 minutes time frames.

To enter a higher time frame, you need to use the instruction:

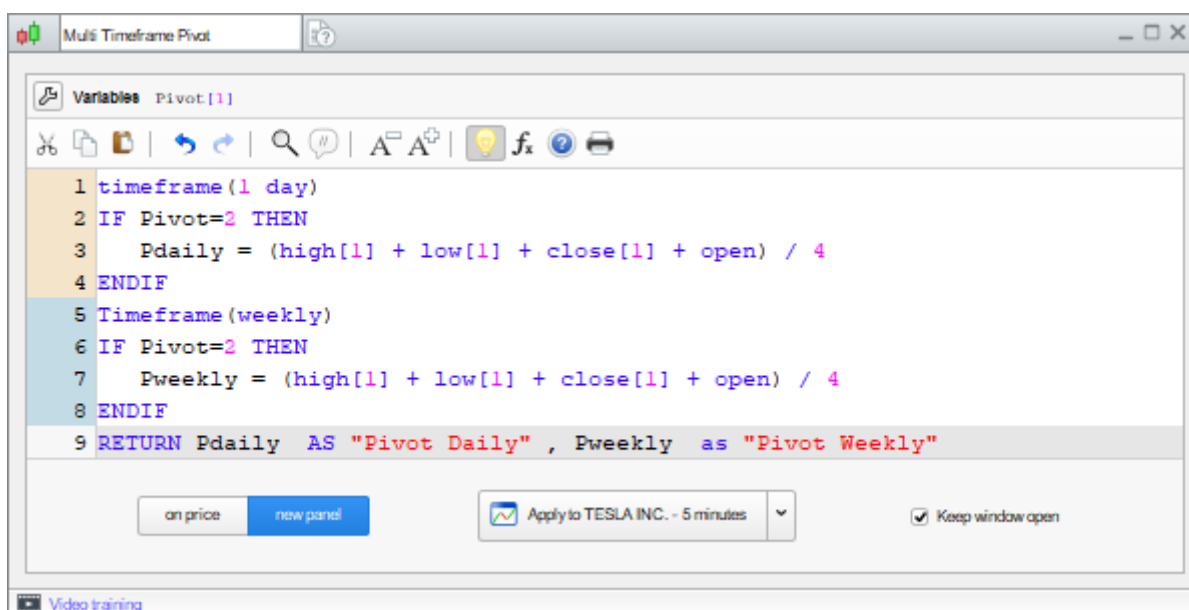
`TIMEFRAME (X TimeUnit)`

To return to the base time frame of the chart, use the following command:

`TIMEFRAME (DEFAULT)`

You can also indicate the time frame of the base chart.

The platform editor colors the background of the code blocks in higher timeframes to help you visualize the pieces of code calculated in each different time frame.



It is also possible to use two calculation modes in a larger time unit in order to have more flexibility in your

calculations:

```
TIMEFRAME(X TimeUnit , DEFAULT)
TIMEFRAME(X TimeUnit , UPDATEONCLOSE)
```

DEFAULT: this is the default mode of the timeframe (mode used when the second parameter is not specified), the calculations in the higher time frames are performed at each new price received in the base time unit of the chart.

UPDATEONCLOSE: the calculations contained in a time frame in this mode are performed at the closing of the candlestick of the higher time frame.

Here is an example of code showing the difference between the two calculation modes:

```
// calculation of an average price between opening and closing in the two available modes.
```

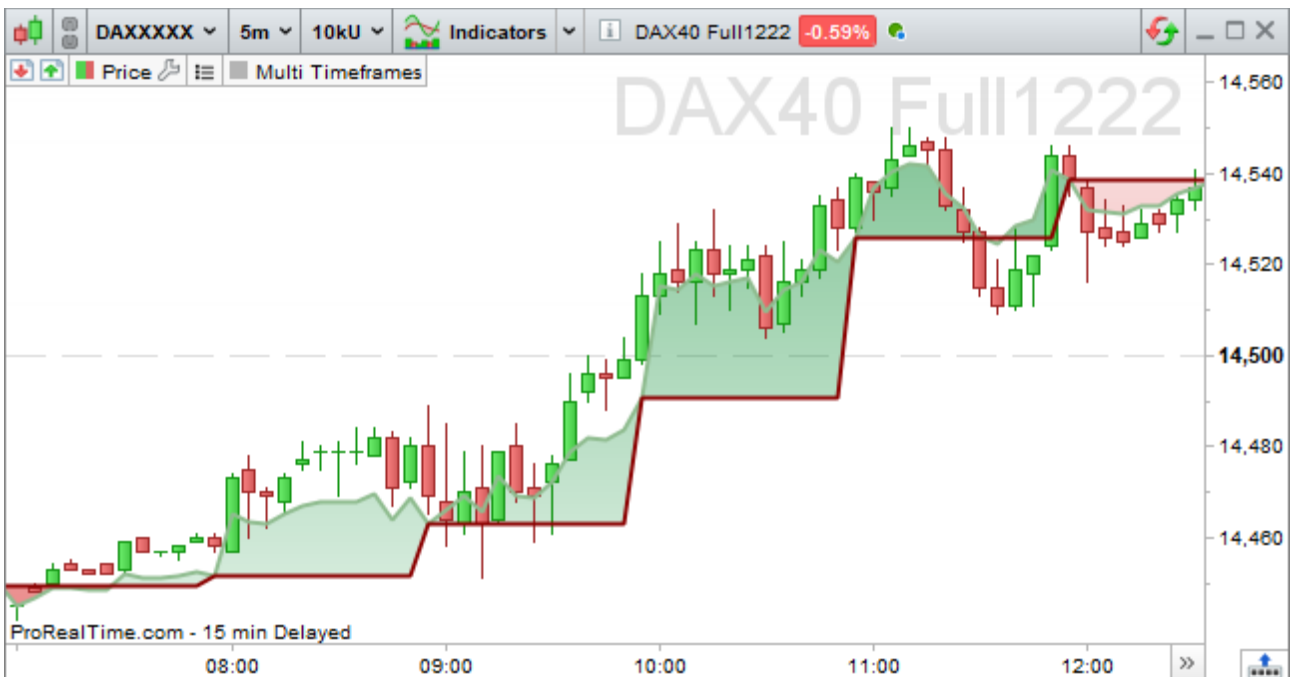
```
TIMEFRAME(1 Hour)
```

```
MidPriceDefault=(open+close)/2
```

```
TIMEFRAME(1 Hour, UPDATEONCLOSE)
```

```
MidPriceUpdateOnClose=(open+close)/2
```

```
Return MidPriceDefault as "Average Price Default mode" COLOURED ("DarkSeaGreen")
, MidPriceUpdateOnClose as "Average Price UpdateOnClose mode" COLOURED ("DarkRed")
```



Here we notice that my `MidPriceDefault` (in green) is updated after every 5 minute candlestick, while `MidPriceUpdateOnClose` (in red) is updated after every 1 hour candlestick.

Note on the use of the TIMEFRAME instruction:

- A variable calculated in one time frame cannot be overwritten by a calculation in another time frame, on the other hand the variables can be used in all time frames contained in the same code.
 - There is a limit of 5 TIMEFRAME intraday instructions (smaller than daily) for automatic trading and backtesting.
 - For the screener, only the **DEFAULT** mode is available, so it is not necessary to specify the mode.
- Moreover, in order to guarantee the performance of calculations on many real time values, only a predefined list of available time frames is authorized for this module.
- For more information, please read the [ProScreener Documentation](#).

List of available time frames

Periods	Examples
Tick / Ticks	TIMEFRAME(1 Tick)
sec / Second / Seconds	TIMEFRAME(10 Seconds)
mn / Minute / Minutes	TIMEFRAME(5 Minutes)
Hour / Hours	TIMEFRAME(1 Hour)
Day / Days	TIMEFRAME(5 Days)
Week / Weeks	TIMEFRAME(1 Weeks)
Month / Months	TIMEFRAME(2 Month)
Year / Years	TIMEFRAME(1 Year)

Arrays (Data tables)

In order to be able to store several values on the same candlestick or to store values only when necessary, we suggest you to use Arrays (data tables) instead of variables.

A code can contain as many arrays as necessary, which can contain up to one million values each.

An array is always prefixed with the \$ symbol.

Syntax of a variable

A

Syntax of an array

\$A

An array starts from index 0 to index 999 999

Index	0	1	2	3	4	5	6	...	999 999
Value									

To insert a value in an array, simply use

```
$Array[Index] = value
```

For example if we want to insert the value of the calculation of the moving average of period 20 at index 0 of array A, we will write :

```
$A[0] = Average[20](close)
```

To read the value of an index of the table, we will use, on the same principle:

```
$Array[Index]
```


For example if we want to create a condition that checks that the close is greater than the value of the first index of the array A:


```
Condition = close > $A[0]
```


When inserting a value at an index **n** of an array, ProBuilder will initialize the values to zero for all the undefined indices from 0 to **n-1** in order to facilitate the use of the data contained in this array.


Specific functions

Several functions specific to arrays are available to facilitate their manipulation and use:

 **ArrayMax**(\$Array): returns the highest value of the array that has been defined. The zeros filled automatically by ProBuilder are not taken into account.

 **ArrayMin**(\$Array): returns the smallest value of the array that has been defined. The zeros filled automatically by ProBuilder are not taken into account.

 **ArraySort**(\$Array, **MODE**): Sorts the array in ascending order (mode=**ASCEND**) or in descending order (mode=**DESCEND**). The zeros filled automatically by ProBuilder will then be removed.

 **IsSet**(\$Array[index]): returns 1 if the index of the table has been defined, 0 if it has not been defined. The zeros filled automatically by ProBuilder are not considered as having been defined so the function will return 0 on these indices.

- `LastSet($Array)`: returns the highest defined index of the array, if no index has been defined in the array, the function will return -1.
- `UnSet($Array)`: Resets the array to 0 by completely deleting its content.






If you want to see use cases of these functions, we recommend [this link](#) from our partner ProRealCode which details the use of arrays through different examples.

Chapter III: Practical aspects

Create a binary or ternary indicator : why and how ?

A binary or ternary indicator is by definition an indicator that can only return two or three possible results (usually 0, 1 or -1). Its main use in a stock market context is to make the verification of the condition that constitutes the indicator immediately identifiable.

Uses of a binary or ternary indicator:

-  Enable the detection of the main Japanese candlestick patterns
-  Facilitate the reading of a chart graph when trying to verify several conditions at once
-  To be able to put classic alerts with 1 condition on an indicator that incorporates several ➔ you will have more alerts available!
-  Detecting complex conditions also on historical data
-  Facilitate the creation or execution of a backtest

Binary or ternary indicators are constructed using the IF function. We advise you to reread the relative section before continuing reading.

Let's picture the creation of these indicators to detect price patterns:

Binary Indicator: hammer detection hammer

```
Hammer = Close>Open AND High = Close AND (Open-Low) >= 3*(Close-Open)
IF Hammer THEN
    Result = 1
ELSE
    Result = 0
ENDIF
RETURN Result AS "Hammer"
```



This simplified code will also give the same results:

```
Hammer = Close>Open AND High = Close AND (Open-Low) >= 3*(Close-Open)
RETURN Hammer AS "Hammer"
```

Ternary Indicator: Golden Cross and Death Cross detection

```
a = ExponentialAverage[10](Close)
b = ExponentialAverage[20](Close)
c = 0
// Golden Cross detection
IF a CROSSES OVER b THEN
    c = 1
ENDIF
// Death Cross detection
IF a CROSSES UNDER b THEN
    c = -1
ENDIF
RETURN c
```



Note: we have displayed the exponential moving average over 10 and 20 periods both applied to the close in order to highlight the results of the indicator.

You can find other candlestick pattern indicators in the "Exercises" chapter later in this manual.

Creating stop indicators to follow a position

It is possible to create STOP indicators, meaning potential places to exit the market defined by personalized parameters.

With the backtesting module ProBacktest, which is the subject of another programming manual, you can also define the stop levels of a backtest. However, programming a stop as an indicator is interesting because:

- It allows to visualize the stop as a line which updates in real-time on the chart (ex: trailing stop)
- It is possible to place real-time alerts to be immediately informed of the situation
- It is not necessary to create long or short orders (contrary to ProBacktest)

Programming Stops is also a means to master the commands you saw in the previous chapters.

These are the 4 categories of stop we will focus on:

- **Static Take Profit STOP**
- **Static STOP Loss**
- **Inactivity STOP**
- **Trailing STOP (trailing stop loss or trailing take profit)**

The indicators presented in the following examples are possible codes to create stop indicators. You will most probably personalize them using the instructions you learned in the previous chapters.

Static Take Profit STOP

A *Static Take-Profit* designates a level that if price reaches it, we plan to close our position and exit with gains. By definition, this STOP is a fixed level (horizontal line). The user of this kind of STOP will exit his position and take his profit when this level is reached.

The indicator coded below indicates two levels and "StartingTime" is the moment you entered your position:

- If you are a buyer, you will take into account the higher curve, representing a 10% profit (110% of the price when you took your long position).
- If you are a seller, you will take into account the lower curve, representing a 10% profit (90% of the price when you took your short position).

```
// We define as a variable : StartingTime = 100000
// Set this variable correctly to the time of your position entry
// Price= Price at the moment of taking the position (we have taken the example of a
position entry date defined at 10 am)
// If you are long, you will be looking at the top curve. If you are short, you will look
at the bottom curve.
// AmplitudeUp represents the rate of change of Price used to plot the Take Profit in a
long position (default 1.1)
// AmplitudeDown represents the rate of change of Price used to plot the Take Profit in a
short position (default: 0.9)
IF Time = StartingTime THEN
    StopLONG = AmplitudeUp * Price
    StopSHORT = AmplitudeDown * Price
ENDIF
RETURN StopLONG COLOURED(0, 0, 0) AS "TakeProfit LONG", StopSHORT COLOURED(0, 255, 0) AS
"TakeProfit SHORT"
```

Static STOP loss

A *Static STOP Loss* is the opposite of a *Static Take-Profit STOP*, meaning if price reaches it, we plan to close our position and exit with losses. This STOP is very useful when you are losing money and try exit the market to limit your losses to the minimum. Just like the *Static Take-Profit*, this STOP defines a fixed level, but this time, the user will exit his position and cut his losses when this level is reached.

The indicator coded below indicates two levels and "StartingTime" is the moment you entered your position:

- If you are a buyer, you will take into account the lower curve, representing a 10% loss (90% of the price when you took your long position).
- If you are a seller, you will take into account the higher curve, representing a 10% loss (110% of the price when you took your short position).

The code of this indicator is:

```
// We define in the variables section:
// StartingTime = 100000 (this is an example for 10 am; set this to the time you entered
your position)
// Price= Price when you took your position
// You can look at StopLONG if looking at a long position and StopShort if you are
looking at a short position. You can also remove StopLONG or StopSHORT if you only work
with long positions or only work with short positions.
// AmplitudeUp represents the variation rate of Price used to draw the Stop Loss for
short positions (default: 0.9)
// AmplitudeDown represents the variation rate of Price used to draw the Stop Loss for
long positions (default: 1.1)
IF Time = StartingTime THEN
    StopLONG = AmplitudeUp * Price
    StopSHORT = AmplitudeDown * Price
ENDIF
RETURN StopLONG COLOURED(0, 0, 0) AS "StopLoss LONG", StopSHORT COLOURED(0, 255, 0) AS
"StopLoss SHORT"
```

Inactivity STOP

An inactivity STOP closes the position when the gains have not obtained a certain objective (defined in % or in points) over a certain period (defined in number of bars).

Remember to define the variables in the "Variables" section.

Example of Inactivity Stop on Intraday Charts:

This stop is to be used with these two indicators:

- The first indicator is to be displayed juxtaposed on the price chart
- The second indicator must be displayed in a separate chart

Indicator1

```
// We define in the variables section:
// MyVolatility = 0.01 represents variation rate between the each part of the range and
the close
IF IntradayBarIndex = 0 THEN
    ShortTarget = (1 - MyVolatility) * Close
    LongTarget = (1 + MyVolatility) * Close
ENDIF
RETURN ShortTarget AS "ShortTarget", LongTarget AS "LongTarget"
```

Indicator2

```
// We define in the variables section:
// We supposed that you take an "On Market Price" position
// MyVolatility = 0.01 represents variation rate between the each part of the range and
the close
// NumberOfBars=20: the close can fluctuate within the range defined during a maximum of
NumberOfBars before the position is cut (Result = 1)
Result = 0
Cpt = 0
IF IntradayBarIndex = 0 THEN
    ShortTarget = (1 - MyVolatility) * Close
    LongTarget = (1 + MyVolatility) * Close
ENDIF
FOR i = IntradayBarIndex DOWNT0 1 DO
    IF Close[i] >= ShortTarget AND Close[i] <= LongTarget THEN
        Cpt = Cpt + 1
    ELSE
        Cpt = 0
    ENDIF
    IF Cpt = NumberOfBars THEN
        Result = 1
    ENDIF
NEXT
RETURN Result
```

Trailing Stop

A *trailing STOP* follows the evolution of the price dynamically and indicates when to close a position.

We suggest you two ways to code a trailing STOP, the first one corresponding to a Dynamic Trailing Stop Loss, and the other one to a Dynamic Trailing Take Profit.

Trailing STOP LOSS (to be used in intraday trading)

```
// Define the following variables in the variable section:
// StartingTime = 090000 (this is an example for 9 am; set this to the time you entered
your position)
// We suppose that you open an "On Market Price" position
// Amplitude represents the variation rate of the "Cut" curve compared to the "Lowest"
curves (for example, we can take Amplitude = 0.95)
IF Time = StartingTime THEN
  IF lowest[5](Close) < 1.2 * Low THEN
    IF lowest[5](Close) >= Close THEN
      Cut = Amplitude * lowest[5](Close)
    ELSE
      Cut = Amplitude * lowest[20](Close)
    ENDIF
  ELSE
    Cut = Amplitude * lowest[20](Close)
  ENDIF
ENDIF
RETURN Cut AS "Trailing Stop Loss"
```

Trailing TAKE Profit (to be used in intraday trading)

```
// Define the following variables in the variable section:
// StartingTime = 090000 (this is an example for 9 am; set this to the time you entered
your position)
// You take an "On Market Price" position
// Amplitude represents the variation rate of the "Cut" curve compared to the "Lowest"
curve (for example, we can take Amplitude = 1.015)
IF Time = StartingTime THEN
  StartingPrice = Close
ENDIF
Price = StartingPrice - AverageTrueRange[10]
TrailingStop = Amplitude * highest[15](Price)
RETURN TrailingStop COLOURED (255, 0, 0) AS "Trailing take profit"
```

Chapter IV: Exercises

Candlesticks patterns

■ GAP UP or DOWN



The color of the candlesticks is not important.

We define as a customizable variable amplitude = 0.001

A gap is defined by these two conditions:

- (the current low is strictly greater than the high of the previous bar) or (the current high is strictly lesser than the low of the previous bar)
- the absolute value of ((the current low – the high of the previous bar)/the high of the previous bar) is strictly greater than amplitude) or ((the current high – the low of the previous bar)/the low of the previous bar) is strictly greater than amplitude)

```
// Initialization of Amplitude
Amplitude = 0.001
// Initialization of detector
Detector = 0
// Gap Up
// 1st condition of the existence of a gap
IF Low > High[1] THEN
    // 2nd condition of the existence of a gap
    IF ABS((Low - High[1]) / High[1]) > Amplitude THEN
        // Behavior of the detector
        Detector = 1
    ENDIF
ENDIF
// Gap Down
// 1st condition of the existence of a gap
IF High < Low[1] THEN
    // 2nd condition of the existence of a gap
    IF ABS((High - Low[1]) / Low[1]) > Amplitude THEN
        // Behavior of the detector
        Detector = -1
    ENDIF
ENDIF
// Result display
RETURN Detector AS "Gap detection"
```

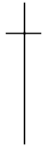
Doji (flexible version)



In this code, we define a doji to be a candlestick with a range (High – Close) greater than 5 times the absolute value of (Open – Close).

```
Doji = Range > ABS(Open - Close) * 5
RETURN Doji AS "Doji"
```

Doji (strict version)



We define the doji with a Close equal to its Open.

```
Doji = (Open = Close)
RETURN Doji AS "Doji"
```

Indicators

BODY MOMENTUM

Body Momentum is mathematically defined by: $\text{BodyMomentum} = 100 * \text{BodyUp} / (\text{BodyUp} + \text{BodyDown})$

BodyUp is a counter of bars for which close is greater than open during a certain number of periods (in this example : 14).

BodyDown is a counter of bars for which open is greater than close during a certain number of periods (in this example : 14).

```
Periods = 14
b = Close - Open
IF BarIndex > Periods THEN
    Bup = 0
    Bdn = 0
    FOR i = 1 TO Periods
        IF b[i] > 0 THEN
            Bup = Bup + 1
        ELSIF b[i] < 0 THEN
            Bdn = Bdn + 1
        ENDIF
    NEXT
    BM = (Bup / (Bup + Bdn)) * 100
ELSE
    BM = Undefined
ENDIF
RETURN BM AS "Body Momentum"
```

• ELLIOT WAVE OSCILLATOR

The Elliot wave oscillator shows the difference between two moving averages.

Parameters:

a: short MA periods (5 by default)

b: long MA periods (35 by default)

This oscillator permits to distinguish between wave 3 and wave 5 using Elliot wave theory.

The short MA shows short-term price action whereas the long MA shows the longer term trend.

When the prices form wave 3, the prices climb strongly which shows a high value of the Elliot Wave Oscillator.

In wave 5, the prices climb more slowly, and the oscillator will show a lower value.

```
RETURN Average[5](MedianPrice) - Average[35](MedianPrice) AS "Elliot Wave Oscillator"
```

• Williams %R

This is an indicator very similar to the Stochastic oscillator. To draw it, we define 2 curves:

1) The curve of the highest of high over 14 periods

2) The curve of the lowest of low over 14 periods

The %R curve is defined by this formula: $(\text{Close} - \text{Lowest Low}) / (\text{Highest High} - \text{Lowest Low}) * 100$

```
HighestH = highest[14](High)
LowestL = lowest[14](Low)
MyWilliams = (Close - LowestL) / (HighestH - LowestL) * 100
RETURN MyWilliams AS "Williams %R"
```

• Bollinger Bands

The middle band is a simple 20-period moving average applied to close.

The upper band is the middle band plus 2 times the standard deviation over 20 periods applied to close.

The lower band is the middle band minus 2 times the standard deviation over 20 periods applied to close.

```
a = Average[20](Close)
// We define the standard deviation.
StdDeviation = STD[20](Close)
Bsup = a + 2 * StdDeviation
Binf = a - 2 * StdDeviation
RETURN a AS "Average", Bsup AS "Bollinger Up", Binf AS "Bollinger Down"
```

You can visit our ProRealTime community on the [ProRealCode forum](#) to find [online documentation](#) and many more examples.

Glossary

A

CODE	SYNTAX	FUNCTION
ABS	ABS(a)	Mathematical function "Absolute Value" of a
AccumDistr	AccumDistr(price)	Classical Accumulation/Distribution indicator
ACOS	ACOS(a)	Mathematical function "Arc cosine"
AdaptiveAverage	AdaptiveAverage[x,y,z](price)	Adaptive Average Indicator
ADX	ADX[N]	Indicator Average Directional Index or "ADX" of n periods
ADXR	ADXR[N]	Indicator Average Directional Index Rate or "ADXR" of n periods
AND	a AND b	Logical AND Operator
ArraySort	ArraySort(\$MyArray, ASCEND)	Sort the table in ascending (ASCEND) or descending (DESCEND) order
AroonDown	AroonDown[P]	Aroon Down indicator
AroonUp	AroonUp[P]	Aroon Up indicator
ATAN	ATAN(a)	Mathematical function "Arctangent"
ANCHOR	ANCHOR(direction, index, yshift)	Anchor function for drawings
AS	RETURN Result AS "ResultName"	Instruction used to name a line or indicator displayed on chart. Used with "RETURN"
ASIN	ASIN(a)	Mathematical function "Arc sine".
Average	Average[N](price)	Simple Moving Average of n periods
AverageTrueRange	AverageTrueRange[N](price)	"Average True Range" - True Range smoothed with the Wilder method

B

CODE	SYNTAX	FUNCTION
BACKGROUNDColor	BACKGROUNDColor(R,G,B,a)	Sets the background color of the chart or a specific bar
BarIndex	BarIndex	Number of bars since the beginning of data loaded (in a chart in the case of a ProBuilder indicator or for a trading system in the case of ProBacktest or ProOrder)
BarsSince	BarsSince(condition)	Returns the number of candlesticks since the last condition was met
Bold	DRAWTEXT("text",barindex,close,Serif,Bold, 10)	Bold style to be applied to the text
BoldItalic	DRAWTEXT("text",barindex,close,Serif,BoldItalic, 10)	Bold italic style to be applied to the text
BollingerBandWidth	BollingerBandWidth[N](price)	Bollinger Bandwidth indicator
BollingerDown	BollingerDown[N](price)	Lower Bollinger band
BollingerUp	BollingerUp[N](price)	Upper Bollinger band
BOTTOM	ANCHOR(BOTTOM,INDEX,YSHIFT)	Anchor at the bottom of the chart
BOTTOMLEFT	ANCHOR(BOTTOMLEFT,INDEX,YSHIFT)	Anchor at the bottom left of the chart
BOTTOMRIGHT	ANCHOR(BOTTOMRIGHT,INDEX,YSHIFT)	Anchor at the bottom right of the chart
BORDERCOLOR	BORDERCOLOR("red")	Adds a colored border to the associated object.
BREAK	(FOR...DO...BREAK...NEXT) or (WHILE...DO...BREAK...WEND)	Instruction forcing the exit of FOR loop or WHILE loop

C

CODE	SYNTAX	FUNCTION
<code>CALCULATEONLASTBARS</code>	DEFPARAM CalculateOnLastBars = 200	Lets you increase the speed at which indicators are calculated by defining the number of bars to display the results, starting with the most recent bar.
<code>CALL</code>	myResult=CALL myFunction	Calls a user indicator to be used in the program you are coding
<code>CCI</code>	CCI[N](price) or CCI[N]	Commodity Channel Index indicator
<code>CEIL</code>	CEIL(N, m)	Returns the smallest number greater than N applied to the decimal m
<code>ChaikinOsc</code>	ChaikinOsc[Ch1, Ch2](price)	Chaikin oscillator
<code>Chandle</code>	Chandle[N](price)	Chande Momentum Oscillator
<code>ChandeKrollStopUp</code>	ChandeKrollStopUp[Pp, Qq, X]	Chande and Kroll Protection Stop on long positions
<code>ChandeKrollStopDown</code>	ChandeKrollStopDown[Pp, Qq, X]	Chande and Kroll Protection Stop on short positions
<code>Close</code>	Close[N]	Closing price of the current bar or of the nth last bar
<code>COLOURED</code>	RETURN x COLOURED(R,G,B)	Colors a curve with the color you defined using the RGB convention
<code>COLORBETWEEN</code>	COLORBETWEEN(a, b, color)	Color the space between two values.
<code>COS</code>	COS(a)	Cosine Function
<code>CROSSES OVER</code>	a CROSSES OVER b	Boolean Operator checking whether a curve has crossed over another one
<code>CROSSES UNDER</code>	a CROSSES UNDER b	Boolean Operator checking whether a curve has crossed under another one
<code>Cumsum</code>	Cumsum(price)	Sums a certain price on the whole data loaded
<code>CurrentDayOfWeek</code>	CurrentDayOfWeek	Represents the current day of the week
<code>CurrentHour</code>	CurrentHour	Represents the current hour
<code>CurrentMinute</code>	CurrentMinute	Represents the current minute
<code>CurrentMonth</code>	CurrentMonth	Represents the current month
<code>CurrentSecond</code>	CurrentSecond	Represents the current second
<code>CurrentTime</code>	CurrentTime	Represents the current time (HHMMSS)
<code>CurrentYear</code>	CurrentYear	Represents the current year
<code>CustomClose</code>	CustomClose[N]	Constant which is customizable in the settings window of the chart (default: Close)
<code>Cycle</code>	Cycle(price)	Cycle Indicator

D

CODE	SYNTAX	FUNCTION
Date	Date[N]	Reports the date of each bar loaded on the chart
DATETOBARINDEX	DATETOBARINDEX(date)	Allows you to use a date for the drawing functions.
Day	Day[N]	Reports the day of each bar loaded in the chart
Days	Days[N]	Counter of days since 1900
Days	TIMEFRAME(X Days)	Set the period to "X Days" for further calculations of the code.
DayOfWeek	DayOfWeek[N]	Day of the week of each bar
DClose	DClose(N)	Close of the nth day before the current one
Decimals	Decimals	Returns the number of decimals of the ticker
DEMA	DEMA[N](price)	Double Exponential Moving Average
DHigh	DHigh(N)	High of the nth bar before the current bar
Dialog	DRAWTEXT("text",barindex,close,Dialog,Bold, 10)	Dialog font applied to text
DI	DI[N](price)	Represents DI+ minus DI-
DIminus	DIminus[N](price)	Represents the DI- indicator
DIplus	DIplus[N](price)	Represents the DI+ indicator
DivergenceCCI	DivergenceCCI[Div1,Div2,Div3,Div4]	Indicator for detecting discrepancies between price and the CCI.
DivergenceMACD	DivergenceMACD[Div1,Div2,Div3,Div4](close)	Indicator for detecting divergences between the price and the MACD.
DLow	DLow(N)	Low of the nth day before the current one
DO	See FOR and WHILE	Optional instruction in FOR loop and WHILE loop to define the loop action
DonchianChannelCenter	DonchianChannelCenter[N]	Middle channel of the Donchian indicator for N periods.
DonchianChannelDown	DonchianChannelDown[N]	Lower channel of the Donchian indicator for N periods.
DonchianChannelUP	DonchianChannelUp[N]	Upper channel of the Donchian indicator for N periods.
DOpen	DOpen(N)	Open of the nth day before the current one
DOTTEDLINE	STYLE(DOTTEDLINE1/2/3/4, width)	Style applicable to the features of an object.
DOWNT0	See FOR	Instruction used in FOR loop to process the loop with a descending order
DPO	DPO[N](price)	Detrended Price Oscillator

CODE	SYNTAX	FUNCTION
DRAWARROW	DRAWARROW(x1,y1)	Draw an arrow pointing right at the selected point. Note: all drawing instructions mentioned hereafter are compatible with version 10.3 and higher of the platform
DRAWARROWDOWN	DRAWARROWDOWN(x1,y1)	Draw a down at the selected point
DRAWARROWUP	DRAWARROWUP(x1,y1)	Draw an up arrow at the selected point
DRAWBARCHART	DRAWBARCHART(open,high,low,close)	Draws a custom bar on the chart. Open, high, low, and close can be constants or variables
DRAWCANDLE	DRAWCANDLE(open,high,low,close)	Draws a custom candlestick. Open, high, low, and close can be constants or variables
DRAWELLIPSE	DRAWELLIPSE(x1,y1,x2,y2)	Draws an ellipse on the chart
DRAWHLINE	DRAWHLINE(y1)	Draws a horizontal line on the chart at the selected point
DRAWLINE	DRAWLINE(x1,y1,x2,y2)	Draws a line on the chart between the two selected points
DRAWONLASTBARONLY	DEFPARAM DrawOnLastBarOnly = true	Parameter that lets you draw drawn objects on the last bar only
DRAWPOINT	DRAWPOINT(x1,y1, <i>optional</i> size)	Draw a point on the chart
DRAWRAY	DRAWRAY(x1,y1,x2,y2)	Draw a ray on the chart
DRAWRECTANGLE	DRAWRECTANGLE(x1,y1,x2,y2)	Draws a rectangle on the chart
DRAWSEGMENT	DRAWSEGMENT(x1,y1,x2,y2)	Draws a segment on the chart
DRAWTEXT	DRAWTEXT("your text", x1, y1)	Adds a text box on the chart at at the selected point with your text
DRAWVLINE	DRAWVLINE(x1)	Draws a vertical line on the chart
DynamicZoneRSIDown	DynamicZoneRSIDown[rsiN, N]	Lower band of the Dynamic Zone RSI indicator.
DynamicZoneRSIUp	DynamicZoneRSIUp[rsiN, N]	Upper band of the Dynamic Zone RSI indicator.
DynamicZoneStochasticDown	DynamicZoneStochasticDown[N]	Lower band of the Dynamic Zone Stochastic indicator.
DynamicZoneStochasticUp	DynamicZoneStochasticUp[N]	Upper band of the Dynamic Zone Stochastic indicator.

E

CODE	SYNTAX	FUNCTION
EaseOfMovement	EaseOfMovement[I]	Ease of Movement indicator
ElderrayBearPower	ElderrayBearPower[N](close)	Elder ray Bear Power indicator
ElderrayBullPower	ElderrayBullPower[N](close)	Elder ray Bull Power indicator
ELSE	See IF/THEN/ELSE/ENDIF	Instruction used to call the second condition of If-conditional statements
ELSEIF	See IF/THEN/ELSEIF/ELSE/ENDIF	Stands for Else If (to be used inside of conditional loop)
EMV	EMV[N]	Ease of Movement Value indicator
ENDIF	See IF/THEN/ELSE/ENDIF	Ending Instruction of IF-conditional statement
EndPointAverage	EndPointAverage[N](price)	End Point Moving Average of a
EXP	EXP(a)	Mathematical Function "Exponential"
ExponentialAverage	ExponentialAverage[N](price)	Exponential Moving Average

F - G

CODE	SYNTAX	FUNCTION
FractalDimensionIndex	FractalDimensionIndex[N] (close)	Fractal Dimension Index indicator.
FOR/TO/NEXT	FOR i=a TO b DO a NEXT	FOR loop (processes all the values with an ascending (TO) or a descending order (DOWNTN))
ForceIndex	ForceIndex(price)	Force Index indicator (determines who controls the market (buyer or seller))
FLOOR	FLOOR(N, m)	Returns the largest number less than N with a precision of m digits after the decimal point
GetTimeFrame	GetTimeFrame	Returns the number of seconds equivalent to the current code time period (ex: 3600 for a one hour time period)

H

CODE	SYNTAX	FUNCTION
High	High[N]	High of the current bar or of the nth last bar
Highest	Highest[N](price)	Highest price over a number of bars to be defined
HighestBars	HighestBars[N]	Returns the candlestick offset of the last highest value
HISTOGRAM	RETURN close STYLE(HISTOGRAM, lineWidth)	Apply the histogram style on the returned value
HistoricVolatility	HistoricVolatility[N](price)	Historic Volatility (or statistic volatility)
Hour	Hour[N]	Represents the hour of each bar loaded in the chart
Hours	TIMEFRAME(X Hours)	Sets the period to "X Hours" for further code calculations. See Multi-timeframe instructions
HullAverage	HullAverage[N](close)	Designates the Hull Average indicator

I - J - K

CODE	SYNTAX	FUNCTION
IF/THEN/ENDIF	IF a THEN b ENDIF	Group of conditional instructions without second instruction
IF/THEN/ELSE/ENDIF	IF a THEN b ELSE c ENDIF	Group of conditional instructions
IntradayBarIndex	IntradayBarIndex[N]	Counts how many bars are displayed in one day on the whole data loaded
INDEX	ANCHOR(TOPLEFT,INDEX,Y SHIFT)	Define the point value of the object on the horizontal axis as a barindex value.
Italic	DRAWTEXT("text",barindex,close,Serif,Italic, 10)	Italic style to be applied to the text
KeltnerBandCenter	KeltnerBandCenter[N]	Central band of the Keltner indicator of N periods.
KeltnerBandDown	KeltnerBandDown[N]	Lower band of the Keltner indicator of N periods.
KeltnerBandUp	KeltnerBandUp[N]	Upper band of the Keltner indicator of N periods.
KijunSen	KijunSen[TPeriod,KPeriod,SP eriod]	Returns the KijunSen value of the Ichimoku indicator

L

CODE	SYNTAX	FUNCTION
LEFT	ANCHOR(LEFT,INDEX,YSHIFT)	Anchor to the left of the chart
LINE	STYLE(LINE, lineWidth)	Standard line style
LinearRegression	LinearRegression[N](price)	Linear Regression indicator
LinearRegressionSlope	LinearRegressionSlope[N](price)	Slope of the Linear Regression indicator
LOG	LOG(a)	Mathematical Function "Neperian logarithm" of a
Low	Low[N]	Low of the current bar or of the nth last bar
Lowest	Lowest[N](price)	Lowest price over a number of bars to be defined
LowestBars	LowestBars[N]	Returns the candlestick offset of the last lowest value

M

CODE	SYNTAX	FUNCTION
MACD	MACD[S,L,Si](price)	Moving Average Convergence Divergence (MACD) in histogram
MACDline	MACDLine[S,L,Si](price)	MACD line indicator
MACDSignal	MACDSignal[S,L,Si](price)	MACD Signal line indicator
MassIndex	MassIndex[N]	Mass Index Indicator applied over N bars
MAX	MAX(a,b)	Mathematical Function "Maximum"
MedianPrice	MedianPrice	Average of the high and the low
MIDDLE	ANCHOR(MIDDLE,INDEX,YSHIFT)	Anchoring in the middle of the chart
MIN	MIN(a,b)	Mathematical Function "Minimum"
Minute	Minute	Represents the minute of each bar loaded in the chart
Minutes	TIMEFRAME(X Minutes)	Sets the period to "X Minutes" for the following code calculations. See Multi-timeframe instructions
MOD	a MOD b	Mathematical Function "remainder of the division"
Momentum	Momentum[I]	Momentum indicator (close – close of the nth last bar)
MoneyFlow	MoneyFlow[N](price)	MoneyFlow indicator (result between -1 and 1)
MoneyFlowIndex	MoneyFlowIndex[N]	MoneyFlow Index indicator
Monospaced	DRAWTEXT("text",barindex,cl	Monospaced font applied to text

CODE	SYNTAX	FUNCTION
	ose, Monospaced, Italic, 10)	
Month	Month[N]	Represents the month of each bar loaded in the chart
Months	TIMEFRAME(X Months)	Sets the period to "X Months" for the following code calculations.

N

CODE	SYNTAX	FUNCTION
NEXT	See FOR/TO/NEXT	Ending Instruction of FOR loops
NOT	Not A	Logical Operator NOT

O

CODE	SYNTAX	FUNCTION
OBV	OBV(price)	On-Balance-Volume indicator
ONCE	ONCE VariableName = VariableValue	Introduces a definition statement which will be processed only once
Open	Open[N]	Open of the current candlestick or of the nth previous candlestick
OpenDay	OpenDay[N]	Designates the opening day of the current candlestick or the nth previous candlestick
OpenDayOfWeek	OpenDay[N]	Designates the day of the week of the opening of the current candlestick or the nth previous candlestick
OpenHour	OpenHour[N]	Designates the opening time of the current candlestick or the nth previous candlestick
OpenMinute	OpenMinute[N]	Designates the opening minute of the current candlestick or the nth previous candlestick
OpenMonth	OpenMonth[N]	Designates the opening month of the current candlestick or the nth previous candlestick
OpenSecond	OpenSecond[N]	Designates the opening second of the current candlestick or the nth previous candlestick
OpenTime	OpenTime[N]	Designates the time (HHMMSS) of the opening of the current candlestick or the nth previous candlestick
OpenTimestamp	OpenTime[N]	Designates the UNIX opening timestamp of the current candlestick or the nth previous candlestick
OpenWeek	OpenWeek[N]	Designates the opening week of the current candlestick or the nth previous candlestick
OpenYear	OpenYear[N]	Designates the opening year of the current

CODE	SYNTAX	FUNCTION
		candlestick or the nth previous candlestick
OR	a OR b	Logical OR Operator

P - Q

CODE	SYNTAX	FUNCTION
Pipsize	Pipsize	Size of a pip (forex)
Point	RETURN close STYLE(POINT, pointWidth)	Apply the dot style on the returned value
PositiveVolumeIndex	PriceVolumeIndex(price)	Positive Volume Index indicator
POW	POW(N,P)	Returns the value of N at power P.
PriceOscillator	PriceOscillator[S,L](price)	Percentage Price oscillator
PRTBANDSUP	pbUp = PRTBANDSUP	Gives the value of the upper band of PRTBands
PRTBANDSDOWN	pbDown = PRTBANDSUP	Gives the value of the lower band of PRTBands
PRTBANDSHORTTERM	pbShort = PRTBANDSSHORTTERM	Gives the value of the short term band of PRTBands
PRTBANDMEDIUMTERM	pbMedium = PRTBANDSMEDIUMTERM	Gives the value of the long term band of PRTBands
PVT	PVT(price)	Price Volume Trend indicator

R

CODE	SYNTAX	FUNCTION
R2	R2[N](price)	R-Squared indicator (error rate of the linear regression on price)
RANDOM	RANDOM(Min, Max)	Generates a random integer between the included Min and Max bounds.
Range	Range[N]	calculates the Range (High minus Low)
Repulse	Repulse[N](price)	Repulse indicator (measure the buyers and sellers force for each candlestick)
RepulseMM	RepulseMM[N,PeriodMM,fact orMM](price)	Moving Average line of the Repulse indicator.
RETURN	RETURN Result	Instruction returning the result
RIGHT	ANCHOR(RIGHT,INDEX,YSH IFT)	Anchor to the right of the chart
ROC	ROC[N](price)	Price Rate of Change indicator
RocnRoll	RocnRoll(price)	Designates the RocnRoll indicator based on the ROC indicator.
ROUND	ROUND(a)	Mathematical Function "Round a to the nearest

CODE	SYNTAX	FUNCTION
		whole number"
RSI	RSI[N](price)	Relative Strength Index indicator

S

CODE	SYNTAX	FUNCTION
SansSerif	DRAWTEXT("text",barindex,close,SansSerif,Italic, 10)	SansSerif font applied to text
SAR	SAR[At,St,Lim]	Parabolic SAR indicator
SARatdmf	SARatdmf[At,St,Lim](price)	Smoothed Parabolic SAR indicator
Second	TIMEFRAME(X Seconds)	Sets the period to "X Seconds" for further code calculations. See Multi-period instructions
SenkouSpanA	SenkouSpanA[TPeriod,KPeriod,SPeriod]	Returns the SenkouSpanA value of the Ichimoku indicator
SenkouSpanB	SenkouSpanB[TPeriod,KPeriod,SPeriod]	Returns the SenkouSpanB value of the Ichimoku indicator
Serif	DRAWTEXT("text",barindex,close,Serif,Italic, 10)	Serif font applied to text
SIN	SIN(a)	Mathematical Function "Sine"
SGN	SGN(a)	Mathematical Function "Sign of" a (it is positive or negative)
SMI	SMI[N,SS,DS](price)	Stochastic Momentum Index indicator
SmoothedRepulse	SmoothedRepulse[N](price)	Smoothed Repulse indicator
SmoothedStochastic	SmoothedStochastic[N,K](price)	Smoothed Stochastic indicator
SQUARE	SQUARE(a)	Mathematical Function "a Squared"
SQRT	SQRT(a)	Mathematical Function "Squared Root" of a
Standard	DRAWTEXT("text",barindex,close,Serif,Standard, 10)	Standard style applied to text
STD	STD[N](price)	Statistical Function "Standard Deviation"
STE	STE[N](price)	Statistical Function "Standard Error"
STYLE	STYLE(dottedline, width)	Applies the <i>dottedline</i> style type with a <i>width</i> on an object.
Stochastic	Stochastic[N,K](price)	%K line of the Stochastic indicator
Stochasticd	Stochasticd[N,K,D](price)	%D line of the Stochastic indicator
Summation	Summation[N](price)	Sums a certain price over the N last candlesticks
Supertrend	Supertrend[STF,N]	Super Trend indicator

T

CODE	SYNTAX	FUNCTION
TAN	TAN(a)	Mathematical Function "Tangent" of a
TEMA	TEMA[N](price)	Triple Exponential Moving Average
TenkanSen	TenkanSen[TPeriod,KPeriod,S Period]	Returns the TenkanSen value of the Ichimoku indicator
THEN	See IF/THEN/ELSE/ENDIF	Instruction following the first condition of "IF"
Ticks	TIMEFRAME(X Ticks)	Sets the period to "X Ticks" for further code calculations. See Multi-period instructions
Ticksize	Ticksize	Minimum price variation of the instrument in the chart
Time	Time[N]	Represents the time of each bar loaded in the chart
TimeSeriesAverage	TimeSeriesAverage[N](price)	Temporal series moving average
Timestamp	Timestamp[N]	UNIX date of the close of the nth previous candlestick
TO	See FOR/TO/NEXT	Directional Instruction in the "FOR" loop
Today	YYYYMMDD	Today's date
TOP	ANCHOR(TOP,INDEX,YSHIF T)	Anchor at the top of the chart
TOLEFT	ANCHOR(TOLEFT,INDEX,Y SHIFT)	Anchor at the top left of the chart
TOPRIGHT	ANCHOR(TORIGHTP,INDEX, YSHIFT)	Anchor at the top right of the chart
TotalPrice	TotalPrice[N]	(Close + Open + High + Low) / 4
TR	TR(price)	True Range indicator
TriangularAverage	TriangularAverage[N](price)	Triangular Moving Average
TRIX	TRIX[N](price)	Triple Smoothed Exponential Moving Average
TypicalPrice	TypicalPrice[N]	Represents the Typical Price (Average of the High, Low and Close)

U

CODE	SYNTAX	FUNCTION
Undefined	a = Undefined	Sets a the value of a variable to undefined
Unset	unset(\$MyArray)	Resets the data in the table

V

CODE	SYNTAX	FUNCTION
VALUE	ANCHOR(TOP, INDEX, VALUE)	Sets the value of the object point for the vertical axis to be a price
Variation	Variation(price)	Difference between the close of the last bar and the close of the current bar in %
ViMinus	ViMinus[N]	Bottom band of the Vortex indicator
ViPlus	ViPlus[N]	Top band of the Vortex indicator
Volatility	Volatility[S, L]	Chaikin volatility
Volume	Volume[N]	Volume indicator
VolumeOscillator	VolumeOscillator[S,L]	Volume Oscillator
VolumeROC	VolumeROC[N]	Volume of the Price Rate Of Change

W

CODE	SYNTAX	FUNCTION
Weeks	TIMEFRAME(X Weeks)	Sets the period to "X Weeks" for further code calculations. See Multi-timeframe instructions
WeightedAverage	WeightedAverage[N](price)	Represents the Weighted Moving Average
WeightedClose	WeightedClose[N]	Average of (2 * Close), (1 * High) and (1 * Low)
WEND	See WHILE/DO/WEND	Ending Instruction of WHILE loop
WHILE/DO/WEND	WHILE (condition) DO (action) WEND	WHILE loop
WilderAverage	WilderAverage[N](price)	Represents Wilder Moving Average
Williams	Williams[N](close)	%R of the Williams indicator
WilliamsAccumDistr	WilliamsAccumDistr(price)	Accumulation/Distribution of Williams Indicator

X

CODE	SYNTAX	FUNCTION
XOR	a XOR b	Logical Operator eXclusive OR
XSHIFT	ANCHOR(TOP, XSHIFT,VALUE)	Defines the value of the object point for the horizontal axis as an offset

Y

CODE	SYNTAX	FUNCTION
<code>Year</code>	<code>Year[N]</code>	Year of the bar n periods before the current bar
<code>Years</code>	<code>TIMEFRAME(X Years)</code>	Set the period to "X Year(s)" for further code calculations See Multi-Timeframe instructions
<code>Yesterday</code>	<code>Yesterday[N]</code>	Date of the day preceding the bar n periods before the current bar
<code>YSHIFT</code>	<code>ANCHOR(TOP, INDEX, YSHIFT)</code>	Defines the value of the object point for the vertical axis as an offset

Z

CODE	SYNTAX	FUNCTION
<code>ZigZag</code>	<code>ZigZag[Zr](price)</code>	Represents the Zig-Zag indicator introduced in the Elliott waves theory
<code>ZigZagPoint</code>	<code>ZigZagPoint[Zp](price)</code>	Represents the Zig-Zag indicator in the Elliott waves theory calculated on Zp points

Other

CODE	FUNCTION	CODE	FUNCTION
<code>+</code>	Addition Operator	<code><></code>	Difference Operator
<code>-</code>	Subtraction Operator	<code><</code>	Strict Inferiority Operator
<code>*</code>	Multiplication Operator	<code>></code>	Strict Superiority Operator
<code>/</code>	Division Operator	<code><=</code>	Inferiority Operator
<code>=</code>	Equality Operator	<code>>=</code>	Superiority Operator

ProRealTime SOFTWARE