visualchart

Functions and properties VBA

Fast guide for user

Indicators

Alert
Angle
CalcDate
CalcTime
Close
CurrentBar
Date
DateSubstract
GetFeedFields
GetBackgroundColor
GetBarColor
GetBarStyle
GetBarRepresentation
GetBarWidth
GetExitOrder
GetHighest
GetHighestBar
GetHistogramBand
GetIndicatorIdentifier GII
GetIndicatorPos
GetIndicatorValue GIV
GetLineName

GetLowest
GetLowestBar
GetNthHighest
GetNthLowest
GetOrderCount
GetOrderDate
GetOrderLabel
GetOrderPrice
GetOrderSide
GetOrderSymbolCode
GetOrderType
GetOrderVolume
GetPivotDown
GetPrice
GetSwingHigh
GetSwingHighBar
GetSwingLow
GetSwingLowBar
GetSymbolIdentifier
GSI
GetSymbolInfo
GetSymbolInfoEx

GetSystemIdentifier GSYSI
GetTrueHigh
GetTrueLow
GetTrueRange
GetTrueRangeCustom
GetVolatility
GetWndBackGroundColor
LimitOrder
LimitPrice
LimitVol
Low
MinutesToTime
NumberOfLines
Open
OpInt
ReleaseDataIdentifier
RDI
RegressionAngle
RegressionSlope
SetBackgroundColor
SetBarColor
SetBarProperties

SetBarRepresentation
SetBarStyle
SetBarWidth
SetHistogramBand
SetIndicatorPos
SetIndicatorValue
SetLineName
SetWndBackgroundColor
ShouldTerminated
Slope
StarBar
Time
TimeEx
TimeToMinutes
Volume

Strategies

- Angle
- AvgBarsInTrade
- AvgLosingTrade
- AvgTrade
- AvgWinningTrade
- BestSeries
- Buy
- CalcDate
- CalcTime
- Close
- ConfigStk
- CurrentBar
- CurrentContract
- CurrentEntries
- Date
- DateSubstract
- EndOfDayTime
- ExitLong
- ExitPositionsAtEndOfDay
- ExitShort
- ExitTrailingLimit
- ExitTrailingStop
- FilledOrders
- GetBarsSinceEntry
- GetBarsSinceExits
- GetConfigStk
- GetDailyLosers
- GetDailyWinners
- GetEntryDate
- GetEntryPrice
- GetEntryTime
- GetExitDate
- GetExitOrder
- GetExitPrice
- GetExitTime
- GetFeedFields
- GetHighest
- GetHighestBar
- GetIndicatorIdentifier GII
- GetIndicatorValue GIV
- GetLowest
- GetLowestBar
- GetMarketPosition
- GetMaxContracts
- GetMaxEntries
- GetNthHighest
- GetNthLowest
- GetOrderCount
- GetOrderCount
- GetOrderDate
- GetOrderLabel
- GetOrderPrice
- GetOrderSide
- GetOrderSymbolCode
- GetOrderType
- GetOrderVolume
- GetOrderVolume
- GetPivotDown
- GetPivotUp
- GetPositionProfit
- GetPrice
- GetStkLength
- GetStkValue
- GetStkValues
- GetSwingHigh
- GetSwingHighBar
- GetSwingLow
- GetSwingLowBar
- GetSymbolIdentifier GSI
- GetSymbolInfo
- GetSymbolInfoEx
- GetSystemIdentifier GSYSI
- GetTrueHigh
- GetTrueLow
- GetTrueRange
- GetTrueRangeCustom
- GetVolatility
- GrossLoss
- GrossProfit
- High
- IsFirstDayBar
- IsLastDayBar
- LargestLosingTrade
- LargestWinningTrade
- LC_Index
- LimitOrder
- LimitPrice
- LimitVol
- Low
- MarketFilledOrders
- MinutesToTime
- NetProfit
- NumberOfLosingTrades
- NumberOfTrades
- NumberOfWinningTrades
- Open
- OpInt
- PercentProfitable
- ProfitFactor
- ReleaseDataIdentifier RDI
- RegressionAngle
- RegressionSlope
- Sell
- ShouldTerminated
- Slope
- StandardDeviation
- StarBar
- Time
- TimeEx
- TimeToMinutes
- TodayCurrentBar
- TodayHigh
- TodayLow
- Volume
- WorstSeries

Studies

Angle
CalcDate
CalcTime
Close
CurrentBar
Date
DateSubstract
FeedFields
GetExitOrder
GetHighest
GetHighestBar
GetIndicatorIdentifier GII
GetIndicatorValue GIV
GetLowest
GetLowestBar
GetNthHighest
GetNthLowest
GetOrderCount
GetOrderDate
GetOrderLabel
GetOrderPrice
GetOrderSide
GetOrderSymbolCode
GetOrderType
GetOrderVolume
GetPivotDown
GetPivotUp
GetPrice
GetSwingHigh
GetSwingHighBar
GetSwingLow
GetSwingLowBar
GetSymbolIdentifier GSI
GetTrueHigh
GetTrueLow
GetTrueRange
GetTrueRangeCustom
GetVolatility
High
LimitOrder
LimitPrice
LimitVol
Low
MinutesToTime
Open
OpInt
PaintBar
PaintCandlestick
PaintMaxMin
PaintSeries
ReleaseDataIdentifier
RDI
RegressionAngle
RegressionSlope
ShouldTerminated
Slope
StarBar
Time
TimeEx
TimeToMinutes
Volume

Alert

Description:
This function is used in indicator programming in order to trigger alerts. When certain conditions defined by the user are fulfilled, a warning message shows up on the screen.

Syntax:
Alert (Description)

Parameters:

Name	Default	Description
Description	“Indicator Alert”	Text showing up when the alert is triggered.

The property **Indicator Alert** must be activated in the indicator properties editor so that the alert message shows up on screen.

Example (VB.NET):

```
Me.Alert ("Warning, Average Crossover")
```

The following message will show up on the screen “Warning, Average Crossover”

Angle

Description:
This function returns the value of the angle between the horizontal line and the regression line formed by the prices StartPrice and EndPrice that related the quotes with the time variable.

Syntax:
Me.Identifier.Angle(StartBar, EndBar, StartPrice, EndPrice)

Parameters:

Name	**Default**	**Description**
StarBar	-	Number of the bar where the straight line starts.
EndBar	-	Number of the bar where the straight line ends up.
StarPrice	-	Start price for the straight line.
EndPrice	-	End price for the straight line.

Example (VB.NET):
Let´s assume that we are willing to know, at any stage, the value in radians between the current closing price and the closing price 30 bars ago. In this case, we should define first the start variable.

```
Dim angleinradians As Double  = Me.Data.Angle(Bar-30,Bar,Me.Data.Close(30),Me.Data.Close(0))
```

The value returned by the call to this property will be assigned to this variable:

AvgBarsInTrade

Description:
This function returns the average number of bars during which a trade is opened. This value will increase by while new bars are generated and its value will depend on the bar on which this property is called.

Syntax:
AvgBarsInTrade

Parameters:

Name	Default	Description
-	-	-

Example (VB.NET):

Let´s assume that we want to know the average number of bars that our trades last. First, we must define a variable:

```
Dim tradeavbars as Double
```

We calculate the average number of bars by trade each time a new trade is generated by asking the following question:

```
If NumberOfTrades > 1 then
```

With this question we make sure that, at least, two trades have been generated.

```
tradeavbars = Me.AvgBarsInTrade
```

The value returned by the property will be assigned to the exit variable.

Then, we can stock the number of trades we have made under another variable previously defined:

```
Dim numcurrenttrades As Long = Me.NumberOfTrades
```

We would calculate again the average number of bars by trades only when NumberOfTrades > NumCurrentTrades

AvgLosingTrade

Description:
This function returns the average results of the losing trades. This value will change as new bars are generated and it will also depend on the bar on which this property is called.

Syntax:
AvgLosingTrade(Show)

Parameters:

Name	Default	Description
Show	Bypoints	Enables to indicate the format on which the information will show up: **SttRepresentation.ByPoints** (points) or **SttRepresentation.Porcentual** (percentage).

Example (VB.NET):

Dim avglosingtrade As Double = Me.AvgLosingTrade(SttRepresentation.ByPoints)

Assigns to the previously defined variable the average losses of all losing trades in points.

AvgTrade

Description:
This function returns the average results of the trades. This value will change by while new bars are generated and its value will depend on the bar on which this property is called.

The returned value is of type *SttRepresentation*.

Syntax:
AvgTrade(Show)

Parameters:

Name	Default	Description
Show	Bypoints	Enables to indicate the format on which the information will show up: **SttRepresentation.ByPoints** (points) or **SttRepresentation.Porcentual** (percentage).

Example (VB.NET):

Dim avgtrade = Me.AvgTrade(SttRepresentation.Porcentual)

This function assigns to the variable the average profit of all trades in %.

AvgWinningTrade

Description:
This function returns the average result of the wining trades. This value will change by while new bars are generated and its value will depend on the bar on which this property is called.

The returned value is of type *SttRepresentation*.

Syntax:
AvgWinningTrade(Show)

Parameters:

Name	Default	Description
Show	Bypoints	Enables to indicate the format on which the information will show up: **SttRepresentation.ByPoints** (points) or **SttRepresentation.Porcentual** (percentage).

Example (VB.NET):

Dim avgwinningtrade = Me.AvgWinningTrade(SttRepresentation.ByPoints)

Assigns to the previously defined variable AvgProfitWinners the average profit of all positive trades (in points).

BestSeries

Description:
This function returns the best value reached by the total profit. This will change by while new bars are generated and its value will depend on the bar on which this property has been called.

Syntax:
BestSeries(Show)

Parameters:

Name	Default	Description
Show	Bypoints	Enables to indicate the format on which the information will show up: **SttRepresentation.ByPoints** (points) or **SttRepresentation.Porcentual** (percentage).

Example (VB.NET):

Me.BestSeries(Percentage)

Returns in % the best value of the total profit reached when applying this property.

Buy

Description:
This function is used to send buy orders in stocks, futures, cfd´s, etc...

Syntax:
Buy(TraderType, Contracts, Price, Label)

Parameters:

Name	Default	Description
Type	AtClose	Type of order to be launched (TraderType.AtClose, TraderType.AtMarket, TraderType.AtLimit and TraderType.AtStop).
Contracts	1	Number of contracts/stocks. The numerical specifications on contracts can be replaced by variables or by any function previously described.
Price	-	Buy price. This parameter must only be indicated for the TraderType.AtStop and TraderType.AtLimit orders. The value can be expressed by using a number, a variable, a function or a mix of a variable and function.
Label	-	Label of the order in text format.

Example (VB.NET):

```
If (Me.GetMarketPosition() <> 1) then
Me.Buy(TraderType.AtStop, 1, Me.Data.High() +10, "C1")
End If
```

Sends a buy order at stop (1 contract), where the stop price is the bar high plus 10 points and the label identifying the order “C1”.

CalcDate

Description:
Sums a certain amount of days to a certain day and returns as a result the resulting date under **military format (AAAAMMDD).**

Syntax:
CalcDate(Date, Days)

Parameters:

Name	Default	Description
Date	-	Date (AAAAMMDD) to which the corresponding amount of days will be added.
Days	-	Amount of days to be added.

Example (VB.NET):

```
Dim newdate As Long = Me.CalcDate(2015110,5)
```

It sums 5 days to the date 10/01/2015, that under military format is 20150110 and as a consequence returns 20150115, that under date format is 15/01/2015.

CalcTime

Description:
Sums an amount of minutes to a certain time and returns the result in military format (HHMM).

Syntax:
CalcTime(Time, Minutes)

Parameters:

Name	Default	Description
Time	-	Time (HHMM) to which a certain amount of minutes will be added.
Minutes	-	Amount of minutes to be added.

Example (VB.NET):

```
Dim newtime As Long = Me.CalcTime(1000,30)
```

Sums 30 minutes to 10:00 am (military format 1000), and thus returns 1030, that under military format is 10:30 am.

Close

Description:
This function returns the value of the close of a certain bar. The property *Close* can be found on any data series or indicator.

Syntax:
Identifier.Close(BarsAgo)

Parameters:

Name	Default	Description
BarsAgo	0	Number of bars backwards. The default value refers to the current bar. In this parameter we can indicate a possible numerical value contained under the variable or type the value directly. It can also be specified as a function replacing the numerical value. This parameter only allows positive values.

Example (VB.NET):

```
Me.Data.Close(3,Data1)
```

Returns the data three bars backwards of the main data source.

ConfigStk

Description:
Enables to set the initial properties of the statistics that we are willing to obtain.

Syntax:
ConfigStk(Sing, Unit, Filt, CompType, Compression, BeginDate, EndDate)

Parameters:

Name	Default	Description
Sing	ssNets	Results filter depending on whether the results are winers (StatisticSign.Wins), losers (StatisticSign.Loss) or all (StatisticSign.Nets).
Unit	suMoney	Data representation in cash (suMoney), in points (suPnts) or in percentage (ssPorc).
Filt	sfAll	Results filter by positions; in fact, depending if they are longs (sfLong) or shorts (sfShort).
CompType	sctTrades	Representation of the data grouped by trades (sctTraders), days (sctDays), weeks (sctWeeks), months (sctMoths) or years (sctYears).
Compression	1	Compression unit.
BeginDate	-	Start date for the data range during which we want to extract the data.
EndDate	-	Final date for the data range during which we want to extract the data.

Example (VB.NET):

```
Me.ConfigStk(StatisticSign.Loss, StatisticUnit.Money, StatisticFilt.Short, StatisticCompType.Days,1,
cDate(”09/08/2010”), cDate(“09/08/2015”)
```

This function sets the properties of a strategy statistics to obtain information in cash for the losses of a single day (only the ones where we have trades short), from August 9th 2010 to August 9th 2015.

CurrentBar

Description:
This function returns the ordinal number of the bar on which the calculations are being run (current bar). Considering the first bar of the data series as bar number 0, the number will be increased in one unity every evaluated bar.

When we work with two data series and one has more historical data than the other one, the calculations will start when one of the bars of the first series coincides in time with one of the bar of the second series. This bar will be considered as first bar (CurrentBar=1).

Syntax:
CurrentBar

Parameters:

Name	Default	Description
-	-	-

Example (VB.NET):

Me.CurrentBar()

If the calculations were being made in the seventh bar of the data series, this function will return for example the value 6.

CurrentContract

Description:
This function is used to know the amount of contracts or shares that are bought or sold in the current opened position.

Syntax:
CurrentContract

Parameters:

Name	Default	Description
-	-	-

Example (VB.NET):

Me.CurrentContract

If there is no position opened, the function will return 0.

CurrentEntries

Description:
This function is used in order to know the number of different entries for an opened position. A position can have different entries in function of the order label or the modality of matching the orders.

Syntax:
CurrentEntries

Parameters:

Name	Default	Description
-	-	-

Example (VB.NET):

Me.CurrentEntries

The function returns the number of entries at the moment of applying the order. If they are no functions, it will return the value 0.

Date

Description:
All the bars of the chart have an associated date, even if they are intraday bars. The function returns the date on which the bar was produced. The returned format is military format, so if the bar was produced on August 10th 2000, the function will return the result 20000810.

The *Date* property can be found in every data series or indicator.

Syntax:
Identifier.Date(BarsAgo)

Parameters:

Name	Default	Description
BarsAgo	0	Number of bars backwards. The default value refers to the current bar. We can indicate for this parameter a numerical value included in a variable or directly type this value. This parameter only allows positive values.

Example (VB.NET):
Me.Data.Date(3)
The function returns the date corresponding to the third bar backwards of the main data series.

DateSubstract

Description:
Returns the difference in days between two dates**.**

Syntax:
DateSubstract(Left, Right)

Parameters:

Name	Default	Description
Left	-	Minuend (date in military format to which we substract)
Right	-	Sustraend (date in military format to which we substract)

Example (VB.NET):
Dim diff As Long = Me.DateSubstract(20150110, 20150120)

The function returns 10.

EndOfDayTime

Description:
Returns the close time of the current session (of main source) in military format (HHMM).

Parameters:

Name	Default	Description

Syntax:
Me.EndOfDayTime()

Example (VB.NET):

We want to enter to a fixed price as long as we are out of the market and this is not the last bar of the session.

```
If GetMarketPosition() = 0 And Me.entryprice <> 0 Then
   If (Me.Data.Time <> Me.EndOfDayTime()) Then
      Me.Buy(TradeType.AtStop, 2, Me.entryprice * (1 + Me.factor / 100))
      Me.Sell(TradeType.AtStop, 2, Me.entryprice * (1 - Me.factor / 100))
   End If
End If
```

We compare the property *Time()* with the property *EndOfDayTime()* in order to check that they are different.

ExitLong

Description:
This function is used when we are willing to close a long position, but not to take, at the same time, a short position. For example, if we have bought 500 shares, and the conditions to exit this trade are fulfilled, we will use this function.

Syntax:
ExitLong Type, Contracts, Price, Label

Parameters:

Name	Default	Description
Type	AtClose	Type of order to be launched (TraderType.AtClose, TraderType.AtMarket, TraderType.AtLimit and TraderType.AtStop).
Contracts	1	Number of contracts/stocks. The numerical specifications on contract numbers can be replaced by variable or any other function previously defined.

Price	---	Buy price. This parameter is only used for TraderType.AtStop and TraderType.AtLimit orders. We can replace this number by a function, a variable or a mixed of both the function and the variable.
Label	---	Order label in text format.

- If the total amount of contracts/shares (Contracts) and neither the label (Label, in case we are using several buy orders) are not specified, the full position will be closed and we will exit the market.
- If the amount of contracts/shares is not specified, but the label is, and if we are using several buy orders, all the contracts/shares corresponding to this label will be closed.
- If the amount of contracts and the label are specified, the amount of contract shares for the order which label has been specified will be closed.
- If the number of contracts/shares is specified, but not the label, and presuming that several positions are opened, the amount of contracts/shares specified in the last order will be closed.

Example (VB.NET):

```
Me.ExitLong(TraderType.AtStop, 1, Me.GetEntryPrice -10, "C1")
```

The position (1 contract), if the order is labelled as “C1”, will be closed with a sell order at stop at the entry price minus 10 points. In this case, we are talking about a stop loss order.

```
Me.ExitLong(TraderType.AtLimit, 1, Me.GetEntryPrice+30)
```

The positions (1 contract) will be closed with a sell limit order at the entry price plus 30 points. In this case, we are talking about a target profit order.

ExitPositionAtEndOfDay

Description:
If the property is established to *True*, then all trades of the strategy will be liquidated at the close of each session.

Attention. This possibility is admitted for all cases of strategy development for backtesting. Nevertherless, if you want to activate the trading of a strategy in real time that use it, it is necessary to contact first your broker to confirm if the method is possible within this broker.

If you want to use this property, we recommend declaring it from the method *OnInitCalculate().*

Syntax:

```
Me.ExitPositionsAtEndOfDay() = True/False
```

Parameters:

Name	**Default**	**Description**
-	-	-

Example (VB.NET):

We want to close all trades at the close of the session according to a parameter. First, we declare the parameter:

```
<Parameter(Name:="EndOfDay", DefaultValue:=1, MinValue:=0, MaxValue:=1, Step:=1)>
Private endofday As Integer
```

Next, we establish the property to true or false according to the parameter:

```
Public Overrides Sub OnInitCalculate()
   Me.ExitPositionsAtEndOfDay = (Me.endofday = 1)
End Sub
```

ExitShort

Description:
This function is used to close a short position without opening a long position. For example, if we have sold 5 future contracts, and the necessary conditions are fulfilled to liquidate the position, we shall use this function.

Syntax:
ExitShort(TraderType, Contracts, Price, Label)

Parameters:

Name	Default	Description
Type	AtClose	Type of order to be launched (TraderType.AtClose, TraderType.AtMarket, TraderType.AtLimit and TraderType.AtStop).
Contracts	1	Number of contracts/stocks. The numercial specifications on contracts can be replaced by a variable or by any function previously defined.
Price	---	Buy price. This parameter must only be indicated for TraderType.AtStop and TraderType.AtLimit orders. The value can be expressed by a number, a variable, a function or by a mix of variable and function.
Label	---	Label of the order in text format.

- If nor the number of contracts/stocks (Contracts) neither the label (Label, if we are using several sell orders) are specified, the full position will be closed and we will exit the market.
- If the number of stokcs is not specified, but the label is (assuming that we are using several sell orders), all the contracts/stocks corresponding to the order with this label will be closed.
- If the number of contracts and the label are specified then this number of contracts/stocks of the order with the specified label will be closed.
- If the number of contracts/stocks is specified, but the label is not, and if there are several opened positions, the number of contracts specified in the last order will be closed.

Example (VB.NET):
```
Me.ExitShort(TraderType.AtStop, 1, Me.Data.High() +10, "ES1")
```

The position (1 contract) of the order labelled as “ES1”, will be closed with a buy stop order at the price of the high of the bar plus 10 points. In this case we are limiting losses with a protection stop.

ExitTrailingLimit

Description:
This method enables to use a *trailing limit* order for a concrete opened position. This function is a novelty at Visual Chart 6 and facilitates the work on the design of dynamic exit orders.

The *trailing limit* order consists of the following (in case of long opened position):

- Stores the highest value reached since the position was opened.
- Calculates the exit price based on said high according to the following function:
  - limitPrice = High + (High x *percentage* x 0.01)
- If it is at the beginning of the trade, it sends an *ExitLong* limit order at the calculated price.
- For the rest of the bars, if the resulting value is **lower** than the last calculated value, the price of the exit order will be replaced and modified.

Syntax:
Me.ExitTrailingLimit(percentage)

Parameters:

Name	Default	Description
Percentage	-	Percentage of margin that will be applied to the most extreme value to position the limit exit order.

Example (VB.NET):

```
If GetMarketPosition() <> 0 Then
   Me.ExitTrailingLimit(pcttrail)
End If
```

Each time there is an opened position, the TrailingLimit will be activated with a percentage depending of the *pcttrail* parameter.

ExitTrailingStop

Description:
This method enables to use a *trailing stop* order for a concrete opened position. This function is a novelty at Visual Chart 6 and facilitates the work on the design of dynamic exit orders.

The *trailing stop* order consists of the following (in case of long opened position):

- Stores the highest value reached since the position was opened.
- Calculates the exit price based on said high according to the following function:
  - stopPrice = High + (High x *percentage* x 0.01)
- If it is at the beginning of the trade, it sends an *ExitLong* stop order at the calculated price.
- For the rest of the bars, if the resulting value is **higher** than the last calculated value, the price of the exit order will be replaced and modified.

Syntax:
Me.ExitTrailingStop(percentage)

Parameters:

Name	Default	Description
Percentage	-	Percentage of margin that will be applied to the most extreme value to position the stop exit order.

Example (VB.NET):

```
If GetMarketPosition() <> 0 Then
   Me.ExitTrailingStop(pcttrail)
End If
```

Each time there is an opened position, the TrailingStop will be activated with a percentage depending of the *pcttrail* parameter.

FeedFields

Description:
This function determines the information fields to which we want to obtain access in real time using the method with the same name.

Syntax:
.FeedFields(Field)

Parameters:

Name	Default	Description
Field	FFLast	**FFAskSize** Amount of offered contracts **FFBidSize** Amount of requested contracts **FFBuy1** Price offered in the first buying position **FFBuyAgency** Buying Agency **FFBuyOrders** Selling Agency **FFDate** Date **FFDecimals** Decimals **FFDescription** Symbol description **FFDiff** Difference **FFDiff_P** Difference % **FFExpiry_Date** Expiry date **FFHigh** High **FFISIN** ISIN code **FFLast** Last price **FFLastVol** Last volume **FFLow** Low **FFMinimumMov** Tick **FFNumTrades** Number of trades **FFOpen** Open

		FFOpenInterest Open Interest **FFPrevious** Previous **FFSell1** Price offered in the first selling position **FFSellAgency** Selling Agency **FFSellOrders** Buy orders **FFSubMarket** Submarket the symbol belongs to **FFTime** Time **FFVolume** Volume

Example (VB.NET):
If we want to obtain the last closed trade, we should previously define an exit variable:
Dim LastTrade as Double
The value returned by this property will be ascribed to this variable:

LastTrade = .FeedFields(FFLast)

FilledOrders

Description:
The property FilledOrders enables to find out if, over a certain bar, buy or sell active orders associated to a certain label have been filled. If so, the function will return 1, if not 0.

Syntax:
FilledOrders(Label, Side, BarsAgo)

Parameters:

Name	Default	Description
Label	-	Label associated to the order that we are willing to check.
Side	-	Position of the order to be checked. (0 buy position; 1 sell position)
BarsAgo	0	Bar from which we are willing to obtain the date. The default value is the current bar.

Example 1 (VB.NET):
Me.FilledOrders(“C1”, 0, 10)
If a buy order labelled as “C1” has been filled 10 bars backwards, the function will return 1.

Example 2 (VB.NET):
In a strategy, we launch three orders at stop, two buy orders with the labels A und B and a sell order with the label C.

In the following bar, we are willing to know how many orders have been executed.

To do so, we proceed as follows:

We define a buy orders counter and a sell orders counter.

Dim buycounter As Integer = 0

```
Dim sellcounter As Integer = 0
If Me.FilledOrders(“A”,0 , 0) =1 then
      buycounter= 1
End If
If Me.FilledOrders(“B”,0 , 0) = 1 then
      buycounter=buycounter+1
End If
If Me.FilledOrders(“C”,1 , 0) =1 then
      sellcounter= 1
End If
```

GetBackGroundColor

Description:

This function returns the background color of a certain bar.

Syntax:

GetBackGrounColor (BarsAgo)

Parameters:

Name	Default	Description
BarsAgo	-	Number of bars backwards. The value 0 refers to the current bar.

Example (VB.NET):

Me.GetBackGroundColor(0)

Returns the value of the window background color for the current bar.

GetBarColor

Description:

This function returns the color of the data line for a certain bar.

Syntax:

GetBarColor (BarsAgo, Line)

Parameters:

Name	Default	Description
BarsAgo	-	Number of bars backwards, 0 corresponds to the current bar.
Line	-	Line in the current bar whose color we are willing to obtain.

Example (VB.NET):

Me.GetBarColor(1,3)

Returns the color in the previous bar for the data line number 3.

GetBarsSinceEntry

Description:
This function is used to find out the number of bars completed since a certain position was opened (long or short). If no position has been opened, it will return 0 as a result.

Syntax:
GetBarsSinceEntry(EntryAgo)

Parameters:

Name	Default	Description
EntryAgo	0	Number of positions backwards. The default value refers to the current position.

Example (VB.NET):
Me.GetBarsSinceEntry (0)

Returns the number of bars formed since the currrent position was opened.

GetBarsSinceExit

Description:
This function is used to know the number of bars completed since a certain position was opened. If we have not exit any position, the function returns the value 0.

Syntax:
GetBarsSinceExit(EntryAgo)

Parameters:

Name	Default	Description
EntryAgo	0	Number of positions backwards. The default value refers to the current positions.

Example (VB.NET):
Me.GetBarsSinceExit (1)

Returns the number of bars formed since the close of the previous trade.

GetBarStyle

Description:
This function returns the style of the line used in a certain bar.

Syntax:
GetBarStyle(BarsAgo, Line)

Parameters:

Name	Default	Description
BarsAgo	-	Number of bars backwards. The value 0 refers to the current bar.

Line	-	Identifies the line to which the bar from which we are willing to obtain the style belongs.

Example (VB.NET):
Me.GetBarStyle (0,1)

It returns the style of the line in the current bar for line number 1. The values that can be returned by the function are:

- **LineStyle.Solid:** Full line.
- **LineStyle.Dash:** Dashed line.
- **LineStyle.Dot:** Dotted line.
- **LineStyle.DashDot:** Dashed line with point.
- **LineStyle.DashDotDot:** Dashed line with two points.

GetBarRepresentation

Description:
This function returns the representation type used in the indicator for a line in a certain bar. The returned value is of the sort *IndicatorRepresentation*.

Syntax:
GetBarsSinceExit(BarsAgo, Line)

Parameter:

Name	Default	Description
BarsAgo	-	Number of bars backwards. The value 0 refers to the current bar.
Line	-	Identifies the line to which the bar from which we are willing to obtain the style belongs.

Example (VB.NET):

```
If Me.GetBarRepresentation(10, 1) = IndicatorRepresentation.Lineal The
    MsgBox("Line 1 was represented with a line 10 bars backwards.")
End If
```

The values that can be returned by the function are::

- **IndicatorRepresentation.Bars:** Representation using bars.
- **IndicatorRepresentation.Candlestic:** Representation using candles.
- **IndicatorRepresentation.DottedLine:** Representation using dotted line.
- **IndicatorRepresentation.FilledHistogram:** Representation using filled histogram.
- **IndicatorRepresentation.Histogram:** Representation using histogram.
- **IndicatorRepresentation.Lineal:** Linear.
- **IndicatorRepresentation.Parabolic:** Representation using parabolic lines.
- **IndicatorRepresentation.Volume:** Representation using volume.

GetBarWidth

Description:
Returns the width of a certain line for the indicated bar.

Syntax:
GetBarWidth(BarsAgo, Line)

Parameters:

Name	Default	Description
BarsAgo	-	Number of bars backwards. The value 0 refers to the current bar.
Line		Identifies the line to which the bar from which we are willing to obtain the width belongs.

Example (VB.NET):
Me.GetBarWidth(0,1)

The function returns the width (1,2,…) of line 1 in the current bar.

GetConfigStk

Description:
This procedure returns the initial properties of the statistics in current use.

Syntax:
GetConfigStk(Sing, Unit, Filt, CompType,Compressión, BeginDate, EndDate)

Parameters:

Name	Default	Description
Sing	-	Result filter depending of the trade results; winners (StatisticSign.Wins) or losers (StatisticSign.Loss).
Unit	-	Representation of the figures in cash (StatisticUnit.Money), in points (StatisticUnit.Pnts) or in percentage (StatisticUnit.Portc).
Filt	-	Result filter by position, long (StatisticFilt.Long), short (StatisticFilt.Short) or all (StatisticFilt.All).
CompType	-	Representation of the figures grouped by trades (StatisticCompType.Trades), days (StatisticCompType.Days), weeks (StatisticCompType.Weeks), months (StatisticCompType.Moths) or years (StatisticCompType.Years).
Compressión	-	Compression units.
BeginDate	-	Start date for the statistics time interval.
EndDate	-	End date for the statistics time interval.

Example (VB.NET):
Lest´s suppose that we have defined the following start variables:

```
    Dim Sign As StatisticSign
    Dim Unit As StatisticUnit
    Dim Filter As StatisticFilt
    Dim TComp As StatisticCompType
    Dim Compression As Long
    Dim DateStart As Date
    Dim DateEnd As Date
```

When calling the property GetConfigStk:

```
    GetConfigStk(Sign, Unit, Filter, TComp, Compression, DateStart , DateEnd)
```

Each of the variables previously defined will be filled with the figures returned by GetConfigStk.

Following with the example given for the property ConfigStk, when using GetConfigStk, we will get the following information as a result:

Sign = ssLoss = 2	Filter = sfShort = 2	Compression = 1	DateEnd = 09/08/2010
Unit = suMoney = 0	Tcomp = sctDays = 3	DateStart = 09/08/2009	

GetDailyLosers

Description:
This function returns the amount of losing trades between two given dates. It will be always compared with the date of the trade start.

Syntax:
GetDailyLosers(FromDate, ToDate)

Parameters:

Name	Default	Description
FromDate	-	Start date for the interval to be analyzed.
ToDate	-1	End date for the interval to be analyzed. If no value is specified or -1 is assigned, it means that the end date of the interval will be the current date.

Example (VB.NET):
If we want to know the number of losing trades between two dates 10/06/2015 and 10/09/2015.

First, we define the variable to which we will assign the number of losing trades and we assing the value returned by this function as follows:

```
    Dim Losingtrades As Long =  Me.GetDailyLosers(“20150610”,”20150910”)
```

GetDailyWinners

Description:
This function returns the amount of winning trades between two given dates. It will be always compared with the date of the trade start.

Syntax:
GetDailyWinners(FromDate, ToDate)

Parameters:

Name	Default	Description
FromDate	-	Start date for the interval to be analyzed.
ToDate	-1	End date for the interval to be analyzed. If no value is specified or -1 is assigned, it means that the end date of the interval will be the current date.

Example (VB.NET):
If we want to know the number of winning trades between two dates 10/06/2015 and 10/09/2015.
First, we define the variable to which we will assign the number of winning trades and we assing the value returned by this function as follows:

```
Dim WinningTrades As Long = Me.GetDailyWinners(“20150610”,”20150910”)
```

GetEntryDate

Description:
This function is used to find out the date on which a position has been opened. The format of the returned date is military (**AAAAMMDD).** Therefore, if the position was opened on June 25th, the indicated function will return the numerical value 20100625. If no position has been started, it will return 0.

Syntax:
GetEntryDate(EntryAgo)

Parameters:

Name	Default	Description
EntryAgo	0	Number of positions backwards. The default value refers to the current position.

Examples (VB.NET):
Me.GetEntryDate(0)   The function returns the date on which the current position was opened.
Me.GetEntryDate(5)   The function returns the date on which the fifth position ago was opened.

GetEntryPrice

Description:
This function is used to find out the price at which a position has been opened.

It returns the entry price of the position indicated in the parameter EntryAgo. If no position has been found, it will return 0.

Syntax:
GetEntryPrice(EntryAgo)

Parameters:

Name	Default	Description
EntryAgo	0	Number of positions backwards, that we are willing to check. The default value refers to the current position.

Examples (VB.NET):

Me.GetEntryPrice(0)	The function returns the entry price of the current position.
Me.GetEntryPrice(5)	The function returns the entry price of the trade 5 trades ago.

GetEntryTime

Description:
This function is used to know the time at which a position has been opened. This information will be returned under 24 hours military format **(HHMM),** so that, if the time is 5:35 pm it will be considered as the numerical format 1735. If no position has been started, the function will return the value 0.

Syntax:
GetEntryTime(EntryAgo)

Parameters:

Name	Default	Description
EntryAgo	0	Number of positions backwards that we are willing to check. The default value refers to the current position.

Examples (VB.NET):

Me.GetEntryTime(0)	The function returns the entry time of the current position.
Me.GetEntryTime(4)	The function returns the entry time four positions backwards.

GetExitDate

Description:
This function is used to know the data on which a position has been closed. The format on which the date is returned is military format (**AAAAMMDD).** If the position was closed on June 25th, the indicated function will return the numerical value 20100625. If the position has not been closed yet, the function will return 0.

Syntax:
GetExitDate(EntryAgo)

Parameters:

Name	Default	Description
EntryAgo	0	Number of positions backwards. The default value refers to the current position.

Notes:
We must take into account that, at all effects, the last opened trade is considered to be closed in the current bar (bar on which the calculations are being made). Thus, if the value 0 is indicated in the parameter **EntryAgo**, this function will return the date of the current bar.

Therefore, if only the functions Buy and Sell are used in the strategy, the value 1 applied to the parameter **EntryAgo** will return the date of the last closed operation before the current one. While the value 0 will always return the date of the current bar, indicating that there is currently a position opened.

But if we also use the functions ExitLong and ExitShort, the case can occur where there is no position opened in the current bar. In these cases, the value 0 applied to the parameter **EntryAgo** will not return anything, as there is no position opened, while the value 1 will return the data when the last position was closed.

Example (VB.NET):
Me.GetExitDate(3)

The function will return the exit date three trades backwards.

GetExitOrder

Description:
Specifies if the nth order given over a certain bar for a Data (System type) is an exit order or not. If it has been an exit order, the function returns **True**. On the other hand, if the order has been an entry order the function will return **False**.

Therefore, it is necessary to declare first a *strategy* type object.

Syntax:
Me.Strategy.GetExitOrder(BarsAgo, NumOrder)

Parameters:

Name	Default	Description
BarsAgo	0	Bar from which we are willing to obtain the data. The default value refers to the current bar.
NumOrder	-	Position of the order from which we are willing to obtain the information. The position list starts at 0.

Example (VB.NET):

We want to close the strategy trades when the strategy *AdxBandSystem 02* closes.

First, we have to declare the *ADXBANDSYS02* type object:

```
Dim adxbdsysdata As ADXBANDSYS02
```

Next, we create the object from the method *OnInitCalculate*:

```
            Me.adxbdsysdata = New ADXBANDSYS02(Me.Data)
    From OnCalculateBar() we add the following code:
            If (Me.adxbdsysdata.GetExitOrder(0, 0)) Then
               Me.ExitLong(TradeType.AtMarket, 1)
               Me.ExitShort(TradeType.AtMarket, 1)
            End If
```

GetExitPrice

Description:
This function is used to know the price at which a position has been closed. If no position has been closed, it will return 0.

Syntax:
GetExitPrice(EntryAgo)

Parameters:

Name	Default	Description
EntryAgo	0	Number of positions backwards that we are willing to check. The default value refers to the current position.

It is important to take into account the observations of the function GetExitDate.

Example (VB.NET):
Me.GetExitPrice(5)   The function returns the exit price 5 trades ago.

GetExitTime

Description:
This function is used to know the time of a position closing. The time will be returned under 24 hours military format **(HHMM),** so that if the time is 5:35 pm we will consider it as the numerical value 1735. If no position has been closed, the function will return the value 0.

Syntax:
GetExitTime(EntryAgo)

Parameters:

Name	Default	Description
EntryAgo	0	Number of positions backwards that we are willing to check. The default value refers to the current position.

Example (VB.NET):
Me.GetExitTime(3)   The function returns the exit time three trades backwards.
It is very important to take into account the observations of the function GetExitDate.

GetFeedFields

Description:

This function determines the information fields to which we want to obtain access in real time using the method with the same name.

Syntax:

GetFeedFields(Field)

Parameters:

Name	Default	Description
Field	FFLast	**FeedFields.AskSize** Amount of offered contracts **FeedFields.BidSize** Amount of requested contracts **FeedFields.Buy1** Price offered in the first buying position **FeedFields.BuyAgency** Buying Agency **FeedFields.BuyOrders** Selling Agency **FeedFields.Date** Date **FeedFields.Decimals** Decimals **FeedFields.Description** Symbol description **FeedFields.Diff** Difference **FeedFields.Diff_P** Difference % **FeedFields.Expiry_Date** Expiry date **FeedFields.High** High **FeedFields.ISIN** ISIN code **FeedFields.Last** Last price **FeedFields.LastVol** Last volume **FeedFields.Low** Low **FeedFields.MinimumMov** Tick **FeedFields.NumTrades** Number of trades **FeedFields.Open** Open **FeedFields.OpenInterest** Open interest **FeedFields.Previous** Previous **FeedFields.Sell1** Price offered in the first selling position **FeedFields.SellAgency** Selling Agency **FeedFields.SellOrders** Sell orders **FeedFields.SubMarket** Submarket the symbol belongs to **FeedFields.Time** Time **FeedFields.Volume** Volume

Example (VB.NET):

If we want to obtain the last closed trade, we should previously define an exit variable to which we will ascribe the value returned by this property:

```
Dim lasttrade as Double = Me.GetFeedFields(FeedFields.Last)
```

GetHighest

Description:
This function is used to obtain the highest value of the last **n** bars.

Syntax:
Me.Identifier.GetHighest(Type.Price, Length)

Parameters:

Name	Default	Description
TPrice	PriceClose	Field of the bar we want to refer to. To do so, we must indicate in this field any of the following values: **PriceHigh:** Equivalent to the High **PriceLow:** Equivalent to the Low **PriceOpen:** Equivalent to the Open **PriceClose:** Equivalent to the Close **PriceVolume:** Equivalent to the Volume If this function is calculated with an indicator as data source, we will use as parameter in **TPrice** the value **PriceClose**, referring to the close of the identifier data series. On the other hand, if we indicated **PriceHigh** or any other, it would not make any difference as it would always return the same value.
Length	1	Number of bars backwards to be considered. Any numerical type variable can be used to replace a number.

Examples (VB.NET):
Me.Data.GetHighest(Price.High, 10)
The function returns the numerical value of the highest high of the last 10 bars and for the indicated data series (Data).

We create the object *AvSimple* from *OnInitCalculate.*

```
Me.avsimpledata = New AvSimple(Me.Data)
So that
Me.avsimpledata.GetHighest(Price.Close, 100)
The function returns the highest value of the moving average of the last 100 bars.
```

GetHighestBar

Description:
This function returns the **distance in bars** of the highest value of the last **n** bars (included the current bar) of a determined source object. The function is included in every class of type source, such as a data series or an indicator.
The distance in bars will be calculated regarding the current bar. As a result, if the highest price is in the current bar, the returned value will be 0. If it is in the 15th bar backwards, the returned value will be 15, and so forth and so on.

Syntax:
Me.Identifier.GetHighestBar (Type.Price, Length)

Parameters:

Name	Default	Description
Tprice	PriceClose	Field of the bar we want to refer to. To do so, we must indicate in this field any of the following values: **PriceHigh:** Equivalent to the High **PriceLow:** Equivalent to the Low **PriceOpen:** Equivalent to the Open **PriceClose:** Equivalent to the Close **PriceVolume:** Equivalent to the Volume If this function is calculated with an indicator as data source, we will use as parameter in **TPrice** the value **PriceClose**, referring to the close of the identifier data series. On the other hand, if we indicated **PriceHigh** or any other it would not make any difference as it would always return the same value.
Length	2	Number of bars backwards to be considered. The default value 2 will return 0 or 1 (depending on the High). Any numerical variable can be used instead of a number.

Example (VB.NET):
Me.Data.GetHighestBar(Price.Low, 20)

The function returns the bar number (backwards), where the highest low of the last 20 bars of the indicated data series (Data1) is obtained.

GetHistogramBand

Description:
This function returns the band line number ascribed to the line specified in the parameter with the same name.

Syntax:
GetHistogramBand(Line)

Parameters:

Name	Default	Description
Line	-	Identifies the data line whose band line number we are willing to know.

Example (VB.NET):
Imagine that we want to create an indicator and we are representing it the following way:

```
Me.SetIndicatorValue(x, 1)
Me.SetIndicatorValue(50, 2)
Me.SetBarRepresentation(0,1,irHistogram)
Me.SetHistogramBand(1, 2)
```

If we define a start variable to which we can assign the value returned by the function:

Dim mybandline as Integer = Me.GetHistogramBand(1)
The function will return the value 2 (the band line).

GetIndicatorIdentifier - GII

Description:
This function is used to create the data series corresponding to any indicator and to obtain an identifier of this series. To do so, we need to declare first a variable of DataIdentifier type.
Once the variable is defined we will always assign to it the value of the function **GetIndicatorIdentifier** in order to create the indicator data series and to obtain an identifier of the same indicator.
The identifier of the indicator must be obtained from the procedure OnInitCalculate.
Later on, in order to obtain the value of an indicator we must use the function GetIndicatorValue and indicate in the parameter **Data** the variable on which we have saved the value of the correponding indicator.
The identifier obtained by this function can be used on any VBA function on which a Data is required (Data series on which the different functions are calculated).

Syntax:
GetIndicatorIdentifier(Name, ParentDataIdentifier, ParamArray())

We can also use the short method **GII**:
GII(Name, ParentDataIdentifier, ParamArray())

Parameters:

Name	Default	Description
Name	-	Indicator code.
ParentDataIdentifier	-	Identifier of the series on which the indicator is calculated. If we set this parameter as data, we will be calculating the indicator on the data or series on which the strategy is being calculated. If we are willing to obtain the identifier of an indicator being calculated on another indicator we shall indicate within this parameter the identifier of the indicator we are willing to use as calculation.

ParamArray	-	First parameter specific of the indicator. It represents a group of parameters which number is not specified, as each of the indicators has a variable number (a moving average has 2, while an RSI has 3). The order on which the parameters must be specified is the same as these indicators figure out on the dialog box showing up when we are ready to plot the indicator into a chart. If the parameter is **Price** type, it refers to one of the fields of the bar (Close, Open, etc.), as it would be the case of a moving average as its second parameter is the data source. In these cases we must specify the field of the bar on which we are going to calculate the indicator. We shall indicate any of the following constants equivalent to those fields: **PriceHigh:** Equivalent to the High **PriceLow:** Equivalent to the Low **PriceOpen:** Equivalent to the Open **PriceClose:** Equivalent to the Close **PriceVolume:** Equivalent to the Volume
ParamArray	-	Second parameter specific of the indicator.
...	-	...
n-th ParamArray	-	n-th parameter specific to the indicator.

Example (VB.NET):

We declare an object *DataIdentifier*:

Dim rsidata As DataIdentifier

We create the object from *OnInitCalculate*:

Me.rsidata = Me.GetIndicatorIdentifier(Indicators.RSI, DataIdentifier.Data1, 14, 70, 30)

The source returns the indicator identifier RSI (14,70 and 30 are the indicator parameters).

Visual Chart 6 novelties:

Visual Chart 6 wants to get the most of the project design using object-oriented programming languages.

As a result, we will see that we can create objects of equivalent classes to the indicator that we want to add.

So, we can define objects of *AvSimple, RSI* or *MACD* type. Every created indicator, public as well as the indicators created with your user, adds a new class of the same type. In this way, it is lighter to add indicators to a project because it is not necessary to use the method *GetIndicatorIdentifier*.

Example (VB.NET):

The previously created object *rsidata* can also be declared as follows:
Dim rsidata As RSI
Then we create it as follows:
Me.rsidata = New RSI(Me.Data)

GetIndicatorPos

Description:
This function obtains the trend of the indicator within a certain bar.

Syntax:
GetIndicatorPos(BarsAgo, Line)

Parameters:

Name	Default	Description
BarsAgo	-	Number of bars backwards.
Line	-	Identifies the line, the bar we are willing to obtain the trend from belongs to.

Example (VB.NET):

```
If Me.GetIndicatorPos(0, 1) = IndicatorPosition.Bull Then
            MsgBox(“Bullish trend indicator.”)
      End If
```

The values that can be returned by this function are the following:

- **IndicatorPosition.Bull**
- **IndicatorPosition.Bear**
- **IndicatorPosition.Neutral**

GetIndicatorValue - GIV

This function is used to obtain the value of an indicator. To do so, we must have previously determined the identifier of the indicator in the procedure InitCalculate via the function GetIndicatorIdentifier.

Syntax:
GetIndicatorValue(Identifier, BarsAgo, LineNumber)
We can also use the abbreviation **GIV**
.GIV(Identifier, BarsAgo,LineNumber)

Parameters:

Name	Default	Description
Identifier	-	Identifier of the indicator.
BarsAgo	0	Represents the number of bars backwards we are referring to in order to obtain the value of the indicator. If we are using a moving average, a value equal to 0 for this parameter, will force

		the function to return the value of the average in the current bar. A value equal to 1 will return the value of the average a bar ago and so forth and so on.
LineNumber	1	Line of the indicator to be obtained. Some indicators have more than one data line. In these cases, if we give to this parameter a value 1, it will return a value referring to the first data line, 2 for the second and so forth and so on.

Example (VB.NET):
We declare an object *DataIdentifier* as follows:
Dim rsidata As DataIdentifier

We create the object from *OnInitCalculate*:
Me.rsidata = Me.GetIndicatorIdentifier(Indicators.RSI, DataIdentifier.Data1, 14, 70, 30)

The following line:
Me.GetIndicatorValue(Me.rsidata, 0, 1)
returns the value of the first indicator line in the current bar.

Visual Chart 6 novelties:
Visual Chart 6 wants to get the most of the project design using object-oriented programming languages. As a result, we will see that we can create objects of equivalent classes to the indicator that we want to add. Objects of *Indicator* type include a property called *Value* that replaces the previously explained function *GetIndicatorValue*.

Syntax:
Me.Indicator.Value(BarsAgo, Line)
Example (VB.NET):

We create a RSI type object:
Me.rsidata = New RSI(Me.Data)

We extract the value of the line 1 of the indicator in the current bar from OnCalculateBar:
Me.rsidata.Value(0, 1)

GetLineName

Description:
This function returns the Name of the indicated data line.

Syntax:
GetLineName(Line)

Parameters:

Name	Default	Description
Line	1	Identifies the line from which we are willing to obtain the Name.

Example (VB.NET):
Me.GetLineName(1)

If we were programming an indicator and had assigned to line number 2 the Name "UpperBand", the function .GetLineName(1) will return "UpperBand".

GetLowest

Description:
This function is used to obtain the lowest value of the last n bars of a data series.

Syntax:
Me.Identifier.GetLowest (TPrice, Length)

Parameters:

Name	Default	Description
TPrice	PriceClose	Field of the bar to which we want to refer. To do so, we must indicate any of the following values: **PriceHigh:** Equivalent to the High **PriceLow:** Equivalent to the Low **PriceOpen:** Equivalent to the Open **PriceClose:** Equivalent to the Close **PriceVolume:** Equivalent to the Volume If this function is calculated using an indicator as data source, we will use as parameter in **TPrice** the value **PriceClose**, that refers to the data series of the identifier. If we indicated **PriceHigh** or any other, we still get the same value as return.
Length	1	Number of bars backwards to consider, including the current bar. The default value wil return the High of the current bar. Any numerical type variable can be used instead of the number.

Example (VB.NET):
Me.Data.GetLowest(Price.Low, 10)
The function returns the lowest price of the 10 latest bars of the main data series (Data1).

We create an *AvSimple* type object from *OnInitCalculate.*

```
    Me.avsimpledata = New AvSimple(Me.Data)
```

So that

Me.avsimpledata.GetLowest(Price.Close, 100)

It returns the lowest price of the moving average of the last 100 bars.

GetLowestBar

Description:

Returns the **distance in bars** of the lowest value of the last **n** bars (included the current bar) of a determined source object. The function is included in every class of type source, such as a data series or an indicator.

The distance in bars will be calculated regarding the current bar. As a result, if the lowest price is in the current bar, the returned value will be 0. If it is in the 15th bar backwards, the returned value will be 15, and so forth and so on.

Syntax:

Me.Identifier.GetLowestBar (Type.Price, Length)

Parameters:

Name	Default	Description
Identifier	Data	Source identifier. Any data series (high, low, closes indicators ...). If there is more than 1 data source inserted in the same window, they will be noted as Data1, Data2, Data3, etc.
Tprice	PriceClose	Field of the bar to which we want to refer. To do so, we must indicate in this field the enumerator of the Types.Price enumeration: **PriceHigh:** Equivalent to the High **PriceLow:** Equivalent to the Low **PriceOpen:** Equivalent to the Open **PriceClose:** Equivalent to the Close **PriceVolume:** Equivalent to the Volume If this function is calculated using an indicator as data source, we will use as parameter in **TPrice** the value **PriceClose**, that refers to the data series of the identifier. If we indicate **PriceHigh** or any other, we still get the same value as return.
Length	2	Number of bars backwards to consider including the current bar. The standard value 2 will return 0 or 1 (depending on the Low). Any numerical variable can be used instead of a number.

Example (VB.NET):

Me.Data.GetLowestBar(Price.High, 20)

The function returns the number of bars (backwards), where the lowest high of the latest 20 bars of the main data series is produced (Data1).

GetMarketPosition

Description:
This function is used to know, while the strategy is being calculated, which is our position in the market, long, short or flat. This function is very useful if we are working with stop or limit orders as we do not know when they have been filled.

The values that can be returned by this function are the following:

- If there are opened positions (long), it returns 1.
- If there are opened positions (short), it returns -1.
- If there is no opened position, it returns 0.

Syntax:
GetMarketPosition (EntryAgo)

Parameters:

Name	Default	Description
EntryAgo	0	Number of trades backwards, by default we are using the current position.

Example (VB.NET):

```
If (Me.GetMarketPosition(0) = 1) Then
          Me.ExitLong(TradeType.AtStop, 1, Me.GetEntryPrice() - 20)
        End If
```

If the function returns 1, then we send a protection stop for long positions.

GetMaxContracts

Description:
This function is used to know the máximum amount of contratcts (long or short) negotiated within a single position.

Syntax:
GetMaxContracts (EntryAgo)

Parameters:

Name	Default	Description
EntryAgo	0	Number of positions backwards, by default we are using the current position.

Example (VB.NET):

```
If (Me.GetMarketPosition(0) = 1) Then
      If (Me.CurrentContracts() = Me.GetMaxContracts()) Then
        If (Me.Data.Close() > Me.GetEntryPrice() + 100) Then
          Me.ExitLong(TradeType.AtMarket, Me.CurrentContracts() / 2)
        End If
```

```
          End If
          Me.ExitLong(TradeType.AtStop, Me.CurrentContracts(), Me.GetEntryPrice() - 20)
      End If
```

If we are opened (long), we pretend to reach a partial aim with half of all contracts. For this purpose, we check if *CurrentContracts* is equal to *GetMaxContracts.* In this case, we check the possible partial close. If the value *CurrentContracts* is lower, it means that we have already done the partial close and, therefore, we do not access to this part of the code.

GetMaxEntries

Description:
This function is used to know the maximum amount of different entries within a single position. A position can have different entries according to the label established in the order and the orders matching modality.

Syntax:
GetMaxEntries (EntryAgo)

Parameters:

Name	Default	Description
EntryAgo	0	Number of positions backwards, by default we are using the current position.

Example (VB.NET):
Me.GetMaxEntries(5) The function returns the highest number of entries 5 trades backwards.

GetNthHighest

Description:
This function is used to obtain the highest value of the last n bars and with a certain order of a specific data source. The function is included in every class of type source, such as a data series or an indicator.

Syntax:
Me.Identifier.GetNthHighest (Nth, TPrice, Length)

Parameters:

Name	Default	Description
Nth	1	This is the ordinal representing the value that we are willing to obtain. If Nth is worth 1, we will obtain the first highest value of the last **n** bars of the series. If Nth is worth 2, we will obtain the second highest value of the series and so forth and so on.
TPrice	PriceClose	Field of the bar to which we are willing to refer. To do so we must indicate in this field any of the following values: **PriceHigh:** Equivalent to the High **PriceLow:** Equivalent to the Low **PriceOpen:** Equivalent to the Open **PriceClose:** Equivalent to the Close **PriceVolume:** Equivalent to the Volume

		If we calculate this function with an indicator as data source, we will pass as parameter in **TPrice** the value **PriceClose**, that refers to the close of the indicator data series. If we had indicated **PriceHigh** or any other, it will still return the same value.
Length	50	Number of bars backwards to be considered.

Example (VB.NET):

```
Dim lasthigh As Double = Me.Data.GetNthHighest(1, Price.High, 100)
            Dim prelasthigh As Double = Me.Data.GetNthHighest(2, Price.High, 100)
            If (lasthigh > prelasthigh + 20) Then
               Me.Sell(TradeType.AtMarket, 1)
            End If
```

We open a short position if the greater high is at least 20 points over the penultimate greater high of the last 100 bars.

GetNthLowest

Description:
This function is used to obtain the lowest value of the last **n** bars and with a certain order of a specific data source. The function is included in every class of type source, such as a data series or an indicator.

Syntax:
Me.Identifier.Get**NthLowest(Nth, TPrice, Length)**

Parameters:

Name	Default	Description
Nth	1	This is the ordinal representing the value that we are willing to obtain. If Nth is worth 1, we will obtain the first lowest value of the last **n** bars of the series. If Nth is worth 2, we will obtain the second lowest value of the series and so forth and so on.
TPrice	PriceClose	Field of the bar to which we are willing to refer. To do so, we must indicate in this field any of the following values: **PriceHigh:** Equivalent to the High **PriceLow:** Equivalent to the Low **PriceOpen:** Equivalent to the Open **PriceClose:** Equivalent to the Close **PriceVolume:** Equivalent to the Volume If we calculate this function with an indicator as data source, we will pass as parameter in **TPrice** the value **PriceClose**, that refers to the close of the indicator data series. If we had indicated **PriceHigh** or any other, it will still return the same value.
Length	50	Number of bars backwards to be considered.

Example (VB.NET):

```
Dim lastlow As Double = Me.Data.GetNthLowest(1, Price.Low, 100)
```

```
    Dim prelastlow As Double = Me.Data.GetNthLowest(2, Price.Low, 100)
    If (lastlow < prelastlow - 20) Then
       Me.Buy(TradeType.AtMarket, 1)
    End If
```

We open a long position if the smaller low is at least 20 points under the penultimate smaller low of the last 100 bars.

GetOrderCount

Description:
Returns the amount of active orders given in a single bar of an object of strategy type.
Therefore, it is necessary to declare first an object of *strategy* type.

Syntax:
Me.Strategy.GetOrderCount(BarsAgo)

Parameters:

Name	Default	Description
BarsAgo	0	Bar from which we are willing to extract the amount of orders. The default value refers to the current bar.

Example (VB.NET):

Assuming we declare an object of type *DEFSYS (Default System)*:

```
Dim defsysdata As DEFSYS
```

Imagine that the object defsysdata to which we want to refer to in the current bar sends to the market the following orders:

- An order to change the position in to short as the strategy is long
- A profit exit order
- A stop loss order

We can assign to a variable previously defined the value returned by this function:

```
Dim ordersnum As Integer = Me.defsysdata.GetOrderCount(0)
```

The function returns the value 3 in the current bar as this is the number of active orders on it.

GetOrderDate

Description:
This function returns the data ascribed to the n^th active order of a strategy given in a certain bar.
Therefore, it is necessary to declare first an object of *strategy* type.

Syntax:

Me.Strategy.GetOrderDate(Identifier, BarsAgo, NumberOrder)

Parameters:

Name	**Default**	**Description**
BarsAgo	0	Bar from which we are willing to extract the order. The default value refers to the current bar.
Number Order	0	Number of reference order ascribed to the order from which we are willing to obtain the information. The default value refers to the first active order in a certain bar.

Example (VB.NET):

Assuming we declare an object of type *DEFSYS (Default System)*:

```
Dim defsysdata As DEFSYS
```

Following the example used in the previous function .GetOrderCount, if we only want to know the date of the latest of the three orders, we will proceed as follows:

```
Dim lastorderdate As Date =Me.defsysdata.GetOrderDate(0, 2)
```

The function returns a DD/MM/YYYY date type. Generally, the date returned will be the corresponding to the current bar (in the case of BarsAgo=0), or to the date of the bar we are referring to (in the case of BarsAgo >0)

The position we are referring to in this case is 2. As indicated in the section Parameters of this function, the value 0 is ascribed to the first order, the value 1 to the second and so forth and so on.

GetOrderLabel

Description:

This function returns the label assigned to the nth active order of a strategy given in a certain bar.

Therefore, it is necessary to declare first an object of *strategy* type.

Syntax:

Me.Strategy.GetOrderLabel(BarsAgo, NumberOrder)

Parameters:

Name	**Default**	**Description**
BarsAgo	0	Bar from which we are willing to extract the label. The default value refers to the current bar.
Number Order	0	Number of reference order ascribed to the order from which we are willing to obtain the information. The default value refers to the first active order in a certain bar.

Example (VB.NET):
Assuming we are the *userx* and we have created the strategy *MyStrategy*, at some point, this strategy proceeds as follows:

1) As entry it has sent a Buy order labelled as "Buy_1"
2) And it prepares 3 new orders:

    a. Me.ExitLong(TradeType.Atlimit 1, GetEntryPrice + 50, "Compra_1")
    b. Me.ExitLong(TradeType.Atstop, 1, GetEntryPrice - 20, "Compra_1")
    c. Me.Sell(TradeType.Atstop, 1, .Low – 100, "Venta_1")

On this basis, we have a second strategy in which we declare an object of *userx_MyStrategy* type:

Dim mystrdata As userx_MyStrategy

Within the method *OnCalculateBar*, we indicate the following:

Dim label As String = Me.mystrdata.GetOrderLabel(0, 1)

In label, we will obtain "Buy_1", which is the label assignated to the second order of *MyStrategy*.

GetOrderPrice

Description:
This function returns the price of the nth active order of a strategy given in a certain bar.
Therefore, it is necessary to declare first an object of *strategy* type.

Syntax:
Me.Strategy.GetOrderPrice(BarsAgo, NumberOrder)

Parameters:

Name	Default	Description
BarsAgo	0	Bar from which we are willing to extract the label. The default value refers to the current bar.
Number Order	0	Number of reference order ascribed to the order from which we are willing to obtain the information. The default value refers to the first active order in a certain bar.

Example (VB.NET):
By keep going with the example used for the function .GetOrderCount, if, from the three orders in the current bar, we are willing to know the price of the first of them, we can define a Double type variable, to which we will assign the value returned by the function .GetOrderPrice:

Dim Price as Double

Price = .GetOrderPrice(My SystemData, 0, 0)

Assuming we declare an object of type *DEFSYS (Default System)*:

```
Dim defsysdata As DEFSYS
```

Imagine that the object defsysdata to which we want to refer to in the current bar sends to the market the following orders:

- An order to change the position in to short as the strategy is long
- A profit exit order
- A stop loss order

We can assign to a variable previously defined the value returned by this function:

```
Dim price as Double= Me.defsysdata.GetOrderPrice(0, 0)
```

The function will return the price of the first from the three sent orders in the current bar.

GetOrderSide

Description:

This function returns the position (long or short) of the nth active order of a strategy given in a certain bar. Therefore, it is necessary to declare first an object of *strategy* type.

The value returned by the function will be of *OrderSide* type. The values it can take are the following:

- OrderSide.Buy
- OrderSide.Sell

Syntax:

```
Me.Strategy.GetOrderside(BarsAgo, NumberOrder)
```

Parameters:

Name	Default	Description
BarsAgo	0	Bar from which we are willing to extract the symbol of the associated value. The default value refers to the current bar.
Number Order	0	Number of reference order ascribed to the order from which we are willing to obtain the information. The default value refers to the first active order in a certain bar.

Example (VB.NET):

Assuming we declare an object of type *ADXBANDSYS02*:

```
Dim adxbdsysdata As ADXBANDSYS02
```

Now we want to send a long order when the strategy sends a new buy order, but 10 points from the buy price:

```
If (Me.adxbdsysdata.GetOrderSide(0, 0) = OrderSide.Buy) Then
   Me.Buy(TradeType.AtStop, 1, Me.adxbdsysdata.GetOrderPrice(0, 0) + 10)
End If
```

GetOrderSymbolCode

Description:
Returns the code (in Visual Chart format, i.e 010072MFXI) of the value associated to the nth order of a strategy given in a certain bar.

Therefore, it is necessary to declare first an object of *strategy* type.

Syntax:
Me.Strategy.GetOrderSymbolCode(BarsAgo, NumberOrder)

Parameters:

Name	Default	Description
BarsAgo	0	Bar from which we are willing to extract the symbol of the associated value. The default value refers to the current bar.
Number Order	0	Number of reference order ascribed to the order from which we are willing to obtain the information. The default value refers to the first active order in a certain bar.

Example (VB.NET):

Assuming we declare an object of type *DEFSYS (Default System)*:

```
Dim defsysdata As DEFSYS
```

By its creation we assign as reference source a secondary data series:

```
Me.defsysdata = New DEFSYS(Me.Data2)
```

Assuming the secondary data series is from Dax Future Continuous, if we create the following variable:

```
Dim symbol as String =Me.defsysdata.GetOrderSymbolCode(0,0)
```

The function will turn “010015DX”, which is the Visual Chart code for the Dax Future Continuous.

GetOrderType

Description:
This function returns the type of order (stop, limit, at close) of the nth active order of a strategy given in a certain bar.

Syntax:
Me.Strategy.GetOrderType(BarsAgo, NumberOrder)

Parameters:

Name	Default	Description
BarsAgo	0	Bar from which we are willing to extract the type of order. The default value refers to the current bar.
Number Order	0	Number of reference order ascribed to the order from which we are willing to obtain the information. The default value refers to the first active order in a certain bar.

Example (VB.NET):

Assuming we declare an object of type *DEFSYS (Default System)*:

```
Dim defsysdata As DEFSYS
```

We are willing to know the type of a certain order of this strategy sent in the previous bar. For this purpose, we create a variable of type Type.TradeType and we assign to it the value returned by the function *GetOrderType*:

```
Dim type As TradeType = Me.defsysdata.GetOrderType(1, 0)
```

The variable will store a value of type *Type.AtMarket* .

GetOrderVolume

Description:
This fucntion returns the amont of contracts /shares of the nth active order of a strategy given in a certain bar.

Syntax:
Me.Strategy.GetOrderVolume(BarsAgo, NumberOrder)

Parameters:

Name	Default	Description
BarsAgo	0	Bar from which we are willing to extract the type of order. The default value refers to the current bar.
Number Order	0	Number of reference order ascribed to the order from which we are willing to obtain the information. The default value refers to the first active order in a certain bar.

Example (VB.NET):

Assuming we declare an object of type *DEFSYS (Default System)*:

```
Dim defsysdata As DEFSYS
```

We are willing to know the contract volume of the second order in the current bar. For this purpose we define a variable and we assign to it the value returned by the function.

Dim volorder As Long = Me.defsysdata.GetOrderVolume(0,0)
Volorder will store the contract number associated to the current order.

GetPivotDow

Description:
This function is used to obtain the value of a pivot. A pivot is a peak in the quote, in this case we could consider it as a support.

Syntax:
Me.Identifier.GetPivotDown**(Occur, TPrice, LeftCount, RightCount, Length)**

Parameters:

Name	Default	Description
Occur	1	Numerical value representing the number of the pivot backwards we are willing to obtain. If Occur is worth 1, we will be obtaining the first pivot from the current bar, if it is worth 2 , the second one and so forth and so on.
TPrice	PriceClose	Field of the series from which we are willing to obtain the pivot. **PriceHigh:** Equivalent to the High **PriceLow:** Equivalent to the Low **PriceOpen:** Equivalent to the Open **PriceClose:** Equivalent to the Close **PriceVolume:** Equivalent to the Volume If we calculate this function on an indicator we will pass as parameter in **TPrice** the value **PriceClose**, that refers to the close of the data series of the identifier. If we had indicated **PriceHigh** or any other value if will not make any change as it will always return the same value.
LeftCount	-	Number of bars in the left side of the pivot.
RightCount	-	Number of bars in the right side of the pivot.
Length	50	Number of bars backwards to be considered while searching for the pivot.

Example (VB.NET):
Me.Data.GetPivotDown(1, Price.Low, 2,4, 50)

The function will search in the latest 50 bars before the current one, the value of the closer pivot (calculated on the lows), finding in the 2 bars on the left of the pivot and the 4 bars on the right, the value of the low superior to this one.

GetPivotUp

Description:
This function is used to obtain the value of a pivot. A pivot is a peak in the quote, in this case we could consider it as a resistance.
Syntax:
Me.Identifier.GetPivotUp**(Occur, TPrice, LeftCount, RightCount, Length)**

Parameters:

Name	Default	Description
Occur	1	Numerical value representing the number of the pivot backwards we are willing to obtain. If Occur is worth 1, we will be obtaining the first pivot from the current bar, if it is worth 2, the second one and so forth and so on.
TPrice	PriceClose	Field of the series from which we are willing to obtain the pivot. **PriceHigh:** Equivalent to the High **PriceLow:** Equivalent to the Low **PriceOpen:** Equivalent to the Open **PriceClose:** Equivalent to the Close **PriceVolume:** Equivalent to the Volume If we calculate this function on an indicator we will pass as parameter in **TPrice** the value **PriceClose**, that refers to the close of the data series of the identifier. If we had indicated **PriceHigh** or any other value if will not make any change as it will always return the same value.
LeftCount	-	Number of bars in the left side of the pivot.
RightCount	-	Number of bars in the right side of the pivot.
Length	50	Number of bars backwards to be considered while searching for the pivot.

Example (VB.NET):

```
Me.Data.GetPivotUp( 1 , Price.High, 2,4, 50)
```

The function will search in the latest 50 bars before the current one, the value of the closer pivot (calculated on the highs), finding in the 2 bars on the left of the pivot and the 4 bars on the right, the value of the high lower to this one.

GetPositionProfit

Description:

This function is used to know the value (monetary) of the profit obtained in a position. This function considers the amount of contracts/stocks bought or sold.

The value returned will be the difference between the close of the last bar and the entry point (taking into account the operation type), multiplied by the number of contracts sold. The total penalty applied to the strategy will be deducted from this result.

If this value is positive, the function will show a profit and if it is negative, it will show a lost.

If the parameter *EntryAgo* is 0, the function will be applied to the current trade. So that:

- If there is no opened position at this moment, then the returned value will be 0.
- If there is an opened position, it will return the profit obtained until the present time, that is to say, taking the last bar as exit point.

If the parameer *EntryAgo* is 1, the function will return the profit of the last ended trade.

Syntax:
GetPositionProfit**(EntryAgo)**

Parameters:

Name	Default	Description
EntryAgo	0	Number of positions backwards. The default value indicates the current position.

Example (VB.NET):
We want to take as profit for the current trade the lost of the last trade, in case of a lost.

```
        If (Me.GetMarketPosition() = 1) Then
          Dim ptsprofit As Double = (Me.GetPositionProfit(1) /
   Me.Data.GetSymbolInfo(SymbolInfo.PointValue))
          If (ptsprofit < 0) Then
            Me.ExitLong(TradeType.AtLimit, 1, Me.GetEntryPrice() + Math.Abs(ptsprofit))
          End If
        End If
```

The profit is divided by the value per point in order to obtain the value in points instead of the monetary value.

GetPrice

Description:
This function is used to know the price of a field of a bar belonging to a series.

Syntax:
Me.Identifier.GetPrice(TPrice, BarsAgo)

Parameters:

Name	Default	Description
TPrice	PriceClose	Field of the bar from which we want to obtain the value. **PriceHigh:** Equivalent to the High **PriceLow:** Equivalent to the Low **PriceOpen:** Equivalent to the Open **PriceClose:** Equivalent to the Close **PriceVolume:** Equivalent to the Volume
BarsAgo	0	Number of bars backwards.

Example (VB.NET):
Me.Data.GetPrice(PriceHigh, 22) Returns the field High of 22 bars backwards belonging to Data1.

GetStkLength

Description:
This function returns the total amount of values given for a certain type of statistical data.

Syntax:
GetStkLength (Statistic)

Parameters:

Name	Default	Description
Statistic	-	Enables to select the statistical data from which we are willing to extract the value. This parameter belongs to type *Types.StatisticValue.*

Example (VB.NET):
Inside a strategy, we want to know, at a certain stage, the length of the drawdown, in fact, how many trades have been calculated. We will proceed as follows:

```
Dim length_dd As Double = Me.GetStkLength(StatisticValue.Drawdown)
```

Length_dd will return the amount of trades taken by the strategy to call the function and that have been used to calculate the drawdown.

GetStkValue

Description:
This function returns the nth value of a certain statistical figure. The returned value can be defined in points, in monetary value or in percentage (default definition).

Syntax:
GetStkValue(Statistic, Index)

Parameters:

Name	Default	Description
Statistic	-	Enables to select the statistical data from which we are willing to extract the value. This parameter belongs to type *Types.StatisticValue.*
Index	-	Nth position from which we are willing to obtain the statistical figure. Each position refers to a trade of the strategy.

About *index* parameter:
If we consider the position 0 as the value of the statistical figure at the present time, each increase of the position will mean a greater distance. In this way, the value of the statistical figure at position 1 refers to the statistical figure at the time of the close of the penultimate trade, while position 2 refers to the statistical figure at the time of the close of the third-to-last trade, and so on.

Example (VB.NET):
The percentage net profit of a strategy at a certain stage is 0.823%, so that if we declare the following variable:

Dim current_net As Double = Me.GetStkValue(StatisticValue.NetProfit, 0)

The current net value will be 0.823.

The last closed trade of said strategy obtained a result of -225€, that in percentage terms means a lost of -0.088%. In this way, if we declare the following variable:

Dim previous_net As Double = Me.GetStkValue(StatisticValue.NetProfit, 1)

The previous net value will be 0.911, that is to say, the percentage net profit before closing the last trade.

GetStkValues

Description:
This function returns the total group of values for a certain statistical data.

Syntax:
GetStkValues(Statistic, aValues)

Parameters:

Name	Default	Description
Statistic	-	Enables to select the statistical data from which we are willing to extract the value. This parameter belongs to type *Types.StatisticValue.*
aValues	-	Buffer where the values returned for the selected statistical data are stocked.

Example (VB.NET):

```
Dim amount As Long = 0
Dim values() As Double = Nothing

amount = Me.GetStkValues(StatisticValue.Drawdown, values)
```

In this case, it returns the array **values** for all the figures and in **amount** the number of values written in the array.

GetSymbolIdentifier-GSI

Description:
This function is used only when we need to obtain and use data from a symbol which is not on the screen while inserting the corresponding strategy as, is this was the case, it would be easier to use the corresponding **Datas**.

This function is also useful to refer to the values of a symbol determined within a Macro, as in this case the historical data are not available on the screen.

In order to create a data source and to obtain its identifier we must previously have declared a DataIdentifier type variable.

Once the variable has been defined, we will assign to it the value of the function **GetSymbolIdentifier** in order to obtain the identifier of the symbol. The identifier of the symbol must be obtained from the procedure OnInitCalculate.

Later on, the obtained identifier can be used with any VBA function requiring a Data (data series from which the different functions are calculated).

Syntax:
GetSymbolIdentifier**(Symbol, Compresion, Cr, FromDate, ToFinalDate)**
We can also use its short name **GSI:**

GSI**(Symbol, Compresion, Cr, FromDate, ToFinalDate)**

Parameters:

Name	**Default**	**Description**
Symbol	-	Code of the required symbol.
Compresion	-	Compression unit (2, 5 ,10...).
Cr	-	Compression type; there are four different types available: **CrMinutes:** To obtain a minutes chart. **CrDays:** To obtain a daily chart. **CrWeeks:** To obtain a weekly chart. **CrMonth:** To obtain a monthly chart.
FromDate	-	Start date of the data source with the required identifier.
ToFinalDate	-	This is the end date of the historical data we are going to load. This data must always be superior to the current date so we recommend using always 01/01/2037 to make sure that the data of the required source are always updated.

Example (VB.NET):
Dim dailydate As DataIdentifier = Me.GetSymbolIdentifier(Me.Data.GetSymbolInfo(SymbolInfo.Code), 1, CompresionRange.Dias, "01/01/1997", "01/01/2037")
We declare an object of *DataIdentifier* type and we assign to it the daily data series of the same titel as the main source.

GetSymbolInfo

Description:
This function is used to obtain a series of characteristics from a certain symbol and not only for a certain bar. These values remain constant all over the chart and are determined by the type of value **Types.SymbolInfo**, that can take the following values:

SbiBarCompresion	Compression used for bars.
SbiCode	Code of the asset.
SbiCompresion	Compression being used (ticks, minutes, days,…).

SbiFirstSessionEnd	Closing time for the session of the asset (format HHMM).
SbiFirstSession Start	Start time for the session of the asset (format HHMM).
SbiMarket	Market the product belongs to.
SbiMinMov	Minimum movement (tick) of the product.
SbiName	Name of the product.
SbiNumDec	Number of decimals considered in the scale of the product.
SbiPath	All the symbols registered in our computer are located in the folder VisualChart\RealServer\Data. With this figure we can extract the path from the folder DATA of a certain asset.
SbiPointValue	Value per point of the product.
SbiTimeD if	Time difference of the product on which it is applied (in seconds).
SbiVendor	Specifies the vendor of the asset.

Synta x:

Me.Identifier.GetSymbolInfo(SymbolInfo)

Parameters:

Name	Default	Description
Info	SbiName	Type of SymbolInfo that we are willing to check.

Example (VB.NET):

We declare a global variable of logical type, that will tell us if we are in an *intraday* compression.

```
Dim isintraday As Boolean
```

From the method *OnInitCalculate* we give value to the variable regarding the type of information SymbolInfo.Compresion:

```
If Me.Data.GetSymbolInfo(SymbolInfo.Compresion) = CompresionRange.Minutos Or
Me.Data.GetSymbolInfo(SymbolInfo.Compresion) = CompresionRange.Ticks Then
   isintraday = True
Else
   isintraday = False
End If
```

That is to say, if we have a minutes or ticks compression, we are in an intraday chart, otherwise, we are not.

GetSymbolInfoEx

Description:

This property returns an object of type **IElementSymbolInfo** regarding the data series assigned to the time the type is being created. This property is used in order to avoid having to do more than one call to obtain all **SymbolInfo** figures.

Once the object has been created, we can use its functions in order to access to the data *SymbolInfo*. The functions that can be used of said type are the following:

GetBarCompresion	Returns the compression used for bars.
GetCode	Returns the titel code.
GetCompresion	Returns the compression type that is being used (ticks, minutes, days,…)
GetFirstSessionEnd	Returns the closing time for the session of the asset (format HHMM).
GetFirstSession Start	Returns the start time for the session of the asset (format HHMM).
GetMarket	Returns the market the product belongs to.
GetMinMov	Returns the minimum movement (pips) which the product can give.
GetName	Returns the name of the product.
GetNumDec	Returns the decimal number considered in the scale used by the product.
GetPath	Returns all symbols registered in our computer, that are located under RealServer\Data from VisualChart. With this figure we can extract the path from the folder DATA of a certain asset.
GetPointValue	Returns the point value of the product.
GetTimeD if	Returns the time difference of the product on which it is applied (in seconds).
GetVendor	Returns the vendor of the asset.

Syntax:

```
Dim x As IElementSymbolInfo = Me.GetSymbolInfoEx(DataIdentifier, [Day])
```

Parameters:

Name	Default	Description
Symbol	Data	Data source from wich we are willing to extract the information. If not specified we wil considered the chart where the strategy is inserted as data.
Day	-	Optional parameter. Day of the week from which we want to obtain the information. Generally it will not be declared.

Example (VB.NET):

We create an object of type IElementSymbolInfo:

```
    Dim ielements As IElementSymbolInfo = Me.GetSymbolInfoEx(Me.Data2.DataSeriesId)
```

The reference data source is the second source associated to the alias Data2. As we require a *DataIdentifier* in the parameter *Symbol*, we specify that the reference is the property *DataSeriesId* of Data2.

Next, we indicate that in case that the source with alias Data2 is the Dax future continuous, then it must send a warning message:

```
If ielements.GetCode = "010015DX" Then
   MsgBox("Is using the dax future cotinuous as Data2.")
End If
```

GetSystemIdentifier-GSYSI

Description:

This function enables to obtain internally the information of a certain strategy. This way, we can extract the information from this strategy without having to calculate it once and once again.

Syntax:

GetSystemIdentifier(Name, ParentDataIdentifier, ParamArray)

We can also use the short method **GSYSI.**

GSYSI(Name, ParentDataIdentifier, ParamArray)

Parameters:

Name	Default	Description
Name	-	Code of the strategy from which we are willing to extract the information.
ParentDataIdentifier	Data	Data source from which we would be loading the strategy.
ParamArray	-	Collection of entry parameters demanded by the strategy (optional).

Example (VB.NET):

We can assign to a DataIdentifier type variable the information of a certain strategy and use it later on with other functions, for example, .GetOrderCount, .GetOrderLabel, etc.

Assuming we have a strategy called *Mysystem*.

```
Dim mysystemdata As DataIdentifier = 0
    mysystemdata =GetSystemIdentifier(MySystem, Data, ContractsNumber, Target, Losses, StartHour, EndHour)
```

From *OnCalculateBar* we extract the number of the strategy orders at a certain stage:

```
Dim ordernum As Long =Me.GetOrderCount(Me.mysystemdata)
```

Visual Chart 6 novelties:

Visual Chart 6 wants to get the most of the project design using object-oriented programming languages.

As a result, we will see that we can create objects of types associated to the public strategies as well as to the private strategies (as long as we have access to them).

So, we can define an object of *ADXBANDSYS*, *STOCHCROSS* or *PVBREAK* type. Every strategy adds a new class of the same type. In this way, it is lighter to add strategies to a project because it is not necessary to use the method *GetSystemIdentifier*.

Use of profit line.

Objects of *strategy* type are also considered data sources. In these cases, the series is made up of the profit curve in points of the concrete strategy. This ability enables to add to the project any indicator calculated on the profit curve of a strategy.

Example (VB.NET):

We want to apply the 20-period RSI to the profit curve of the Stochastic Cross strategy and acts regarding said RSI. For this purpose, we declare an object of *STOCHCROSS* type and another of *RSI* type.

```
Dim stoccrdata As STOCHCROSS
Dim rsidata As RSI
```

From *OnInitCalculate*, we create both objects:

```
Me.stoccrdata = New STOCHCROSS(Me.Data, 14, 6, 3, 74, 23, 0.2, 0.6, 0, 0, 1)
Me.rsidata = New RSI(Me.stoccrdata, 20)
```

Finally from *OnCalculateBar* we act regarding the RSI:

```
If (Me.rsidata.value() > 30 And Me.rsidata.value(1) <= 30) then
      Me.Buy(TradeType.AtMarket)
   End If
```

GetSwingHigh

Description:
This function is used to obtain the value of a pivot. A pivot is a peak in the quote, in this case we could consider it as a resistance.

The difference regarding the *GetPivotUp* is that, in this case, the number of bars to the rigth and to the left is the same and they are defined by the *Strength* parameter.

The function returns the value null if it does not find a bullish pivot within the specific requirements.

Syntax:
Me.Identifier.GetSwinHigh(Occur, Tprice, Strength, Length)

Parameters:

Name	**Default**	**Description**
Occur	1	Numerical value representing the number of the pivot backwards we are willing to obtain. If Occur is worth 1, we will obtain the first pivot from the current bar, if it is worth 2, the second one and so forth and so on.
TPrice	PriceClose	Data series from which we are willing to obtain the pivot. **PriceHigh:** Equivalent to the High **PriceLow:** Equivalent to the Low **PriceOpen:** Equivalent to the Open **PriceClose:** Equivalent to the Close **PriceVolume:** Equivalent to the Volume If this function is calculated on an indicator we will pass as parameter in **TPrice** the value **PriceClose**, that refers to the close of the indicator data series. If we had indicated **PriceHigh** or any other price source, it will still return the same value.
Strength	2	Number of bars to consider in both sides of the pivot.
Length	50	Number of bars backwards to be considered while searching for the pivot.

This function is very useful as it helps us to find support or resistances. We can say that there is an up pivot when a value in a certain data series is superior to a number of previous and subsequent values specified in the parameter Strength.

Example (VB.NET):
Me.Data.GetSwingHigh(1, Price.High, 2, 50)

In this case the function will search in the 50 bars preceeding the current one, the value of the closest pivot (calculated within the highs), being on the 2 bars on each side of the pivot, the value of the high lower than this pivot.

GetSwingHighBar

Description:
This function returns the distance to a bullish pivot. A pivot is a peak in the quote, in this case we could consider it as a resistance.

The function returns the value null, if it does not find a bullish pivot within the specific requirements.

Syntax:
Me.Identifier.GetSwinHighBar(Occur, Tprice, Strength, Length)

Parameters:

Name	Default	Description
Occur	1	Numerical value representing the number of the pivot backwards that we are willing to find. If Occur is worth 1, we will be obtaining the first pivot from the current bar. If it is worth two, the second one and so forth and so.
TPrice	PriceClose	Data series from which we are willing to obtain the pivot. **PriceHigh:** Equivalent to the High **PriceLow:** Equivalent to the Low **PriceOpen:** Equivalent to the Open **PriceClose:** Equivalent to the Close **PriceVolume:** Equivalent to the Volume If this function is calculated on an indicator we will pass as parameter in **TPrice** the value **PriceClose**, that refers to the close of the indicator data series. If we had indicated **PriceHigh** or any other price source, it will still return the same value.
Strength	5	Number of bars to consider in both sides of the pivot.
Length	50	Number of bars backwards to be considered while searching for the pivot. If this number of bars is overcome, the function will return the value null.

Example (VB.NET):
Me.Data.GetSwingHighBar(1, Price.High, 2, 50)

In this case, the function will search over the 50 bars preceeding the current one the distance to the closest pivot (calculated on the highs), being on the 2 bars on each side of the pivot, the value of the high inferior to it. If this pivot is 5 bars away from the current bar, where we are using the function, it will return 5. In the following bar it will return 6 and so forth and so on until a new pivot is found or the distance overcomes the high 50 bars.

GetSwingLow

Description:
This function is used to obtain the value of a bearish pivot. A pivot is a peak in the quote, in this case we will be able to consider it as a support as we are talking about an inverted peak.
The difference regarding the *GetPivotDown* is that in this case the bars number to the left and right of the pivot is the same and they are defined by the parameter *Strength.*
If no bearish pivot within the specific requirements is found the function will return 0.

Syntax:
Me.Identifier.GetSwingLow(Occur, Tprice, Strength, Length)

Parameters:

Name	Default	Description
Occur	1	Numerical value representing the number of the pivot backwards that we are willing to find. If Occur is worth 1, we will be obtaining the first pivot from the current bar, if it is worth two, the second one and so forth and so on.
TPrice	PriceClose	Field of the bar from which we are willing to obtain the pivot **PriceHigh:** Equivalent to the High **PriceLow:** Equivalent to the Low **PriceOpen:** Equivalent to the Open **PriceClose:** Equivalent to the Close **PriceVolume:** Equivalent to the Volume If this function is calculated on an indicator we will pass as parameter in **TPrice** the value **PriceClose**, that refers to the close of the indicator data series. If we had indicated **PriceHigh** or any other price source it will still return the same value.
Strength	2	Number of bars to consider in both sides of the pivot.
Length	50	Number of bars backwards to consider while searching for the pivot.

This function is very useful as it helps us to find support or resistances. We can say that there is a down pivot when a value in a certain data series is inferior or equal to a number of previous and subsequent values specified in the parameter Strength.

Example (VB.NET):
Me.Data.GetSwingLow(1, Price.Low, 2, 50)
In this case, the function will search throughout the 50 bars preceeding the current one, the value of the closest pivot (calculated on the lows), being on the 2 bars on each side of the pivot, the value of the low superior to this low.

GetSwingLowBar

Description:
This function returns the length, in number of bars, since the last bearish pivot occurred. A bearish pivot is a low in the quote, in this case we could consider it as a support.

If no bearish pivot within the specific requirements is found, the function will return 0.

Syntax:
Me.Identifier.GetSwingLowBar(Occur, Tprice, Strength, Length)

Parameters:

Name	Default	Description
Occur	1	Numerical value representing the number of the pivot backwards that we are willing to find. If Occur is worth 1, we will be obtaining the first pivot from the current bar, if it is worth two, the second one and so forth and so on.
TPrice	PriceClose	Field of the bar from which we are willing to obtain the pivot: **PriceHigh:** Equivalent to the High **PriceLow:** Equivalent to the Low **PriceOpen:** Equivalent to the Open **PriceClose:** Equivalent to the Close **PriceVolume:** Equivalent to the Volume If this function is calculated on an indicator we will pass as parameter in **TPrice** the value **PriceClose** that refers to the close of the indicator data series. If we had indicated **PriceHigh** or any other price source, it will still return the same value.
Strength	2	Number of bars to consider in both sides of the pivot.
Length	50	Number of bars backwards to be considered while searching for the pivot. If this number of bars is overcome, the function will return the value null.

Example (VB.NET):
Me.Data.GetSwingLow(1, Price.Low, 2, 50)

In this case, the function will search over the 50 bars preceeding the current one the value of the closest pivot (calculated on the lows). The lowest value superior to the pivot bar one must occur in the two bars before and after the pivot one. If the pivot is found 5 bars before the current one on which the function is used, it will return a 5, is 6 bars then a 6 and so forth and so on until a new pivot is found.

GetTrueHigh

Description:
Returns the highest price between the high of a bar and the close of the previous bar.

Syntax:
Me.Identifier.GetTrueHigh (BarsAgo)

Parameters:

Name	Default	Description
BarsAgo	10	Number of bars backwards that will serve as reference to apply the calculation.

Example (VB.NET):
We want to take as reference for a long entry with a stop order the highest value between the high of the penultimate bar and the close ot the third last bar:

Me.Buy(TradeType.AtStop, 1, Me.Data.GetTrueHigh (1))

GetTrueLow

Description:
Returns the lowest price between the low of a bar and the close of the previous bar.

Syntax:
Me.Identifier.GetTrueLow (BarsAgo)

Parameters:

Name	Default	Description
BarsAgo	10	Number of bars backwards that will serve as reference to apply the calculation.

Example (VB.NET):
Me.Data.GetTrueLow (5)

Compares in the historical figures of the series Data1, the low of a bar with the close of the previous one, indicating the lower of both of them (taking as reference to start the calculation 5 bars backwards).

GetTrueRange

Description:
This function returns the difference between GetTrueHigh and GetTrueLow for a data series.

Syntax:
Me.Identifier.GetTrueRange (BarsAgo)

Parameters:

Name	Default	Description
BarsAgo	10	Bar number on which the function is applied.

Example (VB.NET):
Returns the difference between the value returned by the function GetTrueHigh and GetTrueLow and the second bar backwards of the Data series.

GetTrueRangeCustom

Description:
This function compares the current highest and lowest values with a price interval established by parameters. If the price moves within this interval, the returned value will be the range or difference between the highest value and the lowest value. If the price exceeds the highest value, the returned value will be the range or difference between the high given by the asset and the lowest interval value. In the same way, if the lowest value is exceeded, the returned value will be the range or difference between the interval high and the lowest value given by the asset. As a result, we will quickly know if the underlying is outside the interval and in which proportion if the value returned by the function ovecomes the default interval.

Syntax:
Me.DataIdentifier.GetTrueRangeCustom (BarsAgo, High, Low)

Parameters:

Name	Default	Description
BarsAgo	10	Number of the reference bar
High	-	Highest value
Low	-	Lowest value

Example (VB.NET):

Assuming we create an indicator as follows:

Dim range As Double = Me.Data.GetTrueRangeCustom(0, 9000, 5000)

Me.SetIndicatorValue(difference)

We insert the indicator on the Dax Future Continuous.

The value returned by the indicator from 2010 until 2014 will be 4000 points, while the returned value in 2014 exceeds 4000 points and it is widely exceeded since 2015 (until the range 7000 is reached, wich is the difference between the DAX highs in 2015 and the lowest value 5000).

GetVolatility

Description:
This function returns the volatility between the current bar and the Nth bar backwards in terms of difference in points.

Syntax:
Me.Identifier.GetVolatility (Tprice, Length)

Parameters:

Name	Default	Description
TPrice	PriceClose	Field of the bar to which we want to refer. To do so, we must indicate in this field the enumerator of the Types.Price enumeration: **PriceHigh:** Equivalent to the High **PriceLow:** Equivalent to the Low **PriceOpen:** Equivalent to the Open

		PriceClose: Equivalent to the Close **PriceVolume:** Equivalent to the Volume
Length	1	Distance in relation to the current bar to calculate the volatility.

Example (VB.NET):
Me.GetVolatility(Price.Low, 5)

In this case, the function will return the difference between the low of the current bar and the low 5 bars backwards (from the data series Data 1).

GetWndBackGroundColor

Description:
This function returns the background color of an indicator window.

Syntax:
GetWndBackGrounColor ()

Parameters:

Name	**Default**	**Description**
-	-	-

GrossLoss

Description:
Returns the value of the gross losses for the negative trades accumulated by our strategy until the current bar (on which the calculations are being run). To obtain it, we consider that the last opened trade concludes in the current bar.

Syntax:
Me.GrossLoss(SttRepresentation.Porcentual)

Parameters:

Name	**Default**	**Description**
Show	Bypoints	Enables to indicate the format on which the information will show up: **SttRepresentation.ByPoints** (points) or **SttRepresentation.Porcentual** (percentage).

Example (VB.NET):
Me.GrossLoss(SttRepresentation.Porcentual) Will return the loss percentage (gross) obtained by the strategy.

GrossProfit

Description:
This function returns the value of the gross profits obtained by the strategy in the positive trades until the current bar (bar on which the calculations are being run). To obtain it, we will consider that the last opened trade concludes in the current bar.

Syntax:
GrossProfit(Show As SttRepresentation)

Parameters:

Name	Default	Description
Show	Bypoints	Enables to indicate the format on which the information will show up: **SttRepresentation.ByPoints** (points) or **SttRepresentation.Porcentual** (percentage).

Example (VB.NET):
Me.GrossProfit(SttRepresentation.Puntos)
Will return in points the gross profit obtained by the strategy.

High

Description:
This function returns the value of the bar high of a specific data series.

Syntax:
Me.Identifier.High(BarsAgo)

Parameters:

Name	Default	Description
BarsAgo	0	Number of bars backwardss. The default value refers to the current bar. In this parameter we can use any numerical value contained in a variable or enter the value directly. If can also be specified as function replacing the numerical value.

Example (VB.NET):
Me.Data.High(4)   Will return the highest value of the las 4 bars (from the data source Data1).

IsFirstDayBar

Description:
This function returns true or false based on whether the current bar is the first bar of each session or not (respectively).

Syntax:
Me.IsFirstDayBar()

Parameters:

Name	Default	Description
-	-	

Example (VB.NET):
We want an entry based on the opening price of each session +/- a filter. To do this, we create a variable in which we store the session opening price:

```
Private entryprice As Double
```

From the method *OnCalculateBar()* we indicate to update the variable each time the property *IsFirstDayBar* is true:

```
If (Me.IsFirstDayBar) Then
  Me.entryprice = Me.Data.Open()
End If
```

IsLastDayBar

Description:

This function returns true or false based on whether the current bar is the last bar of each session or not (respectively).

Syntax:
Me.IsLastDayBar()

Parameters:

Name	Default	Description
-	-	

Example (VB.NET):
We want that the strategy checks the entry rules at all bars except the last bar of each session, since we want to avoid a pending order overnight:

```
If (Me.IsLastDayBar = False) Then
  If (Me.rsidata.Value() < 70 And Me.rsidata.Value(1) >= 70) Then
    Me.Buy(TradeType.AtMarket)
  End If
End If
```

LargestLosingTrade

Description:
This function returns the result of the worst operation. This value will change while new bars are generated by and its value will depend on the bar on which the property is called.

Syntax:
LargestLosingTrade(Show As SttRepresentation)

Parameters:

Name	Default	Description
Show	Bypoints	Enables to indicate the format on which the information will show up: **SttRepresentation.ByPoints** (points) or **SttRepresentation.Porcentual** (percentage).

Example (VB.NET):
If we want to know, at a certain stage, the result of the worst operation (in points) done by our strategy, we could define a variable:

Dim largestlosingtrade As Double = 0

An assign to it the value returned by the property:

largestlosingtrade =Me.LargestLosingTrade(SttRepresentation.ByPoints)

LargestWinningTrade

Description:
This function returns the result of the best trade made. This value will change while new bars are generated by and it value will depend on the bar on which the property is being called.

Syntax:
LargestWinningTrade(Show As SttRepresentation)

Parameters:

Name	Default	Description
Show	Bypoints	Enables to indicate the format on which the information will show up: **SttRepresentation.ByPoints** (points) or **SttRepresentation.Porcentual** (percentage).

Example (VB.NET):

We are willing to know, at a certain stage, the result of the best trade in percentage made by our strategy, so we could define a variable:

Dim largestwinningtrade As Double = 0

And assign to it the value returned by the property:
largestwinningtrade = Me.LargestWinningsTrade(SttRepresentation.Porcentual)

Lc_Index

Description:
This function returns the ratio **Profits in long positions /Profits in short positions**. This value will change while new bars are generated by, and its value will depend on the bar on which this property is called.

Syntax:
Lc_Index(Show As SttRepresentation)

Parameters:

Name	Default	Description
Show	Bypoints	Enables to indicate the format on which the information will show up: **SttRepresentation.ByPoints** (points) or **SttRepresentation.Porcentual** (percentage).

Example (VB.NET):
We are willing to know, at a certain stage, the ratio **Profits in long positions /Profits in short positions** (percentage) of our strategy. We could define a variable:

```
Dim lc_index As Double = 0
```

And assign to it the value returned by the property:

```
lc_index = Me.Lc_Index(SttRepresentation.Porcentual)
```

LimitOrder

Description:
This property enables to obtain the existing amount of orders in the bid and in the ask for certain price levels in a certain bar.

Caution. It only returns results in real time as there are not historical data available for the bid and ask positions.

Syntax:
LimitOrder(Level, Side, BarsAgo, Identifier)

Parameters:

Nam e	Default	Description
Level	1	Indicates the nth bid and ask position whose amount we are willing to know.
Side	osBuy	Order position. Figure of Type.OrderSide type, that can be: **OrderSide.Buy** (buy)

		OrderSide.Sell (sell)
Bars Ago	0	Position of the bar we want to consult, although it must be a bar generated during the real time.
Identifier	Data	Data series from which we want to obtain the information.

Example (VB.NET):
Me.LimitOrder(4, OrderSide.Sell, 0, Data1)
This way, we obtain the amount of titles/contracts offered in the forth sell position of the data source Data1.

LimitPrice

Description:
This property enables to obtain the price of the active orders in a bid and in the ask for a certain position and in a certain bar.

Caution. It only returns results in real time as they are not historical data available for the bid and ask positions.

Syntax:
LimitePrice(Level, Side, BarsAgo, Identifier)

Parameters:

Name	Default	Description
Level	1	Indicates the nth bid and ask position whose price we are willing to know.
Side	osBuy	Order position. Figure of Type.OrderSide type, that can be: **OrderSide.Buy** (buy) **OrderSide.Sell** (sell)
BarsAgo	0	Position of the bar we want to consult, although it must be a bar generated during the real time.
Identifier	Data	Data from which we are extracting the information.

Example (VB.NET):
Me.LimitOrder(4, osSell, 0, Data1)   Returns the price offered in the forth sell positions of Data1.

LimitVol

Description:
The property LimitVol enables to obtain the total volume of the active orders in the bid and the ask, for a certain position and for a certain bar.

This property only returns results in real time as there is not historical data kept for the bid and ask positions.

Syntax:
LimitVol(Level, Side, BarsAgo, Identifier)

Parameters:

Name	Default	Description
Level	1	Indicates the nth bid and ask position whose volume we are willing to know.
Side	osBuy	Order position. Figure of Type.OrderSide type, that can be: **OrderSide.Buy** (buy) **OrderSide.Sell** (sell)
BarsAgo	0	Position of the bar we want to consult, although it must be a bar generated during the real time.
Identifier	Data	Data from which we are willing to extract the information.

Example (VB.NET):
Me.LimitVol(2, OrderSide.Buy, 0, Data)

We obtain this way the volume of titles/contracts offered in the second buying position of the data source Data.

Low

Description:
This function returns the lowest value of a bar for a certain data series.

Syntax:
Me.Identifier.Low(BarsAgo)

Parameters:

Name	Default	Description
BarsAgo	0	Number of bars backwards, the default value refers to the current bar. This parameter can be used with any numerical value contained in a variable and also by entering its value. It can also be specified as function replacing the numerical value.

Example (VB.NET):
Me.Data2.Low(0)   Returns the low of the current bar (from the data source Data2).

MarketFilledOrders

Description:
This function enables the access to the filled orders (user´s orders) from the own strategy. As a result, it will only make sense if the broker connection (in real or simulation) has been established.

Sintaxis:
MarketFilledOrders(Label, BarsAgo, Side)

Parameters:

Name	Default	Description

Label	-	Label associated to the order we want to consult.
BarsAgo	0	Number of bars backwards. The default value refers to the current bar.
Side	-	Position of the order we want to consult. (0 long position and 1 short position).

Example (VB.NET):
Assuming that the strategy "A" has a long opened position and set 2 orders at the same time (a target profit and a stop at market).

Once the trading of the strategy has been activated, we can know through the code which order has been filled by doing as follows:

```
Dim MKFO() As MktFilledOrder

MKFO = Me.MarketFilledOrders("A", 0, osSell)    We consult if sell orders with label "A" have been filled in
                                                   the current bar.
n = UBound(MKFO)

If Err.Number = 0 Then                        If MKFO is not empty, it means that at least an order was filled.
   If UBound(MKFO) <> -1 Then
      PriceExitOrder = MKFO(0).Price          We take the price returned by the function.
   End If

End If
```

According to the value of **PreisExitOrder** we can see if the stop or the target was filled.

MinutesToTime

Description:
This function is used to change minutes in standard time.

Syntax:
MinutesToTime(Minutes)

Parameters:

Name	**Default**	**Description**
Minutes	-	Time in numerical format

Example (VB.NET):
Me.MinutesToTime(300)   Will return 5 (300 minutes are worthing 5 hours).

NetProfit

Description:
This function is used to obtain the value of the strategy´s net profit until the current bar (bar on which the calculation is being made). To obtain the total profit we will consider that the last opened trade concludes in the close of the current bar.

Syntax:
NetProfit(Show As SttRepresentation)

Parameters:

Name	Default	Description
Show	Bypoints	Enables to indicate the format on which the information will show up: **SttRepresentation.ByPoints** (points) or **SttRepresentation.Porcentual** (percentage).

Example (VB.NET):
Me.NetProfit(SttRepresentation.ByPoints) Returns in points the net profit obtained by the strategy.

NumberOfLines

Description:
This function returns the number of lines of the indicator on which it is applied. The indicator must have had previously assigned values to these lines, so that the value returned by the function includes them (as the function only returns the values of the current lines).

Syntax:
NumberOfLines

Parameters:

Name	Default	Description
-	-	-

Example (VB.NET):
Supposing that we have created an indicator callled MyIndicator that runs the following calculation:

```
Value1 = Me.Data.High + Me.Data.Low + Me.Data.Close / 3
Value2 = Me.Data.High + Me.Data.Low / 2
Value3 = Me.Data.Open + Me.Data.Close + Me.Data.Close(1) / 3
```

And then we paint:

```
Me.SetIndicatorValue(Value1, 1)
Me.SetIndicatorValue(Value2, 2)
Me.SetIndicatorValue(Value3, 3)
```

If, inside our code, we use the function:

Dim nlines As Integer =Me.NumberOfLines

Nlines will be equal to 3, that is the number of painted lines.

NumberOfLosingTrades

Description:
This function returns the number of losing trades made by our strategy until the current bar. This value will change while new bars are generated and its value will depend on the bar on which the property is called.

Syntax:
NumberOfLosingTrades

Parameters:

Name	Default	Description
-	-	-

Example (VB.NET):
Imagine that we are willing to know, in a certain bar, the number of accumulated losing trades. We could define a variable:

Dim losing As Long = Me.NumberOfLosingTrades

In this case, it returns the number of losing trades accumulated until the moment the property is called.

NumberOfTrades

Description:
Returns the number of trades made until that moment. This value will change while new bars are generated and its value will depend on the bar on which the property is called.

Syntax:
NumberOfTrades

Parameters:

Name	Default	Description
-	-	-

Example (VB.NET):
Supposing that we are willing to define, in a certain bar, the number of accumulated trades. We could define a variable:

Dim numberoftrades As Long =Me.NumberOfTrades

Returns the number of trades accumulated until the moment the property is called.

NumberOfWinningTrades

Description:
This function returns the number of winning trades until that moment. This value will change while new bars are generated and its value will depend on the bar on which the property is called.

Syntax:
NumberOfWinningTrades

Parameters:

Name	Default	Description
-	-	-

Example (VB.NET):
Supposing that we are willing to know, in a certain bar, the number of accumulated winning trades. We could define a variable:

Dim winning As Long =Me.NumberOfWinningTrades

Returns the accumulated number of winning trades until the moment the property is called.

Open

Description:
This function returns the value of the bar´s open.

Syntax:
Me.Identifier.Open(BarsAgo)

Parameters:

Name	Default	Description
BarsAgo	0	Number of bars backwards. The default value refers to the current bar. In this parameter, we can enter a numerical value or indicate a numerical value contained in a variable. It can also be specified as a function replacing the numerical value. This parameter only allows positive values.

Example (VB.NET):
Me.Data.Open(3) Returns the open value three bars backwards from the data source codified as Data1.

OpenInt

Description:
This function returns the value of the **OpenInterest** in a bar for a certain data series.
Caution. This function only returns results in case of future contracts, where this information is provided.

Syntax:
Me.future.OpInt(BarsAgo)

Parameters:

Name	Default	Description
BarsAgo	0	Number of bars backwards. The default value refers to the current bar. It can also be specified as a function replacing the numerical value. This parameter only allows positive values.

Example (VB.NET):
Me.Data.OpInt(5)

Returns the OpenInterest 5 bars backwards (from the data series Data).

PaintBar

Description:
By using this option we can paint bars on the chart with the wished values for open, high, low and close.

Syntax:
PaintBar(Open, High, Low, Close, Color, [LineNumber], [Width], [nBars])

Parameters:

Name	Default	Description
Open	-	Enables to use a function or variable.
High	-	Enables to use a function or variable.
Low	-	Enables to use a function or variable.
Close	-	Enables to use a function or variable.
Color	-	Color used to paint the bars by using the data type **System.Drawing.Color.**
LineNumber	-	Specifies the order number for painting study in studies with different paint orders (i.e., if we want to include within a study a paint bar order (.PaintBar) and a paint line order (.PaintSeries), we must set in the first order line number 0 and in the second order line number 1.
Width	1	Width of the bar to be painted.
nBars Ago	-	Number of bars backwards. The value 0 refers to the current bar.

Example (VB.NET):

We want to create a chart based on averages. For this purpose we declare the following four objects and the parameter *Period* in a new study:

```
<Parameter(Name:="Period", DefaultValue:=15, MinValue:=2, MaxValue:=100, Step:=1)>
 Private period As Integer

 Dim opendata As AvSimple
 Dim closedata As AvSimple
 Dim highdata As AvSimple
 Dim lowdata As AvSimple
```

Next, we create them assigning each of them a different price field:

```
Me.opendata = New AvSimple(Me.Data, Me.period, Price.Open)
Me.closedata = New AvSimple(Me.Data, Me.period, Price.Close)
Me.highdata = New AvSimple(Me.Data, Me.period, Price.High)
Me.lowdata = New AvSimple(Me.Data, Me.period, Price.Low)
```

Finally, from the method *OnCalculateBar* we use the method PaintBar:

```
    If (Me.closedata.Value() > Me.opendata.Value()) Then
        Me.PaintBar(Me.opendata.Value(), Me.highdata.Value(), Me.lowdata.Value(),
Me.closedata.Value(),                    Color.Blue)
    Else
        Me.PaintBar(Me.opendata.Value(), Me.highdata.Value(), Me.lowdata.Value(),
        Me.closedata.Value(), Color.Pink)
    End If
```

Paints the bars in blue or rose depending on the trend.

PaintCandlestick

Description:

By using this option we can paint candles with the wished values for open, high, low and close.

Syntax:

PaintCandlestick(Open, High, Low, Close, Color, [LineNumber], [Width], [nBars])

Parameters:

Name	Default	Description
Open	-	Enables to use a function or variable.
High	-	Enables to use a function or variable.

Low	-	Enables to use a function or variable.
Close	-	Enables to use a function or variable.
Color	-	Color used to paint the candles by using the data type **System.Drawing.Color**.
LineNumber	-	Specifies the order number for painting study in studies with different paint orders (i.e., if we want to include within a study a paint bar order (.PaintBar) and a paint line order (.PaintSeries), we must set in the first order line number 0 and in the second order line number 1.
Width	1	Width of the bar to be painted.
nBars Ago	-	Number of bars backwards. The value 0 refers to the current bar.

Example (VB.NET):
We want to paint candles using the methods GetTrueHigh and GetTrueLow, so that the high of each candle will be the value returned by the function GetTrueHigh and the low by the function GetTrueLow:

```
If (Me.Data.Close() > Me.Data.Open()) Then
    Me.PaintCandlestick(Me.Data.Open, Me.Data.GetTrueHigh, Me.Data.GetTrueLow, Me.Data.Close,
    Color.WhiteSmoke)
Else
    Me.PaintCandlestick(Me.Data.Open, Me.Data.GetTrueHigh, Me.Data.GetTrueLow, Me.Data.Close,
    Color.DarkGray)
End If
```

As a result, the bullish candles will be painted in light grey and the bearish candles in dark grey.

PaintMaxMin

Description:
This function is similar to PaintBar and PaintCandlestick. The difference is that, in the current function, we can only establish values for the high and the low of the bar we are willing to paint.

Syntax:
PaintMaxMin(Top, Bottom, Color, [LineNumber], [Width], [nBars])

Parameters:

Name	**Default**	**Description**
Top	-	High of the bar we are willing to paint. Enables to use a function or variable.
Bottom	-	Low of the bar we are willing to paint. Enables to use a function or variable.
Color	-	Color used to paint the bars by using the data type **System.Drawing.Color**.
LineNumber	-	Specifies the order number for painting study in studies with different paint orders (i.e., if we want to include within a study a paint bar order (.PaintBar) and a paint line order (.PaintSeries), we must set in the first order line number 0 and

		in the second order line number 1.
Width	1	Width used to paint.
nBars Ago	-	Number of bars backwards. The value 0 refers to the current bar.

Example (VB.NET):
Me.PaintMaxMin(Me.Data.Low(), Me.Data.Low(), Color.Blue, 0, 8, 0)
Paints the low of the bar with a circle (width 8) in blue color.

PaintSeries

Description:
This function is used to paint data lines in a window. This task can also be run by creating an indicator, but if we wanted to mix lines with options of painting bars or painting figures, we could decide to create a study including these two types. We must remember that the studies can not be used in other kind of projects as strategies, indicators, etc. Consequently, if we want to use the data line later on, we should use an indicator.

Syntax:
PaintSeries(Price, Color,[LineNumber], [Width], [nBars])

Parameters:

Name	Default	Description
Price	-	In this parameter we indicate the required value for the line in the current bar. Any numerical value or a numerical type variable is accepted in this parameter.
Color	-	Color used to paint the bars by using the data type **System.Drawing.Color**.
LineNumber	-	Specifies the order number for painting study in studies with different paint orders (i.e., if we want to include within a study a paint bar order (.PaintBar) and a paint line order (.PaintSeries), we must set in the first order line number 0 and in the second order line number 1.
Width	1	Width used to paint the candlestick.
nBars Ago	-	Number of bars backwards. The value 0 refers to the current bar.

Example (VB.NET):
PaintSeries((Me.Data.High + Me.Data.Low) / 2, Color.Red, 0, 1, 0)
In this case the function paints a red colored line representing the average point of the bar.

PercentProfitable

Description:
This function returns the reliability ratio. This value will change while new bar are generated and its value will depend on the bar on which the property is called.

Syntax:
PercentProfitable

Parameters:

Na me	Default	Description
-	-	-

Example (VB.NET):

We want to know, at a certain stage, the reliability ratio of our strategy. To do so, we shall previously define a variable:

Dim reliability As Double = 0

At a certain stage, we will assign to this variable the value of the property .PercentProfitable:

Reliability = Me.PercentProfitable

ProfitFactor

Description:
This function returns the **Profit factor**. This value will change while new bars are generated and its value will depend on the bar on which this property is called.

Syntax:
ProfitFactor(Show As SttRepresentation)

Parameters:

Name	Default	Description
Show	Bypoints	Enables to indicate the format on which the information will show up: **SttRepresentation.ByPoints** (points) or **SttRepresentation.Porcentual** (percentage).

Example (VB.NET):
We want to know, at a certain moment, the profit factor (percentage) of our strategy. To do so, we shall previously define a variable:

Dim profitf As Double = 0

At a certain stage, we will assign to this variable the value of this function:

Profitf =Me.ProfitFactor(SttRepresentation.Porcentual)

PRR

Description:
This function returns the ration **Adjusted Profit Factor**. This value will change while new bars are generated and will depend on the bar on which the property is called.

Syntax:
PRR(Show As SttRepresentation)

Parameters:

Name	Default	Description
Show	Bypoints	Enables to indicate the format on which the information will show up: **SttRepresentation.ByPoints** (points) or **SttRepresentation.Porcentual** (percentage).

Example (VB.NET):
If we want to know, at a certain stage, the Adjusted profit factor (in points) of our strategy, we must first define a variable:

Dim adprofitf As Double = 0

We assign to it, at a certain stage, the value of this function:

Adprofitf = Me.PRR(SttRepresentation.ByPoints)

RegressionAngle

Description:
This function returns the angle formed by the horizontal line and the regression line formed by the closing prices located between **StartBar** and **EndBar**.

Syntax:
Me.Identifier.RegressionAngle(StarBar, EndBar, Data)

Parameters:

Name	Default	Description
StarBar	-	Start bar of the regression line.
EndBar	-	End bar of the regression line.

🟨 **RegressionSlope**

Description:
This function returns the value of the slope of the regression line formed by the closing prices situated between **StartBar** and **EndBar**.

Syntax:
Me.Identifier.RegressionSlope(StarBar, EndBar)

Parameters:

Name	Default	Description
StarBar	-	Start bar of the regression line.
EndBar	-	End bar of the regression line.

ReleaseDataIdentifier - RDI

Description:
This function enables to free a **DataIdentifier** previously called via the method GetSymbolIdentifier.

Syntax:
ReleaseDataIdentifier(Identifier)

The short mode **RDI** can also be used.
RDI(Identifier)

Parameters:

Name	Default	Description
Identifier	Data	Data to be freed.

Example (VB.NET):
First, we must create a data identifier that we are going to use to extract the minimum movement of the contract FDAXU5 (Dax Furture September contract-2015)

```
Dim auxdata As DataIdentifer = Me.GetSymbolIdentifier("010015FDAXU5", 1, compresionRange.Dias,
CDate("01/09/2015"), CDate("01/01/2036"))
```

Then we extract the minimum movement that we are willing to use:

```
Dim pip As Double = Me.auxdata.GetSymbolInfo(SymbolInfo.MinMov)
```

Finally, as the new data is consuming ressources on the memory, we freed it as we are no longer going to use it. To do so, we use the function *ReleaseDataIdentifier.*

```
Me.ReleaseDataIdentifier (auxdata)
```

Sell

Description:
This function is used to sell future contracts or to sell stocks at credit. It is important to know that the function is used to open short positions, not only to close the long positions. Therefore, if we want to cancel a long without opening a new short position, we must use the function ExitLong.

Syntax:
Sell(TradeType, Contracts, Price, Label)

Parameters:

Name	Default	Description
Type	AtClose	Type of order we are willing to launch (TraderType.AtClose, TraderType.AtMarket, TraderType.AtLimit and TraderType.AtStop).
Contracts	1	Number of contracts/stocks. The numerical specifications on contracts can be replaced by variables or by any other function previously defined.
Price	-	Sell price. This parameter must only be indicated for **TraderType.AtStop** and **TraderType.AtLimit** orders. The value can be expressed via a number, a variable or a function, or a mix of both.
Label	-	Order label in text format.

Example (VB.NET):

```
If (Me.GetMarketPosition() <> -1) then
        Me.Sell(TradeType.AtStop, 1, Me.Data.Close()-10, "V1")
End If
```

In this case the function sends an AtStop sell order (one contract), the stop price set at the close of the bar minus 10 points and the label “V1”.

SetBackGroundColor

Description:
This function is used to paint the background of the window, for a certain bar, in the indicated color.

Syntax:
SetBackGroundColor (BarsAgo, Color)

Parameters:

Name	Default	Description
BarsAgo	-	Number of bars backwards. The value 0 refers to the current bar.
Color	-	Color on which the background is to be painted for the indicated bar (BarsAgo). It uses the type **System.Drawing.Color.**

Example (VB.NET):
Me.SetBackGroundColor(1, Color.Red)   Will paint the background of the window (in the previous bar) in red.

SetBarColor

Description:
This function assigns a certain color to the indicated bar of a certain indicator line.

Syntax:
SetBarColor (BarsAgo, Line, Color)

Parameters:

Name	Default	Description
BarsAgo	-	Number of bars backwards. The value 0 refers to the current bar.
Line		Identifies the data line to which the bar on which the property will be established belongs.
Color	-	Color with which the indicated bar must be painted. It uses the type **System.Drawing.Color**.

Example (VB.NET):
Me.SetBarColor(0,1, Color.Red) Will paint in red the current bar of line 1.

SetBarProperties

Description:
This function assigns to the indicated bar, of a certain line of the indicator, the color, width and type of line and also the representation used in the rest of the parameters.

Syntax:
SetBarProperties (BarsAgo, Line, Color, Width, Style, Representation)

Parameters:

Name	Default	Description
BarsAgo	-	Number of bars backwards. The value 0 refers to the current bar.
Line	-	Identifies the data line to which the bar on which the property is established belongs.
Color	-	Color to paint the indicated bar. It uses the type **System.Drawing.Color**.
Width	-	Indicates the width to be applied to the bar (1,2,..).
Style	-	Style used for the representation: **LineStyle.Solid** continuous line ———————— **LineStyle.Dash** non-continuous line - - - - - - - - - - - **LineStyle.Dot** dotted line ·················· **LineStyle.Dashdot** dotted line with point -·-·-·-·-·-·-· **LineStyle.Dashdotdot** dotted line with 2 points —··—··—··—··
Representation	-	Type of representation to be used: **IndicatorRepresentation.Bars** bars **IndicatorRepresentation.Candlestic** candlesticks **IndicatorRepresentation.DottedLine** dotted line **IndicatorRepresentation.FilledHistogram** filled histogram **IndicatorRepresentation.Histogram** histogram

		IndicatorRepresentation.Lineal lineal **IndicatorRepresentation.Parabolic** parabolic **IndicatorRepresentation.Volume** volumen

Example (VB.NET):
Me.SetBarProperties(0,1, Color.Red,2,LineStyle.Solid,IndicatorRepresentation.Lineal)

In this case the function will paint in red the line 1 of the current bar (in format of continuous line and thickness 2).

Visual Chart 6 novelties:
As Visual Chart 6 enables to use the .NET Framework capabilities, it is possible to create custom attributes in order to provide additional information about the elements of the programme. Since each indicator´s project consists of a class creation, we can include several attributes to the class at issue. Among them, the attribute *OutputSeriesProperties*. This attribute allows to define the line style of the indicator alternative to the function *SetBarProperties*.

Syntax:
<OutputSeriesProperties(Line:=NumLine, Color:=Chart.Colors, ChartingStyle:=Chart.ChartStyle, Width:=WidthLine)>

The attribute *OutputSeriesProperties* will be defined under the attribute *Indicator*.

SetBarRepresentation

Description:
This function sets the type of representation for a certain bar.

Syntax:
SetBarRepresentation(BarsAgo, Line, Representation)

Parameters:

Name	Default	Description
BarsAgo	-	Number of bars backwards. The value 0 refers to the current bar.
Line		Identifies the data line to which the bar on which the property is established belongs.
Representation	-	Type of representation to be used, **IndicatorRepresentation** (the different types of representation can be checked with the function SetBarProperties).

Example (VB.NET):
Me.SetBarRepresentation(0,1, IndicatorRepresentation.Lineal)
The current bar of the indicator´s line number one will be represented in linear format.

SetBarStyle

Description:
This function sets the style of a certain bar.

Syntax:
SetBarStyle(BarsAgo, Line, LineStyle)

Parameters:

Name	Default	Description
BarsAgo	-	Number of bars backwards. The value 0 refers to the current bar.
Line		Identifies the data line to which the bar on which the property is established belongs.
Representation	-	Type of representation to be used, **LineStyle** (the different types of representation can be checked with the function SetBarProperties).

Example (VB.NET):
Me.SetBarStyle(0,1,LineStyle.Solid)
The current bar of the indicator´s line number one will be represented in continous line format.

SetBarWidth

Description:
Assigns to the indicated bar, of a certain line, the required width.

Syntax:
SetBarWidth(BarsAgo, Line, Width)

Parameters:

Name	Default	Description
BarsAgo	-	Number of bars backwards. The value 0 refers to the current bar.
Line		Identifies the data line to which the bar on which the property is established belongs.
Width	-	Thickness of the bar to be represented in.

Example (VB.NET):
Me.SetBarWidth(0,1, 2) The current bar of the indicator line number 1 will be represented with thickness 2.

SetHistogramBand

Description:
This function assigns to the indicated bar of a certain data series the value of the histogram band, in fact the value serving as limit to paint the histogram (if the representation used is the histogram). This function is strictly associated to the use of the property StartBarRepresentation.

Syntax:
SetHistogramBand(BarsAgo, Line, BandLine)

Parameters:

Name	Default	Description
BarsAgo	-	Number of bars backwards. The value 0 refers to the current bar.
Line		Identifies the data line to which the bar on which the property is established belongs.
BandLine	-	Data line used as reference to draw the histogram.

Example (VB.NET):
Supposing that we use the function .SetBarRepresentation (0,1, IndicatorRepresentation.Histogram). The indicator will need a reference value to oscillate around in order to generate the histogram.

In this case:

Me.SetIndicatorValue (Me.Data.Close – Me.Data.Close(1),1,0)   With line 1 we represent the difference between closes.

Me.SetIndicatorValue(0,2,0)   With line 2 we represent the value 0.

Next, by using the function SetBarRepresentation, we will indicate that line one will be painted with an histogram as default representation:

Me.SetBarRepresentation (0,1, IndicatorRepresentation.Histogram)

And finally we will indicate the line to oscillate around:

Me.SetHistogramBand (1,2)

This way, line 1 will use as line band (reference line) line 2.

SetIndicatorPos

Description:
Indicates to a certain bar, of a certain line, a certain position.

Syntax:
SetIndicatorPos(BarsAgo, Line, IndicatorPosition)

Parameters:

Name	Default	Description
BarsAgo	-	Number of bars backwards. The value 0 refers to the current bar.
Line	-	Identifies the data line to which the bar on which the property is established belongs.
IndicatorPosition	-	The trend can be of type IndicatorPosition, such as **IndicatorPosition.Bull** (bullish), **IndicatorPosition.Bear** (bearish) or **IndicatorPosition.Neutral** (flat).

Example (VB.NET):

Me.SetIndicatorPos(3,2,ipNeutral) Assigns to the bar number three backwards of line number 2 a neutral position.

SetIndicatorValue

Description:
This function assigns a value of the indicator in a certain bar. A certain trend will be ascribed to this value, if we specify something in the property *IndicatorPosition*.

Visual Chart 6 novelties:
If the value of the indicated bar by painting the line x is relative to a previous bar (that is to say, we do not specify 0 in the property *BarsAgo*), we must know that this line will be *blocked* to be used from a strategy. This is done in order to avoid that strategies rely on indicators that throw the information backwards.

Syntax:
SetIndicatorPos(Value, Line, BarsAgo, IndicatorPosition)

Parameters:

Name	Default	Description
Value	-	Numerical value for exit.
Line	1	Identifies the data line to which the bar on which the property is established belongs.
BarsAgo	-	Number of bars backwards. The value 0 refers to the current bar.
IndicatorPosition	-	The trend can be of type IndicatorPosition, such as **IndicatorPosition.Bull** (bullish), **IndicatorPosition.Bear** (bearish) or **IndicatorPosition.Neutral** (flat).

Example (VB.NET):
Imagine that we define a numerical variable and we assign to it the value returned by the following calculation:

```
Dim buffer3 As Double  = Me.Data.High(0) – Me.Data.Low(0) / Me.Data.High(0) * 100
```

Next, we can use the function *SetIndicatorValue* to paint on each bar the value calculated for this variable:

```
If (buffer3 > 50) then
    Me.SetIndicatorValue(buffer3, 1,0,IndicatorPosition.Bull)
Else
    Me.SetIndicatorValue(buffer3, 1,0,IndicatorPosition.Bear)
End If
```

The difference is that, depending on this value, the indicator will be ascribed a bull or bear trend.

SetLineName

Description:
Assigns a name to the indicated line. This method has not to be specified for each bar, since as long as a line

has assigned a name, this is valid for all bars. Therefore, we recommend declaring the function *SetLineName* from the method *OnInitCalculate.*

Syntax:
SetLineName(Line, LineName)

Parameters:

Name	Default	Description
Line	-	Identifies the data line to which the bar on which the property is established belongs.
LineName	-	The name is ascribed to the corresponding property.

Example (VB.NET):
From the method *OnInitCalculate* we specify the following:

Me.SetLineName(2, “DT”)   Assigns the name “DT” to line 2.

Visual Chart 6 novelties:
As Visual Chart 6 enables to use the .NET Framework capabilities, it is possible to create custom attributes in order to provide additional information about the elements of the programme. Since each indicator´s project consists of a class creation, we can include several attributes to the class at issue. Among them, the attribute *OutputSeriesProperties*. This attribute allows to define the line name of the indicator alternative to the function *SetLineName*.

Syntax:
<OutputSeriesProperties(Name:=LineName)>

The attribute *OutputSeriesProperties* is defined under the attribute *Indicator*.

SetWndBackGroundColor

Description:
Assigns the background color to the window of an indicator.

Syntax:
SetWndBackGrounColor()

Parameters:

Name	Default	Description
Color	-	Value of type System.Drawing.Color to indicate the background color of the window.

Example (VB.NET):

Me.SetWndBackgroundColor(Color.Beige)   Assigns the beige color to the indicator window.

ShouldTerminated

Description:
This function is used to stop the calculation process of a strategy. **ShouldTerminated** is a boolean variable, initialized with the value false, so that, in order to interrupt the calculations we must assign to it the value True.

This function turns out to be very useful when working with extended historical data and we want to stop the calculation under certain circumstances.

Syntax:
ShouldTerminated = True/False

Parameters:

Name	Default	Description
-	-	-

Example (VB.NET):

```
If Me.CurrentBar = 1000 And Me.NetProfit < 100 Then
    Me.ShouldTerminate = True
End If
```

In this case, the calculations will stop when 1000 bars of the calculation have gone by and the net profit is lower than 100 euros.

Slope

Description:
This function returns the value of the regression line slope, which is formed by the indicated prices between the bars **StartBar** and **EndBar** for a certain data series.
The function Slope returns for each bar the slope value of the regression equation, which associates the prices with the time variable, so that it can be translated as an indicator of the trend slope.

Syntax:
Me.Identifier.Slope(StarBar, EndBar, StarPrice, EndPrice)

Parameters:

Name	Default	Description
StarBar	-	Bar number of the regression line start.
EndBar	-	Bar number of the regression line end.
StarPrice	-	Start price of the regression line.
EndPrice	-	End price of the regression line.

Example:

We are willing to enter the market each time there is a slope change of the regression line. So if the slope changes from positive to negative we send a sell order and if it changes from negative to positive we send a buy order.

First, we extract the line slope of the last 10 bars in the current bar and the same slope in the previous bar:

```
    Dim slope As Double = Me.avdata.Slope(Bar - 10, Bar, Me.avdata.Value(10), Me.avdata.Value(0))
    Dim slope_ant As Double = Me.avdata.Slope(Bar - 11, Bar - 1, Me.avdata.Value(11),
Me.avdata.Value(1))
```

Next, we use this information to define the trading rules:

```
    If (slope > 0 And slope_ant <= 0) Then
        Me.Buy(TradeType.AtMarket)
    ElseIf (slope < 0 And slope_ant >= 0) Then
        Me.Sell(TradeType.AtMarket)
    End If
```

StandardDeviation

Description:

Returns the standard deviation.

Syntax:

StandardDeviation(Show As SttRepresentation)

Parameters:

Name	Default	Description
Show	Bypoints	Enables to indicate the format on which the information will show up: **SttRepresentation.ByPoints** (points) or **SttRepresentation.Porcentual** (percentage).

Example (VB.NET):

```
Dim DE as double = Me.StandardDeviation(SttRepresentation.ByPoints)
```

When we use this property, the value of the standard deviation in points will be ascribed to the variable DE.

StarBar

Description:

Enables to specify the start bar for the strategy calculation.

Syntax:

StarBar = (Number of bars)

Parameters:

Name	Default	Description
-	-	-

Example (VB.NET):
Assuming that our strategy trades on the basis of certain support and resistance levels calculated regarding the last 25 bars.

As we need at least 24 previous bars, we specify that the process must start from bar number 25:

```
Public Overrides Sub OnInitCalculate()
    Me.StartBar = 25
End Sub
```

Time

Description
Returns the value of the field Time of a certain bar. The time of a bar is given by the end time of the temporary period resuming the bar. The time of a bar is considered in military format (HHMM), so if the time of a bar is 5:35pm, in Visual Chart it will be considered as the numerical format 17:35h.

Syntax:
Me.Identifier.Time(BarsAgo)

Parameters:

Name	Default	Description
BarsAgo	0	Number of bars backward. By default it refers to the current bar.

Example (VB.NET):
Me.Data.Time(3)   Returns the time in military format (HHMM) from Data1 three bars backwards.

TimeEx

Description
Returs the date of the reference bar in date format (DD/MM/AAAA HH:MM:SS).

Syntax:
Me.Identifier.TimeEx(TickIndex, BarsAgo)

Parameters:

Name	Default	Description
TickIndex	0	Start parameter, which must be included. For this purpose, we declare a variable to store the returned value. It only has an effect

		on tick charts.
BarsAgo	0	Bar from which we are willing to extract the date. By default it refers to the current bar.

TickIndex is a Start parameter. When we are trading with a tick chart, some of them may have the same date. This parameter is filled by indicating the nth tick position referring to the same date.

Example (VB.NET):

We declare the variable type Long in which the tick number will be stored:

```
Dim tickindex As Long = 0
```

The function

```
Me.Data2.TimeExe(tickindex, 1)
```

Returns the date (DD/MM/AAAA HH:MM:SS) of the previous bar of the Data2 series.

TimeToMinutes

Description:

Returns the number of minutes passed since 00:00.

Syntax:

TimeToMinutes(Time)

Parameters:

Name	Default	Description
Time	-	Time in military format (HHMM)

Example (VB.NET):

Me.TimeToMinutes(1735)

Returns 1055 minutes.

TodayCurrentBar

Description:

Returns the bar number in relation to the total bars number of the session. In other words, it returns the distance in bars regarding the start bar of the session.

Visual Chart 6 novelties:

If we work with *End Of Day* strategies, to which we apply schedule filters to determine the trading intervals, we recommend that the parameters which establish the schedule margins are specified by a bar number. So instead of specifying *InitTime* equal to 930 (for example), we will define this parameter as *InitBar* equal to 2 (for example). Thanks to the new property *TodayCurrentBar*, it is very easy to check if the corresponding bar was reached or not.

Syntax:

TodayCurrentBar()

Parameters:

Name	Default	Description
-	-	

Example (VB.NET):

We include in a strategy two parameters in order to limit the trading period per session:

```
  <Parameter(Name:="InitBar", DefaultValue:=2, MinValue:=1, MaxValue:=20, Step:=1)>
  Private initbar As Integer
  <Parameter(Name:="EndBar", DefaultValue:=40, MinValue:=25, MaxValue:=50, Step:=1)>
  Private endbar As Integer
```

Then, we specify that only the entry rules within this interval are to be checked:

```
    If (Me.TodayCurrentBar >= Me.initbar And Me.TodayCurrentBar < Me.endbar) Then
      If (Me.rsidata.Value() < 70 And Me.rsidata.Value(1) >= 70) Then
        Me.Buy(TradeType.AtMarket)
      End If
    End If
```

TodayHigh

Description:

Returns the highest price within the last session.

Syntax:

TodayHigh()

Parameters:

Name	Default	Description
-	-	

Example (VB.NET):

```
   If GetMarketPosition() = 1 Then
       Me.ExitLong(TradeType.AtLimit, 1, Me.TodayHigh())
    End If
```

Activates a limit exit order on the basis of the session high.

TodayLow

Description:
Returns the lowest price within the last session.

Syntax:
TodayLow()

Parameters:

Name	Default	Descriptrion
-	-	

Example (VB.NET):

```
If GetMarketPosition() = -1 Then
   Me.ExitShort(TradeType.AtLimit, 1, Me.TodayLow())
End If
```

Activates a limit exit short order on the basis of the session low.

Volume

Description:
This function returns the value of the volume (negotiated stocks/contracts) in a bar for a certain data series.

Syntax:
Me.Identifier.Volume (BarsAgo)

Parameters:

Name	Default	Description
BarsAgo	0	Bar number. The default value refers to the current bar. Upon this parameter, we can indicate any numerical value contained under a variable or even type the number. It can also be specified as a function replacing the numerical value.

Example (VB.NET):
Me.Data2.Volume(15)

Returns the volume negotiated 15 bars backwards in the data series Data2.

WorstSeries

Description:
This function returns the worst series of losses according to their results.

Syntax:
Worstseries(Show)

Parameters:

Name	Default	Description
Show	Bypoints	Enables to indicate the format on which the information will show up: **SttRepresentation.ByPoints** (points) or **SttRepresentation.Porcentual** (percentage).

Example (VB.NET):

```
Dim worstseries As Double = Me.WorstSeries(SttRepresentation.ByPoints)
```

Assings to the variable the worst series in points until now.