PowerLanguage Keyword Reference

MULTICHARTS

Keyword Alphabetical Index

A-D E-N O-S T-Z

Keyword Alphabetical Index

Keyword Alphabetical Index

A-D E-N O-S T-Z

Keyword Alphabetical Index

A-D E-N O-S T-Z

Text_New_s
Text_New_self
Text_New_Self_BN
Text_New_Self_DT
Text_New_self_s
Text_SetAttribute
Text_SetBarNumber
Text_SetBGColor
Text_SetBorder
Text_SetColor
Text_SetFontName
Text_SetLocation
Text_SetLocation_BN
Text_SetLocation_DT
Text_SetLocation_s
Text_SetSize
Text_SetString
Text_SetStyle
Than
The
Then
This
ThreadSafe
Thursday
TickID
Ticks
Time
Time_s
Time_s2Time
Time2Time_s
TimeToString
TL_Anchor_to_Bars
TL_Delete
TL_Get_Anchor_to_Bars
TL_GetActive

GetAccount

Returns the account number for the account at a specific location in the list of accounts.

If the value returned by `GetNumAccounts` is a non-zero, then:

for 1 <= *`AccountLoc`* <= `GetNumAccounts`, the function returns the account number.

For other sequence numbers an empty string ("") is returned.

Usage

`GetAccount(`*`AccountLoc`*`)`

Where: *`AccountLoc`* - the sequence number of the account in the list of accounts.

Notes

This function can be used along with `GetNumAccounts`, to enumerate available accounts returned by broker.

Example

The broker returned the following list of accounts: "DU12345", "DU23456", "DU34567", "DU45678".

`GetAccount(3)` will return `"DU34567"`.

`GetAccount(5)` will return `""`.

GetAccountID

Returns the account number which is used for auto trading on the chart, where the strategy is applied.

Usage

`GetAccountID`()

Example

`GetAccountID`() will return `"DU34567"` if the account number which is used for auto trading on the chart, where the strategy is applied, is DU12345.

GetNumAccounts

Returns the number of accounts in the list of accounts obtained from broker.

If the list is empty, an "N/A" value, i.e. 0, is returned.

Usage

```
GetNumAccounts
```

Example

The broker returned the following list of accounts: "DU12345", "DU23456", "DU34567", "DU45678". `GetNumAccounts` will return a value of 4.

GetNumPositions

Returns the number of positions in MultiCharts Order and Position Tracker (not the actual amount at the broker).

Usage

`GetNumPositions(`*`Account`*`)`

Where: *`Account`* - account number at broker.

Example

`GetNumPositions("DU12345")` will return a value of 3 if the total number of open positions in MultiCharts Order and Position Tracker is 3.

GetPositionAveragePrice

Returns average price of the position.

Usage

`GetPositionAveragePrice(`*`Symbol`*`, `*`Account`*`)`

Where: *`Symbol`* - the name of the instrument.
`Account` - account number at broker.

Example

Will return a value of 1.3456 if the average entry price for the position, defined by specified account/symbol pair is 1.3456.

GetPositionOpenPL

Returns Open PL value in:

- Symbol currency for Avanza (calculated on MultiCharts side)
- Account currency for AvaTrade (calculated on the broker's side)
- Symbol currency for CQG (calculated on the broker's side)
- Symbol currency for Dukascopy (calculated on MultiCharts side)
- Account currency for FXCM (calculated on the broker's side)
- Symbol currency for Interactive Brokers (calculated on MultiCharts side)
- Symbol currency for LMAX (calculated on the broker's side)
- Symbol currency for MB Trading (calculated on MultiCharts side)
- Symbol currency for Open E Cry (calculated on the broker's side)
- Symbol currency for Patsystems (calculated on the broker's side)
- Symbol currency for Trading Technologies (calculated on MultiCharts side)
- Symbol currency for WeBank (calculated on MultiCharts side)
- Symbol currency for Rithmic (calculated on the broker's side)

Usage

`GetPositionOpenPL(`*`Symbol`*`, `*`Account`*`)`

Where: *`Symbol`* - the name of the instrument.
`Account` - account number at broker.

Example

Will return 15 if the current value in "Open PL" column in "Order and Position Tracker" window for the account is 15 units of the selected currency.

GetPositionQuantity

Returns the size of the position, defined by specified account/symbol pair.

Usage

`GetPositionQuantity(`*`Symbol`*, *`Account`*`)`

Where: *`Symbol`* - the name of the instrument.
`Account` - account number at broker.

Example

`GetPositionQuantity("ESZ1", "DU12345")` will return a value of 1000 if the size of the position, defined by specified account/symbol pair position is ¤1000.

GetPositionSymbol

Returns the symbol associated with the position at a specific location (*`PositionLoc`*) in the list of positions for the specified account.

Usage

`GetPositionSymbol(`*`Account`*`, `*`PositionLoc`*`)`

Where: *`Account`* - account number at broker.
1 <= *`PositionLoc`* <= `GetNumPositions` - the sequence number of the position in the position list for the specified account.

Example

`GetPositionSymbol("DU12345", 3)` will return `"ESZ1"` if the position with the sequence number 3 in the position list for account DU12345 is opened on ESZ1.

GetPositionTotalCost

Calculated with the following formula:

PTC (Position Total Cost) = AEP (Average Entry Price) x TMP (Total Market Position).

Usage

`GetPositionTotalCost(`*`Symbol`*`, `*`Account`*`)`

Where: *`Symbol`* - the name of the instrument.
`Account` - account number at broker.

Example

`GetPositionTotalCost("ESZ1", "DU12345")` will return total position cost for ESZ1 on account DU1234.

GetRTAccountEquity

Returns regular Account Equity at broker.

Usage

`GetRTAccountEquity(`*`Account`*`)`

Where: *`Account`* - account name at broker.

Example

`GetRTAccountEquity("DU12345")` will return a value of 100 000 if the Account Equity at broker for DU12345 account is ¤100 000.

GetRTAccountNetWorth

Returns Account Net Liquidation value at broker.

Usage

`GetRTAccountNetWorth(`*`Account`*`)`

Where: *`Account`* - account name at broker.

Example

`GetRTAccountNetWorth("DU12345")` will return a value of 100 000 if the Account Net Liquidity at broker for DU12345 account is 100 000 in base currency.

GetRTUnrealizedPL

Returns the Unrealized/Open P&L; for the specified account.

Usage

`GetRTUnrealizedPL(`*`Account`*`)`

Where: *`Account`* - account number at broker.

Example

Will return a value of 5 000 if Unrealized(Open) P&L; for the account at broker is ¤5 000.

InitialCapital

Returns a numerical value, indicating the amount of initial capital set in the strategy properties.

Usage

```
InitialCapital
```

Example

`InitialCapital` will return 10000 if Initial Capital amount was set to 10000 in the **Strategy Properties**.

Alert

Triggers an alert window where the necessary text can be created. The text can be dynamic, static or absent.

Usage

```
Alert
```

Notes

- If the text isn't set in the alert then Source, Symbol, Resolution, Price is displayed
- Alerts are shown on the last bar only
- Detailed information of alerts' settings is available in **Formatting Studies > Alerts**

Example

This statement will call an alert without text message:

```
Alert;
```

This statement creates a dynamic alert text and displays the number of the weekday:

```
Alert(Text("Day of week is ", DayOfWeek(Date)));
```

This statement will show static text:

```
If Close > Close[1] Then
Alert(Text("Price turning up"))
```

AlertEnabled

Returns `True` if the alerts have been turned on in Format Study > Alerts. This information can make script execution more efficient by discarding redundant calculations. This function can also be used to notify the user that he will not see the alerts until the relevant option is turned on.

Usage

```
AlertEnabled
```

Notes

- Alerts are only generated for the last bar
- There is a difference between `AlertEnabled` and `CheckAlert`: `AlertEnabled` returns `True`/`False` for all the bars on the chart while `CheckAlert` does so for the last bar only
- `AlertEnabled` will return `True` if alerts have been enabled

Example

The example below shows how a user can be notified that he has forgotten to turn on the alerts:

```
Variable: ID(-1);

If AlertEnabled=False And LastBarOnChart_S=True Then

ID = Text_New_S(Date, Time_S, Low, "Alerts are disabled. See Format
Study > Properties > Alerts");
```

Cancel Alert

The expression deactivates alerts. This is necessary if a script contains multiple alert statements and they need to be turned off under certain conditions.

Usage

```
Cancel Alert
```

Notes

Alerts are only generated for the last bar

Example

The example shows how all alerts can be turned off depending on time. The alerts will not be displayed after 10:00 pm.

```
If Close > Close[1] Then
Alert("Price is going up");
If Volume > Volume[1] Then
Alert("Volume is increasing");
If Volatility(5) > Volatility(5)[1] Then
Alert("Volatility is rising");
If OpenInt > OpenInt[1] Then
Alert("Open interest is growing");
If Time >= 2200 Then Cancel Alert;
```

CheckAlert

Returns `True` if the alerts have been turned on in **Formatting Studies > Alerts**.

`True`/`False` is returned on the last bar only.

For the bars other than the last `False` is always returned.

Usage

```
CheckAlert
```

Notes

- There is a difference between `AlertEnabled` and `CheckAlert`. `AlertEnabled` checks the status on all the bars while `CheckAlert` does so for the last bar only.
- Alerts are only generated for the last bar.

Example

The example below shows how `CheckAlert` can help eliminate redundant calculations on historic bars as well as in cases when alerts are not turned on:

```
If CheckAlert Then Begin If Volume >= 2 * Average(Volume, 10) Then Alert
("Volume is going up");
End;
```

Arw_Anchor_to_Bars

Anchors the corresponding arrow drawing to the visible bar index; returns a value of 0 if the operation was performed successfully, and a value of -2 if the specified trendline ID number is invalid.

Usage

`Arw_Anchor_to_Bars(`*`ArrowID`*`,`*`LogicalExpression`*`)`

Where: *`ArrowID`* is a numerical expression specifying the arrow drawing ID number
`LogicalExpression` is a logical value; True = add option and False = remove option

Notes

Arrow ID number is returned by `Arw_New` when the arrow drawing is created.

Example

Anchor the arrow drawing with an ID number of 5 to the visible bar index:

```
Value1=Arw_Anchor_to_Bars(5, true);
```

Arw_Delete

Removes an arrow object with the specified ID number from a chart; returns a value of 0 if the object was successfully removed, and a value of -2 if the specified object ID number is invalid.

Usage

`Arw_Delete(`*`ObjectID`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number

Notes

An object-specific ID number is returned by `Arw_New` when the arrow object is created.

Example

Remove the arrow object with an ID number of 3:

```
Value1=Arw_Delete(3);
```

Arw_GetActive

Returns a numerical value indicating the object ID number of the currently selected arrow object; returns a value of -1 if no arrow objects are currently selected.

Usage

```
Arw_GetActive
```

Notes

An object-specific ID number is assigned by `Arw_New` when the arrow object is created.

Example

Assign a value, indicating the object ID number of the currently selected arrow object, to Value1 variable:

```
Value1=Arw_GetActive;
```

Arw_GetBarNumber

Returns a numerical value representing the barnumber of the arrow object with a specified ID; returns a value of -2 if the specified object ID number is invalid.

Usage

`Arw_GetBarNumber(`*`ref`*`)`

Parameters:

`ref` - ID of the arrow object

Example

Get the number of the bar where the arrow object with ID = 1 is placed:

```
Arw_GetBarNumber(1);
```

Arw_GetColor

Returns an RGB color number or a legacy color value that correspond to the color of the arrow contained in an arrow object with the specified ID number; returns a value of -2 if the specified object ID number is invalid.

Usage

`Arw_GetColor(`*`ObjectID`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number

Notes

An object-specific ID number is returned by `Arw_New` when the arrow object is created.

Example

Assign an RGB color number, corresponding to the color of the arrow contained in an arrow object with the ID number of 3, to Value1 variable:

```
Value1=Arw_GetColor(3);
```

Assign a legacy color value, corresponding to the color of the arrow contained in an arrow object with the ID number of 3, to Value1 variable:

```
[LegacyColorValue=True];
Value1=Arw_GetColor(3);
```

Arw_GetDate

Returns a numerical value, indicating the date of the bar at which an arrow object with the specified ID number has been placed; returns a value of -2 if the specified object ID number is invalid.

The date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month.

Usage

`Arw_GetDate(`*`ObjectID`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number

Notes

An object-specific ID number is returned by `Arw_New` when the arrow object is created.

Example

Assign a value, indicating the date of the bar at which an arrow object with the ID number of 3 has been placed, to Value1 variable:

```
Value1=Arw_GetDate(3);
```

Arw_GetDirection

Returns a logical value indicating the direction of the arrow contained in an arrow object with the specified ID number; returns a value of `True` for Down arrow, and a value of `False` for Up arrow or if the specified object ID number is invalid.

Usage

`Arw_GetDirection(`*`ObjectID`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number

Notes

An object-specific ID number is returned by `Arw_New` when the arrow object is created.

Example

Assign a true/false value, indicating the direction of the arrow contained in an arrow object with the ID number of 3, to DownArrow variable:

```
Variable:DownArrow(False);
DownArrow=Arw_GetDirection(3);
```

Arw_GetFirst

Returns a numerical value, indicating the object ID number of the oldest (the first to be added to the current chart) arrow object of the specified origin; returns a value of -2 if the specified object ID number is invalid.

Usage

`Arw_GetFirst` (*`Origin`*)

Parameters

`Origin` - a numerical expression specifying the origin of the arrow object:

`1` - added by the current study
`2` - added by a study other then the current study, or drawn manually by the user
`3` - added by any study, or drawn manually by the user
`4` - added by the current study, or drawn manually by the user
`5` - added by a study other then the current study
`6` - added by any study
`7` - added manually by the user

Notes

If the oldest (the first added) arrow object is deleted, the next oldest (the second added) arrow object becomes the oldest (the first added) arrow object.

Example

Assign a value, indicating the object ID number of the oldest arrow object added to the chart by the current study, to Value1 variable:

```
Value1=Arw_GetFirst(1);
```

Arw_GetLock

Locked arrow drawings cannot be moved manually. Keyword returns a value of True for locked drawings, and a value of False for unlocked.

Usage

```
Arw_GetLock(ArrowID)
```

Where: *`ArrowID`* - a numerical expression specifying the arrow drawing ID number

Notes

An arrow ID number is returned by `Arw_New` when the arrow drawing is created.

Example

Assign Lock property of the arrow drawing with an ID number of 3 to Condition1 variable:

```
Condition1=Arw_GetLock(3);
```

Arw_GetNext

Returns the ID number of the first existing arrow object added subsequent to an arrow object with the specified ID number, with both objects of a specified origin; returns a value of -2 if the specified object ID number is invalid.

Usage

`Arw_GetNext(`*`ObjectID`*`,`*`Origin`*`)`

Parameters

`ObjectID` - a numerical expression specifying the object ID number

`Origin` - a numerical expression specifying the origin of the arrow objects:

`1` - added by the current study
`2` - added by a study other then the current study, or drawn manually by the user
`3` - added by any study, or drawn manually by the user
`4` - added by the current study, or drawn manually by the user
`5` - added by a study other then the current study
`6` - added by any study
`7` - added manually by the user

Example

Assign a value to Value1 variable, indicating the ID number of the first existing arrow object added subsequent to an arrow object with the ID number of 3, with both objects added by the current study:

```
Value1=Arw_GetNext(3,1);
```

Arw_GetSize

Returns a numerical value indicating the size of the arrow contained in an arrow object with the specified ID number; returns a value of -2 if the specified object ID number is invalid.

Usage

`Arw_GetSize(`*`ObjectID`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number

Notes

An object-specific ID number is returned by `Arw_New` when the arrow object is created.

Example

Assign a value, indicating the size of the arrow in an arrow object with the ID number of 3, to Value1 variable:

```
Value1=Arw_GetSize(3);
```

Arw_GetStyle

Returns a numerical value, indicating the style of the arrow in an arrow object with the specified ID number; returns a value of -2 if the specified object ID number is invalid.

Usage

`Arw_GetStyle(`*`ObjectID`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number

Notes

An object-specific ID number is returned by `Arw_New` when the arrow object is created.

Example

Assign a value, indicating the style of the arrow in an arrow object with the ID number of 3, to Value1 variable:

```
Value1=Arw_GetStyle(3);
```

Arw_GetText

Returns a string expression corresponding to the text contained in an arrow object with the specified ID number.

Usage

`Arw_GetText(`*`ObjectID`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number

Notes

An object-specific ID number is returned by `Arw_New` when the arrow object is created.

Example

`Arw_GetText(3)` will return a string expression corresponding to the text contained in an arrow object with the ID number of 3

Arw_GetTextAttribute

Returns a logical value indicating the setting for an attribute of the text in an arrow object with the specified ID number; returns a value of `True` if the attribute is set to on, and a value of `False` if the attribute is set to off or if the specified object ID number is invalid.

The settings of the following attributes can be returned: border, bold, italic, strike-out, and underline.

Usage

`Arw_GetTextAttribute(`*`ObjectID`*`,`*`Attribute`*`)`

Parameters

`ObjectID` - a numerical expression specifying the object ID number

`Attribute` - a numerical expression specifying the attribute:

`0` - border
`1` - bold
`2` - italic
`3` - strike-out
`4` - underline

Notes

An object-specific ID number is assigned by `Arw_New` when the arrow object is created.

Example

Assign a true/false value, indicating the setting of "bold" attribute for the arrow object with an ID number of 3, to ArwTxtBold variable:

```
Variable:ArwTxtBold(False);
ArwTxtBold=Arw_GetTextAttribute(3,1);
```

Arw_GetTextBGColor

Returns an RGB color number or a legacy color value that correspond to the text background color of an arrow object with the specified ID number; returns a value of -2 if the specified object ID number is invalid.

Usage

`Arw_GetTextBGColor(`*`ObjectID`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number

Notes

An object-specific ID number is returned by `Arw_New` when the arrow object is created.

Example

Assign an RGB color number, corresponding to the text background color of an arrow object with the ID number of 3, to Value1 variable:

```
Value1=Arw_GetTextBGColor(3);
```

Assign a legacy color value, corresponding to the text background color of an arrow object with the ID number of 3, to Value1 variable:

```
[LegacyColorValue=True];
Value1=Arw_GetTextBGColor(3);
```

Arw_GetTextColor

Returns an RGB color number or a legacy color value that correspond to the color of the text contained in an arrow object with the specified ID number; returns a value of -2 if the specified object ID number is invalid.

Usage

`Arw_GetTextColor(`*`ObjectID`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number

Notes

An object-specific ID number is returned by `Arw_New` when the arrow object is created.

Example

Assign an RGB color number, corresponding to the color of the text contained in an arrow object with the ID number of 3, to Value1 variable:

```
Value1=Arw_GetTextColor(3);
```

Assign a legacy color value, corresponding to the color of the text contained in an arrow object with the ID number of 3, to Value1 variable:

```
[LegacyColorValue=True];
Value1=Arw_GetTextColor(3);
```

Arw_GetTextFontName

Returns a string expression corresponding to the name of the text font assigned to an arrow object with the specified ID number.

Usage

`Arw_GetTextFontName(`*`ObjectID`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number

Notes

An object-specific ID number is returned by `Arw_New` when the arrow object is created.

Example

`Arw_GetTextFontName(3)` will return a string expression corresponding to the name of the text font assigned to an arrow object with the ID number of 3

Arw_GetTextSize

Returns a numerical value indicating the font size assigned to the text of an arrow object with the specified ID number; returns a value of -2 if the specified object ID number is invalid.

Usage

`Arw_GetTextSize(`*`ObjectID`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number

Notes

An object-specific ID number is returned by `Arw_New` when the arrow object is created.

Example

Assign a value, indicating the font size of the text in an arrow object with the ID number of 3, to Value1 variable:

```
Value1=Arw_GetTextSize(3);
```

Arw_GetTime

Returns a numerical value, indicating the time of the bar at which an arrow object with the specified ID number has been placed; returns a value of -2 if the specified object ID number is invalid.

The time is indicated in the 24-hour HHmm format, where 1300 = 1:00 PM.

Usage

`Arw_GetTime(`*`ObjectID`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number

Notes

An object-specific ID number is returned by `Arw_New` when the arrow object is created.

Example

Assign a value, indicating the time of the bar at which an arrow object with the ID number of 3 has been placed, to Value1 variable:

```
Value1=Arw_GetTime(3);
```

Arw_GetTime_DT

Returns a double-precision decimal DateTime value indicating the time of the bar at which an arrow object with the specified ID number has been placed; returns a value of -2 if the specified object ID number is invalid.

The time is indicated in the DateTime format, where the integer portion of the DateTime value indicates the number of days that have elapsed since January 1st, 1900, and the fractional portion of the DateTime value indicates the fraction of the day that has passed since midnight. DateTime is a floating point value with high precision. It allows accessing millisecond time stamps of the bar.

Usage

`Arw_GetTime_DT(`*`ObjectID`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number

Notes

An object-specific ID number is returned by `Arw_New_dt` when the arrow object is created.

Example

Assign a value, indicating the time of the bar at which an arrow object with the ID number of 3 has been placed, to Value1 variable:

```
Value1=Arw_GetTime_DT(3);
```

Arw_GetTime_s

Returns a numerical value indicating the time of the bar, including seconds, at which an arrow object with the specified ID number has been placed; returns a value of -2 if the specified object ID number is invalid.

The time is indicated in the 24-hour HHmmss format, where 130000 = 1:00:00 PM.

Usage

`Arw_GetTime_s(`*`ObjectID`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number

Notes

An object-specific ID number is returned by `Arw_New_s` when the arrow object is created.

Example

Assign a value, indicating the time of the bar at which an arrow object with the ID number of 3 has been placed, to Value1 variable:

```
Value1=Arw_GetTime_s(3);
```

Arw_GetVal

Returns the price value (vertical position, corresponding to a value on the price scale of a chart), at which an arrow object with the specified ID number has been placed; returns a value of -2 if the specified object ID number is invalid.

Usage

`Arw_GetVal(`*`ObjectID`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number

Notes

An object-specific ID number is returned by `Arw_New` when the arrow object is created.

Example

Assign a value, indicating the price value at which an arow object with the ID number of 3 has been placed, to Value1 variable:

```
Value1=Arw_GetVal(3);
```

Arw_Get_Anchor_to_Bars

Returns the value of "anchor to bar" option of the arrow drawing with a specified ID.

Usage

```
Arw_Get_Anchor_to_Bars(ArrowID)
```

Where: *`ArrowID`* is a numerical expression specifying the arrow drawing ID number

Notes

An arrow ID number is returned by `Arw_New` when the arrow drawing is created.

Example

Assign "anchor to bars" option of the arrow drawing with an ID number of 3 to the Condition1 variable:

```
Condition1=Arw_Get_Anchor_to_Bars(3);
```

Arw_Lock

Locks corresponding arrow drawing so it cannot be moved manually; returns a value of 0 if the operation was performed successfully, and a value of -2 if the specified trendline ID number is invalid.

Usage

`Arw_Lock(`*`ArrowID`*`,`*`LogicalExpression`*`)`

Where: *`ArrowID`* - a numerical expression specifying the arrow drawing ID number
`LogicalExpression` - a logical value; True = Add and False = Remove

Notes

An arrow ID number is returned by `Arw_New` when the arrow drawing is created.

Example

Lock the arrow drawing with an ID number of 3:

```
Value1=Arw_Lock(3,True);
```

Unlock the arrow drawing with an ID number of 5:

```
Value1=Arw_Lock(5,False);
```

Arw_New

Displays an object, consisting of an up or a down arrow located at the specified bar and specified price value, on the chart that the study is based on; returns an object-specific ID number, required to modify the object.

Usage

`Arw_New` (*BarDate, BarTime, PriceValue, Direction*)

Parameters

`BarDate` - a numerical expression specifying the date of the bar at which the object is to be placed; the date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month

`BarTime` - a numerical expression specifying the time of the bar at which the object is to be placed; the time is indicated in the 24-hour HHmm format, where 1300 = 1:00 PM

`PriceValue` - a numerical expression specifying the price value (vertical position, corresponding to a value on the price scale of a chart), where the object is to be placed

`Direction` - a logical expression specifying the direction of the arrow; True = Down and False = Up

Example

Place, on the chart that the study is based on, an up arrow at the top of a bar if the Open price has increased incrementally over the last three bars:

```
If Open>Open[1] And Open[1]>Open[2] Then
Value1=Arw_New(Date,Time,High,False);
```

Arw_New_BN

Displays an object consisting of an up or a down arrow located at the specified bar and specified price value on the chart that the study is based on; returns an object-specific ID number required to modify the object.

Usage

`Arw_New_BN` (*BarNumber*, *PriceValue*, *Direction*)

Parameters

`BarNumber` - a numerical expression specifying the bar number at which the object is to be placed.

`PriceValue` - a numerical expression specifying the price value (vertical position, corresponding to a value on the price scale of a chart), where the object should be placed.

`Direction` - a logical expression specifying the direction of the arrow; True = Down and False = Up

Example

On the chart that the study is based on, place an up arrow at the top of a current bar if the Open price has increased incrementally over the last three bars:

```
If Open>Open[1] And Open[1]>Open[2] Then
Value1=Arw_New_BN(currentbar,High,False);
```

Arw_New_DT

Displays an object, consisting of an up or a down arrow located at the specified bar and specified price value, on the chart that the study is based on; returns an object-specific ID number, required to modify the object.

Usage

`Arw_New_DT` (*Bar_DateTime, PriceValue, Direction*)

Parameters

`Bar_DateTime` - a numerical expression specifying the date and time of the bar at which the object is to be placed. The date and time are indicated in the DateTime format, where the integer portion of the DateTime value indicates the number of days that have elapsed since January 1st, 1900, and the fractional portion of the DateTime value indicates the fraction of the day that has passed since midnight. DateTime is a floating point value with high precision. It allows accessing millisecond time stamps of the bar.

`PriceValue` - a numerical expression specifying the price value (vertical position, corresponding to a value on the price scale of a chart), where the object is to be placed.

`Direction` - a logical expression specifying the direction of the arrow; True = Down and False = Up

Example

Place, on the chart that the study is based on, an up arrow at the top of a bar if the Open price has increased incrementally over the last three bars:

```
If Open>Open[1] And Open[1]>Open[2] Then
Value1=Arw_New_DT(DateTime,High,False);
```

Arw_New_s

Displays an object, consisting of an up or a down arrow located at the specified bar and specified price value, on the chart that the study is based on; returns an object-specific ID number, required to modify the object.

Usage

`Arw_New_s` (*BarDate*, *BarTime_s*, *PriceValue*, *Direction*)

Parameters

`BarDate` - a numerical expression specifying the date of the bar at which the object is to be placed; the date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month

`BarTime_s` - a numerical expression specifying the time of the bar, including seconds, at which the object is to be placed; the time is indicated in the 24-hour HHmmss format, where 130000 = 1:00:00 PM

`PriceValue` - a numerical expression specifying the price value (vertical position, corresponding to a value on the price scale of a chart), where the object is to be placed

`Direction` - a logical expression specifying the direction of the arrow; True = Down and False = Up

Example

Place, on the chart that the study is based on, an up arrow at the top of a bar if the Open price has increased incrementally over the last three bars:

```
If Open>Open[1] And Open[1]>Open[2] Then
Value1=Arw_New_s(Date,Time_s,High,False);
```

Arw_New_self

Displays an object, consisting of an up or a down arrow located at the specified bar and specified price value, on the SubChart containing the study; returns an object-specific ID number, required to modify the object.

Usage

`Arw_New_self` (*BarDate, BarTime, PriceValue, Direction*)

Parameters

`BarDate` - a numerical expression specifying the date of the bar at which the object is to be placed; the date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month

`BarTime` - a numerical expression specifying the time of the bar at which the object is to be placed; the time is indicated in the 24-hour HHmm format, where 1300 = 1:00 PM

`PriceValue` - a numerical expression specifying the price value (vertical position, corresponding to a value on the price scale of a chart), where the object is to be placed

`Direction` - a logical expression specifying the direction of the arrow; True = Down and False = Up

Example

Place, on the SubChart containing the study, an up arrow at the top of a bar if the Open price has increased incrementally over the last three bars:

```
If Open>Open[1] And Open[1]>Open[2] Then
Value1=Arw_New_self(Date,Time,High,False);
```

ARW_New_Self_BN

The same as `Arw_New_BN`. Difference: Displays a arrow on the SubChart containing the study.

Arw_New_Self_DT

Displays an object, consisting of an up or a down arrow located at the specified bar and specified price value, on the SubChart containing the study; returns an object-specific ID number, required to modify the object.

Usage

`Arw_New_Self_DT` (*Bar_DateTime*, *PriceValue*, *Direction*)

Parameters

`Bar_DateTime` - a numerical expression specifying the date and time of the bar at which the object is to be placed. The date and time are indicated in the DateTime format, where the integer portion of the DateTime value indicates the number of days that have elapsed since January 1st, 1900, and the fractional portion of the DateTime value indicates the fraction of the day that has passed since midnight. DateTime is a floating point value with high precision. It allows accessing millisecond time stamps of the bar.

`PriceValue` - a numerical expression specifying the price value (vertical position, corresponding to a value on the price scale of a chart), where the object is to be placed.

`Direction` - a logical expression specifying the direction of the arrow; True = Down and False = Up

Example

Place, on the SubChart containing the study, an up arrow at the top of a bar if the Open price has increased incrementally over the last three bars:

```
If Open>Open[1] And Open[1]>Open[2] Then
Value1=Arw_New_Self_DT(DateTime,High,False);
```

Arw_New_self_s

Displays an object, consisting of an up or a down arrow located at the specified bar and specified price value, on the SubChart containing the study; returns an object-specific ID number, required to modify the object.

Usage

`Arw_New_self_s` (*BarDate*, *BarTime_s*, *PriceValue*, *Direction*)

Parameters

`BarDate` - a numerical expression specifying the date of the bar at which the object is to be placed; the date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month

`BarTime_s` - a numerical expression specifying the time of the bar, including seconds, at which the object is to be placed; the time is indicated in the 24-hour HHmmss format, where 130000 = 1:00:00 PM

`PriceValue` - a numerical expression specifying the price value (vertical position, corresponding to a value on the price scale of a chart), where the object is to be placed

`Direction` - a logical expression specifying the direction of the arrow; True = Down and False = Up

Example

Place, on the SubChart containing the study, an up arrow at the top of a bar if the Open price has increased incrementally over the last three bars:

```
If Open>Open[1] And Open[1]>Open[2] Then
Value1=Arw_New_self_s(Date,Time_s,High,False);
```

Arw_SetBarNumber

Assigns the specified barnumber to the arrow object with the specified ID number; returns a value of 0 if the barnumber was successfully assigned, and a value of -2 if the specified object ID number is invalid.

Usage

`Arw_SetBarNumber(`*`ref`*`,`*`Barnumber`*`)`

Parameters:

- *`ref`* - ID of the arrow object
- *`Barnumber`* - the new bar number that is to be assigned to the specified object.

Example

Assign the new barnumber value of 100 to the arrow object with ID = 1:

```
Arw_SetBarNumber(1, 100);
```

Arw_SetColor

Assigns the specified color to the arrow contained in an arrow object with the specified ID number; returns a value of 0 if the color was successfully assigned, and a value of -2 if the specified object ID number is invalid.

Usage

`Arw_SetColor(`*ObjectID*,*`ArrowColor`*`)`

Parameters

`ObjectID` - a numerical expression specifying the object ID number

`ArrowColor` - an expression specifying the color of the arrow

The color can be specified by a numerical expression representing an RGB color number or a legacy color value, or by one of 17 base color words.

Notes

An object-specific ID number is returned by `Arw_New` when the arrow object is created.

Example

Assign the color blue to the arrow contained in an arrow object with the ID number of 3:

```
Value1=Arw_SetColor(3,Blue);
```

Assign the RGB color 2138336 (Orange) to the arrow contained in an arrow object with the ID number of 3:

```
Value1=Arw_SetColor(3,2138336);
```

Assign the legacy color 4 (Green) to the arrow contained in an arrow object with the ID number of 3:

```
[LegacyColorValue=True];
Value1=Arw_SetColor(3,4);
```

Arw_SetLocation

Modifies the location of an arrow object with the specified ID number; returns a value of 0 if the location of the object was successfully modified, and a value of -2 if the specified object ID number is invalid.

Usage

`Arw_SetLocation` (*ObjectID*, *BarDate*, *BarTime*, *PriceValue*)

Parameters

`ObjectID` - a numerical expression specifying the object ID number

`BarDate` - a numerical expression specifying the date of the bar at which the object is to be placed; the date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month

`BarTime` - a numerical expression specifying the time of the bar at which the object is to be placed; the time is indicated in the 24-hour HHmm format, where 1300 = 1:00 PM

`PriceValue` - a numerical expression specifying the price value (vertical position, corresponding to a value on the price scale of a chart), where the object is to be placed

Notes

An object-specific ID number is returned by `Arw_New` when the arrow object is created.

Example

Move the arrow object with an ID number of 3 to the top of the current bar:

```
Value1=Arw_SetLocation(3,Date,Time,High);
```

Arw_SetLocation_BN

Modifies location of an arrow object with the specified ID number; returns a value of 0 if location of the object was successfully modified, and a value of -2 if the specified object ID number is invalid.

Usage

`Arw_SetLocation_BN` (*ObjectID*, *BarNumber*, *PriceValue*)

Parameters

`ObjectID` - a numerical expression specifying the object ID number.

`BarNumber` - a numerical expression specifying the bar number at which the object is to be placed.

`PriceValue` - a numerical expression specifying the price value (vertical position, corresponding to a value on the price scale of a chart), where the object should be placed.

Notes

An object-specific ID number is returned by `Arw_New_Dt` when the arrow object is created.

Example

Move the arrow object with an ID number of 3 to the top of the current bar:

```
Value1=Arw_SetLocation_BN(3, currentbar,High);
```

Arw_SetLocation_DT

Modifies the location of an arrow object with the specified ID number; returns a value of 0 if the location of the object was successfully modified, and a value of -2 if the specified object ID number is invalid.

Usage

`Arw_SetLocation_DT` (*ObjectID*, *Bar_DateTime*, *PriceValue*)

Parameters

`ObjectID` - a numerical expression specifying the object ID number.

`Bar_DateTime` - a numerical expression specifying the date and time of the bar at which the object is to be placed. The date and time are indicated in the DateTime format, where the integer portion of the DateTime value indicates the number of days that have elapsed since January 1st, 1900, and the fractional portion of the DateTime value indicates the fraction of the day that has passed since midnight. DateTime is a floating point value with high precision. It allows accessing millisecond time stamps of the bar.

`PriceValue` - a numerical expression specifying the price value (vertical position, corresponding to a value on the price scale of a chart), where the object is to be placed.

Notes

An object-specific ID number is returned by `Arw_New_Dt` when the arrow object is created.

Example

Move the arrow object with an ID number of 3 to the top of the current bar:

```
Value1=Arw_SetLocation_DT(3, DateTime,High);
```

Arw_SetLocation_s

Modifies the location of an arrow object with the specified ID number; returns a value of 0 if the location of the object was successfully modified, and a value of -2 if the specified object ID number is invalid.

Usage

`Arw_SetLocation_s` (*ObjectID*, *BarDate*, *BarTime_s*, *PriceValue*)

Parameters

`ObjectID` - a numerical expression specifying the object ID number

`BarDate` - a numerical expression specifying the date of the bar at which the object is to be placed; the date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month

`BarTime_s` - a numerical expression specifying the time of the bar, including seconds, at which the object is to be placed; the time is indicated in the 24-hour HHmmss format, where 130000 = 1:00:00 PM

`PriceValue` - a numerical expression specifying the price value (vertical position, corresponding to a value on the price scale of a chart), where the object is to be placed

Notes

An object-specific ID number is returned by `Arw_New_s` when the arrow object is created.

Example

Move the arrow object with an ID number of 3 to the top of the current bar:

```
Value1=Arw_SetLocation_s(3,Date,Time_s,High);
```

Arw_SetSize

Assigns the specified size to the arrow contained in an arrow object with the specified ID number; returns a value of 0 if the size was successfully assigned, and a value of -2 if the specified object ID number is invalid.

Usage

`Arw_SetSize(`*ObjectID*,*`ArrowSize`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number
`ArrowSize` - a numerical expression specifying the arrow size

Notes

An object-specific ID number is returned by `Arw_New` when the arrow object is created.

Example

Assign an arrow size of 10 to the arrow object with an ID number of 3:

```
Value1=Arw_SetSize(3,10);
```

Arw_SetStyle

Assigns the specified style to the arrow of an arrow object with the specified ID number; returns a value of 0 if the arrow style was successfully modified, and a value of -2 if the specified object ID number is invalid.

Usage

`Arw_SetStyle(`*ObjectID*,*`ArrowStyle`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number
`ArrowStyle` - a numerical expression specifying the arrow style; arrow styles range from 0 to 13

Notes

An object-specific ID number is returned by `Arw_New` when the arrow object is created.

Example

Assign the arrow style of 5 to an arrow object with the ID number of 3:

```
Value1=Arw_SetStyle(3,5);
```

Arw_SetText

Adds text to, or replaces the existing text contained in an arrow object with the specified ID number; returns a value of -2 if the specified object ID number is invalid.

Usage

`Arw_SetText(`*`ObjectID`*`,"`*`Text`*`")`

Where: *`ObjectID`* - a numerical expression specifying the object ID number
`Text` - the string expression to be displayed in the arrow object

Notes

An object-specific ID number is returned by `Arw_New` when the arrow object is created.

Example

Add text "My Arrow" to the arrow object with an ID number of 3:

```
Value1=Arw_SetText(3,"My Arrow");
```

Replace text contained in the arrow object with an ID number of 3 with the string expression "New Text":

```
Value1=Arw_SetText(3,"New Text");
```

Arw_SetTextAttribute

Sets an attribute of the text in an arrow object with the specified ID number; returns a value of 0 if the attribute was successfully set, and a value of -2 if the specified object ID number is invalid.

The following text attributes can be set: border, bold, italic, strike-out, and underline.

Usage

`Arw_SetTextAttribute(`*`ObjectID`*`,`*`Attribute`*`,`*`LogicalExpression`*`)`

Parameters

`ObjectID` - a numerical expression specifying the object ID number

`Attribute` - a numerical expression specifying the attribute:

`0` - border
`1` - bold
`2` - italic
`3` - strike-out
`4` - underline

`LogicalExpression` - a logical value; True = on and False = off

Notes

An object-specific ID number is returned by `Arw_New` when the arrow object is created.

Example

Set the attribute "bold" to on for the text in an arrow object with the ID number of 3:

```
Value1=Arw_SetTextAttribute(3,1,True);
```

Arw_SetTextBGColor

Assigns the specified background color to the text of an arrow object with the specified ID number; returns a value of 0 if the color was successfully assigned, and a value of -2 if the specified object ID number is invalid.

Usage

`Arw_SetTextBGColor(`*ObjectID*`,`*`TextBGColor`*`)`

Parameters

`ObjectID` - a numerical expression specifying the object ID number

`TextBGColor` - an expression specifying the text background color

The color can be specified by a numerical expression representing an RGB color number or a legacy color value, or by one of 17 base color words.

Notes

An object-specific ID number is returned by `Arw_New` when the arrow object is created.

Example

Assign the color blue to the text background of an arrow object with the ID number of 3:

```
Value1=Arw_SetTextBGColor(3,Blue);
```

Assign the RGB color 2138336 (Orange) to the text background of an arrow object with the ID number of 3:

```
Value1=Arw_SetTextBGColor(3,2138336);
```

Assign the legacy color 4 (Green) to the text background of an arrow object with the ID number of 3:

```
[LegacyColorValue=True];
Value1=Arw_SetTextBGColor(3,4);
```

Arw_SetTextColor

Assigns the specified color to the text contained in an arrow object with the specified ID number; returns a value of 0 if the color was successfully assigned, and a value of -2 if the specified object ID number is invalid.

Usage

`Arw_SetTextColor(`*ObjectID*,*`TextColor`*`)`

Parameters

`ObjectID` - a numerical expression specifying the object ID number

`TextColor` - an expression specifying the color of the text

The color can be specified by a numerical expression representing an RGB color number or a legacy color value, or by one of 17 base color words.

Notes

An object-specific ID number is returned by `Arw_New` when the arrow object is created.

Example

Assign the color blue to the text contained in an arrow object with the ID number of 3:

```
Value1=Arw_SetTextColor(3,Blue);
```

Assign the RGB color 2138336 (Orange) to the text contained in an arrow object with the ID number of 3:

```
Value1=Arw_SetTextColor(3,2138336);
```

Assign the legacy color 4 (Green) to the text contained in an arrow object with the ID number of 3:

```
[LegacyColorValue=True];
Value1=Arw_SetTextColor(3,4);
```

Arw_SetTextFontName

Assigns the specified font to the text of an arrow object with the specified ID number; returns a value of -2 if the specified object ID number is invalid.

Any font in the **Fonts** folder can be used; the folder is accessible from the **Control Panel** in Windows XP operating system.

Usage

`Arw_SetTextFontName(`*`ObjectID`*`,"`*`FontName`*`")`

Where: *`ObjectID`* - a numerical expression specifying the object ID number
`FontName` - a string expression specifying the font name

Notes

An object-specific ID number is returned by `Arw_New` when the arrow object is created.

Example

Assign font Verdana to the text of an arrow object with the ID number of 3:

```
Value1=Arw_SetTextFontName(3,"Verdana");
```

Arw_SetTextSize

Assigns the specified font size to the text of an arrow object with the specified ID number; returns a value of 0 if the font size was successfully assigned, and a value of -2 if the specified object ID number is invalid.

Usage

`Arw_SetTextSize(`*`ObjectID`*`,`*`FontSize`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number
`FontSize` - a numerical expression specifying the font size

Notes

An object-specific ID number is returned by `Arw_New` when the arrow object is created.

Example

Assign the font size of 16 to the text of an arrow object with the ID number of 3:

```
Value1=Arw_SetTextSize(3,16);
```

MC_Arw_GetActive

Returns a numerical value indicating the arrow ID number of the currently selected arrow; returns a value of -1 if no arrows are currently selected.

Usage

```
MC_Arw_GetActive
```

Example

Assign a value, indicating the arrow ID number of the currently selected arrow, to `Value1` variable: `Value1` = `MC_Arw_GetActive`;

AllowSendOrdersAlways

Attribute set to True allows order generation even when barstatus is not defined (-1 will be returned).

Usage

`[AllowSendOrdersAlways = *LogicalValue*]`

Where: `LogicalValue` - a true/false value; True = Enable; False = Disable

If the attribute is set to False, or not present in the study's code, it doesn't allow order generation when barstatus is not defined.

If the attribute is set to True, then allows generation even when barstatus = -1.

Example

Enable AllowSendOrdersAlways value:

```
[AllowSendOrdersAlways = True];
```

IntraBarOrderGeneration

Toggles intra-bar order generation flag on or off.

Usage

```
[IntrabarOrderGeneration = LogicalValue]
```

Where: *`LogicalValue`* - a true/false value; True = Enable; False = Disable

If the attribute is not present in the study's code, intra-bar order generation can be set by the user in the Calculations tab of the Format Signal window.

If set to True, "Enable intra-bar order generation and calculation" check box will be checked and grayed out, and the radio buttons will be available.

If set to False, "Enable intra-bar order generation and calculation" check box will be unchecked and grayed out, and the radio buttons disabled.

Notes

Attributes are applied at the time of compilation and cannot be changed at run-time.

Example

Enable intra-bar order generation:

```
[IntrabarOrderGeneration = True];
```

LegacyColorValue

Specifies the color designation scheme to be used for interpretation of numerical color values.

Usage

`[LegacyColorValue = `*`LogicalValue`*`]`

Where: *`LogicalValue`* - a true/false value; True = Enable; False = Disable

If the attribute is set to False, or not present in the study's code, the RGB (new) 16M color scheme will be used.

If LegacyColorValue the attribute is set to True, the legacy (old) 16-bit, 16-color scheme will be used.

Notes

When a color name is converted to a numerical color value, the specified scheme will be followed.

To ensure compatibility, the attribute [LegacyColorValue = True] is inserted automatically when older versions, prior to v8.1, of EasyLanguage studies are imported.

Example

Enable legacy color value interpretation:

```
[LegacyColorValue = True];
```

Create a Green color plot using the legacy color scheme:

```
[LegacyColorValue = True];
```

```
Plot1(Open);

SetPlotColor(1,4);
```

Create a Green color plot using the RGB color scheme:

```
[LegacyColorValue = False];

Plot1(Open);

SetPlotColor(1,65280);
```

Assign a value, representing the color Green under the legacy color scheme, to Value1 (Value1 will be assigned a value of 4):

```
[LegacyColorValue = True];

Value1=Green;
```

Assign a value, representing the color Green under the RGB color scheme, to Value1 (Value1 will be assigned a value of 65280):

```
[LegacyColorValue = False];

Value1=Green;
```

ProcessMouseEvents

Declares that the study processes mouse events.

Usage

`[ProcessMouseEvents = *LogicalValue*]`

Where: *`LogicalValue`* - a true/false value; True = Enable; False = Disable

Notes

- If this attribute is not present in the study's code, the indicator based in the mouse events will not be calculated.
- Can be used only in signals and indicators

Example

```
[ProcessMouseEvents = True];
```

SameExitFromOneEntryOnce

Attribute defines possibility of position Exit using one and the same order. If it is not specified in the script, then set to True by default.

Usage

```
[SameExitFromOneEntryOnce = LogicalValue]
```

Where: *LogicalValue* - a true/false value; True = Enable; False = Disable

If the attribute is set to True, or not present in the study's code, then exit from the specified position using one and the same order is not possible.

If the attribute is set to False, then specified position can be closed using one and the same order unlimitedly.

Example

Disable SameExitFromOneEntryOnce value interpretation:

```
[SameExitFromOneEntryOnce = False];

buy 5 contracts next bar market;

sell 1 contracts next bar market;
```

When backtesting the given strategy one and the same Exit order (generated by sell next bar market command) is applied for partial position close 5 times.

Commenting out the first script line default behavior will be obtained. Exit order will trigger only once.

Black

Constant, used to designate the color Black.

Usage

```
Black
```

Notes

The base 17 colors can be designated by constants.

Example

Create a plot using the color Black:

```
Plot1(Open);
SetPlotColor(1,Black);
```

Blue

Constant, used to designate the color Blue.

Usage

```
Blue
```

Notes

The base 17 colors can be designated by constants.

Example

Create a plot using the color Blue:

```
Plot1(Open);
SetPlotColor(1,Blue);
```

Cyan

Constant, used to designate the color Cyan.

Usage

```
Cyan
```

Notes

The base 17 colors can be designated by constants.

Example

Create a plot using the color Cyan:

```
Plot1(Open);
SetPlotColor(1,Cyan);
```

DarkBlue

Constant, used to designate the color DarkBlue.

Usage

```
DarkBlue
```

Notes

The base 17 colors can be designated by constants.

Example

Create a plot using the color DarkBlue:

```
Plot1(Open);
SetPlotColor(1,DarkBlue);
```

DarkBrown

Constant, used to designate the color DarkBrown.

Usage

```
DarkBrown
```

Notes

The base 17 colors can be designated by constants.

Example

Create a plot using the color DarkBrown:

```
Plot1(Open);
SetPlotColor(1,DarkBrown);
```

DarkCyan

Constant, used to designate the color DarkCyan.

Usage

```
DarkCyan
```

Notes

The base 17 colors can be designated by constants.

Example

Create a plot using the color DarkCyan:

```
Plot1(Open);
SetPlotColor(1,DarkCyan);
```

DarkGray

Constant, used to designate the color DarkGray.

Usage

```
DarkGray
```

Notes

The base 17 colors can be designated by constants.

Example

Create a plot using the color DarkGray:

```
Plot1(Open);
SetPlotColor(1,DarkGray);
```

DarkGreen

Constant, used to designate the color DarkGreen.

Usage

```
DarkGreen
```

Notes

The base 17 colors can be designated by constants.

Example

Create a plot using the color DarkGreen:

```
Plot1(Open);
SetPlotColor(1,DarkGreen);
```

DarkMagenta

Constant, used to designate the color DarkMagenta.

Usage

```
DarkMagenta
```

Notes

The base 17 colors can be designated by constants.

Example

Create a plot using the color DarkMagenta:

```
Plot1(Open);
SetPlotColor(1,DarkMagenta);
```

DarkRed

Constant, used to designate the color DarkRed.

Usage

```
DarkRed
```

Notes

The base 17 colors can be designated by constants.

Example

Create a plot using the color DarkRed:

```
Plot1(Open);
SetPlotColor(1,DarkRed);
```

DarkYellow

Constant, used to designate the color DarkYellow.

Usage

```
DarkYellow
```

Notes

The base 17 colors can be designated by constants.

Example

Create a plot using the color DarkYellow:

```
Plot1(Open);
SetPlotColor(1,DarkYellow);
```

GetBValue

Returns the Blue color component value of an RGB color number; the value ranges from 0 to 255.

Usage

`GetBValue`(*BigRGBValue*)

Where: ***`BigRGBValue`*** - an RGB color number from 0 to 16777215

Example

Assign a value, representing the Blue color component of the RGB color number 2138336 (Orange), to Value1 (Value1 will be assigned a value of 32):

```
Value1=GetBValue(2138336);
```

GetGValue

Returns the Green color component value of an RGB color number; the value ranges from 0 to 255.

Usage

`GetGValue`(*BigRGBValue*)

Where: *BigRGBValue* - an RGB color number from 0 to 16777215

Example

Assign a value, representing the Green color component of the RGB color number 2138336 (Orange), to Value1 (Value1 will be assigned a value of 160):

```
Value1=GetGValue(2138336);
```

GetRValue

Returns the Red color component value of an RGB color number; the value ranges from 0 to 255.

Usage

`GetRValue(`*BigRGBValue*`)`

Where: *BigRGBValue* - an RGB color number from 0 to 16777215

Example

Assign a value, representing the Red color component of the RGB color number 2138336 (Orange), to Value1 (Value1 will be assigned a value of 224):

```
Value1=GetRValue(2138336);
```

GradientColor

Returns an RGB color number, representing a shade of color from within a defined color range. The gradient shade of color is determined by the value of the specified numerical expression in relation to the defined value range.

For example, if the color range is defined as White to Black, and the value range is defined from 0 to 2, GradientColor will return an RGB color number representing White for the value of 0, Gray for the value of 1, and Black for the value of 2. White will be returned for all values < 0, and Black for all values > 2.

Usage

`GradientColor`(*Value*,*Min*,*Max*,*StartColor*,*EndColor*)

Where: *Value* the specified numerical expression
Min the minimum value of the value range; if Value=Min, StartColor is returned
Max the maximum value of the value range; if Value=Max, EndColor is returned
StartColor the starting color of the color range
EndColor the ending color of the color range

Example

Plot an ADX indicator in Gradient Color, where Magenta gradually changes into White for the indicator values from 5 to 50:

```
Variable:ADXValue(0);
ADXValue=ADX(14);
Plot1(ADXValue,"ADXValue");
SetPlotColor (1,GradientColor (ADXValue,5,50, Magenta,White));
```

Green

Constant, used to designate the color Green.

Usage

```
Green
```

Notes

The base 17 colors can be designated by constants.

Example

Create a plot using the color Green:

```
Plot1(Open);
SetPlotColor(1,Green);
```

LegacyColorToRGB

Returns an RGB color number that corresponds to the specified legacy color value; the number ranges from 0 to 16777215.

Usage

`LegacyColorToRGB`(*LegacyColorValue*)

Where: *`LegacyColorValue`* - a legacy color value from 1 to 16

Example

Assign a value, representing the color 4 (Green) under the legacy color scheme, to Value1 (Value1 will be assigned a value of 65280):

```
Value1 = LegacyColorToRGB(4);
```

LightGray

Constant, used to designate the color LightGray.

Usage

```
LightGray
```

Notes

The base 17 colors can be designated by constants.

Example

Create a plot using the color LightGray:

```
Plot1(Open);
SetPlotColor(1,LightGray);
```

Magenta

Constant, used to designate the color Magenta.

Usage

```
Magenta
```

Notes

The base 17 colors can be designated by constants.

Example

Create a plot using the color Magenta:

```
Plot1(Open);
SetPlotColor(1,Magenta);
```

Red

Constant, used to designate the color Red.

Usage

```
Red
```

Notes

The base 17 colors can be designated by constants.

Example

Create a plot using the color Red:

```
Plot1(Open);
SetPlotColor(1,Red);
```

RGB

Returns an RGB color number that corresponds to the specified combination of red, green, and blue component color values; the number ranges from 0 to 16777215 and represents one of 16 M colors.

Usage

`RGB` (*Red*,*Green*,*Blue*)

Where: *Red* a numerical value from 0 to 255, specifying the Red component of the RGB color
Green a numerical value from 0 to 255, specifying the Green component of the RGB color
Blue a numerical value from 0 to 255, specifying the Blue component of the RGB color

Example

Create an Orange color plot using the RGB color combination:

```
Plot1(Open);
SetPlotColor(1,RGB(224,160,32));
```

RGBToLegacyColor

Returns the legacy color value that most closely matches the specified RGB color number; the value ranges from 0 to 16.

Usage

`RGBToLegacyColor(`*RGBColorValue*`)`

Where: *RGBColorValue* - an RGB color number from 0 to 16777215

Example

Assign a value, representing the color 65280 (Green) under the RGB color scheme, to Value1 (Value1 will be assigned a value of 4):

```
Value1=RGBToLegacyColor(65280);
```

White

Constant, used to designate the color White.

Usage

```
White
```

Notes

The base 17 colors can be designated by constants.

Example

Create a plot using the color White:

```
Plot1(Open);
SetPlotColor(1,White);
```

Yellow

Constant, used to designate the color Yellow.

Usage

```
Yellow
```

Notes

The base 17 colors can be designated by constants.

Example

Create a plot using the color Yellow:

```
Plot1(Open);
SetPlotColor(1,Yellow);
```

Above

Used in combination with `Cross` to specify the direction of the cross.

`Above` specifies an upward (lesser to a greater value) direction.

The plot of A is defined as having crossed `Above` the plot of B if the value of A is greater than the value of B and one of the following is true:

a) The value of A was less than the value of B on the bar immediately preceding the current bar

or

b) The values of A and B were equal for a consecutive sequence of one or more bars immediately preceding the current bar and the value of A was less than the value of B on the bar immediately preceding this sequence of bars.

For more information see `Cross`.

Usage

```
E1 Cross Above E2
```

or:

```
Plot1 Cross Above Plot2
```

Where: *`E`* - a numerical expression

Example

Trigger an alert on the bar where the Close price crosses above 1350.00:

```
Plot1(Close);
If Plot1 Cross Above 1350.50 Then
Alert("Price has crossed above 1350.00");
```

And

A logical (Boolean) operator that returns `True` only if both of its operands are true. Logical operators are used in logical (Boolean) expressions that operate with true/false values.

Usage

`E1 And E2`

Where: `E` - true/false expressions

Example

`2=1 And 2=2` will return a value of `False`

`True And True And True` will return a value of `True`

Begin

Used in combination with `End` to group instructions for conditional execution; a `Begin` must always be followed by an `End`.

`Begin` and `End` can be used with `Then`, `Else`, `For`, and `While` conditional statements.

`Begin` should not be followed by a semicolon (`;`), code lines within an instruction group should end with a semicolon (`;`), and only the last instance of `End` within the same conditional execution statement should be followed by a semicolon (`;`).

Usage

```
CS Begin I1;
I2;
I3;
End;
```

Where: *`CS`* - conditional statement
`I` - conditional instructions

Example

If UpTrend is true then buy, otherwise sell short:

```
If UpTrend Then Begin
Buy Next Bar Market;
End
Else Begin
SellShort Next Bar Market;
End;
```

Below

Used in combination with `Cross` to specify the direction of the cross.

`Below` specifies a downward (greater to a lesser value) direction.

The plot of A is defined as having crossed `Below` the plot of B if the value of A is less than the value of B and one of the following is true:

a) The value of A was more than the value of B on the bar immediately preceding the current bar,

or

b) The values of A and B were equal for a consecutive sequence of one or more bars immediately preceding the current bar and the value of A was more than the value of B on the bar immediately preceding this sequence of bars.

For more information see `Cross`.

Usage

```
E1 Cross Below E2
```

or:

```
Plot1 Cross Below Plot2
```

Where: *`E`* - a numerical expression

Example

Trigger an alert on the bar where the Close price crosses below 1350.00:

```
Plot1(Close);
If Plot1 Cross Below 1350.50 Then
Alert("Price has crossed below 1350.00");
```

Break

Breaks the loop execution.

Usage

```
Break;
```

Example

```
For value1 = 0 to 10 Begin
If ( close[value1] = open[value1] ) then Break;
End;
If value1 <= 10 then print("Open = Close ", value1:0:0, " bars ago.");
```

case

Part of switch-case operator.

Usage

```
See in example
```

Example

```
switch getplotcolor (1) begin case green:
print("Indicator in green zone on bar ", currentbar);
case red:
print("Indicator in red zone on bar ", currentbar);
case red:
print("Indicator is neutral on bar ", currentbar);
end;
```

Cross

Returns a value of `True` if, on the current bar, the plot of a numerical expression A crossed the plot of a numerical expression B in the specified direction.

`Above`, `Over`, `Below`, or `Under` parameters specify the direction of the cross; `Above` and `Over` are transposable and specify an upward (lesser to a greater value) direction, and `Below` and `Under` are transposable and specify a downward (greater to a lesser value) direction.

The plot of A is defined as having crossed `Above` or `Over` the plot of B if the value of A is greater than the value of B and one of the following is true:

a) The value of A was less than the value of B on the bar immediately preceding the current bar

or

b) The values of A and B were equal for a consecutive sequence of one or more bars immediately preceding the current bar and the value of A was less than the value of B on the bar immediately preceding this sequence of bars.

The plot of A is defined as having crossed `Below` or `Under` the plot of B if the value of A is less than the value of B and one of the following is true:

a) The value of A was more than the value of B on the bar immediately preceding the current bar,

or

b) The values of A and B were equal for a consecutive sequence of one or more bars immediately preceding the current bar and the value of A was more than the value of B on the bar immediately preceding this sequence of bars.

Usage

`*E1* Cross *Direction* *E2*`

or:

```
Plot1 Cross Direction Plot2
```

Where: `E` - a numerical expression
`Direction` - a required parameter; specifies the direction of the cross

Example

Trigger an alert on the bar where the Close price crosses above 1350.00:

```
Plot1(Close);
If Plot1 Cross Above 1350.50 Then
Alert("Price has crossed above 1350.00");
```

Crosses

Same as `Cross`

DownTo

Used in combination with `For` to form a loop statement that will execute a set of instructions repeatedly until the loop count reaches the specified final value.

`DownTo` specifies that the value of the counter variable is to be decreased by one on the completion of each loop.

For more information see `For`.

Usage

```
For Counter=IValue DownTo FValue Begin I1;
I2;
End;
```

Where: *`Counter`* - a numerical variable used store the loop count
`IValue` - a numerical expression specifying the initial counter value
`FValue` - a numerical expression specifying the final counter value

Example

Add the high prices of the last 10 bars to the `HighPriceSum` variable:

```
For BarBackNo=9 DownTo 0 Begin
HighPriceSum=HighPriceSum+High[BarBackNo];
End;
```

Else

Used in combination with `If` and `Then` to form a conditional statement that executes specific instructions if a logical expression is false.

The conditional execution statement must contain both `If` and `Then` in addition to `Else`.

For more information see `If`.

Usage

`If` *`E`* `Then` *`I1`* `Else` *`I2`*

Where: *`E`* - a true/false expression
`I` - conditional instructions

Example

If `UpTrend` is true then buy and if `UpTrend` is false then sell short:

```
If UpTrend Then Buy Next Bar Market Else SellShort Next Bar Market;
```

End

Used in combination with `Begin` to group instructions for conditional execution; an `End` must always be preceded by a `Begin`. `Begin` and `End` can be used with `Then`, `Else`, `For`, and `While` conditional statements.

Only the last instance of `End` within the same conditional execution statement should be followed by a semicolon (`;`).

Usage

```
CS Begin I1;
I2;
I3;
End;
```

Where: *`CS`* - conditional statement
`I` - conditional instructions

Example

If `UpTrend` is true then buy, otherwise sell short:

```
If UpTrend Then Begin
Buy Next Bar Market;
End
Else Begin
SellShort Next Bar Market;
End;
```

False

A logical (Boolean) value. Logical values are used in logical (Boolean) expressions and for true/false inputs.

Usage

`False`

Example

`False And True` will return a value of `False`

`2=1` will return a value of `False`

Declare `LogicalVar` as a true/false variable with the initial value of false:

```
Variable:LogicalVar(False);
```

Declare `Overnight` as a true/false input with the default value of false:

```
Input:Overnight(False);
```

For

Used in combination with `To` or `DownTo` to form a loop statement that will execute a set of instructions repeatedly until the loop count reaches the specified final value.

The loop statement specifies a numerical variable that holds the loop count, as well as initial and final counter values. `To` specifies that the value of the counter variable is to be increased by one on the completion of each loop, while `DownTo` specifies that the value of the counter variable is to be decreased by one on the completion of each loop.

The use of `Begin` and `End` statements is required to group the instructions for execution within the loop; a `Begin` must always be followed by an `End`.

`Begin` should not be followed by a semicolon (`;`), code lines within an instruction group should end with a semicolon (`;`), and `End` should be followed by a semicolon (`;`).

Usage

```
For Counter=IValue To FValue Begin I1;
I2;
End;
```

or:

```
For Counter=IValue DownTo FValue Begin
I1;
I2;
End;
```

Where: *`Counter`* - a numerical variable used store the loop count
`IValue` - a numerical expression specifying the initial counter value
`FValue` - a numerical expression specifying the final counter value

Example

Add the high prices of the last 10 bars to the `HighPriceSum` variable:

```
For BarBackNo=0 To 9 Begin
```

```
HighPriceSum=HighPriceSum+High[BarBackNo];
End;
```

Add the high prices of the last 10 bars to the `HighPriceSum` variable:

```
For BarBackNo=9 DownTo 0 Begin
HighPriceSum=HighPriceSum+High[BarBackNo];
End;
```

If

Used in combination with `Then` to form a conditional statement that executes specific instructions if a logical expression is true, and with `Else` to form a conditional statement that executes specific instructions if a logical expression is false.

The conditional execution statement must contain both `If` and `Then`; `Else` is optional.

`Begin` and `End` statements are used to group instructions for conditional execution; a `Begin` must always be followed by an `End`.

`Begin` should not be followed by a semicolon (`;`), code lines within an instruction group should end with a semicolon (`;`), and only the last instance of `End` within the same conditional execution statement should be followed by a semicolon (`;`).

Usage

```
If E Then I1 Else I2
```

or:

```
If E Then Begin I1;
I2;
End
Else Begin
I3;
I4;
End;
```

Where: *E* - a true/false expression
I - conditional instructions

Example

If `UpTrend` is false then sell:

```
If UpTrend=False Then Sell Next Bar Market;
```

If `UpTrend` is true then buy, otherwise sell short:

```
If UpTrend Then Buy Next Bar Market Else SellShort Next Bar Market;
```

If `UpTrend` is true then buy, otherwise sell short:

```
If UpTrend Then Begin
Buy Next Bar Market;
End
Else Begin
SellShort Next Bar Market;
End;
```

Not

Used in True/False statements: **negative**

Example

```
Condition1 = True;

Condition2 = Not Condition1;
```

Assigns to `Condition2` value opposite to `Condition1`.

Or

A logical (Boolean) operator that returns `True` if one or both of its operands are true. Logical operators are used in logical (Boolean) expressions that operate with true/false values.

Usage

`E1 Or E2`

Where: `E` - true/false expressions

Example

`2=1 Or 2>2` will return a value of `False`

`True Or False Or False` will return a value of `True`

Over

Same as Above

switch

Part of switch-case operator.

Usage

```
See in example
```

Example

```
switch getplotcolor (1) begin case green:
print("Indicator in green zone on bar ", currentbar);
case red:
print("Indicator in red zone on bar ", currentbar);
case red:
print("Indicator is neutral on bar ", currentbar);
end;
```

Then

Used in combination with `If` to form a conditional statement that executes specific instructions if a logical expression is true.

For more information see `If`.

Usage

`If` *`E`* `Then` *`I`*

Where: *`E`* - a true/false expression
`I` - conditional instructions

Example

If `UpTrend` is false then sell:

```
If UpTrend=False Then Sell Next Bar Market;
```

To

Used in combination with `For` to form a loop statement that will execute a set of instructions repeatedly until the loop count reaches the specified final value.

`To` specifies that the value of the counter variable is to be increased by one on the completion of each loop.

For more information see `For`.

Usage

```
For Counter=IValue To FValue Begin I1;
I2;
End;
```

Where: *`Counter`* - a numerical variable used store the loop count
`IValue` - a numerical expression specifying the initial counter value
`FValue` - a numerical expression specifying the final counter value

Example

Add the high prices of the last 10 bars to the `HighPriceSum` variable:

```
For BarBackNo=0 To 9 Begin
HighPriceSum=HighPriceSum+High[BarBackNo];
End;
```

True

A logical (Boolean) value. Logical values are used in logical (Boolean) expressions and for true/false inputs.

Usage

`True`

Example

`True Or False` will return a value of `True`

`2=2` will return a value of `True`

Declare `LogicalVar` as a true/false variable with the initial value of true:

```
Variable:LogicalVar(True);
```

Declare `Overnight` as a true/false input with the default value of true:

```
Input:Overnight(True);
```

Under

Same as `Below`

While

Used in combination with `Begin` and `End` to form a conditional loop statement that will execute a set of instructions repeatedly as long as a logical expression is true. If the logical expression is not true, the instructions will not be executed.

`Begin` and `End` statements are used to group instructions for conditional execution; a `Begin` must always be followed by an `End`.

`Begin` should not be followed by a semicolon (`;`), code lines within an instruction group should end with a semicolon (`;`), and `End` should be followed by a semicolon (`;`).

Usage

```
While E Begin I1;
I2;
I3;
End;
```

Where: *E* - a true/false expression
I - conditional instructions

Example

Add the high prices of the last 10 bars to the `HighPriceSum` variable:

```
BarBackNo=0;
While BarBackNo<10 Begin
HighPriceSum=HighPriceSum+High[BarBackNo];
BarBackNo=BarBackNo+1;
End;
```

AUD

Constant used to designate the currency "Australian Dollar".

Usage

```
AUD
```

Notes

The base 14 currencies can be designated by constants.

Example

Checks if the currency of the symbol used for the calculation is "Australian Dollar".

```
condition1 = SymbolCurrencyCode = AUD;
```

CAD

Constant used to designate the currency "Canadian Dollar".

Usage

```
CAD
```

Notes

The base 14 currencies can be designated by constants.

Example

Checks if the currency of the symbol used for the calculation is "Canadian Dollar".

```
condition1 = SymbolCurrencyCode = CAD;
```

CHF

Constant used to designate the currency "Swiss Franc".

Usage

```
CHF
```

Notes

The base 14 currencies can be designated by constants.

Example

Checks if the currency of the symbol used for the calculation is "Swiss Franc".

```
condition1 = SymbolCurrencyCode = CHF;
```

Convert_Currency

Returns the SrcMoney sum, set in the SrcCurrency currency, in the DstCurrency currency using the cross-rate at the DateTime date and time.

Usage

`Convert_Currency(DateTime, SrcCurrency, DstCurrency, SrcMoney);`

Where:

`DateTime` - date and time set in the DateTime format

`SrcCurrency` - initial currency used for conversion

`DstCurrency` - currency to which the sum will be converted to

`SrcMoney` - funds in initial currency (SrcCurrency) to be converted to DstCurrency

Notes

The base 14 currencies can be designated by constants.

Example

`Convert_Currency(39448.25000000, GBP, USD, 12.34);`
will return `24.48`, which means that: on `1/1/2008`: `12.34 GBP` was equal to `24.48 USD`.

EUR

Constant used to designate the currency "Euro".

Usage

```
EUR
```

Notes

The base 14 currencies can be designated by constants.

Example

Checks if the currency of the symbol used for the calculation is "Euro".

```
condition1 = SymbolCurrencyCode = EUR;
```

GBP

Constant used to designate the currency "British Pound".

Usage

```
GBP
```

Notes

The base 14 currencies can be designated by constants.

Example

Checks if the currency of the symbol used for the calculation is "British Pound".

```
condition1 = SymbolCurrencyCode = GBP;
```

HKD

Constant used to designate the currency "Hong Kong Dollar".

Usage

```
HKD
```

Notes

The base 14 currencies can be designated by constants.

Example

Checks if the currency of the symbol used for the calculation is "Hong Kong Dollar".

```
condition1 = SymbolCurrencyCode = HKD;
```

JPY

Constant used to designate the currency "Japanese Yen".

Usage

```
JPY
```

Notes

The base 14 currencies can be designated by constants.

Example

Checks if the currency of the symbol used for the calculation is "Japanese Yen".

```
condition1 = SymbolCurrencyCode = JPY;
```

NOK

Constant used to designate the currency "Norwegian Krone".

Usage

```
NOK
```

Notes

The base 14 currencies can be designated by constants.

Example

Checks if the currency of the symbol used for the calculation is "Norwegian Krone".

```
condition1 = SymbolCurrencyCode = NOK;
```

None

Constant used to designate that currency was not indicated.

Usage

```
None
```

Notes

The base 14 currencies can be designated by constants.

Example

Checks if the currency of the symbol used for the calculation is not indicated in QuoteManager settings.

```
condition1 = SymbolCurrencyCode = None;
```

NZD

Constant used to designate the currency "New Zealand Dollar".

Usage

```
NZD
```

Notes

The base 14 currencies can be designated by constants.

Example

Checks if the currency of the symbol used for the calculation is "New Zealand Dollar".

```
condition1 = SymbolCurrencyCode = NZD;
```

Portfolio_CurrencyCode

Returns currency code from Portfolio settings (View -> Portfolio Settings)

Usage

```
Portfolio_CurrencyCode
```

Notes

The base 14 currencies can be designated by constants.

Example

Checks if the currency of portfolio account used for calculation is US Dollar.

```
condition1 = Portfolio_CurrencyCode = USD;
```

SEK

Constant used to designate the currency "Swedish Krona".

Usage

```
SEK
```

Notes

The base 14 currencies can be designated by constants.

Example

Checks if the currency of the symbol used for the calculation is "Swedish Krona".

```
condition1 = SymbolCurrencyCode = SEK;
```

SGD

Constant used to designate the currency "Singapore Dollar".

Usage

```
SGD
```

Notes

The base 14 currencies can be designated by constants.

Example

Checks if the currency of the symbol used for the calculation is "Singapore Dollar".

```
condition1 = SymbolCurrencyCode = SGD;
```

SymbolCurrencyCode

Returns currency code from Quote Manager symbol settings (Edit Symbol... -> Settings -> Currency)

Usage

```
SymbolCurrencyCode
```

Notes

The base 14 currencies can be designated by constants.

Example

Checks if the currency of the symbol used for calculation is US Dollar.

```
condition1 = SymbolCurrencyCode = USD;
```

TRY_

Constant used to designate the currency "Turkish Lira".

Usage

```
TRY_
```

Notes

The base 14 currencies can be designated by constants.

Example

Checks if the currency of the symbol used for the calculation is "Turkish Lira".

```
condition1 = SymbolCurrencyCode = TRY_;
```

USD

Constant used to designate the currency "US Dollar".

Usage

```
USD
```

Notes

The base 14 currencies can be designated by constants.

Example

Checks if the currency of the symbol used for the calculation is "US Dollar".

```
condition1 = SymbolCurrencyCode = USD;
```

ZAR

Constant used to designate the currency "South African Rand".

Usage

```
ZAR
```

Notes

The base 14 currencies can be designated by constants.

Example

Checks if the currency of the symbol used for the calculation is "South African Rand".

```
condition1 = SymbolCurrencyCode = ZAR;
```

Ago

Used in combination with `Bar` or `Bars` and a numerical expression to reference the bar a specified number of bars back from the current bar.

Bars Ago can also be specified by using the bar offset notation that consists of a numerical expression enclosed in square brackets.

Usage

```
N Bars Ago
```

or:

```
[N]
```

Where: *N* - a numerical expression specifying the number of bars back to reference

Example

Plot the closing price of the previous bar:

```
Plot1(Close Of 1 Bar Ago, "Previous bar's close");
```

Plot the closing price of two bars ago:

```
Plot1(Close[2], "Close 2 bars ago");
```

Bar

Used in combination with `This`, `Next`, or `Ago` to reference a specific bar.

Usage

```
Bar
```

Example

`Close Of 1 Bar Ago` will return the closing price of the previous bar

Buy a user-set number of shares on close of this bar:

```
Buy This Bar On Close;
```

Buy a user-set number of shares on open of next bar:

```
Buy Next Bar On Open;
```

BarInterval

Returns a numerical value, indicating the number of resolution units (bar interval) of the data series that the study is applied to.

Returns the number of Ticks, Contracts, Points, Changes, Seconds, Minutes, Hours, Days, Weeks, Months, Quarters, or Years, depending on the chart resolution; a value of 5 will be returned for a 5-second as well as for a 5-tick chart.

Usage

```
BarInterval
```

Example

Assign a value, indicating the number of resolution units (bar interval) of the data series that the study is applied to, to Value1 variable:

```
Value1=BarInterval;
```

Bars

Same as `Bar`

BarStatus

Returns a numerical value, indicating the status of the most recent tick in the current bar of the specified data series.

A value of 0 indicates that the tick is the opening tick of the bar, 1 indicates that the tick is within the bar, and 2 indicates that the tick is the closing tick of the bar.

Usage

`BarStatus`(*DataNum*)

Where: *DataNum* - a numerical expression specifying the data number of the series

If *DataNum* is not specified, a value for the current data series will be returned.

Example

`BarStatus(1)` will return a value of 2 if the current tick in the data series with the data number 1 is the closing tick of a bar

BarType

Returns a numerical value, indicating the resolution units of the data series that the study is applied to.

Usage

```
BarType
```

The following values are returned for each type of resolution units:

0	Ticks (Ticks & Contracts)
1	Intra-Day (Seconds, Minutes, & Hours)
2	Days
3	Weeks
4	Months, Quarters, & Years
5	Points, Changes, Point & Figure
6	(Reserved for future use)
7	(Reserved for future use)
8	Kagi
9	(Reserved for future use)
10	Line Break
11	(Reserved for future use)
12	(Reserved for future use)
13	Renko
256 -	Heikin Ashi

Example

Assign a value, indicating the resolution units of the data series that the study is applied to, to Value1 variable:

```
Value1=BarType;
```

BarType_ex

An extended version of `BarType`. Indicates the resolution units more specifically.

Returns a numerical value, indicating the resolution units of the data series that the study is applied to.

Usage

`BarType_ex`
The following values are returned for each type of resolution units:

1 Ticks
2 Minutes
3 Hours
4 Days
5 Weeks
6 Months
7 Years
8 Volume
9 Seconds
10 Quarters
11 Points
12 Change
13 Points (Original)
14 - Point & Figure
15 - Kagi
16 - Line Break
17 - Renko
18 - Heikin Ashi

Example

Assign a value, indicating the resolution units of the data series that the study is applied to, to Value1 variable:

```
Value1=BarType_ex;
```

BigPointValue

Returns a numerical value, indicating the currency value of a single whole unit price change for the data series that the study is applied to.

Usage

```
BigPointValue
```

Notes

`BigPointValue` = `PointValue` × `PriceScale`

Example

`BigPointValue` will return 1 for Google

`BigPointValue` will return 50 for E-mini S&P; 500

BoxSize

Returns the price-based interval setting associated with the specified price-based chart type an indicator or signal is applied to.

This value is set in the **Chart Type** section of the **Settings** tab within the **Format Instrument** dialog for a chart.

Usage

```
BoxSize
```

Parameters

Chart Type	**Value Returned**
Point & Figure	Box Size value
Point	Point value
Renko	Box Size value

Example

`Boxsize` will return a value of 3 for Renko chart with the Box Size equal 3.

`Boxsize` will return a value of 0 for Kagi or regular resolutions charts.

C

Same as `Close`

Call

Constant used to designate the Call option instrument.

Usage

```
Call
```

Notes

The Put options could be designated by the `Put` constant.

Example

Checks if the symbol used for the calculation is a Call option.

```
condition1 = OptionType = Call;
```

Category

Returns a numerical value, indicating the category (financial instrument type) of the symbol that study is applied to.

Usage

```
Category
```

The following values are returned for each category:

0	Future
1	Future Option
2	Stock
3	Stock Option
4	Index
5	Currency Option
6	Mutual Fund
7	Money Market Fund
8	Index Option
9	Cash
10	Bond
11	Spread
12	Forex
14	Composite

Example

Assign a value, indicating the type of symbol that the study is applied to, to Value1 variable:

```
Value1=Category;
```

Close

Returns the closing price.

Usage

```
Close
```

Note

The range of returned values is limited by the MaxBarsBack setting.

Example

Plot the closing price of the current bar:

```
Plot1(Close,"Close");
```

Plot the closing price of the previous bar:

```
Plot1(Close Of 1 Bar Ago, "Previous bar's close");
```

Plot the closing price of two bars ago:

```
Plot1(Close[2], "Close 2 bars ago");
```

CurrentBar

Returns the number of the current bar.

Each bar, subsequent to the initial number of bars specified by the **Maximum Bars Back** setting, is assigned a sequential number; the initial bars specified by the setting are not numbered.

For example, if **Maximum Bars Back** is set to 20, the 21st bar will be assigned a number of 1.

Usage

```
CurrentBar
```

Example

`CurrentBar` will return the number of the current bar

D

Same as `Date`

DailyLimit

Retained for backward compatibility.

Data

Used to specify a particular data series in a multi-symbol chart; each data series in a multi-symbol chart has a unique Data Number.

Usage

`DataN`

Where: `N` - the Data Number of the data series

Or:

`Data(`*N*`)`

Where: *`N`* - a numerical expression specifying the Data Number of the data series

Example

`High Of Data2` will return the high price of a bar in the data series with the Data Number of 2

`High Of Data(2)` will return the high price of a bar in the data series with the Data Number of 2

DataCompression

Same as `BarType`

Date

Returns a numerical value indicating the closing date of a bar. The date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month.

Note

The range of returned values is limited by the MaxBarsBack setting.

Usage

`Date`

Example

`Date` will return a value of 1071030 for October 30th, 2007

`Date` will return a value of 990402 for April 2th, 1999

DateTime

The integer portion of the DateTime value indicates the number of days that have elapsed since January 1st, 1900, and the fractional portion of the DateTime value indicates the fraction of the day that has passed since midnight.

DateTime is a double-precision decimal value. It allows accessing millisecond time stamps of the bar.

Note

The range of returned values is limited by the MaxBarsBack setting.

Usage

```
DateTime
```

Example

`DateTime` will return 41422.74 for 5/28/2013 08:41:11.813

DateTime bar update

Returns time of the last update of the current bar if Bar Magnifier mode is on.

If Bar Magnifier mode is off, returns the time of the current bar (as the Time property).

Usage

```
datetime_bar_update(data_stream)
```

Example

Bar Magnifier mode is on. Time and date of the last update of the current bar are 29.04.2013 10:53:59.154

`print(datetime_bar_update:15:8)` will return 41393.45415687.

`print(datetimetostring_ms(datetime_bar_update))` will return "29.04.2013 10:53:59.154".

Day

Retained for backward compatibility; replaced with `Bar`.

Days

Retained for backward compatibility; replaced with `Bar`.

DownTicks

Returns the total number of Down ticks for the current bar if **Build Volume On** is set to **Tick Count**.

Returns the total Down volume for the current bar if **Build Volume On** is set to **Trade Volume**.

A down tick is a tick with the price lower then the preceding tick, and down volume is the volume traded on down ticks.

With **Build Volume On** is set to **Tick Count**:

- the value of 1 will be returned for 1-tick charts
- the total number of Down ticks in the current bar will be returned for multi-tick, volume, and time-based charts

With **Build Volume On** is set to **Trade Volume**:

- the Down volume of the current tick will be returned for 1-tick charts
- the total Down volume of the current bar will be returned for multi-tick, volume, and time-based charts

Please note that most data feeds provide only a limited history of tick and volume data; storing real-time feed data will ensure the availability of historical tick and volume data.

Note

The range of returned values is limited by the MaxBarsBack setting.

Usage

```
DownTicks
```

Example

Plot the number of Down ticks in the current bar (**Build Volume On** is set to **Tick Count**):

```
Plot1(DownTicks,"Down Ticks");
```

Plot the Down volume of the current bar (**Build Volume On** is set to **Trade Volume**):

```
Plot1(DownTicks,"Down Volume");
```

ExpirationDate

Returns a numerical value, indicating the expiration date of the financial instrument the study is applied to. The date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month.

For example, the value returned for the date of October 30th, 2008 will be 1081030, and for April 2nd, 1999 will be 990402.

Usage

```
ExpirationDate
```

A valid expiration date will be returned for futures or options only.

Example

`ExpirationDate` will return a value of 1081030 for October 30th, 2008

`ExpirationDate` will return a value of 990402 for April 2nd, 1999

ExpirationDateFromVendor

Returns a numerical value, indicating the expiration date of the financial instrument the study is applied to. The date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month.

For example, the value returned for the date of October 30th, 2008 will be 1081030, and for April 2nd, 1999 will be 990402.

Usage

`ExpirationDateFromVendor`

A valid expiration date will be returned for futures or options only.

Example

`ExpirationDateFromVendor` will return a value of 1081030 for October 30th, 2008

`ExpirationDateFromVendor` will return a value of 990402 for April 2nd, 1999

GetExchangeName

Returns a string expression containing the exchange name for the symbol that the study is applied to.

Usage

```
GetExchangeName
```

Example

`GetExchangeName` will return "NASD" for Google

`GetExchangeName` will return "CME" for E-mini S&P; 500

GetRTSymbolName

Returns a string expression containing the name of the real-time symbol that the study is applied to in case the merging option is enabled. If the merging option is disabled returns the same value as the Name property. In case of a custom futures instrument the name of the last contract is returned.

Usage

```
GetRTSymbolName
```

Example

`GetRTSymbolName` will return "GOOG" for Google if the merging option is enabled and Google is configured as the real-time instrument.

GetSymbolName

Returns a string expression containing the name of the symbol that the study is applied to.

Usage

```
GetSymbolName
```

Example

`GetSymbolName` will return "GOOG" for Google

H

Same as `High`

High

Returns the high price.

Usage

```
High
```

Note

The range of returned values is limited by the MaxBarsBack setting.

Example

Plot the high price of the current bar:

```
Plot1(High,"High");
```

Plot the high price of the previous bar:

```
Plot1(High Of 1 Bar Ago,"Previous bar's high");
```

Plot the high price of two bars ago:

```
Plot1(High[2],"High 2 Bars ago");
```

I

Same as `OpenInt`

IntervalType

Returns a numerical value, indicating the resolution intervals of the data series that the study is applied to.

Usage

```
IntervalType
```

The following values are returned for each type of resolution units:

0	- Ticks (Ticks & Contracts)
1	- Intra-Day (Seconds, Minutes, & Hours)
2	- Days
3	- Weeks
4	- Months, Quarters, Years
5	- Points, Changes
0 - 4	- Point & Figure
0 - 4	- Kagi
0 - 4	- Line Break
0	- Renko

Example

To assign a value, indicating the resolution units of the data series the study is applied to, to Value1 variable:

```
Value1 = IntervalType;
```

IntervalType_ex

An extended version of `IntervalType`.

Indicates the resolution intervals more specifically.

Returns a numerical value, indicating the resolution intervals of the data series that the study is applied to.

Usage

```
IntervalType_ex
```

The following values are returned for each type of resolution units:

1	- Ticks
2	- Minutes
3	- Hours
4	- Days
5	- Weeks
6	- Months
7	- Years
8	- Volume
9	- Seconds
10	- Quarters
11	- Points
12	- Change
13	- Points (original)
1 - 7, 9, 10	- Point & Figure
1 - 7, 9, 10	- Kagi
1 - 7, 9, 10	- Line Break
1	- Renko

Example

To assign a value, indicating the resolution units of the data series the study is applied to, to Value1 variable:

```
Value1 = IntervalType_ex;
```

L

Same as `Low`

Low

Returns the low price.

Note

The range of returned values is limited by the MaxBarsBack setting.

Usage

```
Low
```

Example

Plot the low price of the current bar:

```
Plot1(Low,"Low");
```

Plot the low price of the previous bar:

```
Plot1(Low Of 1 Bar Ago,"Previous bar's low");
```

Plot the low price of two bars ago:

```
Plot1(Low[2],"Low 2 bars ago");
```

MinMove

Returns a numerical value, indicating the minimum fractional unit price change for the data series that the study is applied to.

Usage

```
MinMove
```

Example

`MinMove` will return 1 for Google

`MinMove` will return 25 for E-mini S&P; 500

`MinMove*PointValue` will return the currency value, corresponding to the minimum price move of a share or contract

Next

Used in combination with `Bar` to reference the next bar.

Usage

```
Next Bar
```

Example

Buy a user-set number of shares at Market price on open of next bar:

```
Buy Next Bar At Market;
```

O

Same as `Open`

Open

Returns a numerical value indicating the open open price.

Note

The range of returned values is limited by the MaxBarsBack setting.

Usage

```
Open
```

Example

Plot the open price of the current bar:

```
Plot1(Open,"Open");
```

Plot the open price of the previous bar:

```
Plot1(Open Of 1 Bar Ago, "Previous bar's open");
```

Plot the open price of two bars ago:

```
Plot1(Open[2],"Open 2 bars ago");
```

OpenInt

Returns the open interest of the current bar for time-based charts with resolutions of 1 day or more.

For tick and volume-based charts and for time-based charts with resolutions of 24 hours or less, OpenInt returns:

- the volume traded on Down ticks will be returned if **Build Volume On** is set to **Trade Volume**
- the number of Down ticks in the current bar will be returned if **Build Volume On** is set to **Tick Count**

Usage

```
OpenInt
```

Notes

The range of returned values is limited by the MaxBarsBack setting.

OpenInt is supported for time-based charts with resolutions of 1 day or more.

Most data feeds provide only a limited history of volume and tick data; storing real-time feed data will ensure the availability of historical volume and tick data.

Example

Plot the open interest of the current bar:

```
Plot1(OpenInt,"Open interest");
```

Plot the open interest of the previous bar:

```
Plot1(OpenInt of 1 Bar Ago,"Previous bar's open interest");
```

Plot the open interest of two bars ago:

```
Plot1(OpenInt[2],"Open interest 2 bars ago");
```

OptionType

Returns the numerical value, indicating the `Put` or `Call` value for the data series that the study is applied to. If the instrument is not an option contract the keyword returns 0.

Usage

```
OptionType
```

Example

`OptionType` will return 0 for Google, since it is not an option instrument

`OptionType` will return the value of the `Put` constant for any Put option, and the value of the `Call` constant for any Call option

Point

Returns a decimal numerical value, equivalent to a single fractional price unit for the data series that the study is applied to.

Usage

```
Point
```

Notes

`Point` = 1/`PriceScale`

Example

`1 Point` will return 0.01 for Google

`8 Point` will return 0.08 for E-mini S&P; 500

`8 Point` will return 0.25 for T-Bond Futures

Points

Same as `Point`

PointValue

Returns a numerical value, indicating the currency value of a single fractional unit price change for the data series that the study is applied to.

Usage

`PointValue`

Notes

`PointValue` = `BigPointValue`/`PriceScale`

Example

`PointValue` will return 0.01 for Google

`PointValue` will return 0.50 for E-mini S&P; 500

PriceScale

Returns a numerical value, indicating the fractional unit equivalent of a single whole unit price change for the data series that the study is applied to.

Usage

`PriceScale`

Notes

`PriceScale` = `BigPointValue`/`PointValue`

Example

`PriceScale` will return 100 for Google

`PriceScale` will return 32 for T-Bond Futures

Put

Constant used to designate the Put option instrument.

Usage

```
Put
```

Notes

The Call options could be designated by the `Call` constant.

Example

Checks if the symbol used for the calculation is a Put option.

```
condition1 = OptionType = Put;
```

RevSize

Returns the Reversal of a Point & Figure chart, the Reversal of a Kagi chart, or the number of Line Breaks in a Line Break chart.

This value is set in the **Chart Type** section of the **Settings** tab within the **Format Instrument** dialog for a chart.

Usage

Notes

Example

`RevSize` returns 2 if the reversal size of a P&F; chart is set to 2.

`RevSize` returns 5 if the number of line breaks for a Line Break chart is set to 5.

`RevSize` returns 4 if the reversal size of a Kagi chart is set to 4%.

ScrollToBar

Centers the chart on the specified bar.

Usage

`ScrollToBar (int` *`DataN`*`, int` *`BarN`*`)`

Where: *`DataN`* - data series number

`BarN` - bar number

Example

```
ScrollToBar(2, 98);
```

Will center the chart on the 98th bar of the second data series.

SessionLastBar

Returns a logical value indicating whether the current bar is the last bar of the session; returns a value of `True` if the current bar is the last bar of the session, and a value of `False` if the current bar is not the last bar of the session.

Usage

```
SessionLastBar
```

Example

Assign a true/false value, indicating whether the current bar is the last bar on the chart, to LastBar variable:

```
Variable:LastBar(False);
LastBar=SessionLastBar;
```

Strike

Returns the numerical value, indicating the strike price for the data series that the study is applied to. If the instrument is not an option contract the keyword returns 0.

Usage

```
Strike
```

Example

`Strike` will return 0 for Google, since it is not an option instrument

`Strike` will return 65 for the January 2012 Put option for Johnson & Johnson (JNJ) at the strike price is $65 (JNJ120121P00065000)

Symbol_Close

Returns the closing price of the bar.

Usage

```
Symbol_Close
```

Note

The range of returned values is not limited by the MaxBarsBack setting. This keyword can return the value of any bar of the data series.

Example

Plot the closing price of the current bar:

```
Plot1(Symbol_Close, "Close");
```

Plot the closing price of the previous bar:

```
Plot1(Symbol_Close of 1 Bar Ago, "Previous Bar's Close");
```

Plot the closing price of two bars ago:

```
Plot1(Symbol_Close [2], "Close 2 bars ago");
```

Symbol_CurrentBar

Returns a numerical value indicating the number of the current bar plus **Maximum Bars Back** setting for the study.

Usage

```
Symbol_CurrentBar
```

Note

Each bar, subsequent to the initial number of bars specified by the **Maximum Bars Back** setting, is assigned a sequential number; the initial bars specified by the setting are not numbered.

For example, if **Maximum Bars Back** is set to 20, the 21st bar will be assigned a number of 1.

Example

```
Symbol_CurrentBar
```

Will return the number of the current bar plus **Maximum Bars Back** setting for the study.

Symbol_Date

Returns a numerical value indicating the closing date of a bar. The date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month.

Usage

```
Symbol_Date
```

Note

The range of returned values is not limited by the MaxBarsBack setting. This keyword can return the value of any bar of the data series.

Example

`Symbol_Date` will return a value of 1071030 for October 30th, 2007

`Symbol_Date` will return a value of 990402 for April 2th, 1999

Symbol_DownTicks

Returns the total number of Down ticks for the current bar if **Build Volume On** is set to **Tick Count**.

Returns the total Down volume for the current bar if **Build Volume On** is set to **Trade Volume**.

A down tick is a tick with the price lower than the preceding tick, and down volume is the volume traded on down ticks.

With **Build Volume On** is set to **Tick Count**:

- the value of 1 will be returned for 1-tick charts
- the total number of Down ticks in the current bar will be returned for multi-tick, volume, and time-based charts

With **Build Volume On** is set to **Trade Volume**:

- the Down volume of the current tick will be returned for 1-tick charts
- the total Down volume of the current bar will be returned for multi-tick, volume, and time-based charts

Usage

```
Symbol_DownTicks
```

Note

- Most data feeds provide only a limited history of tick and volume data; storing real-time feed data will ensure the availability of historical tick and volume data
- The range of returned values is not limited by the MaxBarsBack setting. This keyword can return the value of any bar of the data series.

Example

Plot the number of Down ticks in the current bar (**Build Volume On** is set to **Tick Count**):

```
Plot1(Symbol_DownTicks, "Down Ticks");
```

Plot the Down volume of the current bar (**Build Volume On** is set to **Trade Volume**):

```
Plot1(Symbol_DownTicks, "Down Volume");
```

Symbol_High

Returns the high price of the bar

Usage

```
Symbol_High
```

Note

The range of returned values is not limited by the MaxBarsBack setting. This keyword can return the value of any bar of the data series.

Example

Plot the high price of the current bar:

```
Plot1(Symbol_High, "High");
```

Plot the high price of the previous bar:

```
Plot1(Symbol_High of 1 Bar Ago, "Previous Bar's high");
```

Plot the high price of two bars ago:

```
Plot1(Symbol_High [2], "High 2 Bars Ago");
```

Symbol_Length

Returns a numerical value indicating the actual number of bars of a data series on chart.

Usage

```
Symbol_Length
```

Note

If the series is not updating in real-time (no new bars appear), the keyword returns the same value on each calculation. If the chart updates in real-time the keyword returns the number of bars available by the moment script references the keyword. It means that the keyword possibly can return different values even during calculation within the same bar.

Example

```
Plot1 (Symbol_Length, "Length of series");
```

- Plots a value of 500 if the number of bars of data series is 500 and there is no new bars appear
- Plots a value of 501 if the number of bars of data series becomes 501 because new bar appeared

Symbol_Low

Returns the low price of the bar

Usage

```
Symbol_Low
```

Note

The range of returned values is not limited by the MaxBarsBack setting. This keyword can return the value of any bar of the data series.

Example

Plot the low price of the current bar

```
Plot1 (Symbol_Low, "Low");
```

Plot the low price of the previous bar:

```
Plot1 (Symbol_Low of 1 Bar Ago, "Previous Bar's low");
```

Plot the low price of two bars ago:

```
Plot1 (Symbol_Low [2], "Low 2 bars ago");
```

Symbol_Open

Returns a numerical value indicating the open price of the bar.

Usage

```
Symbol_Open
```

Note

The range of returned values is not limited by the MaxBarsBack setting. This keyword can return the value of any bar of the data series.

Example

Plot the open price of the current bar:

```
Plot1(Symbol_Open, "Open");
```

Plot the open price of the previous bar:

```
Plot1(Symbol_Open of 1 Bar Ago, "Previous bar's open");
```

Plot the open price of two bars ago:

```
Plot1(Symbol_Open [2], "Open 2 bars ago");
```

Symbol_OpenInt

Returns the open interest of the current bar for tick and volume-based charts, and for time-based charts with resolutions of 24 hours or less:

- the volume traded on Down ticks will be returned if **Build Volume On** is set to **Trade Volume**
- the number of Down ticks in the current bar will be returned if **Build Volume On** is set to **Tick Count**

OpenInt is supported for time-based charts with resolutions of 1 day or more.

Usage

```
Symbol_OpenInt
```

Note

- Most data feeds provide only a limited history of volume and tick data; storing real-time feed data will ensure the availability of historical volume and tick data.
- The range of returned values is not limited by the MaxBarsBack setting. This keyword can return the value of any bar of the data series.

Example

Plot the open interest of the current bar:

```
Plot1 (Symbol_OpenInt, "Open Interest");
```

Plot the open interest of the previous bar:

```
Plot1 (Symbol_OpenInt of 1 Bar Ago, "Previous Bar's Open Interest");
```

Plot the open interest of two bars ago:

```
Plot1 (Symbol_OpenInt [2], "Open Interest 2 bars ago");
```

Symbol_TickID

This keyword can be used to distinguish between the bars with the same date and time stamps.

For tick and volume-based charts returns the tick index within a second. For resolutions higher than 1 Tick returns the index of the last tick within the bar.

For time-based charts with resolutions of 1 sec or more not supported. Returns 0.

Realtime ticks stored in the data base are being assigned the last 31 bit - the identifier of realtime affiliation. In order to get the Symbol_TickID value without realtime identifier one needs to calculate the remainder from Symbol_TickID value division by 2147483648.

```
value1=Mod(Symbol_TickID, 2147483648);
```

Usage

```
Symbol_TickID
```

Note

The range of returned values is not limited by the MaxBarsBack setting. This keyword can return the value of any bar of the data series.

Example

Plots the tick index (TickID) of the last tick within the current bar:

```
Plot1(Symbol_TickID,"SymTickID");
```

Symbol_Ticks

Returns the total number of ticks for the current bar if **Build Volume On** is set to **Tick Count**.

Returns the total volume for the current bar if **Build Volume On** is set to **Trade Volume**.

With **Build Volume On** is set to **Tick Count**:

- the value of 1 will be returned for 1-tick charts
- the total number of ticks in the current bar will be returned for multi-tick, volume, and time-based charts

With **Build Volume On** is set to **Trade Volume**:

- the volume of the current tick will be returned for 1-tick charts
- the total volume of the current bar will be returned for multi-tick, volume, and time-based charts

Usage

```
Symbol_Ticks
```

Note

- Most data feeds provide only a limited history of tick and volume data; storing real-time feed data will ensure the availability of historical tick and volume data
- The range of returned values is not limited by the MaxBarsBack setting. This keyword can return the value of any bar of the data series.

Example

Plot the number of ticks in the current bar (**Build Volume On** is set to **Tick Count**):

```
Plot1 (Symbol_Ticks, "Ticks");
```

Plot the volume of the current bar (**Build Volume On** is set to **Trade Volume**):

```
Plot1 (Symbol_Ticks, "Volume");
```

Symbol_Time

Returns a numerical value indicating the closing time of the current bar. The time is indicated in the 24-hour HHmm format, where 1300 = 1:00 PM.

Usage

```
Symbol_Time
```

Note

The range of returned values is not limited by the MaxBarsBack setting. This keyword can return the value of any bar of the data series.

Example

`Symbol_Time` will return a value of 1015 for 10:15 AM

`Symbol_Time` will return a value of 1545 for 3:45 PM

Symbol_Time_S

Returns a numerical value indicating the closing time, including seconds, of the current bar.

The time is indicated in the 24-hour HHmmss format, where 130000 = 1:00:00 PM

Usage

```
Symbol_Time_S
```

Note

The range of returned values is not limited by the MaxBarsBack setting. This keyword can return the value of any bar of the data series.

Example

`Symbol_Time_S` will return a value of 101525 for 10:15:25 AM

`Symbol_Time_S` will return a value of 154500 for 3:45:00 PM

Symbol_UpTicks

Returns the total number of Up ticks for the current bar if **Build Volume On** is set to **Tick Count**.

Returns the total Up volume for the current bar if **Build Volume On** is set to **Trade Volume**.

An up tick is a tick with the price higher then the preceding tick, and up volume is the volume traded on up ticks.

With **Build Volume On** is set to **Tick Count**:

- the value of 1 will be returned for 1-tick charts
- the total number of Up ticks in the current bar will be returned for multi-tick, volume, and time-based charts

With **Build Volume On** is set to **Trade Volume**:

- the Up volume of the current tick will be returned for 1-tick charts
- the total Up volume of the current bar will be returned for multi-tick, volume, and time-based charts

Usage

```
Symbol_UpTicks
```

Note

- Most data feeds provide only a limited history of tick and volume data; storing real-time feed data will ensure the availability of historical tick and volume data
- The range of returned values is not limited by the MaxBarsBack setting. This keyword can return the value of any bar of the data series.

Example

Plot the number of Up ticks in the current bar (**Build Volume On** is set to **Tick Count**):

```
Plot1 (Symbol_UpTicks, "Up Ticks");
```

Plot the Up volume of the current bar (**Build Volume On** is set to **Trade Volume**):

```
Plot1 (Symbol_UpTicks, "Up Volume");
```

Symbol_Volume

Returns the volume of the current bar.

For tick and volume-based charts, and time-based charts with resolutions of 24 hours or less:

- the volume traded on Up ticks will be returned if **Build Volume On** is set to **Trade Volume**
- the number of Up ticks in the current bar will be returned if **Build Volume On** is set to **Tick Count**

For time-based charts with resolutions of 1 day or more:

- the total volume traded will be returned if **Build Volume On** is set to **Trade Volume**
- the total number of ticks in the current bar will be returned if **Build Volume On** is set to **Tick Count**

Usage

```
Symbol_Volume
```

Note

- Most data feeds provide only a limited history of volume and tick data; storing real-time feed data will ensure the availability of historical volume and tick data.
- The range of returned values is not limited by the MaxBarsBack setting. This keyword can return the value of any bar of the data series.

Example

Plot the volume of the current bar

```
Plot1 (Symbol_Volume, "Volume");
```

Plot the volume of the previous bar:

```
Plot1 (Symbol_Volume of 1 Bar Ago, "Previous Bar's Volume");
```

Plot the volume of two bars ago:

```
Plot1(Symbol_Volume [2], "Volume 2 bars ago");
```

T

Same as `Time`

This

Used in combination with `Bar` to reference the current bar.

Usage

```
This Bar
```

Example

Buy a user-set number of shares on close of this bar:

```
Buy This Bar On Close;
```

TickID

This keyword can be used to distinguish between the bars with the same date and time stamps.

For tick and volume-based charts returns the tick index within a second. For resolutions higher than 1 Tick returns the index of the last tick within the bar.

For time-based charts with resolutions of 1 sec or more not supported. Returns 0.

Realtime ticks stored in the data base are being assigned the last 31 bit - the identifier of realtime affiliation. In order to get the TickID value without realtime identifier one needs to calculate the remainder from TickID value division by 2147483648.

```
value1=Mod(TickID, 2147483648);
```

Usage

```
TickID
```

Note

The range of returned values is limited by the MaxBarsBack setting.

Example

Plots the tick index (TickID) of the last tick within the current bar:

```
Plot1(TickID,"TickID");
```

Ticks

Returns the total number of ticks for the current bar if **Build Volume On** is set to **Tick Count**.

Returns the total volume for the current bar if **Build Volume On** is set to **Trade Volume**.

With **Build Volume On** is set to **Tick Count**:

- the value of 1 will be returned for 1-tick charts
- the total number of ticks in the current bar will be returned for multi-tick, volume, and time-based charts

With **Build Volume On** is set to **Trade Volume**:

- the volume of the current tick will be returned for 1-tick charts
- the total volume of the current bar will be returned for multi-tick, volume, and time-based charts

Please note that most data feeds provide only a limited history of tick and volume data; storing real-time feed data will ensure the availability of historical tick and volume data.

Usage

```
Ticks
```

Note

The range of returned values is limited by the MaxBarsBack setting.

Example

Plot the number of ticks in the current bar (**Build Volume On** is set to **Tick Count**):

```
Plot1(Ticks,"Ticks");
```

Plot the volume of the current bar (**Build Volume On** is set to **Trade Volume**):

```
Plot1(Ticks,"Volume");
```

Time

Returns a numerical value indicating the closing time of the current bar. The time is indicated in the 24-hour HHmm format, where 1300 = 1:00 PM.

Usage

```
Time
```

Note

The range of returned values is limited by the MaxBarsBack setting.

Example

`Time` will return a value of 1015 for 10:15 AM

`Time` will return a value of 1545 for 3:45 PM

Time_s

Returns a numerical value indicating the closing time, including seconds, of the current bar. The time is indicated in the 24-hour HHmmss format, where 130000 = 1:00:00 PM.

Usage

```
Time_s
```

Note

The range of returned values is limited by the MaxBarsBack setting.

Example

`Time_s` will return a value of 101525 for 10:15:25 AM

`Time_s` will return a value of 154500 for 3:45:00 PM

Today

Retained for backward compatibility; replaced with `This Bar`

UpTicks

Returns the total number of Up ticks for the current bar if **Build Volume On** is set to **Tick Count**.

Returns the total Up volume for the current bar if **Build Volume On** is set to **Trade Volume**.

An up tick is a tick with the price higher then the preceding tick, and up volume is the volume traded on up ticks.

With **build volume on** is set to **tick count**:

- the value of 1 will be returned for 1-tick charts
- the total number of Up ticks in the current bar will be returned for multi-tick, volume, and time-based charts

With **build volume on** is set to **trade volume**:

- the Up volume of the current tick will be returned for 1-tick charts
- the total Up volume of the current bar will be returned for multi-tick, volume, and time-based charts

Please note that most data feeds provide only a limited history of tick and volume data; storing real-time feed data will ensure the availability of historical tick and volume data.

Usage

```
UpTicks
```

Note

The range of returned values is limited by the MaxBarsBack setting.

Example

`Plot` the number of Up ticks in the current bar (**Build Volume On** is set to **Tick**

Count):

```
Plot1(UpTicks,"Up Ticks");
```

`Plot` the Up volume of the current bar (**Build Volume On** is set to **Trade Volume**):

```
Plot1(UpTicks,"Up Volume");
```

V

Same as `Volume`

Volume

Returns the volume of the current bar.

For tick and volume-based charts, and time-based charts with resolutions of 24 hours or less:

- the volume traded on Up ticks will be returned if **Build Volume On** is set to **Trade Volume**
- the number of Up ticks in the current bar will be returned if **Build Volume On** is set to **Tick Count**

For time-based charts with resolutions of 1 day or more:

- the total volume traded will be returned if **Build Volume On** is set to **Trade Volume**
- the total number of ticks in the current bar will be returned if **Build Volume On** is set to **Tick Count**

Please note that most data feeds provide only a limited history of volume and tick data; storing real-time feed data will ensure the availability of historical volume and tick data.

Usage

```
Volume
```

Note

The range of returned values is limited by the MaxBarsBack setting.

Example

Plot the volume of the current bar:

```
Plot1(Volume,"Volume");
```

Plot the volume of the previous bar:

```
Plot1(Volume Of 1 Bar Ago,"Previous bar's volume");
```

Plot the volume of two bars ago:

```
Plot1(Volume[2],"Volume 2 bars ago");
```

Yesterday

Retained for backward compatibility.

ComputerDateTime

Returns a double-precision decimal DateTime value indicating the computer's current date and time.

The integer portion of the DateTime value indicates the number of days that have elapsed since January 1st, 1900, and the fractional portion of the DateTime value indicates the fraction of the day that has passed since midnight.

Usage

```
ComputerDateTime
```

Example

`ComputerDateTime` will return a value of 39448.25000000 for 6:00 AM on January 1^{st}, 2008

CurrentDate

Returns a numerical value indicating the computer's current date. The date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month.

For example, the value returned for the date of October 30th, 2008 will be 1081030.

Usage

```
CurrentDate
```

Example

`CurrentDate` will return a value of 1081030 for October 30th, 2008

CurrentTime

Returns a numerical value, indicating the computer's current time. The time is indicated in the 24-hour HHmm format, where 1300 = 1:00 PM.

Usage

```
CurrentTime
```

Example

`CurrentTime` will return a value of 1015 for 10:15 AM

`CurrentTime` will return a value of 1545 for 3:45 PM

CurrentTime_s

Returns a numerical value indicating the computer's current time, including seconds. The time is indicated in the 24-hour HHmmss format, where 130000 = 1:00:00 PM.

Usage

```
CurrentTime_s
```

Example

`CurrentTime_s` will return a value of 101525 for 10:15:25 AM

`CurrentTime_s` will return a value of 154500 for 3:45:00 PM

DateTime2ELTime

Returns a numerical value indicating the time from the specified DateTime value. The time is indicated in the 24-hour HHmm format, where 1300 = 1:00 PM.

The integer portion of the DateTime value specifies the number of days since January 1st, 1900, and the fractional portion of the DateTime value specifies the fraction of the day since midnight.

Usage

`DateTime2ELTime(`*`DateTime`*`)`

Where: *`DateTime`* - a double-precision decimal DateTime value

Example

`DateTime2ELTime(39449.65625000)` will return a value of 1545, indicating 3:45 PM

DateTime2ELTime_s

Returns a numerical value indicating the time, including seconds, from the specified DateTime value. The time is indicated in the 24-hour HHmmss format, where 130000 = 1:00:00 PM.

The integer portion of the DateTime value specifies the number of days since January 1st, 1900, and the fractional portion of the DateTime value specifies the fraction of the day since midnight.

Usage

`DateTime2ELTime_s` (*`DateTime`*)

Where: *`DateTime`* - a double-precision decimal DateTime value

Example

`DateTime2ELTime_s(39449.646354167)` will return a value of 153045, indicating 3:30:45 PM

DateTimeToString

Returns a string expression corresponding to the specified DateTime value.

The integer portion of the DateTime value specifies the number of days since January 1st, 1900, and the fractional portion of the DateTime value specifies the fraction of the day since midnight.

Usage

`DateTimeToString` (*`DateTime`*)

Where: *`DateTime`* - a double-precision decimal DateTime value to be converted to a string expression

Notes

The output example is in the default US regional date and time format. Date and time formats are controlled by the Regional Options settings that can be accessed from the Control Panel of the Windows XP operating system.

Example

`DateTimeToString(39448.25000000)` will return the string "1/1/2008 6:00:00 AM"

DateTimeToString_Ms

Returns a string value indicating date and time of a bar with millisecond precision.

Usage

`DateTimeToString_Ms(`*`DT`*`)`

Where: *`DT`* is DateTime value.

Example

`DateTimeToString_Ms(`*`DateTime`*`)` will return a string value corresponding to the time and time of the current bar

"5/28/2013 08:41:11.871".

DateToJulian

Returns a numerical value corresponding to the Julian Date equivalent of the specified date.

The date is specified in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month.

Julian Date indicates the number of days that have elapsed since January 1st, 1900.

Usage

`DateToJulian(`*`YYYMMdd`*`)`

Where: *`YYYMMdd`* - a numerical expression, specifying the date in YYYMMdd format

Example

`DateToJulian (1080101)` will return a value of 39448, corresponding to the specified date of January 1st, 2008

`DateToJulian (990402)` will return a value of 36252, corresponding to the specified date of April 2nd, 1999

DateToString

Returns a string expression corresponding to the date (integer) portion of the specified DateTime value.

The integer portion of the DateTime value specifies the number of days since January 1st, 1900, and the fractional portion of the DateTime value specifies the fraction of the day since midnight.

Usage

`DateToString(`*`DateTime`*`)`

Where: *`DateTime`* - a double-precision decimal DateTime value to be converted to a string expression representing the date

Notes

The output example is in the default US regional date format. Date format is controlled by the Regional Options settings that can be accessed from the Control Panel of the Windows XP operating system.

Example

`DateToString(39448.25000000)` will return the string "1/1/2008".

DayFromDateTime

Returns a numerical value indicating the day of the month for the specified DateTime value.

The integer portion of the DateTime value specifies the number of days since January 1st, 1900, and the fractional portion of the DateTime value specifies the fraction of the day since midnight.

Usage

`DayFromDateTime(`*`DateTime`*`)`

Where: *`DateTime`* - a double-precision decimal DateTime value

Example

`DayFromDateTime(39449.25000000)` will return a value of 2, indicating the 2nd day of the month of January, 2008

DayOfMonth

Returns a numerical value, indicating the day of the month of the specified date.

The date is specified in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month.

Usage

`DayOfMonth(`*`YYYMMdd`*`)`

Where: *`YYYMMdd`* - a numerical expression, specifying the date in YYYMMdd format

Example

`DayOfMonth(1080101)` will return a value of 1, indicating the 1st day of the month of January, 2008

`DayOfMonth(990605)` will return a value of 5, indicating the 5th day of the month of June, 1999

DayOfWeek

Returns a numerical value, indicating the day of the week corresponding to the specified date, where 0 = Sunday, 1 = Monday, etc.

The Date is specified in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month.

Usage

`DayOfWeek(`*`YYYMMdd`*`)`

Where: *`YYYMMdd`* - a numerical expression, specifying the date in YYYMMdd format

Example

`DayOfWeek(1080101)` will return a value of 2, indicating Tuesday, for January 1st, 2008

`DayOfWeek(990603)` will return a value of 4, indicating Thursday, for June 3rd, 1999

DayOfWeekFromDateTime

Returns a numerical value, indicating the day of the week corresponding to the specified DateTime value, where 0 = Sunday, 1 = Monday, etc.

The integer portion of the DateTime value specifies the number of days since January 1st, 1900, and the fractional portion of the DateTime value specifies the fraction of the day since midnight.

Usage

`DayOfWeekFromDateTime(`*`DateTime`*`)`

Where: *`DateTime`* - a double-precision decimal DateTime value

Example

`DayOfWeekFromDateTime(39448.25000000)` will return a value of 2, indicating Tuesday, for January 1st, 2008

ELDateToDateTime

Returns the integer portion of a double-precision decimal DateTime value corresponding to the specified EL Date.

EL Date is specified in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month.

The integer portion of the DateTime value indicates the number of days that have elapsed since January 1st, 1900, and the fractional portion of the DateTime value indicates the fraction of the day that has passed since midnight.

Usage

`ELDateToDateTime`(*YYYMMdd*)

Where: *`YYYMMdd`* - a numerical expression, specifying the date in EL YYYMMdd format

Example

`ELDateToDateTime(1080101)` will return a value of 39448.00000000, corresponding to the specified date of January 1st, 2008

`ELDateToDateTime(990402)` will return a value of 36252.00000000, corresponding to the specified date of April 2nd, 1999

ELTimeToDateTime

Returns the fractional portion of a double-precision decimal DateTime value corresponding to the specified time. The time is specified in the 24-hour HHmm format, where 1300 = 1:00 PM.

The integer portion of the DateTime value indicates the number of days that have elapsed since January 1st, 1900, and the fractional portion of the DateTime value indicates the fraction of the day that has passed since midnight.

Usage

`ELTimeToDateTime(`*HHmm*`)`

Where: *`HHmm`* - a numerical expression specifying the time

Example

`ELTimeToDateTime(1015)` will return a value of 0.42708333, corresponding to the specified time of 10:15 AM

`ELTimeToDateTime(1545)` will return a value of 0.65625000, corresponding to the specified time of 3:45 PM

ELTimeToDateTime_s

Returns the fractional portion of a double-precision decimal DateTime value corresponding to the specified time, including seconds. The time is specified in the 24-hour HHmmss format, where 130000 = 1:00:00 PM.

The integer portion of the DateTime value indicates the number of days that have elapsed since January 1st, 1900, and the fractional portion of the DateTime value indicates the fraction of the day that has passed since midnight.

Usage

`ELTimeToDateTime_s`(*HHmmss*)

Where: *`HHmmss`* - a numerical expression specifying the time

Example

`ELTimeToDateTime_s(101525)` will return a value of 0.427372685, corresponding to the specified time of 10:15:25 AM

`ELTimeToDateTime_s(154500)` will return a value of 0.656250000, corresponding to the specified time of 3:45:00 PM

El_DateStr

Returns an 8-character numerical string corresponding to the specified date. The string is in the yyyyMMdd format, where yyyy is the four-digit year, MM is the month, and dd is the day of the month.

Usage

`El_DateStr`(*dd*, *MM*, *yyyy*)

Where: *`dd`* - a numerical expression specifying the day of the month
`MM` - a numerical expression specifying the month
`yyyy` - a four-digit numerical expression specifying the year

Example

`El_DateStr(02,04,2008)` will return the string "20080402", corresponding to the specified date of April 2nd, 2008.

El_DateToDateTime

Same as the `ElDateToDateTime`

EL_TimeToDateTime

Same as the `ElTimeToDateTime`

EL_TimeToDateTime_s

Same as `ElTimeToDateTime_s`

EncodeDate

Returns the integer portion of a double-precision decimal DateTime value corresponding to the specified date.

The integer portion of the DateTime value indicates the number of days that have elapsed since January 1st, 1900, and the fractional portion of the DateTime value indicates the fraction of the day that has passed since midnight.

Usage

`EncodeDate`(*yy*,*MM*,*dd*)

Where: *`yy`* - a numerical expression specifying the two-digit year
`MM` - a numerical expression specifying the month
`dd` - a numerical expression specifying the day of the month

Example

`EncodeDate(08,01,01)` will return a value of 39448.00000000, corresponding to the specified date of January 1st, 2008

EncodeTime

Returns the fractional portion of a double-precision decimal DateTime value corresponding to the specified time.

The integer portion of the DateTime value indicates the number of days that have elapsed since January 1st, 1900, and the fractional portion of the DateTime value indicates the fraction of the day that has passed since midnight.

Usage

`EncodeTime`(*HH*,*mm*,*ss*,*mmm*)

Where: *`HH`* - a numerical expression specifying the hours in 24-hour format
`mm` - a numerical expression specifying the minutes
`ss` - a numerical expression specifying the seconds
`mmm` - a numerical expression specifying the milliseconds

Example

`EncodeTime(16,29,55,500)` will return a value of 0.6874479167, corresponding to the specified time of 16:29:55.500

FormatDate

Returns a formated string expression corresponding to the date (integer) portion of the specified DateTime value.

The format of the string expression, including the abbreviations and separators, is defined by the specified format string. The format string consists of one or more elements arranged in the desired order. Each element represents a particular part of the date in a specific format. Spaces and separator characters to be used can be inserted within the format string.

Usage

`FormatDate("`*`FormatString`*`",` *DateTime*`)`

Parameters

`FormatString` - a format string, specifying the format of the output string expression representing the date

The following elements can be used in the format string:

d	Day of month as digits with no leading zero for single-digit days
dd	Day of month as digits with leading zero for single-digit days
ddd	Day of week as a three-letter abbreviation
dddd	Day of week as its full name
M	Month as digits with no leading zero for single-digit months
MM	Month as digits with leading zero for single-digit months
MMM	Month as a three-letter abbreviation
MMMM	Month as its full name
y	Year as last two digits, but with no leading zero for years less than 10
yy	Year as last two digits, but with leading zero for years less than 10
yyyy	Year represented by full four digits

`DateTime` - a double-precision decimal DateTime value to be converted to a string

expression representing the date

The integer portion of the DateTime value specifies the number of days since January 1st, 1900, and the fractional portion of the DateTime value specifies the fraction of the day since midnight.

Example

`FormatDate("dddd, MMMM dd, yyyy.",39469.250)` will return the string "Tuesday, January 22, 2008."

`FormatDate("M/d/y",39469.250)` will return the string "1/22/8"

`FormatDate("dd-MM-yy",39469.250)` will return the string "22-01-08"

`FormatDate("Next ddd is: MMM dd",39469.250)` will return the string "Next Tue is: Jan 22"

FormatTime

Returns a formated string expression corresponding to the time (fractional) portion of the specified DateTime value.

The format of the string expression, including separators, is defined by the specified format string. The format string consists of one or more elements arranged in the desired order. Each element represents a particular unit of time in a specific format. Spaces and separator characters to be used can be inserted within the format string.

Usage

`FormatTime("`*`FormatString`*`"`, *DateTime*)

Parameters

`FormatString` - a format string, specifying the format of the output string expression representing the time

The following elements can be used in the format string:

h	Hours in 12-hour AM/PM format with no leading zero for single-digit hours
hh	Hours in 12-hour AM/PM format with leading zero for single-digit hours
H	Hours in 24-hour format with no leading zero for single-digit hours
HH	Hours in 24-hour format with leading zero for single-digit hours
m	Minutes with no leading zero for single-digit minutes
mm	Minutes with leading zero for single-digit minutes
s	Seconds with no leading zero for single-digit seconds
ss	Seconds with leading zero for single-digit seconds
t	One character AM/PM designator
tt	Multicharacter AM/PM designator

`DateTime` - a double-precision decimal DateTime value to be converted to a string expression representing the time

The integer portion of the DateTime value specifies the number of days since January 1st, 1900, and the fractional portion of the DateTime value specifies the fraction of the day since midnight.

Example

`FormatTime("hh:mm:ss t",39469.6674)` will return the string "04:01:03 P"

`FormatTime("h tt",39469.6674)` will return the string "4 PM"

`FormatTime("HH:mm",39469.6674)` will return the string "16:01"

`FormatTime("m MIN s SEC",39469.6674)` will return the string "1 MIN 3 SEC"

Friday

Returns a numerical value of 5, corresponding to Friday.

Usage

`Friday`

Example

`Friday` will return a value of 5

HoursFromDateTime

Returns a numerical value indicating the hours from the specified DateTime value. The hours are indicated in the 24-hour format, where 13 = 1 PM.

The integer portion of the DateTime value specifies the number of days since January 1st, 1900, and the fractional portion of the DateTime value specifies the fraction of the day since midnight.

Usage

`HoursFromDateTime(`*DateTime*`)`

Where: ***`DateTime`*** - a double-precision decimal DateTime value

Example

`HoursFromDateTime(39449.85000000)` will return a value of 20, indicating 8 PM

IncMonth

Returns a numerical value corresponding to a Julian date that is after or before the specified Julian date by a specified number of calendar months.

Julian Date indicates the number of days that have elapsed since January 1st, 1900.

Usage

`IncMonth`(*JulianDate*,*M*)

Where: *JulianDate* - a numerical expression specifying the Julian Date
M - a numerical expression specifying the number of calendar months from the specified Julian date; if the value of *M* is; positive, a date **after** the specified date will be returned; if the value of *M* is; negative, a date **before** the specified date will be returned

Example

`IncMonth(39417,1)` will return a value of 39448, corresponding to January 1st, 2008, one calendar month after the specified date of December 1st, 2007

`IncMonth(36252,-2)` will return a value of 36193, corresponding to February 2nd, 1999, two calendar months before the specified date of April 2nd, 1999

JulianToDate

Returns a numerical value corresponding to the EL Date equivalent of the specified Julian Date.

Julian Date indicates the number of days that have elapsed since January 1st, 1900.

EL Date is in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month.

Usage

`JulianToDate`(*JulianDate*)

Where: *JulianDate* - a numerical expression specifying the Julian Date

Example

`JulianToDate (39448)` will return a value of 1080101, corresponding to the specified date of January 1st, 2008

`JulianToDate (36252)` will return a value of 990402, corresponding to the specified date of April 2nd, 1999

LastCalcDateTime

Returns a numerical value indicating the closing DateTime of the last completed bar.

The integer portion of the DateTime value specifies the number of days since January 1st, 1900, and the fractional portion of the DateTime value specifies the fraction of the day since midnight.

Usage

```
LastCalcDateTime
```

Example

`LastCalcDateTime` will return a value of 41871.83333 for the last bar completion at 08/20/2014 8:00 PM

LastCalcJDate

Returns a numerical value indicating the Julian date for the last completed bar.

Julian Date indicates the number of days that have elapsed since January 1st, 1900.

Usage

```
LastCalcJDate
```

Example

`LastCalcJDate` will return a value of 39448 for the last bar completion date of January 1st, 2008

LastCalcMMTime

Returns a numerical value indicating the closing time of the last completed bar. The time is indicated as the number of minutes that have passed since midnight.

Usage

```
LastCalcMMTime
```

Example

`LastCalcMMTime` will return a value of 850 for the last bar completion time of 2:10 PM

LastCalcmSTime

Returns a numerical value indicating the closing time of the last completed bar. The time is indicated as the number of milliseconds that have passed since midnight.

Usage

```
LastCalcmSTime
```

Example

`LastCalcmSTime` will return a value of 51030150 for the last bar completion time of 2:10:30.150 PM

LastCalcSSTime

Returns a numerical value indicating the closing time of the last completed bar. The time is indicated as the number of seconds that have passed since midnight.

Usage

```
LastCalcSSTime
```

Example

`LastCalcSSTime` will return a value of 51030 for the last bar completion time of 2:10:30 PM

MilliSecondsFromDateTime

Returns a numerical expression indicating the millisecond stamp of DateTime value.

The integer portion of the DateTime value indicates the number of days that have elapsed since January 1st, 1900, and the fractional portion of the DateTime value indicates the fraction of the day that has passed since midnight.

Usage

`MilliSecondsFromDateTime`(*DT*)

Where: *DT* - is a DateTime value.

Example

`MillisecondsFromDateTime(DateTtime)` will return 687.00 if the time stamp of the current bar is 11:57:07.687

MinutesFromDateTime

Returns a numerical value indicating the minutes from the specified DateTime value.

The integer portion of the DateTime value specifies the number of days since January 1st, 1900, and the fractional portion of the DateTime value specifies the fraction of the day since midnight.

Usage

`MinutesFromDateTime`(*DateTime*)

Where: ***`DateTime`*** - a double-precision decimal DateTime value

Example

`MinutesFromDateTime`(`39449.35000000`) will return a value of 24, indicating 24 minutes after 8 AM

Monday

Returns a numerical value of 1, corresponding to Monday.

Usage

`Monday`

Example

`Monday` will return a value of 1

Month

Returns a numerical value, indicating the month of the specified EL Date.

EL Date is specified in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month.

Usage

`Month`(*YYYMMdd*)

Where: *`YYYMMdd`* - a numerical expression, specifying the date in EL YYYMMdd format

Example

`Month` (`1080101`) will return a value of 1, indicating the month of January

`Month` (`990605`) will return a value of 6, indicating the month of June

MonthFromDateTime

Returns a numerical value indicating the month for the specified DateTime value.

The integer portion of the DateTime value specifies the number of days since January 1st, 1900, and the fractional portion of the DateTime value specifies the fraction of the day since midnight.

Usage

`MonthFromDateTime`(*DateTime*)

Where: *`DateTime`* - a double-precision decimal DateTime value

Example

`MonthFromDateTime` (`39600.25000000`) will return a value of 6, indicating the month of June

Saturday

Returns a numerical value of 6, corresponding to Saturday.

Usage

```
Saturday
```

Example

`Saturday` will return a value of 6

SecondsFromDateTime

Returns a numerical value indicating the seconds from the specified DateTime value.

The integer portion of the DateTime value specifies the number of days since January 1st, 1900, and the fractional portion of the DateTime value specifies the fraction of the day since midnight.

Usage

`SecondsFromDateTime`(*DateTime*)

Where: *`DateTime`* - a double-precision decimal DateTime value

Example

`SecondsFromDateTime` (`39449.35440000`) will return a value of 20, indicating 20 seconds from 8:30:20 AM

StringToDate

Returns the integer portion of a double-precision decimal DateTime value, corresponding to the specified date. The date is specified by a string expression "MM/dd/yy"* or "MM/dd/yyyy"*, where MM* is the month, dd* is the day, and yy or yyyy is a two-digit or four-digit year.

The integer portion of the DateTime value indicates the number of days that have elapsed since January 1st, 1900, and the fractional portion of the DateTime value indicates the fraction of the day that has passed since midnight.

Usage

`StringToDate("`*`MM/dd/yy`*`")`

or:

`StringToDate("`*`MM/dd/yyyy`*`")`

Where: *`MM`* - month*
`dd` - day of the month*
`yy` - a two-digit year
`yyyy` - a four-digit year

Notes

* Described usage and examples are for the default US regional date format. If the default UK regional format is selected, the dates will be in **dd/MM/yy** and **dd/MM/yyyy** format instead. Date format is controlled by the Regional Options settings that can be accessed from the Control Panel of the Windows XP operating system.

Example

`StringToDate("01/01/2008")` will return a value of 39448.00000000, corresponding to the specified date of January 1st, 2008

`StringToDate("04/04/99")` will return a value of 36254.00000000, corresponding to the specified date of April 4th, 1999

StringToDateTime

Returns a double-precision decimal DateTime value corresponding to the specified date and time. The date and time are specified by a string expression "MM/dd/yy hh:mm:ss tt"* or "MM/dd/yyyy hh:mm:ss tt"*, where MM* is the month, dd* is the day, yy or yyyy is a two-digit or four-digit year, hh is the hours in 12-hour AM/PM format, mm is the minutes, ss is the seconds, and tt is the AM/PM designator.

The integer portion of the DateTime value indicates the number of days that have elapsed since January 1st, 1900, and the fractional portion of the DateTime value indicates the fraction of the day that has passed since midnight.

Usage

`StringToDateTime("`*`MM/dd/yy hh:mm:ss tt`*`")`

or:

`StringToDateTime("`*`MM/dd/yyyy hh:mm:ss tt`*`")`

Where: *MM* - month*
dd - day of the month*
yy - a two-digit year
yyyy - a four-digit year
hh - hours
mm - minutes
ss - seconds
tt - AM/PM designator

Notes

* Described usage and examples are for the default US regional date and time formats. If the default UK regional format is selected, the dates will be in *dd/MM/yy* and *dd/MM/yyyy* format instead. Date and time formats are controlled by the Regional Options settings that can be accessed from the Control Panel of the Windows XP operating system.

Example

`StringToDateTime(`*`"01/01/2008 08:00:00 AM"`*`)` will return a value of 39448.33333333, corresponding to 08:00:00 AM on January 1st, 2008

`StringToDateTime(`*`"04/04/99 04:48:00 PM"`*`)` will return a value of 36254.70000000, corresponding to 04:48:00 PM on April 4th, 1999

StringToDTFormatted

Returns the DateTime value for the string in a particular format. The format of the string expression, including the abbreviations and separators, is defined by the specified format string. The format string consists of one or more elements arranged in the desired order. Each element represents a particular part of the date in a specific format. Spaces and separator characters to be used can be inserted within the format string.

Usage

`StringToDTFormatted`("DateTimeString", "FormatString")

Parameters

DateTimeString - time and date value in string format.

FormatString - a format string, specifying the format of the output string expression representing the date and time.

The following elements can be used in the format string:

`d` - day of month as digits with no leading zero for single-digit days
`dd` - day of month as digits with leading zero for single-digit days
`ddd` - day of week as a three-letter abbreviation
`dddd` - day of week as its full name
`M` - month as digits with no leading zero for single-digit months
`MM` - onth as digits with leading zero for single-digit months
`MMM` - month as a three-letter abbreviation
`MMMM` - month as its full name
`y` - year as last two digits, but with no leading zero for years less than 10
`yy` - year as last two digits, but with leading zero for years less than 10
`yyyy` - year represented by full four digits
`h` - hours in 12-hour AM/PM format with no leading zero for single-digit hours
`hh` - hours in 12-hour AM/PM format with leading zero for single-digit hours
`H` - hours in 24-hour format with no leading zero for single-digit hours

`HH` - hours in 24-hour format with leading zero for single-digit hours
`m` - minutes with no leading zero for single-digit minutes
`mm` - minutes with leading zero for single-digit minutes
`s` - seconds with no leading zero for single-digit seconds
`ss` - seconds with leading zero for single-digit seconds
`t` - one character AM/PM designator
`tt` - multicharacter AM/PM designator

Return value

DateTime - a double-precision decimal DateTime value, corresponding to the first string parameter with a second
string parameter format applied to it.

The integer portion of the DateTime value specifies the number of days since January 1st, 1900, and the fractional
portion of the DateTime value specifies the fraction of the day since midnight.

Note

Attention! Check if the returned value is correct! If conversion fails, the function returns -1.

Example

`StringToDTFormatted("22/11/2013 15:35", "dd/MM/yyy HH:mm")` will return 41600.649305555555 , which represents November 22nd 2013, 3:35 PM.

`StringToDTFormatted("02/17/11", "MM/dd/yy")` will return 40591 , which represents February 17th 2011.

`StringToDTFormatted("17/02/11", "MM/dd/yy")` will return -1, string conversion to DateTime format failed.

`StringToDTFormatted("4:00 PM", "h:mm tt")` will return 2.66666666666667 , which represents the minimum value of JulianDate = January 1st 1900, 4:00 PM.

StringToTime

Returns the fractional portion of a double-precision decimal DateTime value, corresponding to the specified time.

The time is specified by a string expression "hh:mm:ss tt", where hh is the hours in the 12-hour AM/PM format, mm is the minutes, ss is the seconds, and tt is the AM/PM designator.

The integer portion of the DateTime value indicates the number of days that have elapsed since January 1st, 1900, and the fractional portion of the DateTime value indicates the fraction of the day that has passed since midnight.

Usage

`StringToTime("`*`hh:mm:ss tt`*`")`

Where: *`hh`* - hours in the 12-hour AM/PM format
`mm` - minutes
`ss` - seconds
`tt` - AM/PM designator

Notes

Described usage and examples are for the default US regional time format. Time format is controlled by the Regional Options settings that can be accessed from the Control Panel of the Windows XP operating system.

Example

`StringToTime (`*`"08:00:00 AM"`*`)` will return a value of 0.33333333

`StringToTime (`*`"04:48:00 PM"`*`)` will return a value of 0.70000000

Sunday

Returns a numerical value of 0, corresponding to Sunday.

Usage

```
Sunday
```

Example

`Sunday` will return a value of 0

Thursday

Returns a numerical value of 4, corresponding to Thursday.

Usage

```
Thursday
```

Example

`Thursday` will return a value of 4

Time2Time_s

Returns a numerical value indicating the time in the HHmmss format, corresponding to the specified time in the HHmm format.

The time is returned in the 24-hour HHmmss format, where 130000 = 1:00:00 PM, and specified in the 24-hour HHmm format, where 1300 = 1:00 PM.

Usage

`Time2Time_s(`*`HHmm`*`)`

Where: *`HHmm`* - a numerical expression specifying the time

Example

`Time2Time_s(1015)` will return a value of 101500

`Time2Time_s(1545)` will return a value of 154500

TimeToString

Returns a string expression corresponding to the time (fractional) portion of the specified DateTime value.

The integer portion of the DateTime value specifies the number of days since January 1st, 1900, and the fractional portion of the DateTime value specifies the fraction of the day since midnight.

Usage

`TimeToString(`*DateTime*`)`

Where: *`DateTime`* - a double-precision decimal DateTime value to be converted to a string expression representing the time

Notes

The output example is in the default US regional time format. Time format is controlled by the Regional Options settings that can be accessed from the Control Panel of the Windows XP operating system.

Example

`TimeToString(39448.75000000)` will return the string "6:00 PM"

Time_s2Time

Returns a numerical value indicating the time in the HHmm format, corresponding to the specified time in the HHmmss format; the seconds are truncated.

The time is returned in the 24-hour HHmm format, where 1300 = 1:00 PM, and specified in the 24-hour HHmmss format, where 130000 = 1:00:00 PM.

Usage

`Time_s2Time(`*`HHmmss`*`)`

Where: *`HHmmss`* - a numerical expression specifying the time

Example

`Time_s2Time(101520)` will return a value of 1015

`Time_s2Time(154548)` will return a value of 1545

Tuesday

Returns a numerical value of 2, corresponding to Tuesday.

Usage

`Tuesday`

Example

`Tuesday` will return a value of 2

Wednesday

Returns a numerical value of 3, corresponding to Wednesday.

Usage

`Wednesday`

Example

`Wednesday` will return a value of 3

Year

Returns a numerical value, indicating the year of the specified EL Date.

EL Date is specified in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month.

Usage

`Year(`*`YYYMMdd`*`)`

Where: *`YYYMMdd`* - a numerical expression, specifying the date in EL YYYMMdd format

Example

`Year(1080101)` will return a value of 108, indicating the year of 2008

`Year(990605)` will return a value of 99, indicating the year of 1999

YearFromDateTime

Returns a numerical value indicating the year for the specified DateTime value.

The integer portion of the DateTime value specifies the number of days since January 1st, 1900, and the fractional portion of the DateTime value specifies the fraction of the day since midnight.

Usage

`YearFromDateTime`(*DateTime*)

Where: *`DateTime`* - a double-precision decimal DateTime value

Example

`YearFromDateTime(39449.25000000)` will return a value of 2008, indicating the year of 2008

Array

Declares one or more names as arrays, containing multiple variable data elements; specifies the array structure, data elements type and initial value, update basis, and data number, for each of the arrays.

Data elements type can be numerical, string, or true/false.

The number of elements in an array can be fixed or dynamic (unlimited).

In arrays with a fixed number of elements, the elements can be arranged in single or multiple dimensions. A one-dimensional 10-element array contains 10 elements, a two-dimensional 10-element by 10-element array contains 100 elements, a three-dimensional 10 by 10 by 10 element array contains 1000 elements, a four-dimensional 10 by 10 by 10 by 10 element array contains 10000 elements, etc. The maximum number of array dimensions in PowerLanguage is 9.

Each element in an array is referenced by one or more index numbers, one for each of the dimensions. Indexing starts at 0 for each of the dimensions.

Dynamic arrays (arrays with an unlimited number of elements) are one-dimensional, and are initialized at declaration as having only one element. Declared dynamic arrays can be resized using `Array_SetMaxIndex`.

Elements can be manipulated individually or as a group, in all or part of an array.

Usage

`Array`**:**`<IntraBarPersist>`*`ArrayName1`***[***D1,D2,D3,etc.***](***InitialValue1*<**,**`DataN`>**),**
`<IntraBarPersist>`*`ArrayName2`***[***D1,D2,D3,etc.***](***InitialValue2*<**,**`DataN`>**),***`etc.`*

Parameters inside the angled brackets are optional

Parameters

`IntraBarPersist` - an optional parameter; specifies that the value of the array elements is to be updated on every tick
If this parameter is not specified, the value will be updated at the close of each bar.

`ArrayName` - an expression specifying the array name

The name can consist of letters, underscore characters, numbers, and periods. The name cannot begin with a number or a period and is not case-sensitive.

`D` - a numerical expression specifying the array size in elements, starting at 0, for each of the dimensions; a single expression specifies a one-dimensional array, two expressions specify a two-dimensional (`D1` by `D2`) array, three expressions specify a three-dimensional (`D1` by `D2` by `D3`) array, etc. A dynamic array, with an unlimited number of elements, is specified by the empty square brackets: `[]` and will be a one-dimensional array.

`InitialValue` - an expression, specifying the initial value and defining the data type for all of the elements in the array
The value can be a numerical, string, or true/false expression; the type of the expression defines the data type.

`DataN` - an optional parameter; specifies the Data Number of the data series the array is to be tied to
If this parameter is not specified, the array will be tied to the default data series.

Example

Declare Length and SFactor as 9-element one-dimensional numerical arrays with data elements' initial values of **0**:

```
Array:Length[8](0),SFactor[8](0);
```

Declare Max_Price as a 24-element by 60-element two-dimensional numerical array, updated on every tick, tied to the series with Data #2, and with data elements' initial values equal to the value of Close function:

```
Array:IntraBarPersist Max_Price[23,59](Close,Data2);
```

Declare Highs2 as a dynamic numerical array with data elements' initial values of 0:

```
Array:Highs2[](0);
```

Arrays

Same as the `Array`

Input

Declares one or more names as inputs; specifies the default value and defines the input type for each input.

Inputs can be numerical, string, or true/false. Once declared, the value of the input cannot be modified by the study's code.

Usage

`Input:`*`InputName1`*(*DefaultValue1*), *`InputName2`*(*DefaultValue2*), *`etc.`*

Parameters

`InputName` - an expression specifying the input name

The name can consist of letters, underscore characters, numbers, and periods. The name cannot begin with a number or a period and is not case-sensitive.

`DefaultValue` - an expression specifying the default value and defining the input type

The expression can be numerical, string, or true/false; the type of the expression defines the input type.

Example

Declare Length as a numerical input with the default value of 20:

```
Input:Length(20);
```

Declare Price as a numerical input with the default value equal to the value of Close function, and Name as a character string input with the default value of "Last Close":

```
Input:Price(Close), Name("Last Close");
```

Declare Draw_Line as a true/false input with the default value of True:

```
Input:Draw_Line(True);
```

Inputs

Same as the `Input`

IntraBarPersist

Used in variable and array declaration statements, before a variable or array name, to specify that the value of the variable or array elements are to be updated on every tick.

If `IntraBarPersist` is not specified, the value will be updated at the close of each bar.

Usage

`Declaration`:[`IntraBarPersist`]*Name*(*InitialValue1*)

Example

Declare Max as a numerical variable, updated on every tick, with the initial value of 100:

```
Variable:IntraBarPersist Max(100);
```

Declare Max_Price as a 24-element single-dimension numerical array, updated on every tick, and with data elements' initial values of 0:

```
Array:IntraBarPersist Max_Price[23](0);
```

Numeric

Used in function input declaration statements to define an input as Numerical.

Input defined as Numerical can be used both as a Numerical Simple as well as a Numerical Series input; the value of a Simple input is constant from bar to bar and thus has no history, while the value of a Series input may vary from bar to bar and can be referenced historically.

Usage

`Input:`*`InputName`*`(Numeric)`

Where: *`InputName`* - an expression specifying the input name

The name can consist of letters, underscore characters, numbers, and periods.

The name cannot begin with a number or a period and is not case-sensitive.

Example

Declare `Length` as a Numerical function input:

```
Input:Length(Numeric);
```

NumericArray

Used in function input declaration statements to define an input as a Numerical Array with a specified number of dimensions.

Input defined as Numerical can be used both as a Numerical Simple as well as a Numerical Series input; the value of a Simple input is constant from bar to bar and thus has no history, while the value of a Series input may vary from bar to bar and can be referenced historically.

Usage

`Input:`*`InputName`***[***M1***,***M2***,***M3***,***etc.***]**`(NumericArray)`

Parameters

`InputName` - an expression specifying the input name

The name can consist of letters, underscore characters, numbers, and periods.

The name cannot begin with a number or a period and is not case-sensitive.

`M` - an input variable that represents the maximum index value for each dimension of the array passed to the function; a single input variable specifies a one-dimensional array input, two input variables specify a two-dimensional (*`M1`* by *`M2`*) array input, three input variables specify a three-dimensional (*`M1`* by *`M2`* by *`M3`*) array input, etc.

An input will only accept an array with the specified number of dimensions.

Example

Declare Length as a one-dimensional Numerical Array function input:

```
Input:Length[X](NumericArray);
```

The maximum index value for the array passed to the function will be assigned to input variable X.

Declare Table as a three-dimensional Numerical Array function input:

```
Input:Table[X,Y,Z](NumericArray);
```

The maximum index value for each dimension of the array passed to the function will be assigned to input variables X, Y, and Z.

NumericArrayRef

Used in function input declaration statements to define an input as Passed by Reference Numerical Array with a specified number of dimensions.

Declaring an input as Passed by Reference enables the function to modify the values of variables passed as the input.

Usage

`Input:`*`InputName`***[***M1***,***M2***,***M3***,***etc.***]**`(NumericArrayRef)`

Parameters

`InputName` - an expression specifying the input name

The name can consist of letters, underscore characters, numbers, and periods.

The name cannot begin with a number or a period and is not case-sensitive.

`M` - an input variable that represents the maximum index value for each dimension of the array passed to the function; a single input variable specifies a one-dimensional array input, two input variables specify a two-dimensional (*`M1`* by *`M2`*) array input, three input variables specify a three-dimensional (*`M1`* by *`M2`* by *`M3`*) array input, etc.

An input will only accept an array with the specified number of dimensions.

Example

Declare `Count` as a Passed by Reference one-dimensional Numerical Array function input:

`Input:Count[X](NumericArrayRef);`

The maximum index value for the array passed to the function will be assigned to input variable X.

Declare `Table` as a Passed by Reference three-dimensional Numerical Array

function input:

```
Input:Table[X,Y,Z](NumericArrayRef);
```

The maximum index value for each dimension of the array passed to the function will be assigned to input variables X, Y, and Z.

NumericRef

Used in function input declaration statements to declare a Passed by Reference Numerical input.

Declaring an input as Passed by Reference enables the function to modify the values of variables passed as the input.

Usage

```
Input:InputName(NumericRef)
```

Where: `InputName` - an expression specifying the input name

The name can consist of letters, underscore characters, numbers, and periods.

The name cannot begin with a number or a period and is not case-sensitive.

Example

Declare BarCount as a Passed by Reference Numerical function input:

```
Input:BarCount(NumericRef);
```

NumericSeries

Used in function input declaration statements to define an input as a Numerical Series.

The value of an input defined as a Series may vary from bar to bar and can be referenced historically.

Usage

```
Input:InputName(NumericSeries)
```

Where: `InputName` - an expression specifying the input name

The name can consist of letters, underscore characters, numbers, and periods.

The name cannot begin with a number or a period and is not case-sensitive.

Example

Declare Price as a Numerical Series function input:

```
Input:Price(NumericSeries);
```

NumericSimple

Used in function input declaration statements to define an input as Numerical Simple.

The value of an input defined as Simple is constant from bar to bar and thus has no history.

Usage

`Input:`*`InputName`*`(NumericSimple)`

Where: *`InputName`* - an expression specifying the input name

The name can consist of letters, underscore characters, numbers, and periods.

The name cannot begin with a number or a period and is not case-sensitive.

Example

Declare Length as a Numerical Simple function input:

```
Input:Length(NumericSimple);
```

RecalcPersist

Used in variable declaration statements, before a variable name, to specify that the value of the variable is to be updated on every tick and the latest value of this variable is to be saved after the study recalculation.

Usage

```
Declaration:[RecalcPersist]Name(InitialValue1)
```

Note

- This keyword can be used only with variables.
- The variable with such an attribute cannot be serial type.

Example

Declare Max as a numerical value, updated on every tick, with initial value of 100.

After study recalculation, the latest value of Max will be saved:

```
Variable: RecalcPersist Max(100);
```

String

Used in function input declaration statements to define an input as String.

Input defined as String can be used both as a String Simple as well as a String Series input; the value of a Simple input is constant from bar to bar and thus has no history, while the value of a Series input may vary from bar to bar and can be referenced historically.

Usage

`Input:`*`InputName`*`(String)`

Where: *`InputName`* - an expression specifying the input name

The name can consist of letters, underscore characters, numbers, and periods.

The name cannot begin with a number or a period and is not case-sensitive.

Example

Declare Name as a String function input:

```
Input:Name(String);
```

StringArray

Used in function input declaration statements to define an input as a String Array with a specified number of dimensions.

Input defined as String can be used both as a String Simple as well as a String Series input; the value of a Simple input is constant from bar to bar and thus has no history, while the value of a Series input may vary from bar to bar and can be referenced historically.

Usage

`Input:`*`InputName`*[*M1*,*M2*,*M3*,*etc.*]`(StringArray)`

Parameters

`InputName` - an expression specifying the input name

The name can consist of letters, underscore characters, numbers, and periods.

The name cannot begin with a number or a period and is not case-sensitive.

`M` - an input variable that represents the maximum index value for each dimension of the array passed to the function; a single input variable specifies a one-dimensional array input, two input variables specify a two-dimensional (*`M1`* by *`M2`*) array input, three input variables specify a three-dimensional (*`M1`* by *`M2`* by *`M3`*) array input, etc.

An input will only accept an array with the specified number of dimensions.

Example

Declare Messages as a one-dimensional String Array function input:

```
Input:Messages[X](StringArray);
```

The maximum index value for the array passed to the function will be assigned to input variable X.

Declare MessageTable as a three-dimensional String Array function input:

```
Input:MessageTable[X,Y,Z](StringArray);
```

The maximum index value for each dimension of the array passed to the function will be assigned to input variables X, Y, and Z.

StringArrayRef

Used in function input declaration statements to define an input as Passed by Reference String Array with a specified number of dimensions.

Declaring an input as Passed by Reference enables the function to modify the values of variables passed as the input.

Usage

`Input:`*`InputName`***[***M1***,***M2***,***M3***,***etc.***]**`(StringArrayRef)`

Parameters

`InputName` - an expression specifying the input name

The name can consist of letters, underscore characters, numbers, and periods.

The name cannot begin with a number or a period and is not case-sensitive.

`M` - an input variable that represents the maximum index value for each dimension of the array passed to the function; a single input variable specifies a one-dimensional array input, two input variables specify a two-dimensional (*`M1`* by *`M2`*) array input, three input variables specify a three-dimensional (*`M1`* by *`M2`* by *`M3`*) array input, etc.

An input will only accept an array with the specified number of dimensions.

Example

Declare Messages as a Passed by Reference one-dimensional String Array function input:

`Input:Messages[X](StringArrayRef);`

The maximum index value for the array passed to the function will be assigned to input variable X.

Declare CommentsTable as a Passed by Reference three-dimensional String Array

function input:

```
Input:CommentsTable[X,Y,Z](StringArrayRef);
```

The maximum index value for each dimension of the array passed to the function will be assigned to input variables X, Y, and Z.

StringRef

Used in function input declaration statements to declare a Passed by Reference String input.

Declaring an input as Passed by Reference enables the function to modify the values of variables passed as the input.

Usage

`Input:`*`InputName`*`(StringRef)`

Where: *`InputName`* - an expression specifying the input name

The name can consist of letters, underscore characters, numbers, and periods.

The name cannot begin with a number or a period and is not case-sensitive.

Example

Declare `Message` as a Passed by Reference String function input:

```
Input:Message(StringRef);
```

StringSeries

Used in function input declaration statements to define an input as String Series.

The value of an input defined as a Series may vary from bar to bar and can be referred to historically.

Usage

`Input:`*`InputName`*`(StringSeries)`

Where: *`InputName`* - an expression specifying the input name

The name can consist of letters, underscore characters, numbers, and periods.

The name cannot begin with a number or a period and is not case-sensitive.

Example

Declare `Messages` as a String Series function input:

```
Input:Messages(StringSeries);
```

StringSimple

Used in function input declaration statements to define an input as String Simple.

The value of an input defined as Simple is constant from bar to bar and thus has no history.

Usage

`Input:`*`InputName`*`(StringSimple)`

Where: *`InputName`* - an expression specifying the input name

The name can consist of letters, underscore characters, numbers, and periods.

The name cannot begin with a number or a period and is not case-sensitive.

Example

Declare Name as a String Simple function input:

```
Input:Name(StringSimple);
```

TrueFalse

Used in function input declaration statements to define an input as true/false.

Input defined as true/false can be used both as a true/false Simple as well as a true/false Series input; the value of a Simple input is constant from bar to bar and thus has no history, while the value of a Series input may vary from bar to bar and can be referenced historically.

Usage

```
Input:InputName(TrueFalse)
```

Where: *`InputName`* - an expression specifying the input name

The name can consist of letters, underscore characters, numbers, and periods.

The name cannot begin with a number or a period and is not case-sensitive.

Example

Declare `Overnight` as a true/false function input:

```
Input:Overnight(TrueFalse);
```

TrueFalseArray

Used in function input declaration statements to define an input as a true/false Array with a specified number of dimensions.

Input defined as true/false can be used both as a true/false Simple as well as a true/false Series input; the value of a Simple input is constant from bar to bar and thus has no history, while the value of a Series input may vary from bar to bar and can be referenced historically.

Usage

`Input:`*`InputName`***[***M1***,***M2***,***M3***,***etc.***]**(`TrueFalseArray`)

Parameters

`InputName` - an expression specifying the input name

The name can consist of letters, underscore characters, numbers, and periods.

The name cannot begin with a number or a period and is not case-sensitive.

`M` - an input variable that represents the maximum index value for each dimension of the array passed to the function; a single input variable specifies a one-dimensional array input, two input variables specify a two-dimensional (*`M1`* by *`M2`*) array input, three input variables specify a three-dimensional (*`M1`* by *`M2`* by *`M3`*) array input, etc.

An input will only accept an array with the specified number of dimensions.

Example

Declare `UpTrend` as a one-dimensional true/false Array function input:

`Input:UpTrend[X](TrueFalseArray);`

The maximum index value for the array passed to the function will be assigned to input variable X.

Declare `FlagTable` as a three-dimensional true/false Array function input:

```
Input:FlagTable[X,Y,Z](TrueFalseArray);
```

The maximum index value for each dimension of the array passed to the function will be assigned to input variables X, Y, and Z.

TrueFalseArrayRef

Used in function input declaration statements to define an input as Passed by Reference true/false Array with a specified number of dimensions.

Declaring an input as Passed by Reference enables the function to modify the values of variables passed as the input.

Usage

`Input:`*`InputName`***[***M1***,***M2***,***M3***,***etc.***]**(`TrueFalseArrayRef`)

Parameters

`InputName` - an expression specifying the input name

The name can consist of letters, underscore characters, numbers, and periods.

The name cannot begin with a number or a period and is not case-sensitive.

`M` - an input variable that represents the maximum index value for each dimension of the array passed to the function; a single input variable specifies a one-dimensional array input, two input variables specify a two-dimensional (*`M1`* by *`M2`*) array input, three input variables specify a three-dimensional (*`M1`* by *`M2`* by *`M3`*) array input, etc.

An input will only accept an array with the specified number of dimensions.

Example

Declare `Trend` as a Passed by Reference one-dimensional true/false Array function input:

`Input:Trend`[X](`TrueFalseArrayRef`);

The maximum index value for the array passed to the function will be assigned to input variable X.

Declare `TrendTable` as a Passed by Reference three-dimensional true/false Array

function input:

```
Input:TrendTable[X,Y,Z](TrueFalseArrayRef);
```

The maximum index value for each dimension of the array passed to the function will be assigned to input variables X, Y, and Z.

TrueFalseRef

Used in function input declaration statements to declare a Passed by Reference true/false input.

Declaring an input as Passed by Reference enables the function to modify the values of variables passed as the input.

Usage

```
Input:InputName(TrueFalseRef)
```

Where: `InputName` - an expression specifying the input name

The name can consist of letters, underscore characters, numbers, and periods.

The name cannot begin with a number or a period and is not case-sensitive.

Example

Declare Flag as a Passed by Reference true/false function input:

```
Input:Flag(TrueFalseRef);
```

TrueFalseSeries

Used in function input declaration statements to define an input as a true/false Series.

The value of an input defined as a Series may vary from bar to bar and can be referenced historically.

Usage

```
Input:InputName(TrueFalseSeries)
```

Where: `InputName` - an expression specifying the input name

The name can consist of letters, underscore characters, numbers, and periods.

The name cannot begin with a number or a period and is not case-sensitive.

Example

Declare UpTrend as a true/false Series function input:

```
Input:UpTrend(TrueFalseSeries);
```

TrueFalseSimple

Used in function input declaration statements to define an input as true/false Simple.

The value of an input defined as Simple is constant from bar to bar and thus has no history.

Usage

`Input:`*`InputName`*`(TrueFalseSimple)`

Where: *`InputName`* - an expression specifying the input name

The name can consist of letters, underscore characters, numbers, and periods.

The name cannot begin with a number or a period and is not case-sensitive.

Example

Declare `Overnight` as a true/false Simple function input:

```
Input:Overnight(TrueFalseSimple);
```

Var

Same as the `Variable`

Variable

Declares one or more names as variables; specifies the initial value, variable type, update basis, and data number for each variable.

Variables can be numerical, string, or true/false.

Usage

`Variable:[IntraBarPersist]`*`VariableName1`*(*InitialValue1*[**,**`DataN`]), `[IntraBarPersist]`*`VariableName2`*(*InitialValue2*[**,**`DataN`]),*`etc.`*

Parameters inside the square brackets are optional

Parameters

`IntraBarPersist` - an optional parameter; specifies that the value of the variable is to be updated on every tick
If this parameter is not specified, the value will be updated at the close of each bar.

`VariableName` - an expression, specifying the variable name
The name can consist of letters, underscore characters, numbers, and periods.

The name cannot begin with a number or a period and is not case-sensitive.

`InitialValue` - an expression, specifying the initial value and defining the variable type
The value can be a numerical, string, or true/false expression; the type of the expression defines the variable type.

`DataN` - an optional parameter; specifies the Data Number of the data series the variable is to be tied to
If this parameter is not specified, the variable will be tied to the default data series.

Example

Declare Avg. as a numerical variable with the initial value of 20:

```
Variable:Avg.(20);
```

Declare Max as a numerical variable, updated on every tick, with the initial value of 100:

```
Variable:IntraBarPersist Max(100);
```

Declare Min_Price as a numerical variable, tied to the series with Data #2, and the initial value equal to the value of Close function:

```
Variable:Min_Price(Close,Data2);
```

Declare Overnight as a true/false variable with the initial value of False, and Name as a string variable with the initial value of "Intra-Day":

```
Variable:Overnight(False),Name("Intra-Day");
```

Variables

Same as the `Variable`

Vars

Same as the `Variable`

ArraySize

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

ArrayStartAddr

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

Bool

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

Byte

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

Char

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

DefineDLLFunc

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

Double

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

DWORD

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

External

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

Float

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

iEasyLanguageObject

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

Int

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

Int64

The int64 is a reserved word defining that the passed or returned value of functions exported from the dll has the long long type (64-bit representation of an integer).

Note

Due to architectural specifics of PowerLanguage, before passing the long long type value in a dll (or when the values were returned from a dll) it is converted from the "double" type (or into "double" type). Note that precision of the double type values is 15 decimals. (Values range of the long long type is from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 - 19 decimal points). It means that without loss of accuracy the values with the range from -999,999,999,999,999 to 999,999,999,999,999 will be passed.

Example

Defines the dll function, which receives long long parameter:

```
DEFINEDLLFUNC:"test_int64.dll",void,"int64test_1",int64;
```

Defines the dll function, which returns long long value:

```
DEFINEDLLFUNC:"test_int64.dll",int64,"int64test_2";
```

Long

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

LPBool

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

LPByte

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

LPDouble

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

LPDWORD

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

LPFloat

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

LPInt

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

LPLong

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

LPSTR

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

LPWORD

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

Method

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

OnCreate

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

OnDestroy

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

Self

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

ThreadSafe

Declares the function imported from DLL as thread-safe.

It will increase the performance of calculation and optimization of the studies that use external DLLs.

Usage

```
ThreadSafe
```

Notes

It is not recommended to apply `ThreadSafe` attribute to the functions exported from elkit32.dll (for example `FindAddress_`).

Example

```
DEFINEDLLFUNC: ThreadSafe, "user32.dll", Void, "MessageBeep", Int;
```

Will declare MessageBeep function of user32.dll as thread-safe.

Unsigned

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

VarSize

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

VarStartAddr

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

Void

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

WORD

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

#Events

Supported by PowerLanguage.

A detailed description and usage examples have been published by TradeStation Technologies: EasyLanguage Extension SDK

DOM_AskPrice

Returns ask price for the certain depth level of a particular symbol.

Usage

`DOM_AskPrice(`*`num`*`)` [`Data(`*`N`*`)`]

where:

(*num*) - is the number of depth level

(*N*) - is the number of the data series

Example

`DOM_AskPrice(4)`; will return the ask price for the 5th level of depth for the Data1

`DOM_AskPrice(2) Data2`; will return the ask price for the 3rd level of depth for the Data2

DOM_AsksCount

Returns the number of ask depth levels available for a particular symbol.

Usage

`DOM_AsksCount` [`Data(`*`N`*`)`]

where:

(*`N`*) - is the number of the data series

Example

`DOM_AsksCount` will return 10 if 10 ask levels of the market depth are available for the Data1

`DOM_AsksCount Data2` will return 6 if 6 ask levels of the market depth are available for the Data2

DOM_AskSize

Returns the ask size for the certain depth level of a particular symbol.

Usage

`DOM_AskSize(`*`num`*`)` [`Data(`*`N`*`)`]

where:

(*`num`*) - is the number of depth level

(*`N`*) - is the number of the data series

Example

`DOM_AskSize(2)` will return 1500 if the ask size for the 3rd level of depth on Data1 is 1500

`DOM_AskSize(0) Data2` will return 750 if the ask size for the 1st level of depth on Data2 is 750

DOM_BidPrice

Returns bid price for the certain depth level of a particular symbol.

Usage

`DOM_BidPrice(`*`num`*`)` [`Data(`*`N`*`)`]

where:

(*num*) - is the number of depth level

(*N*) - is the number of the data series

Example

`DOM_BidPrice(4)`; will return the bid price for the 5th level of depth for the Data1

`DOM_BidPrice(2) Data2`; will return the bid price for the 3rd level of depth for the Data2

DOM_BidsCount

Returns the number of bid depth levels available for a particular symbol.

Usage

`DOM_BidsCount` [`Data(`*`N`*`)`]

where:

(*`N`*) - is the number of the data series

Example

`DOM_BidsCount` will return 10 if 10 bid levels of the market depth are available for the Data1

`DOM_BidsCount Data2` will return 6 if 6 bid levels of the market depth are available for the Data2

DOM_BidSize

Returns the bid size for the certain depth level of a particular symbol.

Usage

`DOM_BidSize(`*`num`*`)` [`Data(`*`N`*`)`]

where:

(*num*) - is the number of depth level

(*N*) - is the number of the data series

Example

`DOM_BidSize(2)` will return 1500 if the bid size for the 3rd level of depth on Data1 is 1500

`DOM_BidSize(0) Data2` will return 750 if the bid size for the 1st level of depth on Data2 is 750

DOM_IsConnected

Returns a logical value indicating the availability of the market depth data; returns a value of `True` if the market depth data is available and a value of `False` if the market depth data is not available.

Usage

```
DOM_IsConnected
```

Example

Print the ask size for the 1st level of market depth if the market depth data is available

```
Variables:

var0(0);

If DOM_IsConnected then

var0 = DOM_AskPrice(0);

Print(var0);
```

Array_Compare

Compares a specified range of elements of the specified one-dimensional source array, starting at the specified source index, to the same range of elements of the specified one-dimensional destination array, starting at the specified destination index.

Source and destination can be the same array as well as two different arrays.

For numerical arrays, the numerical values for each pair of elements are compared. For string arrays, the ASCII values of the string characters, with the first character of the string being the most significant, are compared for each pair of elements. For true/false arrays, the logical values of each pair of elements are compared, with the value of true considered to be greater than the value of false.

Returns a value of 0 if each respective pair of elements compared as equal; a value of 1 if, for the first pair that was not equal, the value in the source range was greater then the value in the destination range; and a value of -1 if, for the first pair that was not equal, the value in the source range was less then the value in the destination range.

Usage

`Array_Compare(`*`SourceArray`*,*SourceIndex*,*`DestinationArray`*,*DestinationIndex*,*Nu*

Where: *`SourceArray`* - an expression specifying the name of the source array
`DestinationArray` - an expression specifying the name of the destination ar
`SourceIndex` - a numerical expression specifying the starting index for the s
`DestinationIndex` - a numerical expression specifying the starting index fc
array
`NumberOfElements` - a numerical expression specifying the number of elem

Return

`0` - each respective pair of elements compared as equal
`1` - for the first pair compared that was not equal, the value in the source range was
value in the destination range
`-1` - for the first pair compared that was not equal, the value in the source range was
value in the destination range

Example

Assign a value, indicating the result of comparing two-element segments, of Array1, beginning at the index of 4, and of Array2, beginning at the index of 6, to Value1 variable:

```
Value1=Array_Compare(Array1,4,Array2,6,2);
```

Assign a value, indicating the result of comparing two-element segments that begin at the index of 4 and at the index of 6 within the same array, to Value1 variable:

```
Value1=Array_Compare(Array1,4,Array1,6,2);
```

Array_Contains

Will return True/False value depending on whether specific element is contained in the one-dimensional array, or not.

Usage

`Array_Contains(`*`ArrayName`*`,`*`Value`*`)`

Where: *`ArrayName`* - an expression specifying the name of an numerical/string/bool array.
`Value` - a numerical/string/bool expression specifying the value to search.

Example

Assign a true/false value, indicating that value 4 is contained in the numeric array Array1, to Condition1 variable:

```
Condition1=Array_Contains(Array1,4);
```

Array_Copy

Copies a specified number of elements from the specified one-dimensional source array, starting at the specified source index; the elements are copied to the specified one-dimensional destination array, starting at the specified destination index.

Source and destination can be the same array as well as two different arrays.

Usage

`Array_Copy(`*`SourceArray`*,*SourceIndex*,*`DestinationArray`*,*DestinationIndex*,*Numbe*

Where: *`SourceArray`* - an expression specifying the name of the source array
`DestinationArray` - an expression specifying the name of the destination ar
`SourceIndex` - a numerical expression specifying the starting index for the s
`DestinationIndex` - a numerical expression specifying the starting index fc
destination array
`NumberOfElements` - a numerical expression specifying the number of elem

Example

Copy from Array1 the two-element segment beginning at the index of 4, to the Array2, beginning at the index of 6:

```
Array_Copy(Array1,4,Array2,6,2);
```

Copy from Array1 the two-element segment beginning at the index of 4, to the same array, beginning at the index of 6:

```
Array_Copy(Array1,4,Array1,6,2);
```

Array_GetBooleanValue

Returns the Boolean value with a certain index from numerical array.

Usage

```
Array_GetBooleanValue
```

Example

Assign the second Boolean value of the "arr" array to Value1 variable:

```
Array: arr[10](0);
Value1 = Array_GetBooleanValue (arr, 2);
```

Array_GetFloatValue

Returns the float value with a certain index from numerical array.

Usage

```
Array_GetFloatValue
```

Example

Assign the second float value of arr array to Value1 variable:

```
Array: arr[10](0);
Value1 = Array_GetFloatValue (arr, 2);
```

Array_GetIntegerValue

Returns the integer value with a certain index from numerical array.

Usage

```
Array_GetIntegerValue
```

Example

Assign the second float value of "arr" array to Value1 variable:

```
Array: arr[10](0);
Value1 = Array_GetIntegerValue (arr, 2);
```

Array_GetMaxIndex

Returns a numerical value indicating the maximum index of a one-dimensional array.

Array indexes start at 0, and array size is equal to the value of the maximum index plus one.

Usage

`Array_GetMaxIndex(`*`ArrayName`*`)`

Where: *`ArrayName`* - an expression specifying the name of an array

Example

Assign a value that indicates the maximum index of Array1 to the MaxIndex variable:

```
MaxIndex=Array_GetMaxIndex(Array1);
```

Array_GetStringValue

Returns the string value with a certain index from numerical array.

Usage

```
Array_GetStringValue
```

Example

Assign the second float value of "arr" array to Value1 variable:

```
Array: arr[10](0);
Value1 = Array_GetStringValue (arr, 2);
```

Array_GetType

Returns a numerical value indicating the type of the specified array.

Usage

`Array_GetType(`*`ArrayName`*`)`

Where: *`ArrayName`* - an expression specifying the name of an array

Return

`2` - a true/false array
`3` - a string array
`7` - a double-precision numerical array

Example

Assign a value, indicating the type of Array1, to Value1 variable:

```
Value1=Array_GetType(Array1);
```

Array_IndexOf

Will return index of a specific element in one-dimensional array. Will return -1 if the array will not contain the specific element.

Usage

`Array_IndexOf(`*`ArrayName`*`,`*`Value`*`)`

Where: *`ArrayName`* - an expression specifying the name of an numerical/string/bool array.
`Value` - a numerical/string/bool expression specifying the value to search for.

Example

Assign an index of value 4 to Value1 variable in the numeric array Array1:

```
Value1=Array_IndexOf(Array1,4);
```

Array_SetBooleanValue

Sets the Boolean value for a certain index of numerical array.

Usage

```
Array_SetBooleanValue
```

Example

Set the first and third values of "arr" array to TRUE:

```
Array: arr[10](False);
Array_SetBooleanValue (arr, 1, True);
Array_SetBooleanValue (arr, 1, True);
```

Array_SetFloatValue

Sets the float value for a certain index of numerical array.

Usage

```
Array_SetFloatValue
```

Example

Set the first and third values of "arr" array to 1.789:

```
Array: arr[10](False);
Array_SetFloatValue (arr, 1, 1.789000352);
Array_SetFloatValue (arr, 3, 1.789000352);
```

Array_SetIntegerValue

Sets the integer value for a certain index of numerical array.

Usage

```
Array_SetIntegerValue
```

Example

Set the first and third values of "arr" array to 500:

```
Array: arr[10](False);
Array_SetIntegerValue (arr, 1, 500);
Array_SetIntegerValue (arr, 3, 500);
```

Array_SetMaxIndex

Resizes a declared dynamic array to a specified number of elements; returns a value of `True` to indicate a successful resize.

An array can be resized to a larger or to a smaller number of elements; any elements added to an array will be assigned the initial value that was specified at array declaration.

Usage

`Array_SetMaxIndex(`*`ArrayName`*`,`*MaxIndex*`)`

Where: *`ArrayName`* - an expression specifying the name of an array to be resized
`MaxIndex` - a numerical expression specifying the maximum index of the array*

*Array indexes start at 0, and array size is equal to the value of *`MaxIndex`* plus one.

Return

`True` - resize sucessfull

`False` - resize failed

Example

Resize the dynamic array Array1 to 10 elements by specifying a maximum index value of 9:

```
Array_SetMaxIndex(Array1,9);
```

Assign a value that indicates a status of the resize of Array1 to the ResizeReport variable:

```
ResizeReport=Array_SetMaxIndex(Array1,9);
```

A value of True will indicate a successful resize, and a value of False will indicate a failed resize.

Array_SetStringValue

Sets the string value for a certain index of numerical array.

Usage

```
Array_SetStringValue
```

Example

Set the first and third values of "arr" array to "array string":

```
Array: arr[10](False);
Array_SetStringValue (arr, 1, "array string");
Array_SetStringValue (arr, 3, "array string");
```

Array_SetValRange

Assigns a specified value to each element within a specified range, of the specified one-dimensional array.

Usage

`Array_SetValRange(`*`ArrayName`*,*StartIndex*,*EndIndex*,*Value*)

Where: *`ArrayName`* - an expression specifying the name of an array
`StartIndex` - a numerical expression specifying the starting index for the range
`EndIndex` - a numerical expression specifying the ending index for the range
`Value` - a value to be assigned to each element within the range

Example

Assign a value of True to each element within a segment, beginning at index 4 and ending at index 6, of Array1:

```
Array_SetValRange(Array1,4,6,True);
```

Array_Sort

Sorts, in either ascending or descending order, the range of elements, specified by the starting and ending indexes, of the specified one-dimensional array.

For numerical arrays, the elements are sorted according to the values they contain. For string arrays, the elements are sorted according to the ASCII values of the string characters, with the first character of the string being the most significant. For sorting true/false array elements, the value of true is considered to be greater than the value of false.

Usage

`Array_Sort(`*`ArrayName`*`,`*StartIndex*`,`*EndIndex*`,`*`SortOrder`*`)`

Where: *`ArrayName`* - an expression specifying the name of an array
`StartIndex` - a numerical expression specifying the starting index for the range
`EndIndex` - a numerical expression specifying the ending index for the range
`SortOrder` - a true/false value specifying the sort order; true specifies ascending order, and false specifies descending order

Example

Sort the elements in the segment of Array1 that begins at the index of 4 and ends at the index of 6, in ascending order:

```
Array_Sort(Array1,4,6,True);
```

Array_Sum

Returns a sum of the values contained in a range of elements, specified by the starting and ending indexes, of the specified one-dimensional array; returns the number of true elements if the array contains true/false values; returns a value of 0 if the array contains string values.

Usage

`Array_Sum(`*`ArrayName`*`,`*StartIndex*`,`*EndIndex*`)`

Where: *`ArrayName`* - an expression specifying the name of an array
`StartIndex` - a numerical expression specifying the starting index for the range
`EndIndex` - a numerical expression specifying the ending index for the range

Example

Assign a value, indicating a sum of the values contained in the segment of Array1 that begins at the index of 4 and ends at the index of 6, to Value1 variable:

```
Value1=Array_Sum(Array1,4,6);
```

Fill_Array

Assigns a specified value to each element of the specified one-dimensional array.

Usage

`Fill_Array(`*`ArrayName`*`,`*`Value`*`)`

Where: *`ArrayName`* - an expression specifying the name of an array
`Value` - a value to be assigned to each element of the array

Example

Assign a value of True to each element of Array1:

```
Fill_Array(Array1,True);
```

BaseDataNumber

Returns a numerical value indicating the Data Number of the series that the study is applied to.

Usage

```
BaseDataNumber
```

Example

Assign a value, indicating the Data Number of the series that the study is applied to, to Value1 variable:

```
Value1=BaseDataNumber;
```

CurrentDataNumber

Returns a numerical value indicating the Data Number of the series that the function is being calculated on.

Usage

```
CurrentDataNumber
```

Example

Assign a value, indicating the Data Number of the series that the function is being calculated on, to Value1 variable:

```
Value1=CurrentDataNumber;
```

ExecOffset

Returns a numerical value indicating the function execution offset in bars.

Usage

```
ExecOffset
```

Example

Assign a value, indicating the function execution offset, to Value1 variable:

```
Value1=ExecOffset;
```

GetAppInfo

Returns a numerical value, representing the specified attribute of the calling application.

Usage

`GetAppInfo(Attribute)`

Parameters

`aiApplicationType` - identifies the calling application:

`0` = Unknown

`1` = Chart

`2` = Scanner

`aiBarSpacing` - specifies return of the number of spaces between the bars on a chart GetAppInfo will return a value, indicating the bar spacing of a chart.

`aiCalcReason` - returns a numerical value, indicating the reason of calculation initialization (i.e. returns 0 (CalcReason_Default) when calculation was triggered by a new bar/tick):

`0 (CalcReason_default)` - calculation is to be initialized when the new bar/tick appeared.

`1 (CalcReason_mouseLClick)` calculation is to be initialized after left-click on the chart.

`2 (CalcReason_mouseRClick)` - calculation is to be initialized after right-click on the chart.

`3 (CalcReason_timer)` calculation is to be initialized after expiration of RecalcLastBarAfter timeout.

`4 (CalcReason_MPChange)` calculation is to be initialized after market position for the instrument has been changed (for signals only).

`5 (CalcReason_OrderFilled)` calculation is to be initialized after order filled event (for signals only).

`aiHighestDispValue` - specifies return of the highest price value that could be displayed in the current chart window GetAppInfo will return the highest price value that could be displayed on a chart.

`aiLowestDispValue` - specifies return of the lowest price value that could be displayed in the current chart window GetAppInfo will return the lowest price value that could be displayed on a chart.

`aiLeftDispDateTime` - specifies return of the date and time of the leftmost bar displayed in the current chart window GetAppInfo will return the DateTime value of the leftmost bar displayed on a chart; the integer portion of the DateTime value specifies the number of days since January 1st, 1900, and the fractional portion of the DateTime value specifies the fraction of the day since midnight.

`aiRightDispDateTime` - specifies return of the date and time of the rightmost bar displayed in the current chart window GetAppInfo will return the DateTime value of the rightmost bar displayed on a chart; the integer portion of the DateTime value specifies the number of days since January 1st, 1900, and the fractional portion of the DateTime value specifies the fraction of the day since midnight.

`aiRow` - identifies the symbol's row number in Scanner; returns a positive non-zero value from a Scanner application else returns 0.

`aiSpaceToRight` - specifies return of the right margin, in bars, of the current chart window GetAppInfo will return the right margin, in bars, of a chart.

`aiOptimizing` - specifies return of a numerical value, indicating whether the calling application is currently performing an optimization GetAppInfo will return a value of 1 only if the calling application is currently performing an optimization, and a value of 0 in all other cases.

`aiStrategyAuto` - specifies return of a numerical value, indicating whether the calling application is using Automated Trade Execution GetAppInfo will return a value of 1 only if the calling application is using Automated Trade Execution, and a value of 0 in all other cases.

`aiStrategyAutoConf` - specifies return of a numerical value, indicating whether the calling application is using Automated Trade Execution with order confirmation turned off GetAppInfo will return a value of 0 only if the calling application is using Automated Trade Execution with order confirmation turned off, and a value of 1 in all other cases.

`aiIntrabarOrder` - specifies return of a numerical value, indicating whether the calling application is running the signal with intra-bar order generation turned on GetAppInfo will return a value of 1 only if the calling application is running the signal with intra-bar order generation turned on, and a value of 0 in all other cases.

`aiAppId` - specifies return of a numerical value, used to identify the calling application GetAppInfo will return a unique non-zero integer identifying the calling application.

`aiRealTimeCalc` - specifies return of a numerical value, indicating whether the calling applications calculations are based on real-time data GetAppInfo will return a value of 1 only if the calling application's calculations are based on real-time data, and a value of 0 in all other cases.

`aichartshiftpercent` - returns the ChartShift value in percents from Format Window -> X - Time Scale. The value is updated on the fly.

Example

`GetAppInfo(aiBarSpacing)` will return a value, indicating the bar spacing of a chart

`GetAppInfo(aiStrategyAutoConf)` will return a value of 0 if the calling application is using Automated Trade Execution with order confirmation turned off; otherwise, will return a value of 1

`GetAppInfo(aiRealTimeCalc)` will return a value of 1 if the calling applications calculations are based on real-time data; otherwise, will return a value of 0

```
[ProcessMouseEvents = true];
switch (getappinfo(aicalcreason)) begin
  case CalcReason_MouseLClick : if MouseClickCtrlPressed then begin
  var: var0(0), var1(0);
    repeat
      if 0 = var0 then begin
      var0 = MouseClickDateTime;
      break;
      end;
    until(false);
  end;
end;
var1 = datetime2eltime(var0);
print("Time of the Bar = ", var1);
```

Will return the time of the bar after left click on it pressing Ctrl button on the keyboard.

GetCDRomDrive

Returns a string expression of the drive letter for the first CD-ROM drive detected.

Usage

```
GetCDRomDrive
```

Example

```
Variables: Drive("D");
Drive = GetCDRomDrive;
```

Will set the variable `Drive` equal to the first CD-ROM drive detected.

GetCountry

Returns the locale name, corresponding to the Standards and formats setting selected in the Regional Options settings of the Windows XP Control Panel.

Usage

```
GetCountry
```

Example

`GetCountry` will return a string expression "United States" for the setting "English (United States)"

GetCurrency

Returns the currency symbol selected in the Regional Options settings of the Windows XP Control Panel.

Usage

```
GetCurrency
```

Example

`GetCurrency` will return the "$" symbol for the US Dollar

GetUserID

Returns a unique identification number (UserID) for the PC that the study is being run on.

UserID number is used for study protection.

Usage

```
GetUserID
```

Example

Assign a value, indicating the UserID, to Value1 variable:

```
Value1=GetUserID;
```

GetUserName

Returns MultiCharts registration name (UserName) for MultiCharts installation on the PC that the study is being run on.

UserName is used for study protection.

Usage

```
GetUserName
```

Example

Assign a value, indicating the UserName, to UserName variable:

```
Var: UserName("");

UserName = GetUserName;
```

MaxBarsBack

Returns a numerical value indicating the Maximum Bars Back setting for the study.

All studies based on past data use a certain number of bars for their calculations. The number of bars is called *Maximum number of bars a study will reference*, or Maximum Bars Back.

Usage

```
MaxBarsBack
```

Example

Assign a value, indicating the Maximum Bars Back setting for the study, to Value1 variable:

```
Value1=MaxBarsBack;
```

MaxBarsForward

Returns a numerical value indicating the size, in bars, of the right margin on the chart.

Right margin is used by the studies that visualize the predicted price movement.

Usage

```
MaxBarsForward
```

Example

Assign a value, indicating the size of the right margin of the chart, to Value1 variable:

```
Value1=MaxBarsForward;
```

SetMaxBarsBack

Sets a numerical value indicating the Maximum Bars Back setting for the study.

All studies based on past data use a certain number of bars for their calculations.

The number of bars is called ***Maximum number of bars a study will reference***, or ***Maximum Bars Back***.

Usage

`SetMaxBarsBack(`*BarsBack*`)`

Where: *`BarsBack`* - numerical expression, specifying the number of bars back

Note

`BarsBack` can't be a negative value.

Example

Set a value, indicating the Maximum Bars Back setting for the study to 50:

```
SetMaxBarsBack(50);
```

Abort

Generates a run-time error and aborts the execution of the study.

Usage

```
Abort
```

Example

Abort the studys execution:

```
Abort;
```

CommandLine

Passes a string expression from script to command line.

Usage

`CommandLine`("Expression")

See the [list of supported expressions]() for the command line.

Examples

`CommandLine`(".rld");

Reloads the chart where the script is applied (reloads all chart where the same symbol is used).

`CommandLine`(".at_toggle");

Turns on/off auto trading on the chart where the script is applied (if it is used to turn auto trading on, the popped up confirmation window cannot be skipped and a manual click on "OK" is required).

`CommandLine`(".csy dnum=1, name=AUD/CHF, df=LMAX");

Changes the symbol plotted as data series 1 to AUD/CHF and the data source to LMAX on the chart where the script is applied.

`CommandLine`(".isy name=@ES#, df=IQFeed, res=1 min, desc=E-MINI S&P; 500 MARCH 2013, from=12/31/2012, to=5/10/2013");

Inserts 1 minute graph of S&P; mini 500 from IQFeed starting from 31th of December 2012 up to 10th of May 2013 as additional data series to the chart where the script is applied.

`CommandLine`(".iid name=MACD, base=1, bref=100");

Applies MACD indicator to the data series 1 with MaxBarsBack parameter = 100 to the chart where the script is applied.

`CommandLine(".rld int 2 weeks");`

Reloads 2 weeks of data on the chart where the script is applied.

fpcExactAccuracy

Constant, used in combination with SetFPCompareAccuracy to designate the floating point compare tolerance value of 0.00; can be substituted by a numerical value of 5.

Usage

```
SetFPCompareAccuracy(fpcExactAccuracy)
```

or:

```
SetFPCompareAccuracy(5)
```

Example

Set the floating point compare accuracy to Exact:

```
SetFPCompareAccuracy(fpcExactAccuracy);
```

Set the floating point compare accuracy to Exact:

```
SetFPCompareAccuracy(5);
```

fpcHighAccuracy

Constant, used in combination with SetFPCompareAccuracy to designate the floating point compare tolerance value of 2.2204460492503131e-14; can be substituted by a numerical value of 3.

Usage

```
SetFPCompareAccuracy(fpcHighAccuracy)
```

or:

```
SetFPCompareAccuracy(3)
```

Example

Set the floating point compare accuracy to High:

```
SetFPCompareAccuracy(fpcHighAccuracy);
```

Set the floating point compare accuracy to High:

```
SetFPCompareAccuracy(3);
```

fpcLowAccuracy

Constant, used in combination with SetFPCompareAccuracy to designate the floating point compare tolerance value of 2.2204460492503131e-10; can be substituted by a numerical value of 1.

Usage

```
SetFPCompareAccuracy(fpcLowAccuracy)
```

or:

```
SetFPCompareAccuracy(1)
```

Example

Set the floating point compare accuracy to Low:

```
SetFPCompareAccuracy(fpcLowAccuracy);
```

Set the floating point compare accuracy to Low:

```
SetFPCompareAccuracy(1);
```

fpcMedAccuracy

Constant, used in combination with SetFPCompareAccuracy to designate the floating point compare tolerance value of 2.2204460492503131e-12; can be substituted by a numerical value of 2.

Usage

```
SetFPCompareAccuracy(fpcMedAccuracy)
```

or:

```
SetFPCompareAccuracy(2)
```

Example

Set the floating point compare accuracy to Medium:

```
SetFPCompareAccuracy(fpcMedAccuracy);
```

Set the floating point compare accuracy to Medium:

```
SetFPCompareAccuracy(2);
```

fpcVeryHighAccuracy

Constant, used in combination with SetFPCompareAccuracy to designate the floating point compare tolerance value of 2.2204460492503131e-16; can be substituted by a numerical value of 4.

Usage

```
SetFPCompareAccuracy(fpcVeryHighAccuracy)
```

or:

```
SetFPCompareAccuracy(4)
```

Example

Set the floating point compare accuracy to Very High:

```
SetFPCompareAccuracy(fpcVeryHighAccuracy);
```

Set the floating point compare accuracy to Very High:

```
SetFPCompareAccuracy(4);
```

fpcVeryLowAccuracy

Constant, used in combination with SetFPCompareAccuracy to designate the floating point compare tolerance value of 2.2204460492503131e-8; can be substituted by a numerical value of 0.

Usage

```
SetFPCompareAccuracy(fpcVeryLowAccuracy)
```

or:

```
SetFPCompareAccuracy(0)
```

Example

Set the floating point compare accuracy to Very Low:

```
SetFPCompareAccuracy(fpcVeryLowAccuracy);
```

Set the floating point compare accuracy to Very Low:

```
SetFPCompareAccuracy(0);
```

RaiseRunTimeError

Generates a run-time error and displays the specified error message.

A run-time error will cause the execution of the study to be aborted.

Usage

`RaiseRunTimeError("`*`Message`*`")`

Where: *`Message`* - a string expression specifying the error message

Example

Generate a run-time error and display the message "Strategy Stopped":

```
RaiseRunTimeError("Strategy Stopped");
```

RecalcLastBarAfter

Initializes the calculation after expiration of the timeout, set in seconds. Note: The maximum recalculation frequency is 100 milliseconds (0.1 sec).

`RecalcLastBarAfter(`*`Timeout`*`)`

Where: *`Timeout`* - the number of seconds.

Example

`RecalcLastBarAfter(60)` will initiate new script calculation after one minute timeout since the last calculation.

ReCalculate

Initializes recalculation of the study. All the variables will be re-initialized.

The study will be recalculated from the first bar of the data series.

Usage

```
ReCalculate
```

Note

To avoid infinite looping of the script, use global variables with recalculation conditions.

Example

Set the condition for recalculation:

```
Var: ReCalcPersist recalc_once(True), vo10(0);

If LastBarOnChart Then Begin
  Print ("Last bar volume = ", Volume);

  If recalc_once Then Begin
    Print ("Recalculate study!");
    recalc_once = False
    ReCalculate;
  End;

End;
```

SetFPCompareAccuracy

Sets floating point compare accuracy by specifying the tolerance value to be used when floating point values are compared.

Two floating point values will be considered equal if "abs(Value1 Value2) <= ε", where ε is the tolerance value.

By default, the tolerance value is 2.22044604925031310e-012.

Usage

`SetFPCompareAccuracy`(*Accuracy*)

Parameters

`Accuracy` - an FPC constant or a numerical expression specifying a tolerance value as follows:

`fpcVeryLowAccuracy`	0	02.2204460492503131e-8
`fpcLowAccuracy`	1	12.2204460492503131e-10
`fpcMedAccuracy`	2	22.2204460492503131e-12 (Default)
`fpcHighAccuracy`	3	32.2204460492503131e-14
`fpcVeryHighAccuracy`	4	42.2204460492503131e-16
`fpcExact`	5	50.00

Example

Set floating point compare accuracy to High:

```
SetFPCompareAccuracy(3);
```

Set floating point compare accuracy to High:

```
SetFPCompareAccuracy(fpcHighAccuracy);
```

#Return

Returns the control from the study script by analogy with the corresponding C++ statement.

Does not return any value.

Usage

```
#Return;
```

Notes

Can be used in all types of studies.

AtCommentaryBar

This reserved word returns a value of `True` on the bar clicked by the user.

It will return a value of `False` for all other bars.

This allows you to optimize your trading strategies, analysis techniques, and functions for speed, as it will allow PowerLanguage to skip all commentary-related calculations for all bars except for the one where the commentary is requested.

Usage

```
AtCommentaryBar
```

Notes

The difference between `AtCommentaryBar` and `CommentaryEnabled` is that `CommentaryEnabled` returns a value of True for ALL bars when the Expert Commentary window is open, while the `AtCommentaryBar` returns a value of True only for the bar clicked.

Example

The following statements display a 50-bar average of the volume in the Expert Commentary window but avoid calculating this 50-bar average for every other bar of the chart:

```
If AtCommentaryBar Then

Commentary ("The 50-bar vol avg: ", Average (Volume, 50));
```

CheckCommentary

Returns `True` after left click on a chart with the Expert Commentary pointer on the specified bar.

Returns `False` if the pointer has not been inserted, or if the pointer was inserted on a different bar.

Usage

```
CheckCommentary
```

Example

If you only wanted code to be evaluated for the bar where the user had inserted the Expert Commentary Tool, you could use the following syntax:

```
If CheckCommentary Then Begin {Your Code Here}
End;
```

Commentary

This reserved word sends the expression (or list of expressions) to the Expert Commentary window for whatever bar is selected on the price chart.

Usage

```
Commentary ("My Expression");
```

Where `"My Expression"` is the numerical, text string or true/false expression that is to be sent to the Expert Commentary window.

You can send multiple expressions, commas must separate them.

Example

The following will result in the string "This is one line of commentary" being sent to the commentary window.

Any additional commentary sent will be placed on the same line.

```
Commentary ("This is one line of commentary");
```

CommentaryCL

This reserved word sends the expression (or list of expressions) to the Expert Commentary window for whatever bar is selected by the Expert Commentary pointer.

Usage

```
CommentaryCL("My Expression");
```

Where `"My Expression"` is a single or a comma-separated list of numeric, text string, or true/false expressions that are sent to the Expert Commentary window.

Example

The following will result in the string "This is one line of commentary" being sent to the commentary window.

Any additional commentary sent will be placed on the next line.

```
CommentaryCL("This is one line of commentary");
```

CommentaryEnabled

This reserved word returns a value of `True` only when the Expert Commentary window is open and Commentary has been requested.

This allows you to optimize your trading strategies, analysis techniques, and functions for speed, as it allows PowerLanguage to perform commentary-related calculations only when the Expert Commentary window is open.

Usage

```
CommentaryEnabled
```

Notes

The difference between `CommentaryEnabled` and `AtCommentaryBar` is that `CommentaryEnabled` returns a value of True for ALL bars when the Expert Commentary window is open, while the `AtCommentaryBar` returns a value of True only for the bar clicked with the Expert Commentary pointer.

Example

`CommentaryEnabled` will return True if the Analysis Commentary Tool has been applied to the chart.

#BeginCmtry

The statements between this compiler directive and the reserved word `#End` are evaluated only when the Expert Commentary tool is used to select a bar on a chart or a cell in a grid.

The reserved word `#End` must be used with this reserved word.

Usage

```
#BeginCmtry

Commentary("The indicator value here is " + NumtoStr (Plot1, 2));

#End;
```

Notes

All statements between the `#BeginCmtry` and `#End` are ignored, including calculation of MaxBarsBack, unless commentary is generated.

Example

An indicator that calculates the 10-bar momentum of the closing price needs ten bars in order to start plotting results.

If commentary is added to this indicator and the commentary uses a 50-bar average of the volume, then the MaxBarsBack setting is increased to fifty.

However, the 50-bar average is only used for the commentary, so there is no need to have the indicator wait fifty bars before giving results unless Commentary is requested.

To have the indicator plot after 10 bars and ignore the 50-bar requirement, the indicator can be written as follows:

```
Plot1 (Close - Close [10], "Momentum");
```

```
#BeginCmtry;
If Close - Close [10] > 0 Then
Commentary ("Momentum is positive, ")
Else
Commentary ("Momentum is negative, ");
If Volume > Average (Volume, 50) Then
Commentary (" and volume is greater than average.")
Else
Commentary (" and volume is lower than average.");
#End;
```

This indicator plots the momentum and the commentary states whether the momentum is positive or negative, and if the volume is over or under the 50-bar average of the volume.

When the indicator is applied without using commentary, it will require only 10 bars to start calculating.

When commentary is requested, the indicator is recalculated, the statements within the compiler directives are evaluated, and the new minimum number of bars required is 50.

Any series functions within these reserved words are also ignored.

AbsValue

Returns the absolute value of the specified numerical expression.

Usage

`AbsValue(`*Value*`)`

Where: *`Value`* - a numerical expression

Example

`AbsValue(45.275)` will return a value of 45.275

`AbsValue(-1385)` will return a value of 1385

ArcTangent

Returns the arctangent value, in degrees, of the specified numerical expression.

Usage

`ArcTangent(`*Value*`)`

Where: *`Value`* - a numerical expression

Example

`ArcTangent(2.318)` will return a value of 66.66

AvgList

Returns the average value of the specified numerical expressions.

Usage

`AvgList`(*Value1*,*Value2*,*Value3*, *etc*.)

Where: *`Value1, Value2, Value3, etc.`* - numerical expressions

Example

`AvgList(45,40,0,35)` will return a value of 30

`AvgList(-40,20)` will return a value of -10

Ceiling

Returns the smallest integer greater than or equal to the specified numerical expression.

Usage

`Ceiling(`*Value*`)`

Where: *`Value`* - a numerical expression

Example

`Ceiling(9.1)` will return a value of 10

`Ceiling(-2.85)` will return a value of -2

Cosine

Returns the cosine value for an angle of the specified number of degrees.

Usage

`Cosine`(*Value*)

Where: *`Value`* - a numerical expression, specifying the number of degrees in the angle

Example

`Cosine(60)` will return a value of 0.5

Cotangent

Returns the cotangent value for an angle of the specified number of degrees.

Usage

`Cotangent(`*Value*`)`

Where: *`Value`* - a numerical expression, specifying the number of degrees in the angle

Example

`Cotangent(30)` will return a value of 1.732

ExpValue

Returns the exponential value of the specified numerical expression.

Usage

`ExpValue(`*Value*`)`

Where: *`Value`* - a numerical expression

Example

`ExpValue(2.3)` will return a value of 9.0250

Floor

Returns the greatest integer less than or equal to the specified numerical expression.

Usage

`Floor(`*Value*`)`

Where: *`Value`* - a numerical expression

Example

`Floor(9.1)` will return a value of 9

`Floor(-2.85)` will return a value of -3

FracPortion

Returns the fractional portion of the specified numerical expression while retaing the sign.

Usage

`FracPortion`(*Value*)

Where: *`Value`* - a numerical expression

Example

`FracPortion`(`-45.275`) will return a value of -0.275

`FracPortion`(`1385`) will return a value of 0

IntPortion

Returns the integer portion of the specified numerical expression while retaing the sign.

Usage

`IntPortion`(*Value*)

Where: *`Value`* - a numerical expression

Example

`IntPortion`(`-45.75`) will return a value of -45

`IntPortion`(`1385`) will return a value of 1385

Log

Returns the natural logarithm of the specified numerical expression.

Usage

`Log`(*Value*)

Where: *`Value`* - a numerical expression

Example

`Log(25)` will return a value of 3.2189

MaxList

Returns the value of the greatest of the specified numerical expressions.

Usage

`MaxList`(*Value1*,*Value2*,*Value3*, *etc*.)

Where: *`Value1, Value2, Value3, etc.`* - numerical expressions

Example

`MaxList(-5,0,12,7)` will return a value of 12

MaxList2

Returns the second highest value of the specified numerical expressions.

Usage

`MaxList2`(*Value1*,*Value2*,*Value3*, *etc.*)

Where: *`Value1, Value2, Value3, etc.`* - numerical expressions

Example

`MaxList2`(`-5`,`0`,`12`,`7`) will return a value of 7

MinList

Returns the lowest value of the specified numerical expressions.

Usage

`MinList`(*Value1*,*Value2*,*Value3*, *etc*.)

Where: `Value1,Value2,Value3, etc.` - numerical expressions

Example

`MinList(-5,0,12,7)` will return a value of -5

MinList2

Returns the second lowest value of the specified numerical expressions.

Usage

`MinList2`(*Value1*,*Value2*,*Value3*, *etc.*)

Where: *`Value1`*,*`Value2`*,*`Value3`*, *`etc.`* - numerical expressions

Example

`MinList(-5,0,12,7)` will return a value of 0

Mod

Returns the remainder from dividing one specified numerical expression by another.

Usage

`Mod`(*Dividend*,*Divisor*)

Where: `Dividend` - a numerical expression
`Divisor` - a numerical expression

Example

`Mod(25,7)` will return a value of 4

Neg

Returns the negative absolute value of the specified numerical expression.

Usage

`Neg`(*Value*)

Where: *`Value`* - a numerical expression

Example

`Neg(12)` will return a value of -12

`Neg(-7)` will return a value of -7

NthMaxList

Returns the Nth highest value of the specified numerical expressions.

Usage

`NthMaxList(`*N*,*Value1*,*Value2*,*Value3*, *etc.*)

Where: *`N`* - a numerical expression, indicating the rank of the value to be returned
`Value1`,*`Value2`*,*`Value3`*, *`etc.`* - numerical expressions

Example

`NthMaxList(4,-15,-5,0,6,12)` will return a value of -5

NthMinList

Returns the Nth lowest value of the specified numerical expressions.

Usage

`NthMinList(`*N*,*Value1*,*Value2*,*Value3*, *etc.*)

Where: *`N`* - a numerical expression, indicating the rank of the value to be returned
`Value1, Value2, Value3, etc.` - numerical expressions

Example

`NthMinList(4,-15,-5,0,6,12)` will return a value of 6

Pos

Same as `AbsValue`

Power

Returns the value of one specified numerical expression to the power of another.

Usage

`Power`(*Base*,*Exponent*)

Where: *`Base`* - a numerical expression
`Exponent` - a numerical expression

Example

`Power(5,3)` will return a value of 125

Random

Returns a pseudo-random number between 0 and the value of the specified numerical expression.

Usage

`Random(`*Value*`)`

Where: *`Value`* - a numerical expression

Example

`Random(1.25)` will return a random value anywhere between 0 and 1.25

`Random(-10)` will return a random value anywhere between -10 and 0

Round

Returns the value of one specified numerical expression rounded to the number of decimal places specified by another.

Usage

`Round(`*Value*,*Precision*`)`

Where: *`Value`* - a numerical expression
`Precision` - a numerical expression

Example

`Round(1.237,2)` will return a value of 1.24

`Round(-5.7744,3)` will return a value of 5.774

Sign

Returns a numerical value, indicating the sign of the value of the specified numerical expression.

A value of 1 is returned for a positive value, -1 is returned for a negative value, and 0 is returned for the value of 0.

Usage

`Sign(`*Value*`)`

Where: *`Value`* - a numerical expression

Example

`Sign(5)` will return a value of 1

`Sign(-2.85)` will return a value of -1

`Sign(0)` will return a value of 0

Sine

Returns the sine value for an angle of the specified number of degrees.

Usage

`Sine`(*Value*)

Where: *`Value`* - a numerical expression, specifying the number of degrees in the angle

Example

`Sine(30)` will return a value of 0.5

Square

Returns the square of the value of the specified numerical expression.

Usage

`Square(`*Value*`)`

Where: *`Value`* - a numerical expression

Example

`Square(2.5)` will return a value of 6.25

SquareRoot

Returns the square root of the value of the specified numerical expression.

Usage

`SquareRoot(`*Value*`)`

Where: *`Value`* - a numerical expression

Example

`SquareRoot(57.73)` will return a value of 7.598

SumList

Returns a sum of the values of the specified numerical expressions.

Usage

`SumList`(*Value1*,*Value2*,*Value3*, *etc.*)

Where: *`Value1`*, *`Value2`*, *`Value3`*, *`etc.`* - numerical expressions

Example

`SumList(45,-20,0,35)` will return a value of 60

Tangent

Returns the tangent value for an angle of the specified number of degrees.

Usage

`Tangent(`*`Value`*`)`

Where: *`Value`* - a numerical expression, specifying the number of degrees in the angle

Example

`Tangent(40)` will return a value of 0.839

Ago

Used in combination with `Bar` or `Bars` and a numerical expression to reference the bar a specified number of bars back from the current bar.

Bars Ago can also be specified by using the bar offset notation that consists of a numerical expression enclosed in square brackets.

Usage

```
N Bars Ago
```

or:

```
[N]
```

Where: `N` - a numerical expression specifying the number of bars back to reference

Example

Plot the closing price of the previous bar:

```
Plot1(Close Of 1 Bar Ago, "Previous bar's close");
```

Plot the closing price of two bars ago:

```
Plot1(Close[2], "Close 2 bars ago");
```

Bar

Used in combination with `This`, `Next`, or `Ago` to reference a specific bar.

Usage

```
Bar
```

Example

`Close Of 1 Bar Ago` will return the closing price of the previous bar

Buy a user-set number of shares on close of this bar:

```
Buy This Bar On Close;
```

Buy a user-set number of shares on open of next bar:

```
Buy Next Bar On Open;
```

Bars

Same as `Bar`

Contract

Same as `Contracts`

Contracts

Used in strategy entry or exit statements in combination with a numerical expression to specify the number of contracts or shares to trade.

Usage

```
TradeSize Contracts
```

Where: *`TradeSize`* - a numerical expression, specifying the number of contracts or shares

Example

Buy 2 contracts at Market price on open of next bar:

```
Buy 2 Contracts Next Bar At Market;
```

Market

Used in strategy entry or exit statements to specify a Market price for an entry or an exit.

A Market Buy order will execute at the current ask price and a Maret Sell order will execute at the current bid price.

Usage

```
At Market
```

Where: `At` is a skip word and can be omitted

Example

Buy a user-set number of shares at Market price on open of next bar:

```
Buy Next Bar At Market;
```

Next

Used in combination with `Bar` to reference the next bar.

Usage

```
Next Bar
```

Example

Buy a user-set number of shares at Market price on open of next bar:

```
Buy Next Bar At Market;
```

This

Used in combination with `Bar` to reference the current bar.

Usage

```
This Bar
```

Example

Buy a user-set number of shares on close of this bar:

```
Buy This Bar On Close;
```

Today

Retained for backward compatibility; replaced with `This Bar`

Yesterday

Retained for backward compatibility.

MouseClickBarNumber

After a mouse click on the bar, returns the numerical value indicating the bar number from the beginning of the data series.

Usage

```
MouseClickBarNumber
```

Example

`MouseClickBarNumber` - will return 250 after a mouse click on the 250th bar on the chart from the beginning of the data series.

MouseClickCtrlPressed

Returns `True` if the Ctrl button is pressed on the keyboard simultaneously with a mouse click.

Usage

```
MouseClickCtrlPressed
```

MouseClickDataNumber

Returns the numerical value indicating the data series number after a mouse click on the data series.

Usage

```
MouseClickDataNumber
```

Example

`MouseClickDataNumber` - will return 1 after left-click on the main chart with the main data series.

`MouseClickDataNumber` - will return 2 after left-click on the sub-chart with the second data series.

MouseClickDateTime

Returns a double-precision decimal value in Julian (OLE) date-time format indicating the closing date of the bar after a click on the bar.

Usage

`MouseClickDateTime`

Example

`MouseClickDateTime` - will return a value of 39449.65625000 for 3:45 PM.

MouseClickPrice

Returns a numerical value indicating price level of the mouse pointer position after a click on the chart.

Usage

`MouseClickPrice`

Example

`MouseClickPrice` - will return 139.60 after a click on the chart on the 139.60 price level.

MouseClickShiftPressed

Returns `True` if the Shift button is pressed on the keyboard simultaneously with a mouse click.

Usage

```
MouseClickShiftPressed
```

PlaySound

Plays the specified wave (.wav) sound file.

Usage

`PlaySound("`*`PathFilename`*`")`

Where: *`PathFilename`* - a string expression specifying the path and filename of the wave file to be played

Example

Play ding.wav sound file located in the root directory of the C: hard drive:

```
PlaySound("C:\ding.wav");
```

ClearDebug

Clears the PowerLanguage Editor Output Log.

Usage

```
ClearDebug
```

Example

`ClearDebug;` will clear the PowerLanguage Editor Output Log

ClearPrintLog

Same as `ClearDebug`

File

Used in Print statements to specify an ASCII file as the output location; must precede the expressions to be printed and be followed by a comma. If the specified file does not exist, the file will be created.

Usage

`File("`*`PathFilename`*`")`

Where: *`PathFilename`* - a string expression specifying the path and filename

Example

`Print(File("C:\test.txt"),CurrentDate,CurrentTime);` will save the output of `CurrentDate` and `CurrentTime` to the test.txt file in the root directory of the C: hard drive

FileAppend

Appends the specified string expression to the end of the specified ASCII file; if the specified file does not exist, the file will be created.

Usage

`FileAppend("PathFilename","StringExpression")`

Where: *`PathFilename`* - a string expression specifying the path and filename
`StringExpression` - the string expression to be appended to a file

Example

`FileAppend("C:\test.txt","Appended Text");` will append the string expression "Appended Text" to the end of the test.txt file in the root directory of the C: hard drive

FileDelete

Deletes the specified file.

Usage

`FileDelete("`*`PathFilename`*`")`

Where: *`PathFilename`* - a string expression specifying the path and filename of the file to be deleted

Example

`FileDelete("C:\test.txt");` will delete the file test.txt in the root directory of the C: hard drive

MessageLog

Displays one or more specified expressions in the PowerLanguage Editor Output Log. Any combination of string, true/false, numerical series, or numerical expressions can be specified.

Usage

`MessageLog`(*Expression1*,*Expression2*,*etc.*)

Parameters

`Expression` - a string, true/false, numerical series, or numerical expression; any number of valid expressions, separated by commas, can be used

A *string expression* must be enclosed in quotation marks:

```
"String Expression"
```

A *numerical expression* can be formatted to specify the minimum number of characters, including the decimal point, and the number of decimal places, to be used for the output:

`Expression``:`*`C`*`:`*`D`*

Where: *`C`* - minimum number of characters
`D` - number of decimal places

The default output format for a numerical expression is two decimal places and a minimum of seven characters.

If the number of decimal places in the numerical expression is more than the specified number, the value will be will be rounded off to the specified number of decimal places.

If the number of characters in the output is less than the specified minimum, leading spaces will be added to bring the output to the specified minimum value.

Example

`MessageLog(`**`.1`**`);` will display   0.10 in the PowerLanguage Editor Output Log, with three leading spaces inserted

`MessageLog(`**`1.555555:6:3`**`);` will display 1.556 in the PowerLanguage Editor Output Log, with one leading space inserted

`MessageLog("Current Time is:",CurrentTime:`**`5:0`**`);` will display the string expression "Current Time is:", followed by the output of the `CurrentTime`, with one leading space inserted, in the PowerLanguage Editor Output Log

Print

Sends one or more specified expressions to the PowerLanguage Editor Output Log or another output target, if specified. Any combination of string, true/false, numerical series, or numerical expressions can be specified.

Usage

`Print`([`OutputTarget`],*Expression1*,*Expression2*,*etc*.)

Parameter inside the square brackets is optional
Parameters

`OutputTarget` - an optional parameter; specifies an output target other then the PowerLanguage Editor Output Log; the parameter must be followed by a comma.

There are two optional output targets:

`Printer`

Specifies the default printer as the output target.

`File`("*`PathFilename`*")

Where: *`PathFilename`* - a string expression specifying the path and filename

Specifies an ASCII file as the output target; if the specified file does not exist, the file will be created.

If ***`OutputTarget`*** is not specified, the output will be sent to the PowerLanguage Editor Output Log.

Expression - a string, true/false, numerical series, or numerical expression; any number of valid expressions, separated by commas, can be used

A *string expression* must be enclosed in quotation marks:

`"String Expression"`

A *numerical expression* can be formatted to specify the minimum number of characters, including the decimal point, and the number of decimal places, to be used for the output:

`Expression:C:D`

Where: `C` - minimum number of characters
`D` - number of decimal places

The default output format for a numerical expression is two decimal places and a minimum of seven characters.

If the number of decimal places in the numerical expression is more than the specified number, the value will be will be rounded off to the specified number of decimal places.

If the number of characters in the output is less than the specified minimum, leading spaces will be added to bring the output to the specified minimum value.

Example

`Print(.1);` will print 0.10 in the PowerLanguage Editor Output Log, with three leading spaces inserted

`Print(1.555555:6:3);` will print 1.556 in the PowerLanguage Editor Output Log, with one leading space inserted

`Print(Printer,"Print Test");` will send the string expression "Print Test" to the default printer

`Print(File("C:\test.txt"),CurrentDate,CurrentTime);` will save the output of `CurrentDate` and `CurrentTime` to the test.txt file in the root directory of the C: hard drive

Default

Used in plot statements to specify a default style. Default styles are set by the user.

For more information see Plot

Usage

```
Default
```

Example

Plot the closing price using the default styles (color and width):

```
Plot1(Close,"Close",Default,Default,Default);
```

GetBackgroundColor

Returns an RGB color number or a legacy color value that correspond to the background color of the chart.

Usage

```
GetBackgroundColor
```

Example

Assign an RGB color number, corresponding to the background color of the chart, to Value1 variable:

```
Value1=GetBackgroundColor;
```

Assign a legacy color value, corresponding to the background color of the chart, to Value1 variable:

```
[LegacyColorValue=True];

Value1=GetBackgroundColor;
```

GetPlotBGColor

Returns the numeric color value of the plot (cell) background in a grid.

Usage

`GetPlotBGColor`(*PlotNum*)

Where: *`PlotNum`* - numerical expression representing plot number

Example

Set a variable, Value1, to the cell color of the Plot1 background:

```
Value1 = GetPlotBGColor(1);
```

GetPlotColor

Returns an RGB color number or a legacy color value that correspond to the color of the specified plot.

Usage

`GetPlotColor`(*PlotNumber*)

Where: *`PlotNumber`* - a numerical expression specifying the plot number; plot numbers range from 1 to 999

Example

Assign an RGB color number, corresponding to the color of Plot1, to Value1 variable:

```
Value1=GetPlotColor(1);
```

Assign a legacy color value, corresponding to the color of Plot1, to Value1 variable:

```
[LegacyColorValue=True];

Value1=GetPlotColor(1);
```

GetPlotWidth

Returns the plot line width value of the specified plot. Plot line width values range from 0 to 14.

Usage

`GetPlotWidth`(*PlotNumber*)

Where: *`PlotNumber`* - a numerical expression specifying the plot number; plot numbers range from 1 to 999

Example

Assign the plot line width value of Plot1 to Value1 variable:

```
Value1=GetPlotWidth(1);
```

I_getplotvalue

Gets the value calculated by a signal that is to be used for plotting from an indicator. It can be considered as a bridge between a signal and an indicator. Using i_setplotvalue and i_getplotvalue keywords makes it possible to avoid copying the same script for calculation the same value in both indicator and signal.

Usage

`i_getplotvalue`(*index*)

Parameters

`index` - is the reference number

Notes

1. **i_getplotvalue** can be used in functions and indicators if any signal is applied to the main chart.
2. Values will be transferred between **i_setplotvalue** and **i_getplotvalue** ONLY within 1 chart window,
3. **i_setplotvalue** and **i_getplotvalue** will return 0 if applied to the Market Scanner Window,
4. **i_setplotvalue** and **i_getplotvalue** cannot be used while backtesting a portfolio,
5. **i_setplotvalue** and **i_getplotvalue** cannot be referred historically,
6. it is possible to use unlimited indexes for data transfer.

Usage

Plot in the indicator the max drawdown and open equity values calculated by the signal with indexes 111 and 112

```
plot1(i_getplotvalue(111), "MaxIDDrawdown");

plot2(i_getplotvalue(112), "OpenEquity");
```

I_setplotvalue

Sets the value calculated by a signal that is to be used for plotting from an indicator. It can be considered as a bridge between a signal and an indicator. Using i_setplotvalue and i_getplotvalue keywords makes it possible to avoid copying the same script for calculation the same value in both indicator and signal.

Usage

`i_setplotvalue`(*index*,*value*)

Parameters

`index` - is the reference number

`value` - is the value that is to be transferred.

Notes

1. **i_setplotvalue** can be used in signals, functions and
    - indicators if any signal is applied to the main chart.
2. Values will be transferred between **i_setplotvalue** and **i_getplotvalue** ONLY within 1 chart window,
3. **i_setplotvalue** and **i_getplotvalue** will return 0 if applied to the Market Scanner Window,
4. **i_setplotvalue** and **i_getplotvalue** cannot be used while backtesting a portfolio,
5. **i_setplotvalue** and **i_getplotvalue** cannot be referred historically,
6. it is possible to use unlimited indexes for data transfer.

Usage

Set the max drawdown and open equity values calculated by the signal to be transferred into the indicator with indexes 111 and 112

`i_setplotvalue(`*111*, *maxiddrawdown*)

`i_setplotvalue(`*112*, *netprofit* + *openpositionprofit*)

NoPlot

Removes a specified plot from the current bar.

A conditional plot that is already drawn will remain even if the conditions become no longer true before the bar is closed. NoPlot can be used to remove the conditional plot from the current bar if the conditions are no longer true.

Usage

`NoPlot`(*PlotNumber*)

Where: *`PlotNumber`* - a numerical expression specifying the plot number; plot numbers range from 1 to 999

Example

The example below uses NoPlot remove the PlotPaintBar plot "painted" over the charts bars for which the High price is no longer less then the High price of the previous bar:

```
If High<High[1] Then Begin
PlotPaintBar(High,Low,"",Red);
End
Else Begin
NoPlot(1);
NoPlot(2);
End;
```

Without NoPlot, charts bars for which a High price was initially less then the High price of the previous bar would remain partially painted even if a High price equal to or greater then the High price of the previous bar was reached before the bar was closed.

Plot

Plots the specified numerical or string expression on a chart, up to 999 different plots can be used simultaneously.

Numerical: Plot offset, name, color, and plot line width can be specified by using the optional parameters.

String: Ability to show a user-defined text message on the status line of the chart or in the scanner.

Usage

Numerical: `Plot`*`N`*<[*Offset*]>(*Expression* <,"*`PlotName`*"<,*`PlotColor`* <,*`Scanner Cell Background Color`* <,*LineWidth* >>>>)

Text: `Plot`*`N`*(`"String"`)

Parameters inside the angled brackets are optional

Parameters

`N` - a number used to identify the plot; plot numbers can range from 1 to 999

`Offset` - an optional parameter; a numerical expression specifying the plot offset, in bars; a positive value will displace the plot to the left along the time axis, and a negative value will displace the plot to the right along the time axis

`Expression` - the numerical expression to be plotted

`PlotName` - an optional parameter; assigns a name to the plot

`PlotColor` - an optional parameter; specifies the plot color
Plot color can be specified by a numerical expression representing an RGB color number or a legacy color value, by one of 17 base color words, or by the word `Default` to specify the color chosen by the user. In order for *`PlotColor`* to be used, *`PlotName`* parameter must also be used.

`Scanner Cell Background Color` - an optional parameter; sets the background color of the scanner cell if this plot is applied to a scanner; use the word `Default` to specify the color chosen by the user. In order for `Scanner Cell Background Color` to be used, `PlotName` and `PlotColor` parameters must also be used.

`LineWidth` - an optional parameter; specifies the plot line width, ranging from 1 to 14
Plot line width can be specified as a numerical expression or by the word `Default` to specify the line width chosen by the user. In order for *`LineWidth`* to be used, *`PlotName`*, *`PlotColor`*, and `Scanner Cell Background Color` parameters must also be used.

`String` - text to be displayed

Example

Plot the closing price using the default plot color and line width:

```
Plot1(Close);
```

Plot the closing price using the default plot color and line width, and name the plot "Close":

```
Plot1(Close,"Close",Default,Default,Default);
```

Plot the closing price, offset back by 3 bars, using the plot color of blue, using cell background color of green if this plot is applied to a scanner, line width of 3, and name the plot "Close 3 bars later":

```
Plot1[3](Close,"Close 3 bars later",Blue,Green,3);
```

Plot the closing price, offset forward by 3 bars, using the RGB color 2138336 (Orange), and name the plot "Close 3 bars ago":

```
Plot1[-3](Close,"Close 3 bars ago",2138336);
```

Plot the closing price using the legacy color value of 4 (Green) and name the plot "Close":

```
[LegacyColorValue=True];
Plot1(Close,"Close",4);
```

Show the text "Attention!" on the status bar of the chart or in the scanner.

```
Plot1("Attention!");
```

Show the text "Attention!" on the status bar of the chart or in the scanner if the close price is greater than 100.

```
If close > 100 then Plot1("Attention!");
```

PlotPaintBar

Plots the specified numerical expressions in the form of a bar chart.

Plot name, color, and plot line width can be specified by using the optional parameters.

PlotPaintBar plot can be superimposed on top of one or more bars of a bar chart, effectively "painting" the bars.

Usage

`PlotPaintBar` (*BarHigh*, *BarLow*, *BarOpen*, *BarClose* <,"*`PlotName`*"<,*`PlotColor`* <,`Default` <,*LineWidth* >>>>)

Parameters inside the angled brackets are optional

Parameters

`BarHigh`, *`BarLow`*, *`BarOpen`*, *`BarClose`* - numerical expressions specifying the High, Low, Open, & Close prices for the bars to be plotted; at least two of these parameters are required

`PlotName` - an optional parameter; assigns a name to the plot

`PlotColor` - an optional parameter; specifies the plot color
Plot color can be specified by a numerical expression representing an RGB color number or a legacy color value, by one of 17 base color words, or by the word `Default` to specify the color chosen by the user. In order for *`PlotColor`* to be used, *`PlotName`* parameter must also be used.

`Default` - an optional parameter reserved for future use; should be specified as `Default`; use of this parameter is required in order for *`LineWidth`* to be used

`LineWidth` - an optional parameter; specifies the plot line width, ranging from 1 to 14
Plot line width can be specified as a numerical expression or by the word `Default` to specify the line width chosen by the user. In order for *`LineWidth`* to be used, *`PlotName`*, *`PlotColor`*, and `Default` parameters must also be used.

Notes

`PlotPaintBar`(*BarHigh*,*BarLow*,*BarOpen*,*BarClose*);

is the equivalent of:

`Plot1`(*BarHigh*);
`Plot2`(*BarLow*);
`Plot3`(*BarOpen*);
`Plot4`(*BarClose*);

In order for the PlotPaintBar plot to be displayed in the form of a bar chart, the plot type for each Plot must be set, in the Style section of the General tab of the Format Indicator window, to Bar High, Bar Low, Left Tick, and Right Tick, respectively.

Example

Paint red these bars of an OHLC chart for which the Open price is less then the Open price of the previous bar:

```
If Open<Open[1] Then
PlotPaintBar(High,Low,Open,Close,"",Red);
```

PlotPB

Same as the `PlotPaintBar`

SetPlotBGColor

Assigns a specified color to the cell background for the indicated study plot for the duration of the current bar.

Use of SetBGPlotColor is effective only for the Scanner.

Usage

`SetPlotBGColor`(*PlotNumber*,*`PlotColor`*)

Parameters

`PlotNumber` - a numerical expression specifying the plot number; plot numbers range from 1 to 999

`PlotColor` - an expression specifying the cell background color for the indicated plot. Cell background color for the indicated plot can be specified by a numerical expression representing an RGB color number, a legacy color value, or by one of 17 base color words.

Example

Assign the color of blue to the indicated cell of the plot1 for the duration of the current bar:

```
SetPlotBGColor(1,Blue);
```

Assign RGB color 2138336 (Orange) to the indicated cell of the plot1 for the duration of the current bar:

```
SetPlotColor(1,2138336);
```

Assign legacy color 4 (Green) to the indicated cell of the plot1 for the duration of the current bar:

```
[LegacyColorValue=True];
SetPlotBGColor(1,4);
```

SetPlotColor

Assigns a specified color to the specified plot for the duration of the current bar.

Usage

`SetPlotColor(`*PlotNumber*,*`PlotColor`*`)`

Parameters

`PlotNumber` - a numerical expression specifying the plot number; plot numbers range from 1 to 999

`PlotColor` - an expression specifying the plot color

Plot color can be specified by a numerical expression representing an RGB color number or a legacy color value, or by one of 17 base color words.

Example

Assign the color of blue to plot1 for the duration of the current bar:

```
SetPlotColor(1,Blue);
```

Assign RGB color 2138336 (Orange) to plot1 for the duration of the current bar:

```
SetPlotColor(1,2138336);
```

Assign legacy color 4 (Green) to plot1 for the duration of the current bar:

```
[LegacyColorValue=True];
SetPlotColor(1,4);
```

SetPlotWidth

Assigns a specified line width to the specified plot for the duration of the current bar.

Usage

`SetPlotWidth`(*PlotNumber*,*`LineWidth`*)

Where: *`PlotNumber`* - a numerical expression specifying the plot number; plot numbers range from 1 to 999
`LineWidth` - a numerical expression specifying the plot line width; line width can range from 1 to 14

Example

Assign a plot line width of 10 to plot1 for the duration of the current bar:

```
SetPlotWidth(1,10);
```

pmms_get_strategy_named_num

Returns a numerical value, indicating the value of VariableName of the strategy with StrategyIndex number.

Usage

`pmms_get_strategy_named_num`(*StrategyIndex*, *VariableName*)

Parameters

`StrategyIndex` - numeric variable.

`VariableName` - string variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

`pmms_get_strategy_named_num`(0, "CustomVar")

pmms_get_strategy_named_str

Returns a string value, indicating the value of VariableName of the strategy with StrategyIndex number.

Usage

`pmms_get_strategy_named_str`(*StrategyIndex*, *VariableName*)

Parameters

`StrategyIndex` - numeric variable.

`VariableName` - string variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

`pmms_get_strategy_named_str`(0, "CustomStr")

pmms_set_strategy_named_num

This function sets the value of VariableName of the strategy with StrategyIndex number.

Usage

`pmms_set_strategy_named_num`(*StrategyIndex*, *VariableName*, *VariableValue*)

Parameters

`StrategyIndex` - numeric variable.

`VariableName` - string variable.

`VariableValue` - numeric variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

`pmms_set_strategy_named_num`(0, "CustomVar", 5)

pmms_set_strategy_named_str

This function sets the string value of VariableName of the strategy with StrategyIndex number.

Usage

`pmms_set_strategy_named_str`(*StrategyIndex*, *VariableName*, *VariableValue*)

Parameters

`StrategyIndex` - numeric variable.

`VariableName` - string variable.

`VariableValue` - string variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

`pmms_set_strategy_named_str`(0, "CustomStr", "buy")

pmms_strategies_allow_entries_all

This function allows all the strategies to open positions.

Usage

```
pmms_strategies_allow_entries_all
```

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

```
pmms_strategies_allow_entries_all
```

pmms_strategies_count

Returns a numerical value, indicating the number of trading strategies (which is equal to number of downloaded instruments that are traded).

Usage

```
pmms_strategies_count
```

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

```
pmms_strategies_count
```

pmms_strategies_deny_entries_all

This function denies all the strategies to open positions (entry orders will be excluded from RAW orders collection).

Usage

```
pmms_strategies_deny_entries_all
```

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

```
pmms_strategies_deny_entries_all
```

pmms_strategies_get_by_symbol_name

Returns an index of the strategy based on instrument name (-1, if the instrument is not found). If several strategies are applied to the same instrument, then the number of one of these strategies will be returned.

Usage

`pmms_strategies_get_by_symbol_name`(*SymbolName*)

Parameters

`SymbolName` - string variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

```
Value1 = pmms_strategies_get_by_symbol_name("MSFT");
```

pmms_strategies_in_long_count

Returns a numerical value, indicating the number of strategies with open long position. Accepts one parameter - a one dimensional dynamical array. This array is filled with the index numbers of the strategies that have an open long position at the moment of the strategy calculation.

Usage

`pmms_strategies_in_long_count(`*indexesArray*`)`

Parameters

`indexesArray` - array variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

```
var: positionsLong(0);

array: strategyIndexes[](0);

positionsLong = pmms_strategies_in_long_count(strategyIndexes);
```

pmms_strategies_in_positions_count

Returns a numerical value, indicating the number of strategies with open position. Accepts one parameter - a one dimensional dynamical array. This array is filled with the index numbers of the strategies that have an open position at the moment of the strategy calculation.

Usage

`pmms_strategies_in_positions_count`(*indexesArray*)

Parameters

`indexesArray` - array variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

```
var: positions(0);

array: strategyIndexes[](0);

positions = pmms_strategies_in_positions_count(strategyIndexes);
```

pmms_strategies_in_short_count

Returns a numerical value, indicating the number of strategies with open short position. Accepts one parameter - a one dimensional dynamical array. This array is filled with the index numbers of the strategies that have an open short position at the moment of the strategy calculation.

Usage

`pmms_strategies_in_short_count`(*indexesArray*)

Parameters

`indexesArray` - array variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

```
var: positionsShort(0);

array: strategyIndexes[](0);

positionsShort = pmms_strategies_in_short_count(strategyIndexes);
```

pmms_strategies_pause_all

This function pauses order sending for all strategies of the portfolio.

Usage

```
pmms_strategies_pause_all
```

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

```
pmms_strategies_pause_all
```

pmms_strategies_resume_all

This function resumes order sending for all strategies of the portfolio.

Usage

```
pmms_strategies_resume_all
```

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

```
pmms_strategies_resume_all
```

pmms_strategies_set_status_for_all

This function sets a text (string) status for all strategies of the portfolio (status is indicated in the Custom Text column of Portfolio Real-Time Window).

Usage

`pmms_strategies_set_status_for_all`(*Status*)

Parameters

`Status` - string variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

`pmms_strategies_set_status_for_all`("Calculating")

pmms_strategy_allow_entries

This function allows entry orders for strategy with StrategyIndex number.

Usage

`pmms_strategy_allow_entries`(*StrategyIndex*)

Parameters

`StrategyIndex` - numeric variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

`pmms_strategy_allow_entries`(0)

pmms_strategy_allow_exits

This function allows exit orders for strategy with StrategyIndex number.

Usage

`pmms_strategy_allow_exits`(*StrategyIndex*)

Parameters

`StrategyIndex` - numeric variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

`pmms_strategy_allow_exits`(0)

pmms_strategy_allow_exit_from_long

The same as `pmms_strategy_allow_exits`, but applied for exits from long entries only.

pmms_strategy_allow_exit_from_short

The same as `pmms_strategy_allow_exits`, but applied for exits from short entries only.

pmms_strategy_allow_long_entries

The same as `pmms_strategy_allow_entries`, but applied for long entries only.

pmms_strategy_allow_short_entries

The same as `pmms_strategy_allow_entries`, but applied for short entries only.

pmms_strategy_close_position

This function closes position of the strategy with Index number with market order (orders generated by the strategy will be deleted from Raw Orders collection).

Usage

`pmms_strategy_close_position(`*Index*`)`

Parameters

`Index` - numeric variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

```
pmms_strategy_close_position(0)
```

pmms_strategy_currentcontracts

Returns a numerical value representing number of contracts of the position opened by strategy with StrategyIndex number.

Usage

`pmms_strategy_currentcontracts`(*StrategyIndex*)

Parameters

`StrategyIndex` - numeric variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

`Value1` = `pmms_strategy_currentcontracts`(0);

pmms_strategy_deny_entries

This function denies entry orders for strategy with StrategyIndex number (entry orders will be deleted from Raw Orders collection).

Usage

`pmms_strategy_deny_entries`(*StrategyIndex*)

Parameters

`StrategyIndex` - numeric variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

`pmms_strategy_deny_entries`(0)

pmms_strategy_deny_exits

This function denies exit orders for strategy with StrategyIndex number (exit orders will be deleted from Raw Orders collection).

Usage

`pmms_strategy_deny_exits`(*StrategyIndex*)

Parameters

`StrategyIndex` - numeric variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

`pmms_strategy_deny_exits`(0)

pmms_strategy_deny_exit_from_long

The same as `pmms_strategy_deny_exits`, but applied for exits from long entries only.

pmms_strategy_deny_exit_from_short

The same as `pmms_strategy_deny_exits`, but applied for exits form short entries only.

pmms_strategy_deny_long_entries

The same as `pmms_strategy_deny_entries`, but applied for long entries only.

pmms_strategy_deny_short_entries

The same as `pmms_strategy_deny_entries`, but applied for short entries only.

pmms_strategy_entryprice

Returns a numerical value representing an average entry price of the position opened by the strategy with StrategyIndex number.

Usage

`pmms_strategy_entryprice`(*StrategyIndex*)

Parameters

`StrategyIndex` - numeric variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

`Value1` = `pmms_strategy_entryprice`(0);

pmms_strategy_get_entry_contracts

Returns the number of contracts of the entry order of the strategy with StrategyIndex number.

Usage

`pmms_strategy_get_entry_contracts`(*StrategyIndex*)

Parameters

`StrategyIndex` - numeric variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

`pmms_strategy_get_entry_contracts`(0)

pmms_strategy_is_paused

Returns true/false value indicating if order sending is paused for the strategy with StrategyIndex number.

Usage

`pmms_strategy_is_paused`(*StrategyIndex*)

Parameters

`StrategyIndex` - numeric variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

`Condition1=pmms_strategy_is_paused`(0);

pmms_strategy_marketposition

Returns a numerical value representing market position of the strategy with StrategyIndex number.

Usage

`pmms_strategy_marketposition`(*StrategyIndex*)

Parameters

`StrategyIndex` - numeric variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

```
Value1=pmms_strategy_marketposition(0);
```

pmms_strategy_maxiddrawdown

Returns a numerical value representing max drawdown of the strategy with StrategyIndex number.

Usage

`pmms_strategy_maxiddrawdown`(*StrategyIndex*)

Parameters

`StrategyIndex` - numeric variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

The value is returned in the currency specified in Portfolio Trader: **Portfolio Settings** -> **Base Currency**.

Example

`Value1` = `pmms_strategy_maxiddrawdown`(0);

pmms_strategy_netprofit

Returns a numerical value representing net profit of the strategy with StrategyIndex number.

Usage

`pmms_strategy_netprofit`(*StrategyIndex*)

Parameters

`StrategyIndex` - numeric variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

The value is returned in the currency specified in Portfolio Trader: **Portfolio Settings** -> **Base Currency**.

Example

```
Value1 = pmms_strategy_netprofit(0);
```

pmms_strategy_openprofit

Returns a numerical value representing unrealized profit/loss of the strategy with StrategyIndex number.

Usage

`pmms_strategy_openprofit`(*StrategyIndex*)

Parameters

`StrategyIndex` - numeric variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

The value is returned in the currency specified in Portfolio Trader: **Portfolio Settings** -> **Base Currency**.

Example

```
Value1 = pmms_strategy_openprofit(0);
```

pmms_strategy_pause

This function denies order sending for the strategy with StrategyIndex number (all orders of the strategy will be deleted from Raw Orders Collection).

Usage

`pmms_strategy_pause`(*StrategyIndex*)

Parameters

`StrategyIndex` - numeric variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

`pmms_strategy_pause`(0);

pmms_strategy_resume

This function allows order sending for the strategy with StrategyIndex number (all orders of the strategy will be deleted from Raw Orders Collection).

Usage

`pmms_strategy_resume`(*StrategyIndex*)

Parameters

`StrategyIndex` - numeric variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

`pmms_strategy_resume`(0);

pmms_strategy_riskcapital

Returns a numerical value representing amount of money withheld as a risk capital for the open position of the strategy with StrategyIndex number.

Usage

`pmms_strategy_riskcapital`(*StrategyIndex*)

Parameters

`StrategyIndex` - numeric variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

The value is returned in the currency specified in Portfolio Trader: **Portfolio Settings** -> **Base Currency**.

Example

`Value1` = `pmms_strategy_riskcapital`(0);

pmms_strategy_set_entry_contracts

This function sets the number of contracts for entry orders of the strategy with StrategyIndex number (size calculated by the strategy itself will be ignored). To use the order size calculated by the strategy itself set contracts parameter to -1.

Usage

`pmms_strategy_set_entry_contracts`(*StrategyIndex*, *Contracts*)

Parameters

`StrategyIndex` - numeric variable.

`Contracts` - numeric variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

`pmms_strategy_set_entry_contracts`(0, 5)

pmms_strategy_set_status

This function sets a text (string) status for the strategy with StrategyIndex number of the portfolio (status is indicated in the Custom Text column of Portfolio Real-Time Window).

Usage

`pmms_strategy_set_status`(*StrategyIndex*, *Status*)

Parameters

`StrategyIndex` - numeric variable.

`Status` - string variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

`pmms_strategy_set_status`("Long");

pmms_strategy_symbol

Returns a string value representing the name of instrument to which the signal of the strategy with StrategyIndex number is applied.

Usage

`pmms_strategy_symbol`(*StrategyIndex*)

Parameters

`StrategyIndex` - numeric variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

`pmms_strategy_symbol`(0);

pmm_get_global_named_num

Returns a numerical value, indicating the global numerical value with VariableName name.

Usage

`pmm_get_global_named_num`(*VariableName*)

Parameters

`VariableName` - string variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

`Value1` = `pmm_get_global_named_num`("GlobalVar");

pmm_get_global_named_str

Returns the global string value with VariableName name.

Usage

`pmm_get_global_named_str`(*VariableName*)

Parameters

`VariableName` - string variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

`pmm_get_global_named_str`("GlobalStr");

pmm_get_my_named_num

Returns a numerical value, indicating the value of VariableName of the current strategy.

Usage

`pmm_get_my_named_num`(*VariableName*)

Parameters

`VariableName` - string variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

```
Value1 = pmm_get_my_named_num("CustomVar");
```

pmm_get_my_named_str

Returns a string value, indicating the value of VariableName of the current strategy.

Usage

`pmm_get_my_named_str`(*VariableName*)

Parameters

`VariableName` - string variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

`pmm_get_my_named_str`("CustomStr");

pmm_set_global_named_num

This function sets the global numeric value with VariableName name.

Usage

`pmm_set_global_named_num`(*VariableName*, *VariableValue*)

Parameters

`VariableName` - string variable.

`VariableValue` - numeric variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

`pmm_set_global_named_num`("GlobalVar", 3);

pmm_set_global_named_str

This function sets the global string value with VariableName name.

Usage

`pmm_set_global_named_str`(*VariableName*, *VariableValue*)

Parameters

`VariableName` - string variable.

`VariableValue` - string variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

```
if marketposition > 0 then pmm_set_global_named_str("GlobalStr",
symbolname + " is Long");
```

pmm_set_my_named_num

This function sets the value of VariableName of the current strategy.

Usage

`pmm_set_my_named_num`(*VariableName, VariableValue*)

Parameters

`VariableName` - string variable.

`VariableValue` - numeric variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

`pmm_set_my_named_num`("MarketPosition", `marketposition`);

pmm_set_my_named_str

This function sets the string value of VariableName of the current strategy.

Usage

`pmm_set_my_named_str`(*VariableName, VariableValue*)

Parameters

`VariableName` - string variable.

`VariableValue` - string variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

```
if marketposition > 0 then pmm_set_my_named_str("Position", "Long");
```

pmm_set_my_status

This function sets a text (string) status for the current strategy of the portfolio (status is indicated in the Custom Text column of Portfolio Real-Time Window).

Usage

`pmm_set_my_status`(*Status*)

Parameters

`Status` - string variable.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

```
if marketposition < 0 then pmm_set_my_status("Short");
```

Portfolio_GrossLoss

Returns a negative numerical value, indicating the total currency value of all completed losing trades for a portfolio.

Usage

```
Portfolio_GrossLoss
```

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

The value is returned in the currency specified in Portfolio Trader: **Portfolio Settings** -> **Base Currency**.

Example

`Portfolio_GrossLoss` will return a value of -50 if there were a total of four losing trades, at 10, 5, 20, and 15

`Portfolio_GrossLoss` will return a value of 0 if no losing trades were completed during the entire trading period

Portfolio_GrossProfit

Returns a numerical value, indicating the total currency value of all completed winning trades for a portfolio.

Usage

```
Portfolio_GrossProfit
```

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

The value is returned in the currency specified in Portfolio Trader: **Portfolio Settings** -> **Base Currency**.

Example

`Portfolio_GrossProfit` will return a value of 50 if there were a total of four winning trades, at 10, 5, 20, and 15

`Portfolio_GrossProfit` will return a value of 0 if no winning trades were completed during the entire trading period

Portfolio_InvestedCapital

Returns absolute value indicating the amount of cash assets invested in portfolio securities on the moment of strategy calculation.

Usage

```
Portfolio_InvestedCapital
```

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

The value is returned in the currency specified in Portfolio Trader: **Portfolio Settings** -> **Base Currency**.

Example

`Portfolio_InvestedCapital` will return a value of 100 000 if the amount of cash assets invested in portfolio securities on the moment of strategy calculation is 100 000 units of the selected currency (e.g. there are three open positions: 50 000 long, 20 000 long and 30 000 short).

Portfolio_MaxIDDrawdown

Returns a negative numerical value, indicating the largest decline in equity for the entire portfolio during the trading period.

Usage

```
Portfolio_MaxIDDrawdown
```

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

The value is returned in the currency specified in Portfolio Trader: **Portfolio Settings** -> **Base Currency**.

Example

`Portfolio_MaxIDDrawdown` will return a value of -500 if the largest decline in equity during the entire trading period was ¤500

Portfolio_NetProfit

Returns a numerical value, indicating the total currency value of all completed trades for a portfolio.

Usage

```
Portfolio_NetProfit
```

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

The value is returned in the currency specified in Portfolio Trader: **Portfolio Settings** -> **Base Currency**.

Example

`Portfolio_NetProfit` will return a value of 20 if there were winning trades at 25 and 10, and losing trades at 5 and 10

`Portfolio_NetProfit` will return a value of -15 if there were winning trades at 10 and 5, and losing trades at 20 and 10

`Portfolio_NetProfit` will return a value of 0 no trades were completed during the entire trading period

Portfolio_NumLossTrades

Returns a numerical value, indicating the number of all completed losing trades for a portfolio.

Usage

```
Portfolio_NumLossTrades
```

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

`Portfolio_NumLossTrades` will return a value of 5 if there were a total of five completed losing trades

`Portfolio_NumLossTrades` will return a value of 0 if no losing trades were completed during the entire trading period

Portfolio_NumWinTrades

Returns a numerical value, indicating the number of all completed winning trades for a portfolio.

Usage

```
Portfolio_NumWinTrades
```

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

`Portfolio_NumWinTrades` will return a value of 5 if there were a total of five completed winning trades

`Portfolio_NumWinTrades` will return a value of 0 if no winning trades were completed during the entire trading period

Portfolio_PercentProfit

Returns a numerical value, indicating the percentage of winning trades in all trades completed for a portfolio.

Usage

```
Portfolio_PercentProfit
```

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

`Portfolio_PercentProfit` will return a value of 70 if seven out of the total of 10 completed trades were winning trades

Portfolio_StrategyDrawdown

Returns a negative numerical value, indicating the current decline in equity for the entire portfolio from the peak value for the entire trading period.

Usage

```
Portfolio_StrategyDrawdown
```

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

The value is returned in the currency specified in Portfolio Trader: **Portfolio Settings** -> **Base Currency**.

Example

`Portfolio_StrategyDrawdown` will return a value of -100 if the current decline in equity from the peak value is ¤100

Portfolio_TotalTrades

Returns a numerical value, indicating the total number of all completed trades for a portfolio.

Usage

```
Portfolio_TotalTrades
```

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

`Portfolio_TotalTrades` will return a value of 5 if there were a total of five completed trades

`Portfolio_TotalTrades` will return a value of 0 if no trades were completed during the entire trading period

Portfolio_CalcMaxPotentialLossForEnt

Calculates and returns maximum potential loss (not including margin, commision or slippage) if user entered the position with the number of `Contracts` and `Price` of entry.

Usage

`Portfolio_CalcMaxPotentialLossForEntry` (*`Side`* <,*`Contracts`* <,*`Price`*>>);

Parameters inside the angled brackets are optional

Parameters

`Side` is a numerical expression specifying the entry type (e.g. 1 Long entry or -1 Short entry).

`Contracts` is an optional parameter specifying the number of contracts. If the Contracts parameter is not specified, then the number of contracts indicated in the **Format Settings** dialog window under the **Properties** tab is used by default.

`Price` is an optional parameter specifying the price value. If the `Price` parameter is not specified, then the Close price value of the current bar will be used by default. This parameter can be rounded down if entered a Short position or rounded up if entered a Long position.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

The value is returned in the currency specified in Portfolio Trader: **Portfolio Settings** -> **Base Currency**.

Example

`Portfolio_CalcMaxPotentialLossForEntry` (`0`, `25`, `High`) will return a value of 0, since the parameter `Side`=0

`Portfolio_CalcMaxPotentialLossForEntry` (`1`, `100`, `Close`) will return the maximum potential loss (not including margin, commision or slippage) if user entered a Long position for 100 contracts at the Close price.

`Portfolio_CalcMaxPotentialLossForEntry` (`-1`, `5`, `Open`) will return the maximum potential loss (not including margin, commision or slippage) if user entered a Short position for 5 contracts at Open price.

`Portfolio_CalcMaxPotentialLossForEntry` (`1`) will return the maximum potential loss (not including margin, commision or slippage) if the user entered a Long position for a number of contracts indicated in the **Format Settings** dialog window under the **Properties** tab at Close price.

Portfolio_CurrentEntries

Returns a numerical value, indicating the combined number of entries currently open within a portfolio.

Usage

```
Portfolio_CurrentEntries
```

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

Assign a value, indicating the combined number of entries currently open within a portfolio, to Value1 variable:

```
Value1=Portfolio_CurrentEntries;
```

Portfolio_MaxOpenPositionPotentialLo

Returns a value indicating the combined potential loss (not including margin, commision or slippage) for the traded symbol's open position within the portfolio.

Usage

```
SetStopPosition;

SetStopLoss(Portfolio_MaxOpenPositionPotentialLoss);
```

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

The value is returned in the currency specified in Portfolio Trader: **Portfolio Settings** -> **Base Currency**.

Example

```
value1 = Portfolio_MaxOpenPositionPotentialLoss;

if value1 <> 0 then begin

SetStopPosition;

SetStopLoss (value1);

end;
```

`Portfolio_MaxOpenPositionPotentialLoss` will return a value of 0 if there are currently no open positions within a portfolio.

`Portfolio_MaxOpenPositionPotentialLoss` will return a value of 100 if the combined potential loss for all open positions within a portfolio is ¤100 since the positions were entered.

Portfolio_OpenPositionProfit

Returns a numerical value, indicating the current combined profit or loss for all open positions within a portfolio.

Usage

```
Portfolio_OpenPositionProfit
```

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

The value is returned in the currency specified in Portfolio Trader: **Portfolio Settings** -> **Base Currency**.

Example

`Portfolio_OpenPositionProfit` will return a value of 0 if there are currently no open positions within a portfolio

`Portfolio_OpenPositionProfit` will return a value of 100 if the combined value of all open positions within a portfolio has increased by ¤100 since the positions were entered

`Portfolio_OpenPositionProfit` will return a value of -50 if the combined value of all open positions within a portfolio has decreased by ¤50 since the positions were entered

Portfolio_SetMaxPotentialLossPerCont

Redefines the values for the indicated symbol. The values in the ¤ box if the **Absolute Max Potential Loss** option is selected or the values in the **%** box if the **Max Potential Loss** is selected.

The newly set value is valid during the strategy calculation or until the `Portfolio_SetMaxPotentialLossPerContract` is requested again assigning a new value.

Usage

`Portfolio_SetMaxPotentialLossPerContract`(`NewValue`);

Parameters

`NewValue` is a numerical value that can be:

- an absolute value in the range [-100, -0.001]; defines percentage of the maximum potential loss per contract. (Max Potential Loss: %)
- a value in the range [0.001, 1e+29]; defines the maximum potential loss per contract in the selected currency. (Absolute Max Potential Loss: ¤)
- equal 0; in this case the value entered in the **Format Settings** dialog window under the **Portfolio Settings** tab is used.

`Portfolio_SetMaxPotentialLossPerContract` returns:

- True if the value is in one of the ranges indicated above. Redefining is considered to be succesfull.
- False if the if the value is out of the ranges indicated above. Redefining is considered unsuccessful and the Max Potential Loss value is unchanged.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

The value is returned in the currency specified in Portfolio Trader: **Portfolio**

Settings -> **Base Currency**.

Example

`If "MSFT"=SymbolName then Portfolio_SetMaxPotentialLossPerContract (-10)`; the new value of -10 is assigned for the symbol "MSFT" only if such symbol is exist in the portfolio.

`Portfolio_SetMaxPotentialLossPerContract (-5)`; redefines the percentage value of the maximum potential loss per contract for 5%.

`Portfolio_SetMaxPotentialLossPerContract (200)`; redefines the maximum potential loss per contract for ¤200.

PortfolioEntriesPriority

Assigns a priority to each entry order within a portfolio.

If execution of all entries generated for a portfolio would cause the capital limits to be exceeded, the entries with the highest priority will receive preference, while the entries with the lowest priority will not be executed.

If PortfolioEntriesPriority is not specified, the entries will be executed according to the order the symbols are listed in the symbol grid of the Portfolio Trader.

Usage

`PortfolioEntriesPriority`=*Priority*

Where: *`Priority`* - a numerical expression specifying the entry order execution priority; a greater value indicates a higher priority

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

Assign higher execution priority to entry orders for symbols with lower share prices:

```
PortfolioEntriesPriority=(-Close);
```

Portfolio_GetMarginPerContract

Returns:

- a numerical value indicated in the **% of contract cost** box multiplied by **-1**, if the **Margin value** option is selected; or:
- the same value as a reserved word Margin if the **Absolute Margin Value** option is selected.

Usage

`Portfolio_GetMarginPerContract`

Parameters

This function can only be used in signals intended to be used with the Portfolio Trader.

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

The absolute margin value is returned in the currency specified in Portfolio Trader: **Portfolio Settings** -> **Base Currency**.

Example

`Portfolio_GetMarginPerContract` will return a value of 0 if **Absolute Margin Value** option is selected and the margin does not exist in the QuoteManager symbol settings. Not all of the securities can be bought on margin. The margin value can be returned for futures or options.

`Portfolio_GetMarginPerContract` will return a value of 25 if **Absolute Margin Value** option is selected and the margin box in the **Edit Symbol** dialog window under **Future** tab in QuoteManager contains the value of 25.

`Portfolio_GetMarginPerContract` will return a value of -10 if the **Margin Value** option is selected and the **% of contract cost** box contains the value of 10.

Portfolio_GetMaxPotentialLossPerCon

Returns:

- a numerical value indicated in the **%** box multiplied by **-1**, if the **Max Potential Loss** option is selected in the **Format Settings** dialog window under the **Portfolio Settings** tab; or:
- a numerical value indicated in the **Absolute Max Potential Loss** option box in the **Format Settings** dialog window under the **Portfolio Settings** tab.

Usage

```
Portfolio_GetMaxPotentialLossPerContract
```

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

The absolute max potential loss value is returned in the currency specified in Portfolio Trader: **Portfolio Settings** -> **Base Currency**.

Example

`Portfolio_GetMaxPotentialLossPerContract` will return -5 if the **Max Potential Loss** option is selected and **%** box value is 5 under the **Portfolio Settings** tab in the **Format Settings** dialog window.

`Portfolio_GetMaxPotentialLossPerContract` will return 0.001 if the **Absolute Max Potential Loss** option is selected and ¤ box value is 0.001 under the **Portfolio Settings** tab in the **Format Settings** dialog window.

Portfolio_MaxRiskEquityPerPosPercen

Returns the *Max % of Equity at Risk per Position* numerical value set by the user in the Portfolio Settings tab of the Portfolio Trader Format Settings window.

Usage

```
Portfolio_MaxRiskEquityPerPosPercent
```

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

`Portfolio_MaxRiskEquityPerPosPercent` will return the *Max % of Equity at Risk per Position* numerical value set by the user

Portfolio_TotalMaxRiskEquityPercent

Returns the equity *Exposure %* numerical value set by the user in the Portfolio Settings tab of the Portfolio Trader Format Settings window.

Usage

```
Portfolio_TotalMaxRiskEquityPercent
```

Notes

This function can only be used in signals intended to be used with the Portfolio Trader.

Example

`Portfolio_TotalMaxRiskEquityPercent` will return the equity *Exposure %* numerical value set by the user.

AskSize

Returns a numerical value indicating the current best Ask volume for the symbol that the study is applied to.

Usage

```
AskSize
```

Notes

Quote Fields cannot be referenced historically.

Example

`AskSize` will return the current best Ask volume

BidSize

Returns a numerical value indicating the current best Bid volume for the symbol that the study is applied to.

Usage

`BidSize`

Notes

Quote Fields cannot be referenced historically.

Example

`BidSize` will return the current best Bid volume

CurrentOpenInt

Returns a numerical value indicating the last known open interest for the symbol that the study is applied to.

Usage

```
CurrentOpenInt
```

Notes

Quote Fields cannot be referenced historically.

Example

`CurrentOpenInt` will return the last known open interest

DailyClose

Returns a numerical value indicating the most recent Close price.

Usage

```
DailyClose
```

Notes

Quote Fields cannot be referenced historically.

Example

`DailyClose` will return the most recent Close price

DailyHigh

Returns a numerical value indicating the High price for the current trading session.

Usage

`DailyHigh`

Notes

Quote Fields cannot be referenced historically.

Example

`DailyHigh` will return the High price for the current trading session

DailyLow

Returns a numerical value indicating the Low price for the current trading session.

Usage

`DailyLow`

Notes

Quote Fields cannot be referenced historically.

Example

`DailyLow` will return the Low price for the current trading session

DailyOpen

Returns a numerical value indicating the opening price for the current trading session.

Usage

```
DailyOpen
```

Notes

Quote Fields cannot be referenced historically.

Example

`DailyOpen` will return the Open price for the current trading session

DailyVolume

Returns a numerical value indicating the current total trade volume for the trading session.

Usage

```
DailyVolume
```

Notes

Quote Fields cannot be referenced historically.

Example

`DailyVolume` will return the current total trade volume

Description

Returns a string expression containing the description for the symbol that the study is applied to; if no description is available, a blank ("") string expression will be returned.

Usage

```
Description
```

Example

`Description` will return "GOOGLE INC" for Google

ExchListed

Returns a string expression containing the exchange name for the symbol that the study is applied to.

Usage

```
ExchListed
```

Example

`ExchListed` will return "NASD" for Google

`ExchListed` will return "CME" for E-mini S&P; 500

InsideAsk

Returns a numerical value indicating the current best Ask for the symbol that the study is applied to.

Usage

```
InsideAsk
```

Notes

Quote Fields cannot be referenced historically.

Example

`InsideAsk` will return the current best Ask

InsideBid

Returns a numerical value indicating the current best Bid for the symbol that the study is applied to.

Usage

```
InsideBid
```

Notes

Quote Fields cannot be referenced historically.

Example

`InsideBid` will return the current best Bid

Last

Returns a numerical value indicating the price of the last completed trade.

Usage

```
Last
```

Notes

Quote Fields cannot be referenced historically.

Example

`Last` will return the price of the last trade

PrevClose

Returns a numerical value indicating the closing price of the previous trading session.

Usage

```
PrevClose
```

Notes

Quote Fields cannot be referenced historically.

Example

`PrevClose` will return the Close of the previous trading session

q_Ask

Retained for backward compatibility; replaced with `InsideAsk`.

q_asksize

Retained for backward compatibility; replaced with `AskSize`.

q_Bid

Retained for backward compatibility; replaced with `InsideBid`.

q_bidsize

Retained for backward compatibility; replaced with `BidSize`.

q_BigPointValue

Retained for backward compatibility; replaced with `BigPointValue`.

q_Date

Retained for backward compatibility; replaced with `TradedDate`.

q_ExchangeListed

Retained for backward compatibility; replaced with ExchListed.

q_Last

Retained for backward compatibility; replaced with `Last`.

q_OpenInterest

Retained for backward compatibility; replaced with `CurrentOpenInt`.

q_PreviousClose

Retained for backward compatibility; replaced with `PrevClose`.

q_Time

Retained for backward compatibility; replaced with `TradeTime`.

q_Time_Dt

Returns a double-precision decimal DateTime value indicating current time from the status line.

Usage

`q_Time_Dt`

Example

`q_Time_Dt` will return a value of 39448.25000000 for 6:00 AM on January 1st, 2008

q_Time_s

Same as `q_Time`.

Time is indicated in **HHmmss** format.

q_TotalVolume

Retained for backward compatibility; replaced with `DailyVolume`.

q_tradevolume

Retained for backward compatibility; same as `TradeVolume`.

RTSymbol

Same as the `RTSymbolName`.

RTSymbolName

Returns a string expression containing the name of the real-time symbol that the study is applied to in case the merging option is enabled. If the merging option is disabled returns the same value as the Name property. In case of a custom futures instrument the name of the last contract is returned.

See also `GetRTSymbolName`

Usage

```
RTSymbolName
```

Example

`RTSymbolName` will return "GOOG" for Google if the merging option is enabled and Google is configured as the real-time instrument.

Symbol

Same as the `SymbolName`.

SymbolName

Returns a string expression containing the name of the symbol that the study is applied to.

See also `GetSymbolName`

Usage

```
SymbolName
```

Example

`SymbolName` will return "GOOG" for Google

TradeDate

Returns a numerical value indicating the date of the most recent price field update for the symbol. The date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month.

Usage

```
TradeDate
```

Notes

Quote Fields cannot be referenced historically.

Example

`TradeDate` will return a value of 1071030 for October 30th, 2007

`TradeDate` will return a value of 990402 for April 2th, 1999

TradeTime

Returns a numerical value indicating the time of the most recent price field update for the symbol. The time is indicated in the 24-hour HHmm format, where 1300 = 1:00 PM.

Usage

`TradeTime`

Notes

Quote Fields cannot be referenced historically.

Example

`TradeTime` will return a value of 1015 for 10:15 AM

`TradeTime` will return a value of 1545 for 3:45 PM

TradeVolume

Returns a numerical value indicating the volume of Last Price. Information is taken from the Status Line.

Usage

```
TradeVolume
```

Notes

Quote Fields cannot be referenced historically.

Example

`TradeVolume` will return the value of 5 if Last Size was 5.

AutoSession

Returns 0 and is used as an argument in session information functions.

RegularSession

Returns 1 and is used as an argument in session information functions.

Sess1EndTime

This word is retained for backward compatibility.

Sess1FirstBarTime

Returns the First bar's time for the first session of the trading day in 24-hour format. Please note that the Time Zone setting affects the value returned.

Example

`Sess1FirstBarTime` returns 0945 when applied to MSFT data with a 15-minute interval

`Sess1FirstBarTime` returns 0835 when applied to E-mini S&P500; Data with a 5-minute interval

Sess1StartTime

This word is retained for backward compatibility.

Sess2EndTime

This word is retained for backward compatibility.

Sess2FirstBarTime

Returns the First bars time for the second session of the trading day in 24-hour format. Please note that the Time Zone setting affects the value returned.

Example

`Sess2FirstBarTime` returns 1725 when applied to US Treasury Bond Data with a 5-minute interval.

Sess2StartTime

This word is retained for backward compatibility.

SessionCount

Returns the number of sessions for the trading week.

Usage

`SessionCount`(*SessionType*);

Where: *`SessionType`* - a numerical expression: 0 = Auto Detect, 1 = Regular Session*

* Custom Sessions parameters will be used if selected in QuoteManager.

Example

In this example, we have assigned to Value 1 the total number of sessions for the week in the current bar.

```
Value1 = SessionCount(0);
```

In this example, we have assigned to Value 1 the total number of regular sessions for the week in the current bar.

```
Value1 = SessionCount(1);
```

SessionCountMS

Returns the number of merged sessions for the trading week. Merged sessions are sessions beginning at the earliest start time for all symbols and ending at the latest end time for all symbols each trading day.

Example

In this example, we have assigned to Value1 the total number of merged sessions for the week in the current chart:

```
Value1=SessionCountMS
```

SessionEndDay

Returns a numerical value indicating the day of the week that the specified session ends, where 0 = Sunday, 1 = Monday, etc.

Usage

`SessionEndDay`(*SessionType*,*SessionNum*)

Where: *`SessionType`* - a numerical expression: 0 = Auto Detect, 1 = Regular Session`*`
`SessionNum` - a numerical expression specifying the Session Number

`*` Custom Sessions parameters will be used if selected in QuoteManager.

Example

Assign a value, indicating the day of the week that the 4th regular session ends, to Value1 variable:

```
Value1=SessionEndDay(1,4)
```

SessionEndDayMS

Returns a numerical value indicating the day of the week the specified merged session in a multi-data series chart ends, where 0 = Sunday, 1 = Monday, etc.

Merged session extends from the earliest start time to the latest end time of all sessions merged.

Usage

`SessionEndDayMS`(*SessionNum*)

Where: *SessionNum* - a numerical expression specifying the Session Number

Example

Assign a value, indicating the day of the week that the 4th merged session ends, to Value1 variable:

```
Value1 = SessionEndDayMS(4);
```

SessionEndTime

Returns a numerical value, indicating the time of the day that the specified session ends. The time is indicated in the 24-hour HHmm format, where 1300 = 1:00 PM.

Usage

`SessionEndTime`(*SessionType*,*SessionNum*)

Where: *`SessionType`* - a numerical expression: 0 = Auto Detect, 1 = Regular Session`*`
`SessionNum` - a numerical expression specifying the Session Number

`*` Custom Sessions parameters will be used if selected in QuoteManager.

Example

Assign a value, indicating the time of the day that the 4th regular session ends, to Value1 variable:

```
Value1 = SessionEndTime(1,4);
```

SessionEndTimeMS

Returns a numerical value, indicating the time of the day that the specified merged session in a multi-data series chart ends. The time is indicated in the 24-hour HHmm format, where 1300 = 1:00 PM.

Merged session extends from the earliest start time to the latest end time of all sessions merged.

Usage

`SessionEndTimeMS`(*SessionNum*)

Where: *`SessionNum`* - a numerical expression specifying the Session Number

Example

Assign a value, indicating the time of the day that the 4th merged session ends, to Value1 variable:

```
Value1 = SessionEnd(4);
```

SessionStartDay

Returns a numerical value indicating the day of the week that the specified session starts, where 0 = Sunday, 1 = Monday, etc.

Usage

`SessionStartDay`(*SessionType*,*SessionNum*)

Where: *`SessionType`* - a numerical expression: 0 = Auto Detect, 1 = Regular Session*
`SessionNum` - a numerical expression specifying the Session Number

* Custom Sessions parameters will be used if selected in QuoteManager.

Example

Assign a value, indicating the day of the week that the 4th regular session starts, to Value1 variable:

```
Value1 = SessionStartDay(1,4);
```

SessionStartDayMS

Returns a numerical value indicating the day of the week that the specified merged session in a multi-data series chart starts, where 0 = Sunday, 1 = Monday, etc.

Merged session extends from the earliest start time to the latest end time of all sessions merged.

Usage

`SessionStartDayMS`(*SessionNum*)

Where: *`SessionNum`* - a numerical expression specifying the Session Number

Example

Assign a value, indicating the day of the week that the 4th merged session starts, to Value1 variable:

```
Value1 = SessionStartDayMS(4);
```

SessionStartTime

Returns a numerical value, indicating the time of the day that the specified session starts. The time is indicated in the 24-hour HHmm format, where 1300 = 1:00 PM.

Usage

`SessionStartTime`(*SessionType*,*SessionNum*)

Where: *`SessionType`* - a numerical expression: 0 = Auto Detect, 1 = Regular Session*
`SessionNum` - a numerical expression specifying the Session Number

* Custom Sessions parameters will be used if selected in QuoteManager.

Example

Assign a value, indicating the time of the day that the 4th regular session starts, to Value1 variable:

```
Value1 = SessionStartTime(1,4);
```

SessionStartTimeMS

Returns a numerical value, indicating the time of the day that the specified merged session in a multi-data series chart starts. The time is indicated in the 24-hour HHmm format, where 1300 = 1:00 PM.

Merged session extends from the earliest start time to the latest end time of all sessions merged.

Usage

`SessionStartTimeMS`(*SessionNum*)

Where: *`SessionNum`* - a numerical expression specifying the Session Number

Example

Assign a value, indicating the time of day that the 4th merged session starts, to Value1 variable:

```
Value1 = SessionStartTimeMS(4);
```

A

A Skip Word. Skip Words serve solely to improve the readability of PowerLanguage code and are skipped (ignored) during the compilation and execution.

The use of Skip Words is optional; they can be inserted anywhere within the PL code and will appear in red in PowerLanguage Editor.

Example

The Skip Words in the example below are colored in red:

```
If an Open is < than Close of 4 Bars Ago was Then Buy This Bar on Close;
If Plot1 does Cross Above a 1350 Then Sell From Entry("My Entry") Next
Bar at the Market;
```

An

A Skip Word. Skip Words serve solely to improve the readability of PowerLanguage code and are skipped (ignored) during the compilation and execution.

The use of Skip Words is optional; they can be inserted anywhere within the PL code and will appear in red in PowerLanguage Editor.

Example

The Skip Words in the example below are colored in red:

```
If an Open is < than Close of 4 Bars Ago was Then Buy This Bar on Close;
If Plot1 does Cross Above a 1350 Then Sell From Entry("My Entry")Next
Bar at the Market;
```

At

A Skip Word. Skip Words serve solely to improve the readability of PowerLanguage code and are skipped (ignored) during the compilation and execution.

The use of Skip Words is optional; they can be inserted anywhere within the PL code and will appear in red in PowerLanguage Editor.

Example

The Skip Words in the example below are colored in red:

```
If an Open is < than Close of 4 Bars Ago was Then Buy This Bar on Close;
If Plot1 does Cross Above a 1350 Then Sell From Entry("My Entry")Next
Bar at the Market;
```

Based

A Skip Word. Skip Words serve solely to improve the readability of PowerLanguage code and are skipped (ignored) during the compilation and execution.

The use of Skip Words is optional; they can be inserted anywhere within the PL code and will appear in red in PowerLanguage Editor.

Example

The Skip Words in the example below are colored in red:

```
If an Open is < than Close of 4 Bars Ago was Then Buy This Bar on Close;
If Plot1 does Cross Above a 1350 Then Sell From Entry("My Entry")Next
Bar at the Market;
```

By

A Skip Word. Skip Words serve solely to improve the readability of PowerLanguage code and are skipped (ignored) during the compilation and execution.

The use of Skip Words is optional; they can be inserted anywhere within the PL code and will appear in red in PowerLanguage Editor.

Example

The Skip Words in the example below are colored in red:

```
If an Open is < than Close of 4 Bars Ago was Then Buy This Bar on Close;
If Plot1 does Cross Above a 1350 Then Sell From Entry("My Entry")Next
Bar at the Market;
```

Does

A Skip Word. Skip Words serve solely to improve the readability of PowerLanguage code and are skipped (ignored) during the compilation and execution.

The use of Skip Words is optional; they can be inserted anywhere within the PL code and will appear in red in PowerLanguage Editor.

Example

The Skip Words in the example below are colored in red:

```
If an Open is < than Close of 4 Bars Ago was Then Buy This Bar on Close;
If Plot1 does Cross Above a 1350 Then Sell From Entry("My Entry")Next
Bar at the Market;
```

From

A skip word; used in strategy exit statements in combination with `Entry` word that ties an exit to the particular entry that was assigned the *`EntryLabel`* name.

An exit can only be tied to an entry within the same signal; for more information, see `Buy` or `SellShort`.

Skip Words serve solely to improve the readability of PowerLanguage code and are skipped (ignored) during the compilation and execution.

The use of Skip Words is optional; they can be inserted anywhere within the PL code and will appear in red in PowerLanguage Editor.

Usage

```
From Entry("EntryLabel")
```

Where: *`EntryLabel`* - the name that was assigned to the entry that the exit is to be tied to

`From` - a skip word and can be omitted

Example

Completely exit from the long position established by the entry labeled "Original Entry", at Market price on open of next bar:

```
Sell From Entry("Original Entry")Next Bar at Open;
```

Is

A Skip Word. Skip Words serve solely to improve the readability of PowerLanguage code and are skipped (ignored) during the compilation and execution.

The use of Skip Words is optional; they can be inserted anywhere within the PL code and will appear in red in PowerLanguage Editor.

Example

The Skip Words in the example below are colored in red:

```
If an Open is < than Close of 4 Bars Ago was Then Buy This Bar on Close;
If Plot1 does Cross Above a 1350 Then Sell From Entry("My Entry")Next
Bar at the Market;
```

Of

A Skip Word. Skip Words serve solely to improve the readability of PowerLanguage code and are skipped (ignored) during the compilation and execution.

The use of Skip Words is optional; they can be inserted anywhere within the PL code and will appear in red in PowerLanguage Editor.

Example

The Skip Words in the example below are colored in red:

```
If an Open is < than Close of 4 Bars Ago was Then Buy This Bar on Close;
If Plot1 does Cross Above a 1350 Then Sell From Entry("My Entry")Next
Bar at the Market;
```

On

A Skip Word. Skip Words serve solely to improve the readability of PowerLanguage code and are skipped (ignored) during the compilation and execution.

The use of Skip Words is optional; they can be inserted anywhere within the PL code and will appear in red in PowerLanguage Editor.

Example

The Skip Words in the example below are colored in red:

```
If an Open is < than Close of 4 Bars Ago was Then Buy This Bar on Close;
If Plot1 does Cross Above a 1350 Then Sell From Entry("My Entry")Next
Bar at the Market;
```

Place

A Skip Word retained for backward compatibility. Skip Words serve solely to improve the readability of PowerLanguage code and are skipped (ignored) during the compilation and execution.

The use of Skip Words is optional; they can be inserted anywhere within the PL code and will appear in red in PowerLanguage Editor.

Example

The Skip Words in the example below are colored in red:

```
If an Open is < than Close of 4 Bars Ago was Then Buy This Bar on Close;
If Plot1 does Cross Above a 1350 Then Sell From Entry("My Entry")Next
Bar at the Market;
```

Than

A Skip Word. Skip Words serve solely to improve the readability of PowerLanguage code and are skipped (ignored) during the compilation and execution.

The use of Skip Words is optional; they can be inserted anywhere within the PL code and will appear in red in PowerLanguage Editor.

Example

The Skip Words in the example below are colored in red:

```
If an Open is < than Close of 4 Bars Ago was Then Buy This Bar on Close;
If Plot1 does Cross Above a 1350 Then Sell From Entry("My Entry")Next
Bar at the Market;
```

The

A Skip Word. Skip Words serve solely to improve the readability of PowerLanguage code and are skipped (ignored) during the compilation and execution.

The use of Skip Words is optional; they can be inserted anywhere within the PL code and will appear in red in PowerLanguage Editor.

Example

The Skip Words in the example below are colored in red:

```
If an Open is < than Close of 4 Bars Ago was Then Buy This Bar on Close;
If Plot1 does Cross Above a 1350 Then Sell From Entry("My Entry")Next
Bar at the Market;
```

Was

A Skip Word. Skip Words serve solely to improve the readability of PowerLanguage code and are skipped (ignored) during the compilation and execution.

The use of Skip Words is optional; they can be inserted anywhere within the PL code and will appear in red in PowerLanguage Editor.

Example

The Skip Words in the example below are colored in red:

```
If an Open is < than Close of 4 Bars Ago was Then Buy This Bar on Close;
If Plot1 does Cross Above a 1350 Then Sell From Entry("My Entry")Next
Bar at the Market;
```

All

Used in strategy exit statements in place of a numerical expression, preceding the words `Shares` or `Contracts`, to denote all of the shares or all of the contracts held without specifying the actual number of shares or contracts.

Usage

```
All Contracts
```

Example

Sell all of the shares held at market price on open of next bar:

```
Sell All Shares Next Bar At Market;
```

Buy

Enters a long position as specified by the parameters.

The entry point is visually indicated on a chart by an Arrow and a Tick. The Arrow identifies the time and the Tick identifies the price value of the entry point. Labels, displaying the entry name and the number of contracts or shares traded, are displayed below the Buy Arrow.

An order is executed at the point specified by the parameters; if the order is not filled within the specified bar, the order is cancelled.

When a Buy order is filled, any open short positions will also be closed.

Usage

`Buy`[("*`EntryLabel`*")][*TradeSize*]*`EntryType`*;

Parameters inside the square brackets are optional

Parameters

`EntryLabel` - an optional parameter; assigns a name that will be displayed in the name label below the entry, and can be used to identify the particular entry and to tie an exit to it.

An exit can only be tied to an entry within the same signal; for more information, see `Sell`.

If *`EntryLabel`* is not specified, the name "Buy" will be used for the first entry, "Buy#2" for the second entry, "Buy#3" for the third entry, etc.

`TradeSize` - an optional parameter; a numerical expression, specifying the number of contracts or shares to buy; the expression must be followed by one of the following transposable words: `Share`, `Shares`, `Contract` or `Contracts`.

If *`TradeSize`* value equals 0 or is negative, a long position will not be entered but any open short positions will be closed.

If *`TradeSize`* is not specified, the trade size value set by the user in the Properties tab of the Strategy Properties window will be used.

`EntryType` - a required parameter; specifies the timing and price of entry.

There are four kinds of EntryType:

`This Bar[On]Close`

Where: - `On` is a skip word and can be omitted

A Buy Arrow will be placed at the current bar's `Close` tick.

`Next Bar[At]Open` or `Next Bar[At]Market`

Where: - words "`Market`" and "`Open`" are transposable
- `At` is a skip word and can be omitted

A Buy Arrow will be placed at the next bar's `Open` tick.

`Next Bar[At]Price Limit`

Where: - `Price` is a numerical expression, specifying the Limit Price
- `At` is a skip word and can be omitted

A Buy Arrow will be placed on the next bar at the first tick with a price value less than or equal to `Price` ; if there are no such ticks within the next bar, the order will be cancelled.

`Next Bar[At]Price Stop`

Where: - `Price` is a numerical expression, specifying the Stop Price
- `At` is a skip word and can be omitted

A Buy Arrow will be placed on the next bar at the first tick with a price value equal to or greater than the `Price` ; if there are no such ticks within the next bar, the order will be cancelled.

Example

Buy a user-set number of shares at Market price on close of this bar:

```
Buy This Bar On Close;
```

Buy 1 share at Market price on open of next bar and label the entry "Entry":

```
Buy("Entry")1 Share Next Bar At Open;
```

Buy 1 contract at Market price on open of next bar and label the entry "Entry":

```
Buy("Entry")1 Contract Next Bar Market;
```

Buy 2 shares within the next bar on the first tick with a price of 100 or less:

```
Buy 2 Shares Next Bar At 100 Limit;
```

Buy 10 contracts within the next bar on the first tick with a price of 50 or more:

```
Buy 10 Contracts Next Bar 50 Stop;
```

BuyToCover

Completely or partially exits one or all of the short entries as specified by the parameters.

The exit point is visually indicated on a chart by an Arrow and a Tick. The Arrow identifies the time and the Tick identifies the price value of the exit point. Labels, displaying the exit name and the number of contracts or shares traded, are displayed below the Cover Arrow.

An order is executed at the point specified by the parameters; if the order is not filled within the specified bar, the order is cancelled.

Usage

`BuyToCover`[("*`ExitLabel`*")][`From Entry`("*`EntryLabel`*")][*TradeSize*[`Total`]]*`Exit`*

or:

`Buy To Cover`[("*`ExitLabel`*")][`From Entry`("*`EntryLabel`*")]
[*TradeSize*[`Total`]]*`Exit`*

Parameters inside the square brackets are optional

Parameters

`ExitLabel` - an optional parameter; assigns a name that will be displayed in the name label above the exit

If *`ExitLabel`* is not specified, the name "Cover" will be used for the first exit, "Cover#2" for the second exit, "Cover#3" for the third exit, etc.

`EntryLabel` - an optional parameter; ties the exit to the particular entry that was assigned the *`EntryLabel`* name; the name must be preceded by the word `Entry`, the word `From` is a skip word and can be omitted

An exit can only be tied to an entry within the same signal. For more information, see `SellShort`

If *`EntryLabel`* is not specified, **all** of the open short entries will be closed.

`TradeSize` - an optional parameter; a numerical expression, specifying the number of contracts or shares to buy; the expression must be followed by one of the following transposable words: `Share`, `Shares`, `Contract` or `Contracts`.

By default, the number of contracts or shares specified by the *`TradeSize`* parameter will be covered from **each** of the open short entries.

If *`TradeSize`* is followed by the word `Total`, only the number of contracts or shares specified by the *`TradeSize`* parameter will be covered, regardless of the number of open short entries. The contracts or shares will be covered in the same order they were shorted: First In, First Out.

If *`TradeSize`* is not specified, the entire short position will be closed out.

`Exit` - a required parameter; specifies the timing and price of exit.

There are four types of Exit:

`This Bar[On]Close`

Where: - `On` is a skip word and can be omitted

A Cover Arrow will be placed at the current bar's `Close` tick.

`Next Bar[At]Open` or `Next Bar[At]Market`

Where: - words "`Market`" and "`Open`" are transposable
- `At` is a skip word and can be omitted

A Cover Arrow will be placed at the next bar's `Open` tick.

`Next Bar[At]`*`Price`*` Limit`

Where: - *`Price`* is a numerical expression, specifying the Limit Price
- `At` is a skip word and can be omitted

A Cover Arrow will be placed on the next bar at the first tick with a price value less than or equal to *`Price`* ; if there are no such ticks within the next bar, the order will be cancelled.

```
Next Bar[At]Price Stop
```

Where: - `Price` is a numerical expression, specifying the Stop Price
- `At` is a skip word and can be omitted

A Cover Arrow will be placed on the next bar at the first tick with a price value equal to or greater than the `Price` ; if there are no such ticks within the next bar, the order will be cancelled.

Example

Completely exit all open short entries at Market price on close of this bar and label the exit "Complete Exit":

```
BuyToCover("Complete Exit")This Bar On Close;
```

Completely exit from the short position established by the entry labeled "Original Entry", at Market price on open of next bar:

```
BuyToCover From Entry("Original Entry")Next Bar At Open;
```

Cover 10 shares of the short position established by the entry labeled "Original Entry", at Market price on open of next bar:

```
BuyToCover Entry("Original Entry")10 Shares Next Bar At Market;
```

Cover 5 contracts for each one of the open short entries at Market price on open of next bar:

```
BuyToCover 5 Contracts Next Bar Market;
```

Cover a total of 1 share, regardless of the number of open short entries, within the next bar on the first tick with a price of 100 or less (the first share shorted will be covered if the price is met):

```
BuyToCover 1 Share Total Next Bar At 100 Limit;
```

Completely exit all short entries within the next bar on the first tick with a price of 50 or more:

```
BuyToCover Next Bar 50 Stop;
```

Contract

Same as `Contracts`

Contracts

Used in strategy entry or exit statements in combination with a numerical expression to specify the number of contracts or shares to trade.

Usage

```
TradeSize Contracts
```

Where: *`TradeSize`* - a numerical expression, specifying the number of contracts or shares

Example

Buy 2 contracts at Market price on open of next bar:

```
Buy 2 Contracts Next Bar At Market;
```

Cover

Used in combination with the words `Buy To`; same as `BuyToCover`.

Usage

`Buy To Cover`[("*`ExitLabel`*")][`From Entry`("*`EntryLabel`*")]
[*TradeSize*[`Total`]]*`Exit`*;

Entry

Used in strategy exit statements to tie an exit to the particular entry that was assigned the *`EntryLabel`* name.

An exit can only be tied to an entry within the same signal; for more information, see `Buy` or `SellShort`.

Usage

`From Entry("EntryLabel")`

Where: *`EntryLabel`* - the name that was assigned to the entry that the exit is to be tied to

`From` - a skip word and can be omitted

Example

Completely exit from the long position established by the entry labeled "Original Entry", at Market price on open of next bar:

```
Sell From Entry("Original Entry")Next Bar At Open;
```

Higher

Used in strategy entry or exit statements to specify a price range for an entry or an exit; must be preceded by the word `Or`.

Usage

```
At Price Or Higher
```

Where: *`Price`* - a numerical expression, specifying the base Price
`At` - a skip word and can be omitted

Notes

`At` *`Price`* `Or Higher` is an equivalent of a Limit when used with `Sell` or `SellShort` statements, and an equivalent of a Stop when used with `Buy` or `BuyToCover` statements.

Example

Buy within the next bar on the first tick with a price of 100 or more:

```
Buy Next Bar At 100 Or Higher;
```

Sell short within the next bar on the first tick with a price of 50 or more:

```
SellShort Next Bar At 50 Or Higher;
```

Limit

Used in strategy entry or exit statements to specify a Limit price for an entry or an exit.

A Limit order will execute at the specified price or better. A better price is a lower price for Buy and Buy to cover orders, and a higher price for Sell and Sell short orders.

Usage

```
At Price Limit
```

Where: `Price` - a numerical expression, specifying the Limit Price
`At` - a skip word and can be omitted

Example

Buy within the next bar on the first tick with a price of 100 or less:

```
Buy Next Bar At 100 Limit;
```

Sell short within the next bar on the first tick with a price of 50 or more:

```
SellShort Next Bar 50 Limit;
```

Lower

Used in strategy entry or exit statements to specify a price range for an entry or an exit.

Must be preceded by the word `Or`.

Usage

```
At Price Or Lower
```

Where: *`Price`* - a numerical expression, specifying the base Price
`At` - a skip word and can be omitted

Notes

`At` *`Price`* `Or Lower` is an equivalent of a Limit when used with `Buy` or `BuyToCover` statements, and an equivalent of a Stop when used with `Sell` or `SellShort` statements.

Example

Buy within the next bar on the first tick with a price of 100 or less:

```
Buy Next Bar At 100 Or Lower;
```

Sell short within the next bar on the first tick with a price of 50 or less:

```
SellShort Next Bar At 50 Or Lower;
```

Market

Used in strategy entry or exit statements to specify a Market price for an entry or an exit.

A Market Buy order will execute at the current ask price and a Maret Sell order will execute at the current bid price.

Usage

```
At Market
```

Where: `At` is a skip word and can be omitted

Example

Buy a user-set number of shares at Market price on open of next bar:

```
Buy Next Bar At Market;
```

Sell

Completely or partially exits one or all of the long entries as specified by the parameters.

The exit point is visually indicated on a chart by an Arrow and a Tick. The Arrow identifies the time and the Tick identifies the price value of the exit point. Labels, displaying the exit name and the number of contracts or shares traded, are displayed above the Sell Arrow.

An order is executed at the point specified by the parameters; if the order is not filled within the specified bar, the order is cancelled.

Usage `Sell`[("*`ExitLabel`*")][`From Entry`("*`EntryLabel`*")][*TradeSize*[`Total`]]*`Exit`*

Parameters inside the square brackets are optional

Parameters
`ExitLabel` - an optional parameter; assigns a name that will be displayed in the name label above the exit

If *`ExitLabel`* is not specified, the name "Sell" will be used for the first exit, "Sell#2" for the second exit, "Sell#3" for the third exit, etc.

`EntryLabel` - an optional parameter; ties the exit to the particular entry that was assigned the *`EntryLabel`* name; the name must be preceded by the word "`Entry`", the word "`From`" is a skip word and can be omitted

An exit can only be tied to an entry within the same signal. For more information, see `Buy`

If *`EntryLabel`* is not specified, **all** of the open long entries will be closed.

TradeSize - an optional parameter; a numerical expression, specifying the number of contracts or shares to sell; the expression must be followed by one of the following transposable words: `Share`, `Shares`, `Contract` or `Contracts`.

By default, the number of contracts or shares specified by the *`TradeSize`* parameter will be sold from **each one** of the open long entries.

If *`TradeSize`* is followed by the word "`Total`", only the number of contracts or shares specified by the *`TradeSize`* parameter will be sold, regardless of the number of open long entries. The contracts or shares will be sold in the same order they were bought: First In, First Out.

If *`TradeSize`* is not specified, the entire long position will be closed out.

`Exit` - a required parameter; specifies the timing and price of exit.

There are four types of Exit:

`This Bar[On]Close`

Where: - `On` is a skip word and can be omitted

A Sell Arrow will be placed at the current bar's `Close` tick.

`Next Bar[At]Open or Next Bar[At]Market`

Where: - words "`Market`" and "`Open`" are transposable
- `At` is a skip word and can be omitted

A Sell Arrow will be placed at the next bar's `Open` tick.

`Next Bar[At]`*`Price`*` Limit`

Where: - *`Price`* is a numerical expression, specifying the Limit Price
- `At` is a skip word and can be omitted

A Sell Arrow will be placed on the next bar at the first tick with a price value greater than or equal to *`Price`* ; if there are no such ticks within the next bar, the order will be cancelled.

`Next Bar[At]`*`Price`*` Stop`

Where: - *`Price`* is a numerical expression, specifying the Stop Price
- `At` is a skip word and can be omitted

A Sell Arrow will be placed on the next bar at the first tick with a price value equal to or less than the *`Price`* ; if there are no such ticks within the next bar, the order will be cancelled.

Example

Completely exit all open long entries at Market price on close of this bar and label the exit "Complete Exit":

```
Sell("Complete Exit")This Bar On Close;
```

Completely exit from the long position established by the entry labeled "Original Entry", at Market price on open of next bar:

```
Sell From Entry("Original Entry")Next Bar At Open;
```

Sell 10 shares of the long position established by the entry labeled "Original Entry", at Market price on open of next bar:

```
Sell Entry("Original Entry")10 Shares Next Bar At Market;
```

Sell 5 contracts for all open long entries at Market price on open of next bar:

```
Sell 5 Contracts Next Bar Market;
```

Sell a total of 1 share, regardless of the number of open long entries, within the next bar on the first tick with a price of 100 or more (the longest-held share will be sold if the price is met):

```
Sell 1 Share Total Next Bar At 100 Limit;
```

Completely exit all long entries within the next bar on the first tick with a price of 50 or less:

```
Sell Next Bar 50 Stop;
```

SellShort

Enters a short position as specified by the parameters.

The entry point is visually indicated on a chart by an Arrow and a Tick. The Arrow identifies the time and the Tick identifies the price value of the entry point. Labels, displaying the entry name and the number of contracts or shares traded, are displayed above the Short Arrow.

An order is executed at the point specified by the parameters; if the order is not filled within the specified bar, the order is cancelled.

When a Sell Short order is filled, any open long positions will also be closed.

Usage

`SellShort`[("*`EntryLabel`*")][*TradeSize*]*`Entry`*

or:

`Sell Short`[("*`EntryLabel`*")][*TradeSize*]*`Entry`*

Parameters inside the square brackets are optional

Parameters

`EntryLabel` - an optional parameter; assigns a name that will be displayed in the name label above the entry, and can be used to identify the particular entry and to tie an exit to it.

An exit can only be tied to an entry within the same signal; for more information, see `BuyToCover`.

If *`EntryLabel`* is not specified, the name "Short" will be used for the first entry, "Short#2" for the second entry, "Short#3" for the third entry, etc.

`TradeSize` - an optional parameter; a numerical expression, specifying the number of contracts or shares to sell short; the expression must be followed by one of the following transposable words: `Share`, `Shares`, `Contract` or `Contracts`.

If *`TradeSize`* value equals 0 or is negative, a short position will not be entered but any open long positions will be closed.

If *`TradeSize`* is not specified, the trade size value set by the user in the Properties tab of the Strategy Properties window will be used.

`Entry` - a required parameter; specifies the timing and price of entry.

There are four types of Entry:

`This Bar[On]Close`

Where: - `On` is a skip word and can be omitted

A Short Arrow will be placed at the current bar's `Close` tick.

`Next Bar[At]Open or Next Bar[At]Market`

Where: - words "`Market`" and "`Open`" are transposable
- `At` is a skip word and can be omitted

A Short Arrow will be placed at the next bar's `Open` tick.

`Next Bar[At]`*`Price`*` Limit`

Where: - *`Price`* is a numerical expression, specifying the Limit Price
- `At` is a skip word and can be omitted

A Short Arrow will be placed on the next bar at the first tick with a price equal to or greater than the *`Price`* ; if there are no such ticks within the next bar, the order will be cancelled.

`Next Bar[At]`*`Price`*` Stop`

Where: - *`Price`* is a numerical expression, specifying the Stop Price
- `At` is a skip word and can be omitted

A Short Arrow will be placed on the next bar at the first tick with a price value less than or equal to *`Price`* ; if there are no such ticks within the next bar, the order will be cancelled.

Example

SellShort a user-set number of shares at Market price on close of this bar:

```
SellShort This Bar On Close;
```

SellShort 1 share at Market price on open of next bar and label the entry "Entry":

```
SellShort("Entry")1 Share Next Bar At Open;
```

SellShort 1 contract at Market price on open of next bar and label the entry "Entry":

```
SellShort("Entry")1 Contract Next Bar Market;
```

SellShort 2 shares within the next bar on the first tick with a price of 100 or more:

```
SellShort 2 Shares Next Bar At 100 Limit;
```

SellShort 10 contracts within the next bar on the first tick with a price of 50 or less:

```
SellShort 10 Contracts Next Bar 50 Stop;
```

SetBreakEven

Closes out the entire position or the entry if it is at the breakeven point after the profit has reached the specified value; generates the appropriate Stop order depending on whether the position is long or short.

`SetStopPosition` and `SetStopContract` or `SetStopShare` functions determine whether SetBreakEven will be applied to the entire position or to each contract or share individually; by default, SetBreakEven is applied to the entire position.

SetBreakEven function is evaluated intra-bar and not only on close of a bar, and can exit within the same bar as the entry.

Usage

`SetBreakEven(`*`Profit`*`)`

Where: *`Profit`* - a numerical expression, specifying the currency value of the profit that must be reached first

Notes

This function can only be used in signals.

SetBreakEven function does not factor in commissions or slippage.

Amount can be set either in the currency of the symbol or in the currecy of the strategy, depending on the key set in Windows Registry.

Go to HKEY_CURRENT_USER\Software\TS Support\[ProductName]\StrategyProp and create a key DWORD Value: SpecOrdersAmountIsStrategyCurr.

0 - to calculate Amount in the currency of the symbol.

1 - to calculate Amount in the currency of the strategy/Portfolio (by default).

[ProductName] is name of product, for example, for 32-bit MultiCharts = "MultiCharts", for 64-bit version = "MultiCharts64".

If there is no such a key, Amount is calculated in the currency of the strategy/Portfolio.

Example

Generate an exit order for the entire position if it is at the breakeven point after position profit has reached ¤50:

```
SetStopPosition;

SetBreakEven(50);
```

Generate an exit order for the entry if it is at the breakeven point after per contract profit has reached ¤10:

```
SetStopContract;

SetBreakEven(10);
```

SetBreakEven_pt

Closes out the entire position or the entry if it is at the breakeven point after the profit has reached the specified tick value; generates the appropriate Stop order depending on whether the position is long or short.

`SetStopPosition` and `SetStopContract` or `SetStopShare` functions determine whether single SetBreakEven_pt order will be applied to the entire position or multiple SetBreakEven_pt orders will be applied to each entry in position individually; by default, SetBreakEven_pt is applied to the entire position.

SetBreakEven_pt function is evaluated intra-bar and not only on close of a bar, and can exit within the same bar as the entry.

Usage

`SetBreakEven_pt(`*`Profit`*`)`

Where: *`Profit`* - a numerical expression, specifying the amount of the profit in ticks that must be reached first

Notes

This function can only be used in signals.

SetBreakEven_pt function does not factor in commissions or slippage.

Example

Generate an exit order for the entire position if it is at the breakeven point after position profit has reached 50 ticks:

```
SetStopPosition;
```

```
SetBreakEven_pt(50);
```

Generate an exit order for the entry if it is at the breakeven point after per entery profit has reached 10 ticks:

```
SetStopContract;

SetBreakEven_pt(10);
```

SetDollarTrailing

Closes out the entire position or the entry if the current profit is less than the maximum profit by the specified amount; generates the appropriate Stop order depending on whether the position is long or short.

For example, if the specified ammount is ¤50 and the profit has reached the maximum of ¤120, the position will be closed once the profit drops to ¤70.

`SetStopPosition` and `SetStopContract` or `SetStopShare` functions determine whether SetDollarTrailing will be applied to the entire position or to each contract or share individually; by default, SetDollarTrailing is applied to the entire position.

SetDollarTrailing function is evaluated intra-bar and not only on close of a bar, and can exit within the same bar as the entry.

Usage

`SetDollarTrailing(`*`Amount`*`)`

Where: *`Amount`* - a numerical expression, specifying the currency value of the maximum loss of profit

Notes

This function can only be used in signals.

Amount can be set either in the currency of the symbol or in the currecy of the strategy, depending on the key set in Windows Registry.

Go to HKEY_CURRENT_USER\Software\TS Support\[ProductName]\StrategyProp and create a key DWORD Value: SpecOrdersAmountIsStrategyCurr.

0 - to calculate Amount in the currency of the symbol.

1 - to calculate Amount in the currency of the strategy/Portfolio (by default).

[ProductName] is name of product, for example, for 32-bit MultiCharts =

"MultiCharts", for 64-bit version = "MultiCharts64".

If there is no such a key, Amount is calculated in the currency of the strategy/Portfolio.

Example

Generate an exit order for the entire position if position profit drops by ¤50:

```
SetStopPosition;

SetDollarTrailing(50);
```

Generate an exit order for the entry if per contract profit drops by ¤10:

```
SetStopContract;

SetDollarTrailing(10);
```

SetExitOnClose

Closes out the current position at the Close tick of the last bar of the trading session on an intra-day chart; generates the appropriate Market exit order depending on whether the position is long or short.

SetExitOnClose function uses the session closing time specified in the session settings for the symbol in the QuoteManager.

Usage

```
SetExitOnClose
```

Notes

This function can only be used in signals.

Example

Generate an exit order at the Close tick of the last bar of the trading session:

```
SetExitOnClose;
```

SetPercentTrailing

Closes out the entire position or the entry if the specified percentage of the maximum profit is lost after the profit has reached the specified value; generates the appropriate Stop order depending on whether the position is long or short.

For example, if the specified profit is ¤100 and the specified percentage is 50, and the profit has reached the maximum of ¤120, the position will be closed once the profit falls back to ¤60.

`SetStopPosition` and `SetStopContract` or `SetStopShare` functions determine whether SetPercentTrailing will be applied to the entire position or each contract or share individually; by default, SetPercentTrailing is applied to the entire position.

SetPercentTrailing function is evaluated intra-bar and not only on close of a bar, and can exit within the same bar as the entry.

Usage

`SetPercentTrailing(`*`Profit`*`,`*`Percentage`*`)`

Where: *`Profit`* - a numerical expression, specifying the currency value of the profit that must be reached first
`Percentage` - a numerical expression, specifying the maximum loss of profit in percent

Notes

This function can only be used in signals.

Amount can be set either in the currency of the symbol or in the currecy of the strategy, depending on the key set in Windows Registry.

Go to HKEY_CURRENT_USER\Software\TS Support\[ProductName]\StrategyProp and create a key DWORD Value: SpecOrdersAmountIsStrategyCurr.

0 - to calculate Amount in the currency of the symbol.

1 - to calculate Amount in the currency of the strategy/Portfolio (by default).

[ProductName] is name of product, for example, for 32-bit MultiCharts = "MultiCharts", for 64-bit version = "MultiCharts64".

If there is no such a key, Amount is calculated in the currency of the strategy/Portfolio.

Example

Generate an exit order for the entire position if 50 percent of maximum position profit is lost after the profit has reached ¤25:

```
SetStopPosition;

SetPercentTrailing(25,50);
```

Generate an exit order for the entry if 25 percent of maximum per share profit is lost after the profit has reached ¤5:

```
SetStopShare;

SetPercentTrailing(5,25);
```

SetPercentTrailing_pt

Closes out the entire position or the entry if the specified percentage of the maximum profit is lost after the profit has reached the specified tick value; generates the appropriate Stop order depending on whether the position is long or short.

For example, if the specified profit is 100 ticks and the specified percentage is 50, and the profit has reached the maximum of 120 ticks, the position will be closed once the profit falls back to 60 ticks.

`SetStopPosition` and `SetStopContract` or `SetStopShare` functions determine whether single SetPercentTrailing_pt order will be applied to the entire position or multiple SetPercentTrailing_pt orders will be applied to each entry in position individually; by default, SetPercentTrailing_pt is applied to the entire position.

SetPercentTrailing_pt function is evaluated intra-bar and not only on close of a bar, and can exit within the same bar as the entry.

Usage

`SetPercentTrailing_pt(`*`Profit`*`,`*`Percentage`*`)`

Where: *`Profit`* - a numerical expression, specifying the tick value of the profit that must be reached first
`Percentage` - a numerical expression, specifying the maximum loss of profit in percent

Notes

This function can only be used in signals.

SetPercentTrailing_pt function does not factor in commissions or slippage.

Example

Generate an exit order for the entire position if 50 percent of maximum position profit is lost after the profit has reached 25 ticks:

```
SetStopPosition;

SetPercentTrailing_pt(25,50);
```

Generate an exit order for the entry if 25 percent of maximum per entry profit is lost after the profit has reached 5 ticks:

```
SetStopShare;

SetPercentTrailing_pt(5,25);
```

SetProfitTarget

Closes out the entire position or the entry if profit reaches the specified currency value; generates the appropriate Limit exit order depending on whether the position is long or short.

`SetStopPosition` and `SetStopContract` or `SetStopShare` functions determine whether the profit target will be applied to the entire position or to each contract or share individualy; by default, profit target is applied to the entire position.

SetProfitTarget function is evaluated intra-bar and not only on close of a bar, and can exit within the same bar as the entry.

Usage

`SetProfitTarget(`*`Amount`*`)`

Where: *`Amount`* - a numerical expression, specifying the profit target amount

Notes

This function can only be used in signals.

Amount can be set either in the currency of the symbol or in the currecy of the strategy, depending on the key set in Windows Registry.

Go to HKEY_CURRENT_USER\Software\TS Support\[ProductName]\StrategyProp and create a key DWORD Value: SpecOrdersAmountIsStrategyCurr.

0 - to calculate Amount in the currency of the symbol.

1 - to calculate Amount in the currency of the strategy/Portfolio (by default).

[ProductName] is name of product, for example, for 32-bit MultiCharts = "MultiCharts", for 64-bit version = "MultiCharts64".

If there is no such a key, Amount is calculated in the currency of the strategy/Portfolio.

Example

Generate an exit order for the entire position if the position profit reaches ¤100:

```
SetStopPosition;

SetProfitTarget(100);
```

Generate an exit order for the entry if the profit per contract reaches ¤10:

```
SetStopContract;

SetProfitTarget(10);
```

SetProfitTarget_pt

Closes out the entire position or the entry if profit reaches the specified tick value; generates the appropriate Limit exit order depending on whether the position is long or short.

`SetStopPosition` and `SetStopContract` or `SetStopShare` functions determine whether single profit target order will be applied to the entire position or multiple SetProfitTarget_pt orders will be applied to each entry in position individualy; by default, profit target is applied to the entire position.

SetProfitTarget_pt function is evaluated intra-bar and not only on close of a bar, and can exit within the same bar as the entry.

Usage

`SetProfitTarget_pt(`*`Amount`*`)`

Where: *`Amount`* - a numerical expression, specifying the profit target amount in ticks.

Notes

This function can only be used in signals.

SetProfitTarget_pt function does not factor in commissions or slippage.

Example

Generate an exit order for the entire position if the position profit reaches 100 ticks:

```
SetStopPosition;
```

```
SetProfitTarget_pt(100);
```

Generate an exit order for the entry if the profit per entry reaches 10 ticks:

```
SetStopContract;

SetProfitTarget_pt(10);
```

SetStopContract

Forces the built-in strategy exit functions to be applied on per contract or share basis.

The built-in strategy exit functions are: `SetStopLoss` , `SetProfitTarget` , `SetBreakEven` , `SetDollarTrailing` , and `SetPercentTrailing`.

Usage

```
SetStopContract
```

Notes

If `SetStopPositon`, `SetStopContract`, and `SetStopShare` were not used, the exit functions will be applied on the entire position basis as a default.

If `SetStopPositon`, `SetStopContract`, and `SetStopShare` were used in multiple instances or in different signals applied to the same chart, the last instance will be controlling.

Example

Force the SetStopLoss strategy exit function to be applied on per contract basis:

```
SetStopContract;

SetStopLoss(10);
```

An exit order for the entry will be generated if the loss per contract reaches ¤10.

SetStopLoss

Closes out the entire position or the entry if the loss reaches the specified currency value; generates the appropriate Stop order depending on whether the position is long or short.

`SetStopPosition` and `SetStopContract` or `SetStopShare` functions determine whether the stop loss will be applied to the entire position or to each contract or share individually; by default, stop loss is applied to the entire position.

SetStopLoss function is evaluated intra-bar and not only on close of a bar, and can exit within the same bar as the entry.

Usage

`SetStopLoss(`*`Amount`*`)`

Where: *`Amount`* - a numerical expression, specifying the stop loss amount

Notes

This function can only be used in signals.

Amount can be set either in the currency of the symbol or in the currecy of the strategy, depending on the key set in Windows Registry.

Go to HKEY_CURRENT_USER\Software\TS Support\[ProductName]\StrategyProp and create a key DWORD Value: SpecOrdersAmountIsStrategyCurr.

0 - to calculate Amount in the currency of the symbol.

1 - to calculate Amount in the currency of the strategy/Portfolio (by default).

[ProductName] is name of product, for example, for 32-bit MultiCharts = "MultiCharts", for 64-bit version = "MultiCharts64".

If there is no such a key, Amount is calculated in the currency of the strategy/Portfolio.

Example

Generate an exit order for the entire position if the position loss reaches ¤100:

```
SetStopPosition;

SetStopLoss(100);
```

Generate an exit order for the entry if the loss per contract reaches ¤10:

```
SetStopContract;

SetStopLoss(10);
```

SetStopLoss_pt

Closes out the entire position or the entry if the loss reaches the specified tick value; generates the appropriate Stop order depending on whether the position is long or short.

`SetStopPosition` and `SetStopContract` or `SetStopShare` functions determine whether single stop loss will be applied to the entire position or multiple SetStopLoss_pt orders will be applied to each entry in position individually; by default, stop loss is applied to the entire position.

SetStopLoss_pt function is evaluated intra-bar and not only on close of a bar, and can exit within the same bar as the entry.

Usage

`SetStopLoss_pt(`*`Amount`*`)`

Where: *`Amount`* - a numerical expression, specifying the stop loss amount in ticks.

Notes

This function can only be used in signals.

SetStopLoss_pt function does not factor in commissions or slippage.

Example

Generate an exit order for the entire position if the position loss reaches 100 ticks:

```
SetStopPosition;

SetStopLoss_pt(100);
```

Generate an exit order for the entry if the loss per entry reaches 10 ticks:

```
SetStopContract;
```

```
SetStopLoss_pt(10);
```

SetStopPosition

Forces the built-in strategy exit functions to be applied on the entire position basis.

The built-in strategy exit functions are: `SetStopLoss` , `SetProfitTarget` , `SetBreakEven` , `SetDollarTrailing` , and `SetPercentTrailing`.

Usage

```
SetStopPosition
```

Notes

If `SetStopPositon`, `SetStopContract`, and `SetStopShare` were not used, the exit functions will be applied on the entire position basis as a default.

If `SetStopPositon`, `SetStopContract`, and `SetStopShare` were used in multiple instances or in different signals applied to the same chart, the last instance will be controlling.

Example

Force `SetStopLoss` strategy exit function to be applied on the entire position basis:

```
SetStopPosition;

SetStopLoss(100);
```

An exit order for the entire position will be generated if the position loss reaches ¤100.

SetStopShare

Same as `SetStopContract`

SetTrailingStop_pt

Closes out the entire position or the entry if the current profit is less than the maximum profit by the specified amount; generates the appropriate Stop order depending on whether the position is long or short.

For example, if the specified ammount is 50 ticks and the profit has reached the maximum of 120 ticks, the position will be closed once the profit drops to 70 ticks.

`SetStopPosition` and `SetStopContract` or `SetStopShare` functions determine whether SetTrailingStop_pt will be applied to the entire position or to each entry in position individually; by default, SetTrailingStop_pt is applied to the entire position.

SetTrailingStop_pt function is evaluated intra-bar and not only on close of a bar, and can exit within the same bar as the entry.

Usage

`SetDollarTrailing_pt(`*`Amount`*`)`

Where: *`Amount`* - a numerical expression, specifying the tick value of the maximum loss of profit in ticks.

Notes

This function can only be used in signals.

SetTrailingStop_pt function does not factor in commissions or slippage.

Example

Generate an exit order for the entire position if position profit drops by 50 ticks:

```
SetStopPosition;
```

```
SetDollarTrailing_pt(50);
```

Generate an exit order for the entry if per entry profit drops by 10 ticks:

```
SetStopContract;

SetDollarTrailing_pt(10);
```

Share

Same as Shares

Shares

Used in strategy entry or exit statements in combination with a numerical expression to specify the number of shares or contracts to trade.

Usage

```
TradeSize Shares
```

Where: *`TradeSize`* - a numerical expression, specifying the number of shares or contracts

Example

Buy 2 shares at Market price on open of next bar:

```
Buy 2 Shares Next Bar At Market;
```

Short

Used in combination with the word `Sell`; same as `SellShort`.

Usage

`Sell Short`[("*`EntryLabel`*")][*TradeSize*]*`Entry`*;

Stop

Used in strategy entry or exit statements to specify a Stop price for an entry or an exit.

A Stop order will execute at the specified price or worse. A worse price is a higher price for Buy and Buy to cover orders, and a lower price for Sell and Sell short orders.

Usage

```
At Price Stop
```

Where: `Price` - a numerical expression, specifying the Limit Price
`At` - a skip word and can be omitted

Example

Sell within the next bar on the first tick with a price of 100 or less:

```
Sell Next Bar 100 Stop;
```

Cover within the next bar on the first tick with a price of 50 or more:

```
BuyToCover Next Bar At 50 Stop;
```

Total

Used in strategy exit statements, following a numerical expression and the words `Shares` or `Contracts`, to indicate that only the number of contracts or shares specified by the numerical expression is to be sold or covered in total, regardless of the number of open entries. The contracts or shares will be sold or covered in the same order they were bought or shorted: First In, First Out.

If the word `Total` is not used, the number of contracts or shares specified by the numerical expression will be sold or covered for **each one** of the open entries.

Usage

```
TradeSize Shares Total
```

Where: *`TradeSize`* - a numerical expression, specifying the number of shares or contracts

Example

Sell a total of 2 contracts, regardless of the number of open long entries, at Market price on open of next bar:

```
Sell 2 Contracts Total Next Bar At Market;
```

AvgBarsEvenTrade

Returns a numerical value, indicating the average number of bars a position was held during all completed even trades.

Usage

```
AvgBarsEvenTrade
```

Notes

This function can only be used in signals.

Example

`AvgBarsEvenTrade` will return a value of 3.5 if four even trades held positions for 2, 5, 3, and 4 bars

AvgBarsLosTrade

Returns a numerical value, indicating the average number of bars a position was held during all completed losing trades.

Usage

```
AvgBarsLosTrade
```

Notes

This function can only be used in signals.

Example

`AvgBarsLosTrade` will return a value of 3.5 if four losing trades held positions for 2, 5, 3, and 4 bars

AvgBarsWinTrade

Returns a numerical value, indicating the average number of bars a position was held during all completed winning trades.

Usage

```
AvgBarsWinTrade
```

Notes

This function can only be used in signals.

Example

`AvgBarsWinTrade` will return a value of 3.5 if four winning trades held positions for 2, 5, 3, and 4 bars

AvgEntryPrice

Returns a numerical value, indicating the average entry price for all open entries in a pyramided position.

Usage

```
AvgEntryPrice
```

Notes

This function can only be used in signals.

Example

`AvgEntryPrice` will return a value of 101 if there were three open entries at 95, 105, and 103

AvgEntryPrice_at_Broker

Returns a numerical value, indicating the average entry price at the broker for the symbol.

A positive value indicates a long position and a negative value indicates a short position.

A zero ('0') is returned when the current position is flat, or if Automated Trading is not turned on.

Usage

```
AvgEntryPrice_at_Broker
```

Notes

This function can only be used in signals and functions.

Important

If Automated Trading was manually turned off by the user, the value returned by the keyword stops changing, and may remain unequal to '0'.

Example

`AvgEntryPrice_at_Broker` will return a value of 102 if broker has returned the value of 102 for the current trading instrument.

AvgEntryPrice_at_Broker_for_The_Str

Returns a numerical value, indicating the average entry price at the broker for the strategy.

A positive value indicates a long position and a negative value indicates a short position.

A zero ('0') is returned when the current position is flat, or if Automated Trading is not turned on.

Usage

`AvgEntryPrice_at_Broker_for_The_Strategy`

Notes

This function can only be used in signals and functions.

Important

If Automated Trading was manually turned off by the user, the value returned by the keyword stops changing, and may remain unequal to '0'.

Example

`AvgEntryPrice_at_Broker_for_The_Strategy` will return a value of 100 if there were two 1 contract entries for the current strategy: at 98 and 102.

GrossLoss

Returns a negative numerical value, indicating the total currency value of all completed losing trades.

Usage

```
GrossLoss
```

Notes

This function can only be used in signals.

Example

`GrossLoss` will return a value of -50 if there were a total of four losing trades, at 10, 5, 20, and 15

`GrossLoss` will return a value of 0 if no losing trades were completed during the entire trading period

GrossProfit

Returns a numerical value, indicating the total currency value of all completed winning trades.

Usage

```
GrossProfit
```

Note

This function can only be used in signals.

Example

`GrossProfit` will return a value of 50 if there were a total of four winning trades, at 10, 5, 20, and 15

`GrossProfit` will return a value of 0 if no winning trades were completed during the entire trading period

i_AvgEntryPrice

Same as `AvgEntryPrice` except used in indicators.

i_AvgEntryPrice_at_Broker

Returns the Average entry price of each open entry in a pyramided position.

Usage

```
i_AvgEntryPrice_at_Broker
```

Notes

`i_AvgEntryPrice_at_Broker` only returns the average entry price for open trades.

`i_AvgEntryPrice_at_Broker` can only be used in an indicator.

`i_AvgEntryPrice_at_Broker` will only return a value if a signal is applied to the same data.

Example

`i_AvgEntryPrice_at_Broker` returns 170 if three trades are currently open and were entered at a price of 140, 170, and 200.

`i_AvgEntryPrice_at_Broker` returns 53 if four trades are currently open and were entered at a price of 54, 48, 60, and 50.

i_AvgEntryPrice_at_Broker_for_The_S

Is used for the extraction of strategy information in indicator.

Returns the same information as `AvgEntryPrice at Broker for The Strategy`.

i_ClosedEquity

Returns the profit or loss realized when a position has been closed.

Usage

```
i_ClosedEquity
```

Notes

This function can only be used in indicators.

Example

`i_ClosedEquity` will return 100 if the closed equity is 100.

i_CurrentContracts

Same as `CurrentContracts` except used in indicators.

i_CurrentShares

Same as `CurrentShares` except used in indicators.

i_MarketPosition

Returns a numerical value, indicating the type of the specified position.

A value of 1 indicates the current bar has a long position, -1 indicates the current bar has a short position, and 0 is returned only if the current bar has a flat position.

Usage

```
i_MarketPosition
```

Notes

This function can only be used in indicators.

Example

`i_MarketPosition` will return a value of 0 if the position on the current bar is flat

`i_MarketPosition` will return a value of 1 if the position on the current bar is long

`i_MarketPosition` will return a value of -1 if the position on the current bar is short

i_OpenEquity

Returns the current equity (`netprofit` + `openpositionprofit`)

Usage

```
i_OpenEquity
```

Notes

This function can only be used in indicators.

Example

`i_OpenEquity` will return 100 if the current equity is 100.

`i_OpenEquity` will return -100 if the current equity is -100.

LargestLosTrade

Returns a negative numerical value, indicating the currency value of the largest completed losing trade.

Usage

```
LargestLosTrade
```

Notes

This function can only be used in signals.

Example

`LargestLosTrade` will return a value of -20 if there were a total of four losing trades at 10, 5, 20, and 15

`LargestLosTrade` will return a value of 0 if no losing trades were completed during the entire trading period

LargestWinTrade

Returns a numerical value, indicating the currency value of the largest completed winning trade.

Usage

```
LargestWinTrade
```

Notes

This function can only be used in signals.

Example

`LargestWinTrade` will return a value of 20 if there were a total of four winning trades at 10, 5, 20, and 15

`LargestWinTrade` will return a value of 0 if no winning trades were completed during the entire trading period

MaxConsecLosers

Returns a numerical value, indicating the number of trades in the longest sequence of consecutive completed losing trades.

Usage

```
MaxConsecLosers
```

Notes

This function can only be used in signals.

Example

`MaxConsecLosers` will return a value of 3 if there were three consecutive completed losing trades

`MaxConsecLosers` will return a value of 0 if no losing trades were completed during the entire trading period

MaxConsecWinners

Returns a numerical value, indicating the number of trades in the longest sequence of consecutive completed winning trades.

Usage

```
MaxConsecWinners
```

Notes

This function can only be used in signals.

Example

`MaxConsecWinners` will return a value of 3 if there were three consecutive completed winning trades

`MaxConsecWinners` will return a value of 0 if no winning trades were completed during the entire trading period

MaxContractsHeld

Returns a numerical value, indicating the maximum number of contracts or shares held at any one time.

Usage

`MaxContractsHeld`

Notes

This function can only be used in signals.

Example

`MaxContractsHeld` will return a value of 10 if a maximum of ten contracts were held at any one time

MaxIDDrawDown

Returns a negative numerical value, indicating the largest decline in equity during the entire trading period.

Usage

```
MaxIDDrawDown
```

Notes

This function can only be used in signals.

Example

`MaxIDDrawDown` will return a value of -500 if the largest decline in equity during the entire trading period was ¤500

MaxSharesHeld

Same as MaxContractsHeld

NetProfit

Returns a numerical value, indicating the total currency value of all completed trades.

Usage

```
NetProfit
```

Notes

This function can only be used in signals.

Example

`NetProfit` will return a value of 20 if there were winning trades at 25 and 10, and losing trades at 5 and 10

`NetProfit` will return a value of -15 if there were winning trades at 10 and 5, and losing trades at 20 and 10

`NetProfit` will return a value of 0 no trades were completed during the entire trading period

NumEvenTrades

Returns a numerical value, indicating the total number of all completed even trades.

Usage

```
NumEvenTrades
```

Notes

This function can only be used in signals.

Example

`NumEvenTrades` will return a value of 10 if there were ten completed even trades

`NumEvenTrades` will return a value of 0 if no even trades were completed during the entire trading period

NumLosTrades

Returns a numerical value, indicating the number of all completed losing trades.

Usage

```
NumLosTrades
```

Notes

This function can only be used in signals.

Example

`NumLosTrades` will return a value of 5 if there were a total of five completed losing trades

`NumLosTrades` will return a value of 0 if no losing trades were completed during the entire trading period

NumWinTrades

Returns a numerical value, indicating the number of all completed winning trades.

Usage

```
NumWinTrades
```

Notes

This function can only be used in signals.

Example

`NumWinTrades` will return a value of 5 if there were a total of five completed winning trades

`NumWinTrades` will return a value of 0 if no winning trades were completed during the entire trading period

PercentProfit

Returns a numerical value, indicating the percentage of winning trades in all trades completed.

Usage

```
PercentProfit
```

Notes

This function can only be used in signals.

Example

`PercentProfit` will return a value of 70 if seven out of the total of 10 completed trades were winning trades

SetCustomFitnessValue

Sets a value of the custom criterion that is to be used for optimization.

Usage

`SetCustomFitnessValue`(*Criterion*)

Where: *`Criterion`* - an expression specifying a custom criterion value.

Note

- This function can be used only in signals
- To use the custom fitness value:
  1. Open genetic algorithm properties window;
  2. Set the number of simulations by changing the inputs range;
  3. Select the **Algorithm-Specified Properties** tab;
  4. Select the **Custom Fitness Value** from the list under the **Standard Criteria** section.

Example

Set the gross profit to be a custom criterion for genetic optimization

```
SetCustomFitnessValue (GrossProfit);
```

Set the formula to be a custom criterion for genetic optimization

```
SetCustomFitnessValue (TotalTrades / (GrossLoss + GrossProfit));
```

TotalBarsEvenTrades

Returns a numerical value, indicating the total number of bars a position was held during all completed even trades.

Usage

```
TotalBarsEvenTrades
```

Notes

This function can only be used in signals.

Example

`TotalBarsEvenTrades` will return a value of 14 if four even trades held positions for 2, 5, 3, and 4 bars

`TotalBarsEvenTrades` will return a value of 0 if no even trades were completed during the entire trading period

TotalBarsLosTrades

Returns a numerical value, indicating the total number of bars a position was held during all completed losing trades.

Usage

```
TotalBarsLosTrades
```

Notes

This function can only be used in signals.

Example

`TotalBarsLosTrades` will return a value of 14 if four losing trades held positions for 2, 5, 3, and 4 bars

`TotalBarsLosTrades` will return a value of 0 if no losing trades were completed during the entire trading period

TotalBarsWinTrades

Returns a numerical value, indicating the total number of bars a position was held during all completed winning trades.

Usage

```
TotalBarsWinTrades
```

Notes

This function can only be used in signals.

Example

`TotalBarsWinTrades` will return a value of 14 if four winning trades held positions for 2, 5, 3, and 4 bars

`TotalBarsWinTrades` will return a value of 0 if no winning trades were completed during the entire trading period

TotalTrades

Returns a numerical value, indicating the total number of all completed trades.

Usage

```
TotalTrades
```

Notes

This function can only be used in signals.

Example

`TotalTrades` will return a value of 5 if there were a total of five completed trades

`TotalTrades` will return a value of 0 if no trades were completed during the entire trading period

BarsSinceEntry

Returns a numerical value, indicating the number of bars since the initial entry into the specified position.

Usage

`BarsSinceEntry`(*PosBack*)

Where: *PosBack* - a numerical expression, specifying the position:

0 - open position;
1 - one position back (the last position closed);
2 - two positions back, etc.

If *`PosBack`* is not specified, a value for the open position will be returned.

Notes

This function can only be used in signals.

Example

Assign a value, indicating the number of bars since the current position has been entered, to Value1 variable:

```
Value1=BarsSinceEntry;
```

Assign a value, indicating the number of bars since the most recently closed position has been entered, to Value1 variable:

```
Value1=BarsSinceEntry(1);
```

BarsSinceEntry_Checked

Returns a numerical value, indicating the number of bars since the initial entry into the specified position.

Usage

`BarsSinceEntry_Checked`(*PosBack*)

Where: *PosBack* - a numerical expression, specifying the position:

0 - open position;
1 - one position back (the last position closed);
2 - two positions back, etc.

If *`PosBack`* is not specified, a value for the open position will be returned.

Notes

1. This function can only be used in signals.
2. If *`PosBack`* value is greater that the real number of previously opened positions, `BarsSinceEntry_Checked` will generate an error.

Example

Assign a value, indicating the number of bars since the current position has been entered, to Value1 variable:

```
Value1 = BarsSinceEntry_Checked;
```

Assign a value, indicating the number of bars since the most recently closed position has been entered, to Value1 variable:

```
Value1 = BarsSinceEntry_Checked (1);
```

BarsSinceExit

Returns a numerical value, indicating the number of bars since a complete exit from the specified position.

Usage

`BarsSinceExit`(*PosBack*)

Where: *PosBack* - a numerical expression, specifying the position:

> 1 - the last position closed (one position back);
> 2 - two positions back, etc.

Notes

This function can only be used in signals.

Example

Assign a value, indicating the number of bars since the last position has been closed, to Value1 variable:

```
Value1=BarsSinceExit(1);
```

BarsSinceExit_Checked

Returns a numerical value, indicating the number of bars since a complete exit from the specified position.

Usage

`BarsSinceExit_Checked`(*PosBack*)

Where: *PosBack* - a numerical expression, specifying the position:

0 - open position;
1 - one position back (the last position closed);
2 - two positions back, etc.

If *`PosBack`* is not specified, a value for the open position will be returned.

Notes

1. This function can only be used in signals.
2. If *`PosBack`* value is greater that the real number of previously opened positions, `BarsSinceExit_Checked` will generate an error.

Example

Assign a value, indicating the number of bars since the last position has been closed, to Value1 variable:

```
Value1 = BarsSinceExit_Checked (1);
```

ContractProfit

Returns a numerical value, indicating the current profit or loss per each contract or share of a multi-share or multi-contract position.

Usage

```
ContractProfit
```

Notes

This function can only be used in signals.

Example

Assign a value, indicating the current profit or loss per contract or share, to Value1 variable:

```
Value1=ContractProfit;
```

CurrentContracts

Returns an absolute numerical value, indicating the number of contracts or shares held in the current position.

Usage

```
CurrentContracts
```

Notes

This function can only be used in signals.

Example

`CurrentContracts` will return a value of 1 for 1 contract long

`CurrentContracts` will return a value of 5 for 5 shares short

CurrentEntries

Returns a numerical value, indicating the number of open entries for the current position.

Usage

```
CurrentEntries
```

Notes

This function can only be used in signals.

Example

Assign a value, indicating the number of open entries in the current position, to Value1 variable:

```
Value1=CurrentEntries;
```

CurrentShares

Same as `CurrentContracts`

EntryDate

Returns a numerical value, indicating the date of initial entry into the specified position. The date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month.

Usage

`EntryDate(`*PosBack*`)`

Where: *PosBack* - a numerical expression, specifying the position:

0 - open position;
1 - one position back (the last position closed);
2 - two positions back, etc.

If *`PosBack`* is not specified, a value for the open position will be returned.

Notes

This function can only be used in signals.

Example

Assign a value, indicating the date that the current position has been entered, to Value1 variable:

```
Value1=EntryDate;
```

Value 1 will equal to 1081030 for the entry date of October 30th, 2008

Assign a value, indicating the date that the most recently closed position has been entered, to Value1 variable:

```
Value1=EntryDate(1);
```

Value 1 will equal to 990402 for the entry date of April 2nd, 1999

EntryDateTime

Returns a double-precision decimal DateTime value indicating the date and time of the bar where the order that opened a specified position was generated. The integer portion of the DateTime value indicates the number of days that have elapsed since January 1st, 1900, and the fractional portion of the DateTime value indicates the fraction of the day that has passed since midnight. DateTime is a floating point value with high precision. It allows accessing millisecond time stamps of the bar.

Usage

`EntryDateTime`(*PosBack*)

Where: *PosBack* - a numerical expression, specifying the position:

0 - open position;
1 - one position back (the last position closed);
2 - two positions back, etc.

If *`PosBack`* is not specified, a value for the open position will be returned.

Notes

This function can only be used in signals.

Example

Assign a value, indicating the date that the current position has been entered, to Value1 variable:

```
Value1 = EntryDateTime;
```

EntryDateTime_Checked

Returns a double-precision decimal DateTime value indicating the date and time of the bar where the order that opened a specified position was generated. The integer portion of the DateTime value indicates the number of days that have elapsed since January 1st, 1900, and the fractional portion of the DateTime value indicates the fraction of the day that has passed since midnight. DateTime is a floating point value with high precision. It allows accessing millisecond time stamps of the bar.

Usage

`EntryDateTime_Checked`(*PosBack*)

Where: *PosBack* - a numerical expression, specifying the position:

0 - open position;
1 - one position back (the last position closed);
2 - two positions back, etc.

If *`PosBack`* is not specified, a value for the open position will be returned.

Notes

This function can only be used in signals.

If PosBack value is greater that the real number of previously opened positions, EntryDateTime_Checked will generate an error.

Example

Assign a value, indicating the date that the current position has been entered, to Value1 variable:

```
Value1 = EntryDateTime_Checked;
```

EntryDate_Checked

Returns a numerical value, indicating the date of initial entry into the specified position.

The date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month.

Usage

`EntryDate_Checked`(*PosBack*)

Where: *PosBack* - a numerical expression, specifying the position:

0 - open position;
1 - one position back (the last position closed);
2 - two positions back, etc.

If *`PosBack`* is not specified, a value for the open position will be returned.

Notes

1. This function can only be used in signals.
2. If *`PosBack`* value is greater that the real number of previously opened positions, `EntryDate_Checked` will generate an error.

Example

Assign a value, indicating the date that the current position has been entered, to Value1 variable:

`Value1` = `EntryDate_Checked`;

Value 1 will equal to 1081030 for the entry date of October 30th, 2008

Assign a value, indicating the date that the most recently closed position has been entered, to Value1 variable:

```
Value1 = EntryDate_Checked (1);
```

Value 1 will equal to 990402 for the entry date of April 2nd, 1999

EntryName

Returns the name of the order which opened the position.

Usage

`EntryName(`*`PosBack`*`)`

Where: *`PosBack`* - a numerical expression, specifying the position:

0 - open position;

1 - one position back (the last position closed);

2 - two positions back, etc.

Notes

If *`PosBack`* is not specified, a value for the open position will be returned.

Example

`EntryName(1)` will return a value of "buy LE" for the last closed position, if this position was opened by the order with "buy LE" name.

EntryPrice

Returns a numerical value, indicating the price at the initial entry into the specified position.

Usage

`EntryPrice(`*PosBack*`)`

Where: *PosBack* - a numerical expression, specifying the position:

0 - open position;
1 - one position back (the last position closed);
2 - two positions back, etc.

If *`PosBack`* is not specified, a value for the open position will be returned.

Notes

This function can only be used in signals.

Example

Assign a value, indicating the entry price for the current position, to Value1 variable:

```
Value1=EntryPrice;
```

Assign a value, indicating the entry price for the most recently closed position, to Value1 variable:

```
Value1=EntryPrice(1);
```

EntryPrice_Checked

Returns a numerical value, indicating the price at the initial entry into the specified position.

Usage

`EntryPrice_Checked`(*PosBack*)

Where: *PosBack* - a numerical expression, specifying the position:

0 - open position;
1 - one position back (the last position closed);
2 - two positions back, etc.

If *`PosBack`* is not specified, a value for the open position will be returned.

Notes

1. This function can only be used in signals.
2. If *`PosBack`* value is greater that the real number of previously opened positions, `EntryPrice_Checked` will generate an error.

Example

Assign a value, indicating the entry price for the current position, to Value1 variable:

`Value1` = `EntryPrice_Checked`;

Assign a value, indicating the entry price for the most recently closed position, to Value1 variable:

`Value1` = `EntryPrice_Checked` (`1`);

EntryTime

Returns a numerical value, indicating the time of initial entry into the specified position. The time is indicated in the 24-hour HHmm format, where 1300 = 1:00 PM.

Usage

`EntryTime`(*PosBack*)

Where: *PosBack* - a numerical expression, specifying the position:

0 - open position;
1 - one position back (the last position closed);
2 - two positions back, etc.

If *`PosBack`* is not specified, a value for the open position will be returned.

Notes

This function can only be used in signals.

Example

Assign a value, indicating the time that the current position has been entered, to Value1 variable:

```
Value1=EntryTime;
```

Value 1 will equal to 1015 for 10:15 AM

Assign a value, indicating the time that the most recently closed position has been entered, to Value1 variable:

```
Value1=EntryTime(1);
```

Value 1 will equal to 1545 for 3:45 PM

EntryTime_Checked

Returns a numerical value, indicating the time of initial entry into the specified position.

The time is indicated in the 24-hour HHmm format, where 1300 = 1:00 PM.

Usage

`EntryTime_Checked`(*PosBack*)

Where: *PosBack* - a numerical expression, specifying the position:

> 0 - open position;
> 1 - one position back (the last position closed);
> 2 - two positions back, etc.

If *`PosBack`* is not specified, a value for the open position will be returned.

Notes

1. This function can only be used in signals.
2. If *`PosBack`* value is greater that the real number of previously opened positions, `EntryTime_Checked` will generate an error.

Example

Assign a value, indicating the time that the current position has been entered, to Value1 variable:

`Value1` = `EntryTime_Checked`;

Value 1 will equal to 1015 for 10:15 AM.

Assign a value, indicating the time that the most recently closed position has been entered, to Value1 variable:

```
Value1 = EntryTime_Checked (1);
```

Value 1 will equal to 1015 for 10:15 AM.

ExitDate

Returns a numerical value, indicating the date of the complete exit from the specified position. The date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month.

Usage

`ExitDate`(*PosBack*)

Where: *PosBack* - a numerical expression, specifying the position:

1 - the last position closed (one position back);
2 - two positions back, etc.

Notes

This function can only be used in signals.

Example

Assign a value, indicating the date that the most recently closed position has been exited, to Value1 variable:

```
Value1=ExitDate(1);
```

Value 1 will equal to 1081030 for the exit date of October 30th, 2008

ExitDateTime

Returns a double-precision decimal DateTime value indicating the date and time of the bar where the order that closed a specified position was generated. The integer portion of the DateTime value indicates the number of days that have elapsed since January 1st, 1900, and the fractional portion of the DateTime value indicates the fraction of the day that has passed since midnight. DateTime is a floating point value with high precision. It allows accessing millisecond time stamps of the bar.

Usage

`ExitDateTime`(*PosBack*)

Where: *PosBack* - a numerical expression, specifying the position:

    1 - the last position closed (one position back);
    2 - two positions back, etc.

Notes

This function can only be used in signals.

Example

Assign a value, indicating the date that the most recently closed position has been exited, to Value1 variable:

```
Value1 = ExitDateTime(1);
```

ExitDateTime_Checked

Returns a double-precision decimal DateTime value indicating the date and time of the bar where the order that closed a specified position was generated. The integer portion of the DateTime value indicates the number of days that have elapsed since January 1st, 1900, and the fractional portion of the DateTime value indicates the fraction of the day that has passed since midnight. DateTime is a floating point value with high precision. It allows accessing millisecond time stamps of the bar.

Usage

`ExitDateTime_Checked`(*PosBack*)

Where: *PosBack* - a numerical expression, specifying the position:

1 - the last position closed (one position back);
2 - two positions back, etc.

Notes

This function can only be used in signals.

If PosBack value is greater that the real number of previously opened positions, ExitDateTime_Checked will generate an error.

Example

Assign a value, indicating the date that the most recently closed position has been exited, to Value1 variable:

```
Value1 = ExitDateTime_Checked(1);
```

ExitDate_Checked

Returns a numerical value, indicating the date of the complete exit from the specified position. The date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month.

Usage

`ExitDate_Checked`(*PosBack*)

Where: *PosBack* - a numerical expression, specifying the position:

1 - the last position closed (one position back);
2 - two positions back, etc.

Notes

1. This function can only be used in signals.
2. If *`PosBack`* value is greater that the real number of previously opened positions, `ExitDate_Checked` will generate an error.

Example

Assign a value, indicating the date that the most recently closed position has been exited, to Value1 variable:

`Value1` = `ExitDate_Checked` (`1`);

Value 1 will equal to 1081030 for the exit date of October 30th, 2008.

ExitName

Returns the name of the order which closed the position.

Usage

`ExitName(`*`PosBack`*`)`

Where: *`PosBack`* - a numerical expression, specifying the position:

0 - open position;

1 - one position back (the last position closed);

2 - two positions back, etc.

Notes

If *`PosBack`* is not specified, a value for the open position will be returned.

Example

`ExitName(1)` will return a value of "sell LX" for the last closed position, if this position was closed by the order with "sell LX" name.

ExitPrice

Returns a numerical value, indicating the price at a complete exit from the specified position.

Usage

`ExitPrice`(*PosBack*)

Where: *PosBack* - a numerical expression, specifying the position:

1 - the last position closed (one position back);
2 - two positions back, etc.

Notes

This function can only be used in signals.

Example

Assign a value, indicating the exit price of the most recently closed position, to Value1 variable:

```
Value1=ExitPrice(1);
```

ExitPrice_Checked

Returns a numerical value, indicating the price at a complete exit from the specified position.

Usage

`ExitPrice_Checked`(*PosBack*)

Where: *PosBack* - a numerical expression, specifying the position:

1 - one position back (the last position closed);
2 - two positions back, etc.

Notes

1. This function can only be used in signals.
2. If *`PosBack`* value is greater that the real number of previously opened positions, `ExitPrice_Checked` will generate an error.

Example

Assign a value, indicating the exit price of the most recently closed position, to Value1 variable:

```
Value1 = ExitPrice_Checked (1);
```

ExitTime

Returns a numerical value, indicating the time at the complete exit from the specified position. The time is indicated in the 24-hour HHmm format, where 1300 = 1:00 PM.

Usage

`ExitTime`(*PosBack*)

Where: *PosBack* - a numerical expression, specifying the position:

1 - the last position closed (one position back);
2 - two positions back, etc.

Notes

This function can only be used in signals.

Example

Assign a value, indicating the time that the most recently closed position has been exited, to Value1 variable:

```
Value1=ExitTime(1);
```

Value 1 will equal to 1545 for 3:45 PM

ExitTime_Checked

Returns a numerical value, indicating the time at the complete exit from the specified position. The time is indicated in the 24-hour HHmm format, where 1300 = 1:00 PM.

Usage

`ExitTime_Checked`(*PosBack*)

Where: *PosBack* - a numerical expression, specifying the position:

1 - one position back (the last position closed);
2 - two positions back, etc.

Notes

1. This function can only be used in signals.
2. If *`PosBack`* value is greater that the real number of previously opened positions, `ExitTime_Checked` will generate an error.

Example

Assign a value, indicating the time that the most recently closed position has been exited, to Value1 variable:

`Value1` = `ExitTime_Checked(1)`;

Value 1 will equal to 1545 for 3:45 PM.

i_MarketPosition_at_Broker

Returns a numerical value, indicating the number of contracts and the type of position at the broker for the symbol.

A positive value indicates a long position and a negative value indicates a short position.

A zero ('0') is returned when the current position is flat, or if Automated Trading is not turned on.

Usage

```
i_MarketPosition_at_Broker
```

Notes

This function can only be used in indicators.

This function differs from the marketposition keyword in that it cannot take an argument to reference past values. By saving the value of i_MarketPosition_at_Broker to a Numeric Variable, it's possible to reference the position at the broker for previous bars or tick.

I_MarketPosition_at_Broker works with all brokers, though there are some peculiarities with MIG Bank and Trading Technologies: For MIG Bank, MultiCharts calculates the broker position after the broker profile has been connected since this isn't provided by MIG Bank. The Trading Technologies API doesn't provide information about positions opened the day before, in which case MultiCharts uses the average of all complete long (short) trades for the day to calculate the EntryPrice.

Important

If Automated Trading was manually turned off by the user, the value returned by the keyword stops changing, and may remain unequal to '0'.

Example

`i_MarketPosition_at_Broker` will will return '17' if the current position at the broker for the strategy is 17 contracts long.

`i_MarketPosition_at_Broker` will return '-132' if the current position at the broker for the strategy is 132 contracts short.

`i_MarketPosition_at_Broker` will return '0' if the current position at the broker for the strategy is flat.

`i_MarketPosition_at_Broker` will return '0' if the Automated Trading Mode is not turned on.

i_MarketPosition_at_Broker_for_The_

Returns a numerical value, indicating the number of contracts and the type of position at the broker for the strategy.

A positive value indicates a long position and a negative value indicates a short position.

A zero ('0') is returned when the current position is flat, or if Automated Trading is not turned on.

Usage

```
i_MarketPosition_at_Broker_for_The_Strategy
```

Notes

This function can only be used in indicators.

This function differs from the marketposition keyword in that it cannot take an argument to reference past values. By saving the value of i_MarketPosition_at_Broker_for_The_Strategy to a Numeric Variable, it's possible to reference the position at the broker for previous bars or tick.

Important

If Automated Trading was manually turned off by the user, the value returned by the keyword stops changing, and may remain unequal to '0'.

Example

`i_MarketPosition_at_Broker_for_The_Strategy` will will return '17' if the current position at the broker for the strategy is 17 contracts long.

`i_MarketPosition_at_Broker_for_The_Strategy` will return '-132' if the current position at the broker for the strategy is 132 contracts short.

`i_MarketPosition_at_Broker_for_The_Strategy` will return '0' if the current position at the broker for the strategy is flat.

`i_MarketPosition_at_Broker_for_The_Strategy` will return '0' if the Automated Trading Mode is not turned on.

MarketPosition

Returns a numerical value, indicating the type of the specified position.

A value of 1 indicates a long position, -1 indicates a short position, and 0 is returned only if the current position is specified and indicates that the current position is flat.

Usage

`MarketPosition(`*PosBack*`)`

Where: *PosBack* - a numerical expression, specifying the position:

0 - open position;
1 - one position back (the last position closed);
2 - two positions back, etc.

If *`PosBack`* is not specified, a value for the open position will be returned.

Notes

This function can only be used in signals.

Example

`MarketPosition` will return a value of 0 if the current position is flat

`MarketPosition(1)` will return a value of -1 if the most recently closed position was a short position

MarketPosition_at_Broker

Returns a numerical value, indicating the number of contracts and the type of position at the broker for the symbol.

A positive value indicates a long position and a negative value indicates a short position.

A zero ('0') is returned when the current position is flat, or if Automated Trading is not turned on.

Usage

```
MarketPosition_at_Broker
```

Notes

This function can only be used in signals and functions.

This function differs from the marketposition keyword in that it cannot take an argument to reference past values.

This function can only be used with Interactive Brokers, Patsystems, and Zen-Fire.

Important

If Automated Trading was manually turned off by the user, the value returned by the keyword stops changing, and may remain unequal to '0'.

Example

`MarketPosition_at_Broker` will will return '17' if the current position at the broker for the strategy is 17 contracts long.

`MarketPosition_at_Broker` will return '-132' if the current position at the broker for the strategy is 132 contracts short.

`MarketPosition_at_Broker` will return '0' if the current position at the broker for the strategy is flat.

`MarketPosition_at_Broker` will return '0' if the Automated Trading Mode is not turned on.

MarketPosition_at_Broker_for_The_St

Returns a numerical value, indicating the number of contracts and the type of position at the broker for the strategy.

A positive value indicates a long position and a negative value indicates a short position.

A zero ('0') is returned when the current position is flat, or if Automated Trading is not turned on.

Usage

`MarketPosition_at_Broker_for_The_Strategy`

Notes

This function can only be used in signals and functions.

This function differs from the marketposition keyword in that it cannot take an argument to reference past values.

Important

If Automated Trading was manually turned off by the user, the value returned by the keyword stops changing, and may remain unequal to '0'.

Example

`MarketPosition_at_Broker_for_The_Strategy` will will return '17' if the current position at the broker for the strategy is 17 contracts long.

`MarketPosition_at_Broker_for_The_Strategy` will return '-132' if the current position at the broker for the strategy is 132 contracts short.

`MarketPosition_at_Broker_for_The_Strategy` will return '0' if the current position at the broker for the strategy is flat.

`MarketPosition_at_Broker_for_The_Strategy` will return '0' if the Automated Trading Mode is not turned on.

MarketPosition_Checked

Returns a numerical value, indicating the type of the specified position.

A value of 1 indicates a long position, -1 indicates a short position, and 0 is returned only if the current position is specified and indicates that the current position is flat.

Usage

`MarketPosition_Checked(`*PosBack*`)`

Where: *PosBack* - a numerical expression, specifying the position:

0 - open position;
1 - one position back (the last position closed);
2 - two positions back, etc.

If *`PosBack`* is not specified, a value for the open position will be returned.

Notes

1. This function can only be used in signals.
2. If *`PosBack`* value is greater that the real number of previously opened positions, `MarketPosition_Checked` will generate an error.

Example

`MarketPosition_Checked` - will return a value of 0 if the current position is flat.

`MarketPosition_Checked (1)` - will return a value of -1 if the most recently closed position was a short position.

MaxContractProfit

Returns a numerical value, indicating the largest gain reached per each contract or share of a current multi-share or multi-contract position.

Usage

```
MaxContractProfit
```

Notes

This function can only be used in signals.

Example

Assign a value, indicating the largest gain reached per contract or share, to Value1 variable:

```
Value1=MaxContractProfit;
```

MaxContractProfit_Checked

Returns a numerical value, indicating the largest gain reached per each contract or share of a current multi-share or multi-contract position.

Usage

```
MaxContractProfit_Checked
```

Notes

1. This function can only be used in signals.
2. If `PosBack` value is greater that the real number of previously opened positions, `MaxContractProfit_Checked` will generate an error.

Example

Assign a value, indicating the largest gain reached per contract or share, to Value1 variable:

```
Value1 = MaxContractProfit_Checked
```

MaxContracts

Returns an absolute numerical value, indicating the maximum number of contracts held during the specified position.

Usage

`MaxContracts`(*PosBack*)

Where: *PosBack* - a numerical expression, specifying the position:

> 0 - open position;
> 1 - one position back (the last position closed);
> 2 - two positions back, etc.

If *`PosBack`* is not specified, a value for the open position will be returned.

Notes

This function can only be used in signals.

Example

`MaxContracts` will return a value of 0 if the current position is flat

`MaxContracts (1)` will return a value of 10 if the most recently closed position was long or short a maximum of 10 contracts

MaxContracts_Checked

Returns an absolute numerical value, indicating the maximum number of contracts held during the specified position.

Usage

`MaxContracts_Checked`(*PosBack*)

Where: *PosBack* - a numerical expression, specifying the position:

0 - open position;
1 - one position back (the last position closed);
2 - two positions back, etc.

If *`PosBack`* is not specified, a value for the open position will be returned.

Notes

1. This function can only be used in signals.
2. If *`PosBack`* value is greater that the real number of previously opened positions, `MaxContracts_Checked` will generate an error.

Example

`MaxContracts_Checked` - will return a value of 0 if the current position is flat.

`MaxContracts_Checked (1)` - will return a value of 10 if the most recently closed position was long or short a maximum of 10 contracts.

MaxEntries

Returns a numerical value, indicating the total number of entries for the specified position.

Usage

`MaxEntries`(*PosBack*)

Where: *PosBack* - a numerical expression, specifying the position:

0 - open position;
1 - one position back (the last position closed);
2 - two positions back, etc.

If *`PosBack`* is not specified, a value for the open position will be returned.

Notes

This function can only be used in signals.

Example

`MaxEntries (1)` will return a value of 2 if there were two separate entries for the most recently closed position

MaxEntries_Checked

Returns a numerical value, indicating the total number of entries for the specified position.

Usage

`MaxEntries_Checked`(*PosBack*)

Where: *PosBack* - a numerical expression, specifying the position:

0 - open position;
1 - one position back (the last position closed);
2 - two positions back, etc.

If *`PosBack`* is not specified, a value for the open position will be returned.

Notes

1. This function can only be used in signals.
2. If *`PosBack`* value is greater that the real number of previously opened positions, `MaxEntries_Checked` will generate an error.

Example

`MaxEntries_Checked (1)` - will return a value of 2 if there were two separate entries for the most recently closed position.

MaxPositionLoss

Returns a negative numerical value, indicating the largest loss reached while the specified position was held.

Usage

`MaxPositionLoss`(*PosBack*)

Where: *PosBack* - a numerical expression, specifying the position:

0 - open position;
1 - one position back (the last position closed);
2 - two positions back, etc.

If *`PosBack`* is not specified, a value for the open position will be returned.

Notes

This function can only be used in signals.

Example

`MaxPositionLoss` will return a value of 0 if the value of the open position has not dropped below the entry price at any time while it was held

`MaxPositionLoss(1)` will return a value of -10 if the most recently closed position has dropped in value as much as ¤10 while it was held

MaxPositionLoss_Checked

Returns a negative numerical value, indicating the largest loss reached while the specified position was held.

Usage

`MaxPositionLoss_Checked`(*PosBack*)

Where: *PosBack* - a numerical expression, specifying the position:

0 - open position;
1 - one position back (the last position closed);
2 - two positions back, etc.

If *`PosBack`* is not specified, a value for the open position will be returned.

Notes

1. This function can only be used in signals.
2. If *`PosBack`* value is greater that the real number of previously opened positions, `MaxPositionLoss_Checked` will generate an error.

Example

`MaxPositionLoss_Checked`; - will return a value of 0 if the value of the open position has not dropped below the entry price at any time while it was held.

`MaxPositionLoss_Checked (1)`; - will return a value of -10 if the most recently closed position has dropped in value as much as ¤10 while it was held.

MaxPositionProfit

Returns a numerical value, indicating the largest gain reached while the specified position was held.

Usage

`MaxPositionProfit`(*PosBack*)

Where: *PosBack* - a numerical expression, specifying the position:

0 - open position;
1 - one position back (the last position closed);
2 - two positions back, etc.

If *`PosBack`* is not specified, a value for the open position will be returned.

Notes

This function can only be used in signals.

Example

`MaxPositionProfit` will return a value of 0 if the value of the open position has not increased at any time while it was held

`MaxPositionProfit (1)` will return a value of 10 if the most recently closed position has gained in value as much as ¤10 while it was held

MaxPositionProfit_Checked

Returns a numerical value, indicating the largest gain reached while the specified position was held.

Usage

`MaxPositionProfit_Checked`(*PosBack*)

Where: *PosBack* - a numerical expression, specifying the position:

0 - open position;
1 - one position back (the last position closed);
2 - two positions back, etc.

If *`PosBack`* is not specified, a value for the open position will be returned.

Notes

1. This function can only be used in signals.
2. If *`PosBack`* value is greater that the real number of previously opened positions, `MaxPositionProfit_Checked` will generate an error.

Example

`MaxPositionProfit_Checked`; - will return a value of 0 if the value of the open position has not increased at any time while it was held.

`MaxPositionProfit_Checked (1)`; - will return a value of 10 if the most recently closed position has gained in value as much as ¤10 while it was held.

MaxPositionsAgo

Returns the number of closed positions for the strategy on the current moment.

Usage

```
MaxPositionsAgo
```

Note

This function can only be used in Signals.

Example

Calculate the PnL for all closed positions up to the current moment:

```
var: TotalProfitLoss(0), idx(0);
for idx = 1 to MaxPositionsAgo Begin
TotalProfitLoss = PositionProfit(idx);
End;
```

MaxShares

Same as MaxContracts

MaxShares_Checked

Same as `MaxContracts_Checked`

OpenPositionProfit

Returns a numerical value, indicating the current unrealized profit or loss for the open position.

Usage

```
OpenPositionProfit
```

Notes

This function can only be used in signals.

Example

`OpenPositionProfit` will return a value of 0 if the current position is flat

`OpenPositionProfit` will return a value of 10 if the value of the open position has increased by ¤10 since it was entered

`OpenPositionProfit` will return a value of -5 if the value of the open position has decreased by ¤5 since it was entered

PositionProfit

Returns a numerical value, indicating the total realized profit or loss for the specified closed position.

Usage

`PositionProfit`(*PosBack*)

Where: *PosBack* - a numerical expression, specifying the closed position:

1 - one position back (the last position closed);
2 - two positions back, etc.

If *`PosBack`* is not specified, a value for the open position will be returned.

Notes

This function can only be used in signals.

Example

`PositionProfit (0)` will return a value of 5 if the value of the open position has increased by ¤5 since it was entered

`PositionProfit (1)` will return a value of -5 if the most recently closed position has generated a loss of ¤5

PositionProfit_Checked

Returns a numerical value, indicating the profit or loss for the specified position.

Usage

`PositionProfit_Checked`(*PosBack*)

Where: *PosBack* - a numerical expression, specifying the position:

0 - open position;
1 - one position back (the last position closed);
2 - two positions back, etc.

If *`PosBack`* is not specified, a value for the open position will be returned.

Notes

1. This function can only be used in signals.
2. If *`PosBack`* value is greater that the real number of previously opened positions, `PositionProfit_Checked` will generate an error.

Example

`PositionProfit_Checked`; - will return a value of 0 if the current position is flat.

`PositionProfit_Checked (0)`; - will return a value of 5 if the value of the open position has increased by ¤5 since it was entered.

`PositionProfit_Checked (1)`; - will return a value of -5 if the most recently closed position has generated a loss of ¤5.

ChangeMarketPosition

Places the order with set name and price on the chart.

Usage

`ChangeMarketPosition (Delta, Price, Name)`

Where: *Delta* - number of contracts by which current market position should be changed;
Price - order filling price;
Name - name of the order that changes the position.

Notes

Can be used as a mean of synchronization of strategy market position with a broker.

Example

```
If MarketPosition = 2 then ChangeMarketPosition (-2,100,"LX")
```

Will place close order with the name "LX" and the price 100 if current marketposition =2

```
If MarketPosition = 0 then ChangeMarketPosition (-2,100,"SE")
```

Will place open order with the name "SE" and the price 100 if current marketposition =0

Please refer to "!From Strategy To Broker MP Synchronizer!" default signal for more examples.

PlaceMarketOrder

Places market order at the broker without position changing on the chart.

Usage

`PlaceMarketOrder(IsBuy, IsEntry, Contracts)`

Where: *IsBuy* - indicates whether order is buy or sell;
IsEntry - indicates whether order is entry or exit;
Contracts - indicates the number of contracts/shares of the order.

Notes

Works only with auto trading turned on.

Can be used as a mean of synchronization of strategy market position with a broker.

Example

```
If MarketPosition*CurrentContracts = 2 and MarketPosition_at_Broker = 4
then PlaceMarketOrder (false, false, 2);
```

will generate sell market order for 2 contracts to synchronize market position at broker with the strategy position

Please refer to "!From Strategy To Broker MP Synchronizer!" default signal for more examples.

OpenEntriesCount

Same as `CurrentEntries`.

OpenEntryComission

Returns a numerical value, indicating the amount of cash assets in the units of the selected currency spent on the commission for specified trade.

Usage

`OpenEntryComission(`*`EntryIndex`*`)`

Where: *`EntryIndex`* - a numerical expression, specifying the number of trade (zero-based).

Notes

This function can only be used in signals.

To retrieve the total number of trades in open position use `OpenEntriesCount`

Example

`OpenEntryComission(1)` will return a value of 1 if there has been one unit of the selected currency comission for the second trade of the open position.

OpenEntryContracts

Returns a numerical value, indicating the quantity of contracts of specified entry order into the open position.

Usage

`OpenEntryContracts(`*`EntryIndex`*`)`

Where: *`EntryIndex`* - a numerical expression, specifying the number of trade (zero-based).

Notes

This function can only be used in signals.

To retrieve the total number of trades in open position use `OpenEntriesCount`

Example

`OpenEntryContracts` will return a value of 1 for the first trade of the 3 contracts open position if it has two open trades for 1 and 2 contracts.

`OpenEntryContracts(1)` will return a value of 2 for the second trade of the 3 contracts open position if it has two open trades for 1 and 2 contracts.

OpenEntryDate

Returns a numerical value, indicating the date of specified entry into the open position.

The date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month.

Usage

`OpenEntryDate(`*`EntryIndex`*`)`

Where: *`EntryIndex`* - a numerical expression, specifying the number of trade (zero-based).

Notes

This function can only be used in signals.

To retrieve the total number of trades in open position use `OpenEntriesCount`

Example

`OpenEntryDate(1)` will return 1110402 for the open position if the second trade was generated on April 2nd, 2011.

OpenEntryMaxProfit

Returns a numerical value, indicating maximal value of `OpenEntryProfit` for the time from entry order execution.

Usage

`OpenEntryMaxProfit(`*`EntryIndex`*`)`

Where: *`EntryIndex`* - a numerical expression, specifying the number of trade (zero-based).

Notes

This function can only be used in signals.

To retrieve the total number of trades in open position use `OpenEntriesCount`

Example

`OpenEntryMaxProfit` will return a value of 20 for the first trade of the open position if it has ever reached 20 units of the specified currency maximum profit.

OpenEntryMaxProfitPerContract

Returns a numerical value, indicating maximal value of `OpenEntryProfitPerContract` for the time from entry order execution.

Usage

`OpenEntryMaxProfitPerContract(`*`EntryIndex`*`)`

Where: *`EntryIndex`* - a numerical expression, specifying the number of trade (zero-based).

Notes

This function can only be used in signals.

To retrieve the total number of trades in open position use `OpenEntriesCount`

Example

`OpenEntryMaxProfitPerContract(1)` will return a value of 1.5 for the first trade of the open position if it has ever reached 1.5 units of the specified currency per contract profit.

OpenEntryMinProfit

Returns a numerical value, indicating minimal value of `OpenEntryProfit` for the time from entry order execution.

Usage

`OpenEntryMinProfit(`*`EntryIndex`*`)`

Where: *`EntryIndex`* - a numerical expression, specifying the number of trade (zero-based).

Notes

This function can only be used in signals.

To retrieve the total number of trades in open position use `OpenEntriesCount`

Example

`OpenEntryMinProfit(1)` will return a value of -15 for the second trade of the open position if it has ever reached 15 units of the specified currency loss.

OpenEntryMinProfitPerContract

Returns a numerical value, indicating minimal value of `OpenEntryProfitPerContract` for the time from entry order execution.

Usage

`OpenEntryMinProfitPerContract(`*`EntryIndex`*`)`

Where: *`EntryIndex`* - a numerical expression, specifying the number of trade (zero-based).

Notes

This function can only be used in signals.

To retrieve the total number of trades in open position use `OpenEntriesCount`

Example

`OpenEntryMinProfitPerContract(1)` will return a value of -1.5 for the second trade of the open position if it has ever reached 1.5 units of the specified currency per contract loss.

OpenEntryPrice

Returns a numerical value, indicating the price of specified entry into the open position.

Usage

`OpenEntryPrice(`*`EntryIndex`*`)`

Where: *`EntryIndex`* - a numerical expression, specifying the number of trade (zero-based).

Notes

This function can only be used in signals.

To retrieve the total number of trades in open position use `OpenEntriesCount`

Example

`OpenEntryPrice` will return a value of 101 for the first trade of the open position if it was executed at 101 price level.

`OpenEntryPrice(1)` will return a value of 101.5 for the second trade of the open position if it was executed at 101.5 price level.

OpenEntryProfit

Returns a numerical value, indicating the profit (loss if negative) of specified entry into the open position in the specified currency.

Usage

`OpenEntryProfit(`*`EntryIndex`*`)`

Where: *`EntryIndex`* - a numerical expression, specifying the number of trade (zero-based).

Notes

This function can only be used in signals.

To retrieve the total number of trades in open position use `OpenEntriesCount`

Example

`OpenEntryProfit` will return a value of 10 for the first trade of the open position if it has 10 units of the specified currency profit at the moment.

`OpenEntryProfit(1)` will return a value of -10 for the second trade of the open position if it has reached 10 units of the specified currency loss at the moment.

OpenEntryProfitPerContract

Returns a numerical value, indicating the profit (loss if negative) per contract of specified entry in the specified currency.

Usage

`OpenEntryProfitPerContract(`*`EntryIndex`*`)`

Where: *`EntryIndex`* - a numerical expression, specifying the number of trade (zero-based).

Notes

This function can only be used in signals.

To retrieve the total number of trades in open position use `OpenEntriesCount`

Example

`OpenEntryProfitPerContract` will return a value of 1 for the first trade of the open position if it has 1 unit of the selected currency per contract profit at the moment.

`OpenEntryProfitPerContract(1)` will return a value of -0.5 for the second trade of the open position if it has reached 0.5 units of the selected currency per contract loss at the moment.

OpenEntryTime

Returns a numerical value, indicating the time of specified entry into the open position.

The time is indicated in the HHmm format, where HH is the hour in 24 hours format and mm are minutes.

Usage

`OpenEntryTime(`*`EntryIndex`*`)`

Where: *`EntryIndex`* - a numerical expression, specifying the number of trade (zero-based).

Notes

This function can only be used in signals.

To retrieve the total number of trades in open position use `OpenEntriesCount`

Example

`OpenEntryTime` will return a value of 1015 for the first trade of the open position if it was executed at 10:15 AM.

`OpenEntryTime(1)` will return a value of 1545 for the second trade of the open position if it was executed at 3:45 PM.

PosTradeCommission

Returns an absolute numerical value, indicating the commission amount spent for the specified trade.

Usage

`PosTradeCommission(`*`PosAgo`*`,`*`TradeNumber`*`)`

Where: *`PosAgo`* - a numerical expression, specifying the position:

0 - open position;

1 - one position back (the last position closed);

2 - two positions back, etc.

`TradeNumber` - a numerical expression, specifying the number of trade (zero-based).

Notes

This function can only be used in signals.

To retrieve the total number of trades in specified position use `PosTradeCount`

Example

`PosTradeCommission(0,1)` will return a value of 5 for the second trade of the open position, if the commission for this trade is 5 units of the selected currency.

PosTradeCount

Returns a numerical value, indicating the total number of entries for the specified position.

Usage

`PosTradeCount(`*`PosBack`*`)`

Where: *`PosBack`* - a numerical expression, specifying the position:

0 - open position;

1 - one position back (the last position closed);

2 - two positions back, etc.

Notes

This function can only be used in signals. Please note that if there is a single entry and multiple scalping exits, each exit will have its own entry in this case and the count will increase after each partial exit.

Example

`PosTradeCount (1)` will return a value of 2 if there were two separate entries for the most recently closed position.

PosTradeEntryBar

Returns an absolute numerical value, indicating bar number of the trade entry order.

Usage

`PosTradeEntryBar(`*`PosAgo`*`,`*`TradeNumber`*`)`

Where: *`PosAgo`* - a numerical expression, specifying the position:

0 - open position;

1 - one position back (the last position closed);

2 - two positions back, etc.

`TradeNumber` - a numerical expression, specifying the number of trade (zero-based).

Notes

This function can only be used in signals.

To retrieve the total number of trades in specified position use `PosTradeCount`

Example

`PosTradeEntryBar(0,1)` will return a value of 25 for the second trade of the open position, if this trade was opened on 25th bar.

PosTradeEntryCategory

Returns an absolute numerical value, indicating trade entry order category.

The following types are possible:

1 = Stop order (buy next bar at close - 1 point stop)
2 = Limit order (buy next bar at close + 1 point limit)
3 = Market order (buy next bar market)
4 = Market at Close order (buy this bar at close)
5 = Market at Open order (buy next bar open)
6 = Reserved for special orders
7 = Reserved for special orders
8 = StopLimit order (buy 1 contracts next bar at close - 2 point stop close + 2 point limit)

Usage

`PosTradeEntryCategory(`*`PosAgo`*`,`*`TradeNumber`*`)`

Where: *`PosAgo`* - a numerical expression, specifying the position:

0 - open position;

1 - one position back (the last position closed);

2 - two positions back, etc.

`TradeNumber` - a numerical expression, specifying the number of trade (zero-based).

Notes

This function can only be used in signals.

To retrieve the total number of trades in specified position use `PosTradeCount`

Example

`PosTradeEntryCategory(0,1)` will return a value of 1 for the second trade of the open position, if the order type was Stop Order.

PosTradeEntryDateTime

Returns double-precision decimal DateTime for entry order. As an example see `ComputerDateTime`

Usage

`PosTradeEntryDateTime(`*`PosAgo`*`,`*`TradeNumber`*`)`

Where: *`PosAgo`* - a numerical expression, specifying the position:

0 - open position;

1 - one position back (the last position closed);

2 - two positions back, etc.

`TradeNumber` - a numerical expression, specifying the number of trade (zero-based).

Notes

This function can only be used in signals.

To retrieve the total number of trades in specified position use `PosTradeCount`

Example

`PosTradeEntryDateTime(0,1)` will return a value of 39448.25000000 for the second trade of the open position, if this trade was opened at 6:00 AM on January 1st, 2008.

PosTradeEntryName

Returns entry order name. Entry Name is indicated on the chart and in Order and Position Tracker Window.

Usage

`PosTradeEntryName(`*`PosAgo`*`,`*`TradeNumber`*`)`

Where: *`PosAgo`* - a numerical expression, specifying the position:

0 - open position;

1 - one position back (the last position closed);

2 - two positions back, etc.

`TradeNumber` - a numerical expression, specifying the number of trade (zero-based).

Notes

This function can only be used in signals.

To retrieve the total number of trades in specified position use `PosTradeCount`

Example

`PosTradeEntryName(0,1)` will return a value of "buy LE" for the second trade of the open position, if this trade was opened by the order with "buy LE" name.

PosTradeEntryPrice

Returns an absolute numerical value, indicating the execution price of trade entry order.

Usage

`PosTradeEntryPrice(`*`PosAgo`*`,`*`TradeNumber`*`)`

Where: *`PosAgo`* - a numerical expression, specifying the position:

0 - open position;

1 - one position back (the last position closed);

2 - two positions back, etc.

`TradeNumber` - a numerical expression, specifying the number of trade (zero-based).

Notes

This function can only be used in signals.

To retrieve the total number of trades in specified position use `PosTradeCount`

Example

`PosTradeEntryPrice(0,1)` will return a value of 100.2 for the second trade of the open position, if this trade was opened by the order filled at 100.2.

PosTradeExitBar

Returns an absolute numerical value, indicating bar number of the trade exit order.

Usage

`PosTradeExitBar(`*`PosAgo`*`,`*`TradeNumber`*`)`

Where: *`PosAgo`* - a numerical expression, specifying the position:

0 - open position;

1 - one position back (the last position closed);

2 - two positions back, etc.

`TradeNumber` - a numerical expression, specifying the number of trade (zero-based).

Notes

This function can only be used in signals.

To retrieve the total number of trades in specified position use `PosTradeCount`

Example

`PosTradeExitBar(0,1)` will return a value of 28 for the second trade of the open position, if this trade was closed on 28th bar.

PosTradeExitCategory

Returns an absolute numerical value, indicating trade exit order category.

The following types are possible:

1 = Stop order (buy next bar at close - 1 point stop)
2 = Limit order (buy next bar at close + 1 point limit)
3 = Market order (buy next bar market)
4 = Market at Close order (buy this bar at close)
5 = Market at Open order (buy next bar open)
6 = Reserved for special orders
7 = Reserved for special orders
8 = StopLimit order (buy 1 contracts next bar at close - 2 point stop close + 2 point limit)

Usage

`PosTradeExitCategory(`*`PosAgo`*`,`*`TradeNumber`*`)`

Where: *`PosAgo`* - a numerical expression, specifying the position:

0 - open position;

1 - one position back (the last position closed);

2 - two positions back, etc.

`TradeNumber` - a numerical expression, specifying the number of trade (zero-based).

Notes

This function can only be used in signals.

To retrieve the total number of trades in specified position use `PosTradeCount`

Example

`PosTradeExitCategory(0,1)` will return a value of 3 for the second trade of the open position, if the closing order type was Market Order.

PosTradeExitDateTime

Returns double-precision decimal DateTime for exit order. As an example see `ComputerDateTime`

Usage

`PosTradeExitDateTime(`*`PosAgo`*`,`*`TradeNumber`*`)`

Where: *`PosAgo`* - a numerical expression, specifying the position:

0 - open position;

1 - one position back (the last position closed);

2 - two positions back, etc.

`TradeNumber` - a numerical expression, specifying the number of trade (zero-based).

Notes

This function can only be used in signals.

To retrieve the total number of trades in specified position use `PosTradeCount`

Example

`PosTradeEntryDateTime(1,1)` will return a value of 39448.25000000 for the second trade of the last closed position, if this trade was closed at 6:00 AM on January 1st, 2008.

PosTradeExitName

Returns exit order name. Exit Name is indicated on the chart and in Order and Position Tracker Window.

Usage

`PosTradeExitName(`*`PosAgo`*`,`*`TradeNumber`*`)`

Where: *`PosAgo`* - a numerical expression, specifying the position:

0 - open position;

1 - one position back (the last position closed);

2 - two positions back, etc.

`TradeNumber` - a numerical expression, specifying the number of trade (zero-based).

Notes

This function can only be used in signals.

To retrieve the total number of trades in specified position use `PosTradeCount`

Example

`PosTradeExitName(0,1)` will return a value of "sell LX" for the second trade of the open position, if this trade was closed by the order with "sell LX" name.

PosTradeExitPrice

Returns an absolute numerical value, indicating the execution price of trade exit order.

Usage

`PosTradeExitPrice(`*`PosAgo`*`,`*`TradeNumber`*`)`

Where: *`PosAgo`* - a numerical expression, specifying the position:

0 - open position;

1 - one position back (the last position closed);

2 - two positions back, etc.

`TradeNumber` - a numerical expression, specifying the number of trade (zero-based).

Notes

This function can only be used in signals.

To retrieve the total number of trades in specified position use `PosTradeCount`

Example

`PosTradeExitPrice(0,1)` will return a value value of 100.5 for the second trade of the open position, if this trade was closed by the order filled at 100.5.

PosTradeIsLong

Returns True value if the trade was opened by buy order, otherwise False value is returned.

Usage

`PosTradeIsLong`(*`PosAgo`*,*`TradeNumber`*)

Where: *`PosAgo`* - a numerical expression, specifying the position:

0 - open position;

1 - one position back (the last position closed);

2 - two positions back, etc.

`TradeNumber` - a numerical expression, specifying the number of trade (zero-based).

Notes

This function can only be used in signals.

To retrieve the total number of trades in specified position use `PosTradeCount`

Example

`PosTradeIsLong`(`0`,`1`) will return True for the second trade of the open position, if this trade was opened by "buy" order.

PosTradeIsOpen

Returns True value if the trade is open, False value if the trade is closed.

It makes sense to check the trades of the open position.

For other positions False is always returned.

Usage

`PosTradeIsOpen(`*`PosAgo`*`,`*`TradeNumber`*`)`

Where: *`PosAgo`* - a numerical expression, specifying the position:

0 - open position;

1 - one position back (the last position closed);

2 - two positions back, etc.

`TradeNumber` - a numerical expression, specifying the number of trade (zero-based).

Notes

This function can only be used in signals.

To retrieve the total number of trades in specified position use `PosTradeCount`

Example

`PosTradeIsOpen(0,1)` will return True for the second trade of the open position, if this trade is opened (there was no closing order).

PosTradeProfit

Returns an absolute numerical value, indicating the profit (or loss if negative) of the specified trade.

Usage

`PosTradeProfit(`*`PosAgo`*`,`*`TradeNumber`*`)`

Where: *`PosAgo`* - a numerical expression, specifying the position:

0 - open position;

1 - one position back (the last position closed);

2 - two positions back, etc.

`TradeNumber` - a numerical expression, specifying the number of trade (zero-based).

Notes

This function can only be used in signals.

To retrieve the total number of trades in specified position use `PosTradeCount`

Example

`PosTradeProfit(0,1)` will return a value of 3 for the second trade of the open position, if the profit for this trade is 3 units of the selected currency.

PosTradeSize

Returns an absolute numerical value, indicating the number of contracts or shares in the specified trade.

Usage

`PosTradeSize(`*`PosAgo`*`,`*`TradeNumber`*`)`

Where: *`PosAgo`* - a numerical expression, specifying the position:

0 - open position;

1 - one position back (the last position closed);

2 - two positions back, etc.

`TradeNumber` - a numerical expression, specifying the number of trade (zero-based).

Notes

This function can only be used in signals.

To retrieve the total number of trades in specified position use `PosTradeCount`

Example

`PosTradeSize(0,1)` will return a value of 2 for the second trade of the open position, if this trade had a quantity of 2.

Commission

Returns the commission currency value entered in the Strategy Properties window.

Usage

`Commission`

Notes

This function can only be used in signals.

Example

`Commission` will return a value of 10.00 if the commission has been set to ¤10

GetStrategyName

Retained for backward compatibility.

Margin

Returns a value in units of the selected currency that indicates the margin value per contract.

Usage

```
Margin
```

Notes

The margin value may not be returned for all types of the securities. The margin value is returned for futures and options.

Example

```
Margin
```

Slippage

Returns the slippage currency value entered in the Strategy Properties window.

Usage

```
Slippage
```

Notes

This function can only be used in signals.

Example

`Slippage` will return a value of 0.25 if the slippage has been set to ¤0.25

MC_Text_GetActive

Returns a numerical value indicating the text ID number of the currently selected text; returns a value of -1 if no text is currently selected.

Usage

```
MC_Text_GetActive
```

Example

Assign a value, indicating the text ID number of the currently selected text, to `Value1` variable: `Value1` = `MC_Text_GetActive`;

Text_Anchor_to_Bars

Anchors the corresponding text drawing to the visible bar index; returns a value of 0 if the operation was performed successfully, and a value of -2 if the specified trendline ID number is invalid.

Usage

`Text_Anchor_to_Bars(`*`Text_ID`*`,`*`LogicalExpression`*`)`

Where: *`Text_ID`* is a numerical expression specifying the text drawing ID number
`LogicalExpression` is a logical value; True = add option and False = remove option

Notes

Text ID number is returned by `Text_New` when the text drawing is created.

Example

Anchor the text drawing with an ID number of 3 to the visible bar index:

```
Value1=Text_Anchor_to_Bars(3,True);
```

Text_Delete

Removes a text object with the specified ID number from a chart; returns a value of 0 if the object was successfully removed, and a value of -2 if the specified object ID number is invalid.

Usage

`Text_Delete(`*`ObjectID`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number

Notes

An object-specific ID number is returned by `Text_New` when the text object is created.

Example

Remove the text object with an ID number of 3:

```
Value1=Text_Delete(3);
```

Text_GetActive

Returns a numerical value indicating the object ID number of the currently selected text object; returns a value of -1 if no text objects are currently selected.

Usage

```
Text_GetActive
```

Notes

An object-specific ID number is assigned by `Text_New` when the text object is created.

Example

Assign a value, indicating the object ID number of the currently selected text object, to Value1 variable:

```
Value1=Text_GetActive;
```

Text_GetAttribute

Returns a logical value indicating the setting for an attribute of a text object with the specified ID number; returns a value of `True` if the attribute is set to on, and a value of `False` if the attribute is set to off or if the specified object ID number is invalid.

The settings of the following attributes can be returned: border, bold, italic, strike-out, and underline.

Usage

`Text_GetAttribute(`*`ObjectID`*`,`*`Attribute`*`)`

Parameters

`ObjectID` - a numerical expression specifying the object ID number

`Attribute` - a numerical expression specifying the attribute:

`0` - border
`1` - bold
`2` - italic
`3` - strike-out
`4` - underline

Notes

An object-specific ID number is returned by `Text_New` when the text object is created.

Example

Assign a true/false value, indicating the setting of "bold" attribute for the text object with an ID number of 3, to Bold variable:

```
Variable:Bold(False);
```

```
Bold=Text_GetAttribute(3,1);
```

Text_GetBarNumber

Returns a numerical value representing the barnumber of the text object with a specified ID; returns a value of -2 if the specified object ID number is invalid.

Usage

`Text_GetBarNumber(`*`ref`*`);`

Parameters:

`ref` - ID of the text object;

Example

Get the number of the bar where a text object with ID = 1 is placed:

```
Text_GetBarNumber(1);
```

Text_GetBGColor

Returns an RGB color number or a legacy color value that correspond to the background color of a text object with the specified ID number; returns a value of -2 if the specified object ID number is invalid.

Usage

`Text_GetBGColor(`*`ObjectID`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number

Notes

An object-specific ID number is returned by `Text_New` when the text object is created.

Example

Assign an RGB color number, corresponding to the background color of the text object with an ID number of 3, to Value1 variable:

```
Value1=Text_GetBGColor(3);
```

Assign a legacy color value, corresponding to the background color of the text object with an ID number of 3, to Value1 variable:

```
[LegacyColorValue=True];
Value1=Text_GetBGColor(3);
```

Text_GetBorder

Returns a logical value, indicating whether a border is added to a text object with the specified ID number; returns a value of `True` if the border has been added, and a value of `False` if the border has not been added or if the specified object ID number is invalid.

Usage

```
Text_GetBorder(ObjectID)
```

Where: ***`ObjectID`*** - a numerical expression specifying the object ID number

Notes

An object-specific ID number is returned by `Text_New` when the text object is created.

Example

Assign a true/false value, indicating whether a border is added to the text object with an ID number of 3, to Border variable:

```
Variable:Border(False);
Border=Text_GetBorder(3);
```

Text_GetColor

Returns an RGB color number or a legacy color value that correspond to the color of the text contained in a text object with the specified ID number; returns a value of -2 if the specified object ID number is invalid.

Usage

`Text_GetColor(`*`ObjectID`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number

Notes

An object-specific ID number is returned by `Text_New` when the text object is created.

Example

Assign an RGB color number, corresponding to the color of the text object with an ID number of 3, to Value1 variable:

```
Value1=Text_GetColor(3);
```

Assign a legacy color value, corresponding to the color of the text object with an ID number of 3, to Value1 variable:

```
[LegacyColorValue=True];
Value1=Text_GetColor(3);
```

Text_GetDate

Returns a numerical value, indicating the date of the bar at which a text object with the specified ID number has been placed; returns a value of -2 if the specified object ID number is invalid.

The date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month.

Usage

`Text_GetDate(`*`ObjectID`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number

Notes

An object-specific ID number is returned by `Text_New` when the text object is created.

Example

Assign a value, indicating the date of the bar at which a text object with the ID number of 3 has been placed, to Value1 variable:

```
Value1=Text_GetDate(3);
```

Text_GetFirst

Returns a numerical value, indicating an object ID number of the oldest (the first to be added to the current chart) text object of the specified origin; returns a value of -2 if the specified object ID number is invalid.

Usage

`Text_GetFirst` (*`Origin`*)

Parameters

`Origin` - a numerical expression specifying the origin of the text object:

`1` - added by the current study
`2` - added by a study other then the current study, or drawn manually by the user
`3` - added by any study, or drawn manually by the user
`4` - added by the current study, or drawn manually by the user
`5` - added by a study other then the current study
`6` - added by any study
`7` - added manually by the user

Notes

If the oldest (the first added) text object is deleted, the next oldest (the second added) text object becomes the oldest (the first added) text object.

Example

Assign a value, indicating an object ID number of the oldest text object added to the chart by the current study, to Value1 variable:

```
Value1=Text_GetFirst(1);
```

Text_GetFontName

Returns a string expression corresponding to the name of the font assigned to a text object with the specified ID number.

Usage

`Text_GetFontName(`*`ObjectID`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number

Notes

An object-specific ID number is returned by `Text_New` when the text object is created.

Example

`Text_GetFontName(3)` will return a string expression corresponding to the name of the font assigned to a text object with the ID number of 3

Text_GetHStyle

Returns the horizontal placement style of a text object with the specified ID number; returns a value of -2 if the specified object ID number is invalid.

Usage

`Text_GetHStyle(`*`ObjectID`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number

Return

Horizontal placement style:

`0` - to the right of the specified bar
`1` - to the left of the specified bar
`2` - centered on the specified bar

Notes

An object-specific ID number is returned by `Text_New` when the text object is created.

Example

Assign a value, indicating the horizontal placement style of the text object with an ID number of 3, to Value1 variable:

```
Value1=Text_GetHStyle(3);
```

Text_GetLock

Locked text drawings cannot be moved manually. Keyword returns a value of True for Locked drawings, and a value of False for others.

Usage

```
Text_GetLock(Text_ID)
```

Where: *`Text_ID`* - a numerical expression specifying the text drawing ID number

Notes

A text ID number is returned by `Text_New` when the text drawing is created.

Example

Assign Lock property of the text drawing with an ID number of 3 to Condition1 variable:

```
Condition1=Text_GetLock(3);
```

Text_GetNext

Returns an ID number of the first existing text object added subsequent to a text object with the specified ID number, with both objects of a specified origin; returns a value of -2 if the specified object ID number is invalid.

Usage

`Text_GetNext(`*`ObjectID`*`,`*`Origin`*`)`

Parameters

`ObjectID` - a numerical expression specifying the object ID number

`Origin` - a numerical expression specifying the origin of the text objects:

`1`	- added by the current study
`2`	- added by a study other then the current study, or drawn manually by the user
`3`	- added by any study, or drawn manually by the user
`4`	- added by the current study, or drawn manually by the user
`5`	- added by a study other then the current study
`6`	- added by any study
`7`	- added manually by the user

Example

Assign a value to Value1 variable, indicating an ID number of the first existing text object added subsequent to a text object with the ID number of 3, with both objects added by the current study:

```
Value1=Text_GetNext(3,1);
```

Text_GetSize

Returns a numerical value indicating the font size assigned to a text object with the specified ID number; returns a value of -2 if the specified object ID number is invalid.

Usage

`Text_GetSize(`*`ObjectID`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number

Notes

An object-specific ID number is returned by `Text_New` when the text object is created.

Example

Assign a value, indicating the font size of the text object with an ID number of 3, to Value1 variable:

```
Value1=Text_GetSize(3);
```

Text_GetString

Returns a string expression corresponding to the text contained in a text object with the specified ID number.

Usage

`Text_GetString(`*`ObjectID`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number

Notes

An object-specific ID number is returned by `Text_New` when the text object is created.

Example

`Text_GetString(3)` will return a string expression corresponding to the text contained in the text object with an ID number of 3

Text_GetTime

Returns a numerical value, indicating the time of the bar at which a text object with the specified ID number has been placed; returns a value of -2 if the specified object ID number is invalid.

The time is indicated in the 24-hour HHmm format, where 1300 = 1:00 PM.

Usage

`Text_GetTime(`*`ObjectID`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number

Notes

An object-specific ID number is returned by `Text_New` when the text object is created.

Example

Assign a value, indicating the time of the bar at which a text object with the ID number of 3 has been placed, to Value1 variable:

```
Value1=Text_GetTime(3);
```

Text_GetTime_DT

Returns a double-precision decimal DateTime value indicating the time of the bar at which a text object with the specified ID number has been placed; returns a value of -2 if the specified object ID number is invalid.

The time is indicated in the DateTime format, where the integer portion of the DateTime value indicates the number of days that have elapsed since January 1st, 1900, and the fractional portion of the DateTime value indicates the fraction of the day that has passed since midnight. DateTime is a floating point value with high precision. It allows accessing millisecond time stamps of the bar.

Usage

`Text_GetTime_DT(`*`ObjectID`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number

Notes

An object-specific ID number is returned by `Text_New_Dt` when the text object is created.

Example

Assign a value, indicating the time of the bar at which a text object with the ID number of 3 has been placed, to Value1 variable:

```
Value1=Text_GetTime_DT(3);
```

Text_GetTime_s

Returns a numerical value indicating the time, including seconds, of the bar at which a text object with the specified ID number has been placed; returns a value of -2 if the specified object ID number is invalid.

The time is indicated in the 24-hour HHmmss format, where 130000 = 1:00:00 PM.

Usage

`Text_GetTime_s(`*`ObjectID`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number

Notes

An object-specific ID number is returned by `Text_New_s` when the text object is created.

Example

Assign a value, indicating the time of the bar at which a text object with the ID number of 3 has been placed, to Value1 variable:

```
Value1=Text_GetTime_s(3);
```

Text_GetValue

Returns the price value (vertical position, corresponding to a value on the price scale of a chart), at which a text object with the specified ID number has been placed; returns a value of -2 if the specified object ID number is invalid.

Usage

`Text_GetValue(`*`ObjectID`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number

Notes

An object-specific ID number is returned by `Text_New` when the text object is created.

Example

Assign a value, indicating the price value at which a text object with the ID number of 3 has been placed, to Value1 variable:

```
Value1=Text_GetValue(3);
```

Text_GetVStyle

Returns the vertical placement style of a text object with the specified ID number; returns a value of -2 if the specified object ID number is invalid.

Usage

`Text_GetVStyle(`*`ObjectID`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number

Return

Vertical placement style:

`0` - below the specified price value
`1` - above the specified price value
`2` - centered on the specified price value

Price value represents the vertical position corresponding to a value on the price scale of a chart.

Notes

An object-specific ID number is returned by `Text_New` when the text object is created.

Example

Assign a value, indicating the vertical placement style of the text object with an ID number of 3, to Value1 variable:

```
Value1=Text_GetVStyle(3);
```

Text_Get_Anchor_to_Bars

Returns the value of the "anchor to bar" option of the text drawing with a specified ID.

Usage

`Text_Get_Anchor_to_Bars(`*`Text_ID`*`)`

Where: *`Text_ID`* is a numerical expression specifying the text drawing ID number

Notes

Text ID number is returned by `Text_New` when the text drawing is created.

Example

Assign "anchor to bars" option of the text drawing with an ID number of 3 to the Condition1 variable:

```
Condition1=Text_Get_Anchor_to_Bars(3);
```

Text_Lock

Locks corresponding text drawing so it cannot be moved manually; returns a value of 0 if the operation was performed successfully, and a value of -2 if the specified trendline ID number is invalid.

Usage

`Text_Lock(`*`Text_ID`*`,`*`LogicalExpression`*`)`

Where: *`Text_ID`* - a numerical expression specifying the text drawing ID number
`LogicalExpression` - a logical value; True = Add and False = Remove

Notes

A text ID number is returned by `Text_New` when the text drawing is created.

Example

Lock the text drawing with an ID number of 3:

```
Value1=Text_Lock(3,True);
```

Unlock the text drawing with an ID number of 5:

```
Value1=Text_Lock(5,False);
```

Text_New

Displays a text object, consisting of the specified string expression located at the specified bar and specified price value, on the chart that the study is based on; returns an object-specific ID number, required to modify the object.

Usage

`Text_New` (*BarDate*, *BarTime*, *PriceValue*,"*`Text`*")

Parameters

`BarDate` - a numerical expression specifying the date of the bar at which the object is to be placed; the date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month

`BarTime` - a numerical expression specifying the time of the bar at which the object is to be placed; the time is indicated in the 24-hour HHmm format, where 1300 = 1:00 PM

`PriceValue` - a numerical expression specifying the price value (vertical position, corresponding to a value on the price scale of a chart), where the object is to be placed

`Text` - the string expression to be displayed

Example

Place, on the chart that the study is based on, the text "UpT" at the top of a bar if the Open price has increased incrementally over the last three bars:

```
If Open>Open[1] And Open[1]>Open[2] Then
Value1=Text_New(Date,Time,High,"UpT");
```

Text_New_BN

Displays a text object consisting of the specified string expression located at the specified bar and specified price value on the chart that the study is based on; returns an object-specific ID number required to modify the object.

Usage

`Text_New_BN` (*BarNumber*, *PriceValue*,"*`Text`*")

Parameters

`BarNumber` - Numerical expression specifying number of the bar (horizontal position).

`PriceValue` - a numerical expression specifying the price value (vertical position, corresponding to a value on the price scale of a chart), where the object should be placed.

`Text` - the string expression to be displayed.

Example

On the chart that the study is based on place the text "UpT" at the top of the bar if the Open price has increased incrementally over the last three bars:

```
If Open>Open[1] And Open[1]>Open[2] Then
Value1=Text_New_BN(currentbar,High,"UpT");
```

Text_New_Dt

Displays a text object, consisting of the specified string expression located at the specified bar and specified price value, on the chart that the study is based on; returns an object-specific ID number, required to modify the object.

Usage

`Text_New_Dt` (*Bar_DateTime*, *PriceValue*,"*`Text`*")

Parameters

`Bar_DateTime` - Numerical expression specifying date and time of the bar (horizontal position). The integer portion of the DateTime value indicates the number of days that have elapsed since January 1st, 1900, and the fractional portion of the DateTime value indicates the fraction of the day that has passed since midnight. DateTime is a floating point value with high precision. It allows accessing millisecond time stamps of the bar.

`PriceValue` - a numerical expression specifying the price value (vertical position, corresponding to a value on the price scale of a chart), where the object is to be placed.

`Text` - the string expression to be displayed

Example

Place, on the chart that the study is based on, the text "UpT" at the top of a bar if the Open price has increased incrementally over the last three bars:

```
If Open>Open[1] And Open[1]>Open[2] Then
Value1=Text_New_Dt(DateTime,High,"UpT");
```

Text_New_s

Displays a text object, consisting of the specified string expression located at the specified bar and specified price value, on the chart that the study is based on; returns an object-specific ID number, required to modify the object.

Usage

`Text_New_s` (*BarDate*, *BarTime_s*, *PriceValue*,"*`Text`*")

Parameters

`BarDate` - a numerical expression specifying the date of the bar at which the object is to be placed; the date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month

`BarTime_s` - a numerical expression specifying the time of the bar, including seconds, at which the object is to be placed; the time is indicated in the 24-hour HHmmss format, where 130000 = 1:00:00 PM

`PriceValue` - a numerical expression specifying the price value (vertical position, corresponding to a value on the price scale of a chart), where the object is to be placed

`Text` - the string expression to be displayed

Example

Place, on the chart that the study is based on, the text "UpT" at the top of a bar if the Open price has increased incrementally over the last three bars:

```
If Open>Open[1] And Open[1]>Open[2] Then
Value1=Text_New_s(Date,Time_s,High,"UpT");
```

Text_New_self

Displays a text object, consisting of the specified string expression located at the specified bar and specified price value, on the SubChart containing the study; returns an object-specific ID number, required to modify the object.

Usage

`Text_New_self` (*BarDate*, *BarTime*, *PriceValue*,"*`Text`*")

Parameters

`BarDate` - a numerical expression specifying the date of the bar at which the object is to be placed; the date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month

`BarTime` - a numerical expression specifying the time of the bar at which the object is to be placed; the time is indicated in the 24-hour HHmm format, where 1300 = 1:00 PM

`PriceValue` - a numerical expression specifying the price value (vertical position, corresponding to a value on the price scale of a chart), where the object is to be placed

`Text` - the string expression to be displayed

Example

Place, on the SubChart containing the study, the text "UpT" at the points of the plot where the Open price has increased incrementally over the last three bars:

```
Plot1(Close);
If Open>Open[1] And Open[1]>Open[2] Then
Value1=Text_New_self(Date,Time,High,"UpT");
```

Text_New_Self_BN

The same as `Text_New_BN`. Difference: Displays a text on the SubChart containing the study.

Text_New_Self_DT

Displays a text object, consisting of the specified string expression located at the specified bar and specified price value, on the SubChart containing the study; returns an object-specific ID number, required to modify the object.

Usage

`Text_New_Self_DT` (*Bar_DateTime*, *PriceValue*,"*`Text`*")

Parameters

`Bar_DateTime` - Numerical expression specifying date and time of the bar (horizontal position). The integer portion of the DateTime value indicates the number of days that have elapsed since January 1st, 1900, and the fractional portion of the DateTime value indicates the fraction of the day that has passed since midnight. DateTime is a floating point value with high precision. It allows accessing millisecond time stamps of the bar.

`PriceValue` - a numerical expression specifying the price value (vertical position, corresponding to a value on the price scale of a chart), where the object is to be placed.

`Text` - the string expression to be displayed

Example

Place, on the chart that the study is based on, the text "UpT" at the top of a bar if the Open price has increased incrementally over the last three bars:

```
If Open>Open[1] And Open[1]>Open[2] Then
Value1=Text_New_Self_DT(DateTime,High,"UpT");
```

Text_New_self_s

Displays a text object, consisting of the specified string expression located at the specified bar and specified price value, on the SubChart containing the study; returns an object-specific ID number, required to modify the object.

Usage

`Text_New_self_s` (*BarDate*, *BarTime_s*, *PriceValue*,"*`Text`*")

Parameters

`BarDate` - a numerical expression specifying the date of the bar at which the object is to be placed; the date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month

`BarTime_s` - a numerical expression specifying the time of the bar, including seconds, at which the object is to be placed; the time is indicated in the 24-hour HHmmss format, where 130000 = 1:00:00 PM

`PriceValue` - a numerical expression specifying the price value (vertical position, corresponding to a value on the price scale of a chart), where the object is to be placed

`Text` - the string expression to be displayed

Example

Place, on the SubChart containing the study, the text "UpT" at the points of the plot where the Open price has increased incrementally over the last three bars:

```
Plot1(Close);
If Open>Open[1] And Open[1]>Open[2] Then
Value1=Text_New_self_s(Date,Time_s,High,"UpT");
```

Text_SetAttribute

Sets an attribute of the text in a text object with the specified ID number; returns a value of 0 if the attribute was successfully set, and a value of -2 if the specified object ID number is invalid.

The following text attributes can be set: border, bold, italic, strike-out, and underline.

Usage

`Text_SetAttribute(`*`ObjectID`*`,`*`Attribute`*`,`*`LogicalExpression`*`)`

Parameters

`ObjectID` - a numerical expression specifying the object ID number

`Attribute` - a numerical expression specifying the attribute:

`0`	- border
`1`	- bold
`2`	- italic
`3`	- strike-out
`4`	- underline

`LogicalExpression` - a logical value; True = on and False = off

Notes

An object-specific ID number is returned by `Text_New` when the text object is created.

Example

Set the attribute "bold" to on for the text in a text object with the ID number of 3:

```
Value1=Text_SetAttribute(3,1,True);
```

Text_SetBarNumber

Assigns the specified barnumber to the text object with the specified ID number; returns a value of 0 if the barnumber was successfully assigned, and a value of -2 if the specified object ID number is invalid.

Usage

`Text_SetBarNumber(`*`ref`*`,`*`Barnumber`*`)`

Parameters:

- *`ref`* - ID number of the text object;
- *`Barnumber`* - the new bar number that is to be assigned to the specified object.

Example

Assign the new barnumber value of 100 to the text object with ID = 1:

```
Text_SetBarNumber(1, 100);
```

Text_SetBGColor

Assigns the specified background color to a text object with the specified ID number; returns a value of 0 if the color was successfully assigned, and a value of -2 if the specified object ID number is invalid.

Usage

`Text_SetBGColor(`*ObjectID*,*`BGColor`*`)`

Parameters

`ObjectID` - a numerical expression specifying the object ID number

`BGColor` - an expression specifying the background color

The color can be specified by a numerical expression representing an RGB color number or a legacy color value, or by one of 17 base color words.

Notes

An object-specific ID number is returned by `Text_New` when the text object is created.

Example

Assign the color blue to the background of a text object with the ID number of 3:

```
Value1=Text_SetBGColor(3,Blue);
```

Assign the RGB color 2138336 (Orange) to the background of a text object with the ID number of 3:

```
Value1=Text_SetBGColor(3,2138336);
```

Assign the legacy color 4 (Green) to the background of a text object with the ID number of 3:

```
[LegacyColorValue=True];
Value1=Text_SetBGColor(3,4);
```

Text_SetBorder

Adds or removes a border around the text object with the specified ID number; returns a value of 0 if the border was successfully set, and a value of -2 if the specified object ID number is invalid.

The color of the border is the same as the color of the text in the text object.

Usage

`Text_SetBorder(`*`ObjectID`*`,`*`LogicalExpression`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number
`LogicalExpression` - a logical value; True = Add and False = Remove

Notes

An object-specific ID number is returned by `Text_New` when the text object is created.

Example

Add a border to the text object with an ID number of 3:

```
Value1=Text_SetBorder(3,True);
```

Remove a border from the text object with an ID number of 3:

```
Value1=Text_SetBorder(3,False);
```

Text_SetColor

Assigns the specified color to the text of a text object with the specified ID number; returns a value of 0 if the color was successfully assigned, and a value of -2 if the specified object ID number is invalid.

Usage

`Text_SetColor(`*ObjectID*`,`*`TextColor`*`)`

Parameters

`ObjectID` - a numerical expression specifying the object ID number

`TextColor` - an expression specifying the text color

The color can be specified by a numerical expression representing an RGB color number or a legacy color value, or by one of 17 base color words.

Notes

An object-specific ID number is returned by `Text_New` when the text object is created.

Example

Assign the color blue to the text of a text object with the ID number of 3:

```
Value1=Text_SetColor(3,Blue);
```

Assign the RGB color 2138336 (Orange) to the text of a text object with the ID number of 3:

```
Value1=Text_SetColor(3,2138336);
```

Assign the legacy color 4 (Green) to the text of a text object with the ID number of

3:

```
[LegacyColorValue=True];
Value1=Text_SetColor(3,4);
```

Text_SetFontName

Assigns the specified font to a text object with the specified ID number; returns a value of -2 if the specified object ID number is invalid.

Any font in the **Fonts** folder can be used; the folder is accessible from the **Control Panel** in Windows XP operating system.

Usage

`Text_SetFontName(`*`ObjectID`*`,"`*`FontName`*`")`

Where: *`ObjectID`* - a numerical expression specifying the object ID number
`FontName` - a string expression specifying the font name

Notes

An object-specific ID number is returned by `Text_New` when the text object is created.

Example

Assign font Verdana to the text object with the ID number of 3:

```
Value1=Text_SetFontName(3,"Verdana");
```

Text_SetLocation

Modifies the location of a text object with the specified ID number; returns a value of 0 if the location of the object was successfully modified, and a value of -2 if the specified object ID number is invalid.

Usage

`Text_SetLocation` (*ObjectID*, *BarDate*, *BarTime*, *PriceValue*)

Parameters

`ObjectID` - a numerical expression specifying the object ID number

`BarDate` - a numerical expression specifying the date of the bar at which the object is to be placed; the date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month

`BarTime` - a numerical expression specifying the time of the bar at which the object is to be placed; the time is indicated in the 24-hour HHmm format, where 1300 = 1:00 PM

`PriceValue` - a numerical expression specifying the price value (vertical position, corresponding to a value on the price scale of a chart), where the object is to be placed

Notes

An object-specific ID number is returned by `Text_New` when the text object is created.

Example

Move the text object with an ID number of 3 to the top of the current bar:

```
Value1=Text_SetLocation(3,Date,Time,High);
```

Text_SetLocation_BN

Modifies location of a text object with the specified ID number; returns a value of 0 if location of the object was successfully modified, and a value of -2 if the specified object ID number is invalid.

Usage

`Text_SetLocation_BN` (*ObjectID*, *BarNumber*, *PriceValue*)

Parameters

`ObjectID` - a numerical expression specifying the object ID number.

`BarNumber` - a numerical expression specifying the bar number at which the object is to be placed.

`PriceValue` - a numerical expression specifying the price value (vertical position, corresponding to a value on the price scale of a chart), where the object should be placed.

Notes

An object-specific ID number is returned by `Text_New_Dt` when the text object is created.

Example

Move the text object with an ID number of 3 to the top of the current bar:

```
Value1=Text_SetLocation_BN(3,currentbar,High);
```

Text_SetLocation_DT

Modifies the location of a text object with the specified ID number; returns a value of 0 if the location of the object was successfully modified, and a value of -2 if the specified object ID number is invalid.

Usage

`Text_SetLocation_DT` (*ObjectID*, *Bar_DateTime*, *PriceValue*)

Parameters

`ObjectID` - a numerical expression specifying the object ID number.

`Bar_DateTime` - a numerical expression specifying the date and time of the bar at which the object is to be placed; the date is indicated in the DateTime format, where the integer portion of the DateTime value indicates the number of days that have elapsed since January 1st, 1900, and the fractional portion of the DateTime value indicates the fraction of the day that has passed since midnight. DateTime is a floating point value with high precision. It allows accessing millisecond time stamps of the bar.

`PriceValue` - a numerical expression specifying the price value (vertical position, corresponding to a value on the price scale of a chart), where the object is to be placed.

Notes

An object-specific ID number is returned by `Text_New_Dt` when the text object is created.

Example

Move the text object with an ID number of 3 to the top of the current bar:

```
Value1=Text_SetLocation_DT(3,DateTime,High);
```

Text_SetLocation_s

Modifies the location of a text object with the specified ID number; returns a value of 0 if the location of the object was successfully modified, and a value of -2 if the specified object ID number is invalid.

Usage

`Text_SetLocation_s` (*ObjectID*, *BarDate*, *BarTime_s*, *PriceValue*)

Parameters

`ObjectID` - a numerical expression specifying the object ID number

`BarDate` - a numerical expression specifying the date of the bar at which the object is to be placed; the date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month

`BarTime_s` - a numerical expression specifying the time of the bar, including seconds, at which the object is to be placed; the time is indicated in the 24-hour HHmmss format, where 130000 = 1:00:00 PM

`PriceValue` - a numerical expression specifying the price value (vertical position, corresponding to a value on the price scale of a chart), where the object is to be placed

Notes

An object-specific ID number is returned by `Text_New_s` when the text object is created.

Example

Move the text object with an ID number of 3 to the top of the current bar:

```
Value1=Text_SetLocation_s(3,Date,Time_s,High);
```

Text_SetSize

Assigns the specified font size to the text of a text object with the specified ID number; returns a value of 0 if the font size was successfully assigned, and a value of -2 if the specified object ID number is invalid.

Usage

`Text_SetSize(`*`ObjectID`*`,`*`FontSize`*`)`

Where: *`ObjectID`* - a numerical expression specifying the object ID number
`FontSize` - a numerical expression specifying the font size

Notes

An object-specific ID number is returned by `Text_New` when the text object is created.

Example

Assign the font size of 16 to the text of a text object with the ID number of 3:

```
Value1=Text_SetSize(3,16);
```

Text_SetString

Replaces with the specified string expression the text contained in a text object with the specified ID number; returns a value of 0 if the string expression was successfully replaced, and a value of -2 if the specified object ID number is invalid.

Usage

`Text_SetString(`*`ObjectID`*`,"`*`Text`*`")`

Where: *`ObjectID`* - a numerical expression specifying the object ID number
`Text` - the new string expression to be displayed in the text object

Notes

An object-specific ID number is returned by `Text_New` when the text object is created.

Example

Replace the text, contained in a text object with the ID number of 3, with the string expression "New Text":

```
Value1=Text_SetString(3,"New Text");
```

Text_SetStyle

Sets the placement style of a text object relative to the bar and to the price value specified for the object; returns a value of 0 if the placement style of the object was successfully modified, and a value of -2 if the specified object ID number is invalid.

Usage

`Text_SetStyle` (*ObjectID*, *HorizPl*, *VertPl*)

Parameters

`ObjectID` - a numerical expression specifying the object ID number

`HorizPl` - a numerical expression specifying the horizontal placement style for the text object:

`0` - to the right of the specified bar
`1` - to the left of the specified bar
`2` - centered on the specified bar

`VertPl` - a numerical expression specifying the vertical placement style for the text object:

`0` - below the specified price value
`1` - above the specified price value
`2` - centered on the specified price value

Price value represents the vertical position corresponding to a value on the price scale of a chart.

Notes

An object-specific ID number is returned by `Text_New` when the text object is created.

Example

Center the text object with an ID number of 3 on the bar and on the price value, specified for the object:

```
Value1=Text_SetStyle(3,2,2);
```

DoubleQuote

Displays the double-quote (") character in a string.

Example

This example demonstrates how the word Hello in double quotes can be displayed on the last bar above High:

```
Variable: ID(-1);
If LastBarOnChart Then
ID = Text_New(Date, Time, High + 1 Point, DoubleQuote + "Hello" +
DoubleQuote);
```

InStr

Returns the position of a specified string expression inside another specified string expression.

The position of the string being located is indicated by the number of characters from the left side of the string being evaluated.

Usage

`InStr(`*`String1`*`,`*`String2`*`)`

Where: *`String1`* - a string to be evaluated
`String2` - a string to be located

Notes

In case the search returns no results 0 is returned

In case if the sought word occurs more than once, the first position will be returned

Search is case sensetive

Example

```
InStr("Friday is the expiration day", "Friday");
```

will return a value of 1, indicating that the string "Friday" begins at position 1 of the String "Friday is the expiration day"

```
InStr("Friday is the expiration day", "Monday");
```

will return a value of 0, indicating that the string "Monday" does not exist in the String "Friday is the expiration day"

LeftStr

Returns one or more leftmost characters from the specified string expression.

Usage

`LeftStr(`*`String`*,*sSize*)

Where: *`String`* - a string expression from which the characters are to be taken
`sSize` - a numerical expression specifying the number of characters to be returned

Example

`LeftStr("Hello World",5);` will return a string expression "Hello"

LowerStr

Converts the uppercase letters of a specified string expression to a lowercase.

Usage

`LowerStr`("*`String`*")

Where: *`String`* - a string expression to be converted

Example

`LowerStr("Return on Account");` will return a string expression "return on account"

MidStr

Returns a part, starting from a specified position and of a specified length, of a specified string expression.

Usage

`MidStr("`*`String`*`"`**,***Pos*,*Num*)

Where: *`String`* - a string expression the specified part is to be taken from
`Pos` - a numerical expression specifying the position, from the left side of the string, of the starting character of the part
`Num` - a numerical expression specifying the length, in characters, of the part

Example

`MidStr("Largest winning trade",1,7);` will return a string expression "Largest"

NewLine

Starts a new line and returns carriage.

Notes

Use the "+" character to add `NewLine` to a string expression.

Example

```
FileAppend("c:\testfile.txt","We can see a new bar for"+
NumToStr(Date,0)+ NewLine);
```

NumToStr

Returns a specified numerical expression in the form of a string expression.

Usage

`NumToStr`(*Expression*,*Dec*)

Where: *`Expression`* - a numerical expression to be converted to a string expression

`Dec` - a numerical expression specifying the number of decimal places the returned string expression is to contain

Example

`NumToStr(1500.5,3);` will return a string expression "1500.500"

RightStr

Returns one or more rightmost characters from the specified string expression.

Usage

`RightStr(`*`String`*,*sSize*)

Where: *`String`* - a string expression from which the characters are to be taken
`sSize` - a numerical expression specifying the number of characters to be returned

Example

`RightStr("Hello World",5);` will return a string expression "World"

Spaces

Returns a string expression consisting of a specified number of spaces.

Usage

`Spaces(`*Num*`)`

Where: *`Num`* - a numerical expression specifying the number of spaces

Example

The example inserts two blank spaces between the letters "a" and "b":

```
Print("a"+Spaces(2)+"b");
```

StrLen

Returns the length, in characters, of a specified string expression.

Usage

`StrLen`("*`String`*")

Where: *`String`* - a string expression to be evaluated

Example

`StrLen("Drawdown")`; will return a value of 8

StrToNum

Converts a specified string expression to a numerical value.

Usage

`StrToNum("`*`String`*`")`

Where: *`String`* - a string expression to be converted

Notes

If non-numeric characters are encountered the rest of the expression is ignored.

Example

`StrToNum("2500.70");` will return a value of 2500.70

Text

Returns the string, formed up by the transferred arguments.

Usage

`Text`(Param1, Param2, ..., ParamN);

Where: `Param(i)` is string expression, numeric expression or `True/False` value.

Example

```
Text_new(D,T,C,Text("Bar Date/Time is ",Date:0:0,"/",Time:0:0));
```

UpperStr

Converts the lowercase letters of a specified string expression to uppercase.

Usage

`UpperStr("`*`String`*`")`

Where: *`String`* - a string expression to be converted

Example

`UpperStr("msft");` will return a string expression "MSFT"

MC_TL_GetActive

Returns a numerical value indicating the trendline ID number of the currently selected trendline; returns a value of -1 if no trendlines are currently selected.

Usage

```
MC_TL_GetActive
```

Notes

A trendline-specific ID number is assigned by `MC_TL_New` when the trendline is created.

Example

Assign a value, indicating the trendline ID number of the currently selected trendline, to `Value1` variable:

```
Value1 = MC_TL_GetActive;
```

MC_TL_New

The same as `TL_New`.

MC_TL_New_BN

The same as `TL_New_BN`.

MC_TL_New_DT

The same as `TL_New_DT`.

MC_TL_New_Self

The same as `TL_New_Self`.

MC_TL_New_Self_BN

The same as `TL_New_Self_BN`.

MC_TL_New_Self_DT

The same as `TL_New_Self_DT`.

TL_Anchor_to_Bars

Anchors the corresponding trendline drawing to the visible bar index; returns a value of 0 if the operation was performed successfully, and a value of -2 if the specified trendline ID number is invalid.

Usage

`TL_Anchor_to_Bars(`*`TL_ID`*`,`*`LogicalExpression`*`)`

Where: *`TL_ID`* is a numerical expression specifying the trendline ID number
`LogicalExpression` is a logical value; True = Add option and False = Remove option

Notes

A trendline-specific ID number is returned by `TL_New` when the trendline is created.

Example

Anchor the trendline with an ID number of 3 to the visible bar index:

```
Value1=TL_Anchor_to_Bars(3,True);
```

TL_Delete

Removes a trendline with the specified ID number from a chart; returns a value of 0 if the trendline was successfully removed, and a value of -2 if the specified trendline ID number is invalid.

Usage

`TL_Delete(`*`TL_ID`*`)`

Where: *`TL_ID`* - a numerical expression specifying the trendline ID number

Notes

A trendline-specific ID number is returned by `TL_New` when the trendline is created.

Example

Remove the trendline with an ID number of 3:

```
Value1=TL_Delete(3);
```

TL_GetActive

This reserved word returns a numeric value representing the ID of the currently active trendline.

Usage

```
TL_GetActive
```

Notes

When the reserved word performs its operation successfully, trendline ID is returned.

When a reserved word cannot perform its operation, it returns an error code.

Example

```
Value1=TL_GetActive;
```

Assigns the ID of the currently active trendline to Value1.

Value1 is any numeric variable or array.

Trendline reserved word should be assigned to a numeric variable or array in order to determine whether the reserved word performed its operation successfully or not.

TL_GetAlert

Returns the alert status for a trendline with the specified ID number; returns a value of -2 if the specified trendline ID number is invalid.

Usage

`TL_GetAlert(`*`TL_ID`*`)`

Where: *`TL_ID`* - a numerical expression specifying the trendline ID number

Return

Alert status:

`0` - Alert disabled

`1` - Breakout intra-bar

An alert is triggered if the High crosses over the trendline or the Low crosses under the trendline. Alert triggering conditions are evaluated intra-bar.

`2` - Breakout on close

An alert is triggered if the Close of the previous bar was below the trendline and the Close of the current bar is above the trendline, or if the Close of the previous bar was above the trendline and the Close of the current bar is below the trendline. Alert triggering conditions are evaluated at Close of a bar.

Notes

A trendline-specific ID number is returned by `TL_New` when the trendline is created.

Example

Assign a value, indicating the alert status for a trendline with the ID number of 3, to Value1 variable:

```
Value1=TL_GetAlert(3);
```

TL_GetBeginDate

Returns a numerical value, indicating the date of the starting point of a trendline with the specified ID number; returns a value of -2 if the specified trendline ID number is invalid.

Of the two end points of a trendline, the point with the earlier date and time is always considered to be the starting point; if the trendline is vertical, the point with the lower price value is considered to be the starting point.

The date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month.

Usage

`TL_GetBeginDate(`*`TL_ID`*`)`

Where: *`TL_ID`* - a numerical expression specifying the trendline ID number

Notes

A trendline-specific ID number is returned by `TL_New` when the trendline is created.

Example

Assign a value, indicating the date of the starting point of a trendline with the ID number of 3, to Value1 variable:

```
Value1=TL_GetBeginDate(3);
```

TL_GetBeginTime

Returns a numerical value, indicating the time of the starting point of a trendline with the specified ID number; returns a value of -2 if the specified trendline ID number is invalid.

Of the two end points of a trendline, the point with the earlier date and time is always considered to be the starting point; if the trendline is vertical, the point with the lower price value is considered to be the starting point.

The time is indicated in the 24-hour HHmm format, where 1300 = 1:00 PM.

Usage

`TL_GetBeginTime(`*`TL_ID`*`)`

Where: *`TL_ID`* - a numerical expression specifying the trendline ID number

Notes

A trendline-specific ID number is returned by `TL_New` when the trendline is created.

Example

Assign a value, indicating the time of the starting point of a trendline with the ID number of 3, to Value1 variable:

```
Value1=TL_GetBeginTime(3);
```

TL_GetBeginTime_s

Returns a numerical value indicating the time, including seconds, of the starting point of a trendline with the specified ID number; returns a value of -2 if the specified trendline ID number is invalid.

Of the two end points of a trendline, the point with the earlier date and time is always considered to be the starting point; if the trendline is vertical, the point with the lower price value is considered to be the starting point.

The time is indicated in the 24-hour HHmmss format, where 130000 = 1:00:00 PM.

Usage

`TL_GetBeginTime_s(`*`TL_ID`*`)`

Where: *`TL_ID`* - a numerical expression specifying the trendline ID number

Notes

A trendline-specific ID number is returned by `TL_New_s` when the trendline is created.

Example

Assign a value, indicating the time of the starting point of a trendline with the ID number of 3, to Value1 variable:

```
Value1=TL_GetBeginTime_s(3);
```

TL_GetBeginVal

Returns the price value (vertical position, corresponding to a value on the price scale of a chart) of the starting point of a trendline with the specified ID number; returns a value of -2 if the specified trendline ID number is invalid.

Of the two end points of a trendline, the point with the earlier date and time is always considered to be the starting point; if the trendline is vertical, the point with the lower price value is considered to be the starting point.

Usage

`TL_GetBeginVal(`*`TL_ID`*`)`

Where: *`TL_ID`* - a numerical expression specifying the trendline ID number

Notes

A trendline-specific ID number is returned by `TL_New` when the trendline is created.

Example

Assign a value, indicating the price value of the starting point of a trendline with the ID number of 3, to Value1 variable:

```
Value1=TL_GetBeginVal(3);
```

TL_GetBegin_BN

Returns a bar number value of the starting point of a trend line with the specified ID number; returns a value of -2 if the specified trendline ID number is invalid.

Usage

`TL_GetBegin_BN(`*`TL_ID`*`)`

Where: *`TL_ID`* - a numerical expression specifying the trendline ID number

Notes

A trendline-specific ID number is returned by `TL_New` when the trendline is created.

Example

`TL_GetBegin_BN(value1)`; will return 5 if Trend Line with ID = Value1 starts on fifth bar of price series.

TL_GetBegin_Dt

Returns a DateTime value, indicating the date and time of the starting point of a trend line with the specified ID number; returns a value of -2 if the specified trendline ID number is invalid.

Usage

`TL_GetBegin_Dt(`*`TL_ID`*`)`

Where: *`TL_ID`* - a numerical expression specifying the trendline ID number

Notes

A trendline-specific ID number is returned by `TL_New_Dt` when the trendline is created.

Example

`TL_GetBegin_Dt(value1)`; will return 41422.752623935186 if Trend Line with ID = Value1 starts at 08:30:00.722 on 5/28/2013

TL_GetColor

Returns an RGB color number or a legacy color value that correspond to the color of a trendline with the specified ID number; returns a value of -2 if the specified trendline ID number is invalid.

Usage

`TL_GetColor(`*`TL_ID`*`)`

Where: *`TL_ID`* - a numerical expression specifying the trendline ID number

Notes

A trendline-specific ID number is returned by `TL_New` when the trendline is created.

Example

Assign an RGB color number, corresponding to the color of a trendline with the ID number of 3, to Value1 variable:

```
Value1=TL_GetColor(3);
```

Assign a legacy color value, corresponding to the color of a trendline with the ID number of 3, to Value1 variable:

```
[LegacyColorValue=True];
Value1=TL_GetColor(3);
```

TL_GetEndDate

Returns a numerical value, indicating the date of the ending point of a trendline with the specified ID number; returns a value of -2 if the specified trendline ID number is invalid.

Of the two end points of a trendline, the point with the later date and time is always considered to be the ending point; if the trendline is vertical, the point with the higher price value is considered to be the ending point.

The date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month.

Usage

`TL_GetEndDate(`*`TL_ID`*`)`

Where: *`TL_ID`* - a numerical expression specifying the trendline ID number

Notes

A trendline-specific ID number is returned by `TL_New` when the trendline is created.

Example

Assign a value, indicating the date of the ending point of a trendline with the ID number of 3, to Value1 variable:

```
Value1=TL_GetEndDate(3);
```

TL_GetEndTime

Returns a numerical value, indicating the time of the ending point of a trendline with the specified ID number; returns a value of -2 if the specified trendline ID number is invalid.

Of the two end points of a trendline, the point with the later date and time is always considered to be the ending point; if the trendline is vertical, the point with the higher price value is considered to be the ending point.

The time is indicated in the 24-hour HHmm format, where 1300 = 1:00 PM.

Usage

`TL_GetEndTime(`*`TL_ID`*`)`

Where: *`TL_ID`* - a numerical expression specifying the trendline ID number

Notes

A trendline-specific ID number is returned by `TL_New` when the trendline is created.

Example

Assign a value, indicating the time of the ending point of a trendline with the ID number of 3, to Value1 variable:

```
Value1=TL_GetEndTime(3);
```

TL_GetEndTime_s

Returns a numerical value indicating the time, including seconds, of the ending point of a trendline with the specified ID number; returns a value of -2 if the specified trendline ID number is invalid.

Of the two end points of a trendline, the point with the later date and time is always considered to be the ending point; if the trendline is vertical, the point with the higher price value is considered to be the ending point.

The time is indicated in the 24-hour HHmmss format, where 130000 = 1:00:00 PM.

Usage

`TL_GetEndTime_s(`*`TL_ID`*`)`

Where: *`TL_ID`* - a numerical expression specifying the trendline ID number

Notes

A trendline-specific ID number is returned by `TL_New_s` when the trendline is created.

Example

Assign a value, indicating the time of the ending point of a trendline with the ID number of 3, to Value1 variable:

```
Value1=TL_GetEndTime_s(3);
```

TL_GetEndVal

Returns the price value (vertical position, corresponding to a value on the price scale of a chart) of the ending point of a trendline with the specified ID number; returns a value of -2 if the specified trendline ID number is invalid.

Of the two end points of a trendline, the point with the later date and time is always considered to be the ending point; if the trendline is vertical, the point with the higher price value is considered to be the ending point.

Usage

`TL_GetEndVal(`*`TL_ID`*`)`

Where: *`TL_ID`* - a numerical expression specifying the trendline ID number

Notes

A trendline-specific ID number is returned by `TL_New` when the trendline is created.

Example

Assign a value, indicating the price value of the ending point of a trendline with the ID number of 3, to Value1 variable:

```
Value1=TL_GetEndVal(3);
```

TL_GetEnd_BN

Returns a bar number value of the ending point of a trend line with the specified ID number; returns a value of -2 if the specified trendline ID number is invalid.

Usage

`TL_GetEnd_BN(`*`TL_ID`*`)`

Where: *`TL_ID`* - numerical expression specifying the trendline ID number

Notes

A trendline-specific ID number is returned by `TL_New` when the trendline is created.

Example

`TL_GetEnd_BN(value1)`; will return 8 if Trend Line with ID = Value1 ends on eighth bar of price series.

TL_GetEnd_Dt

Returns a DateTime value, indicating the date and time of the ending point of a trend line with the specified ID number; returns a value of -2 if the specified trendline ID number is invalid.

Usage

`TL_GetEnd_Dt(`*`TL_ID`*`)`

Where: *`TL_ID`* - numerical expression specifying the trendline ID number

Notes

A trendline-specific ID number is returned by `TL_New_Dt` when the trendline is created.

Example

`TL_GetEnd_Dt(value1)`; will return 41422.752623935186 if Trend Line with ID = Value1 ends at 08:30:00.722 on 5/28/2013

TL_GetExtLeft

Returns a logical value, indicating whether the trendline with the specified ID number is extended to the left; returns a value of `True` if the trendline is extended, and a value of `False` if the trendline is not extended or if the specified trendline ID number is invalid.

Usage

`TL_GetExtLeft(`*`TL_ID`*`)`

Where: *`TL_ID`* - a numerical expression specifying the trendline ID number

Notes

A trendline-specific ID number is returned by `TL_New` when the trendline is created.

Example

Assign a true/false value, indicating whether the trendline with an ID number of 3 is extended to the left, to ExtL variable:

```
Variable:ExtL(False);
ExtL=TL_GetExtLeft(3);
```

TL_GetExtRight

Returns a logical value, indicating whether the trendline with the specified ID number is extended to the right; returns a value of `True` if the trendline is extended, and a value of `False` if the trendline is not extended or if the specified trendline ID number is invalid.

Usage

`TL_GetExtRight(`*`TL_ID`*`)`

Where: *`TL_ID`* - a numerical expression specifying the trendline ID number

Notes

A trendline-specific ID number is returned by `TL_New` when the trendline is created.

Example

Assign a true/false value, indicating whether the trendline with an ID number of 3 is extended to the right, to ExtR variable:

```
Variable:ExtR(False);
ExtR=TL_GetExtRight(3);
```

TL_GetFirst

Returns a numerical value, indicating the trendline ID number of the oldest (the first to be drawn on the current chart) trendline of the specified origin; returns a value of -2 if the specified trendline ID number is invalid.

Usage

`TL_GetFirst` (*`Origin`*)

Parameters

`Origin` - a numerical expression specifying the origin of the trendline:

`1` - drawn by the current study
`2` - drawn by a study other then the current study, or drawn manually by the user
`3` - drawn by any study, or drawn manually by the user
`4` - drawn by the current study, or drawn manually by the user
`5` - drawn by a study other then the current study
`6` - drawn by any study
`7` - drawn manually by the user

Notes

If the oldest (the first to be drawn) trendline is deleted, the next oldest (the second to be drawn) trendline becomes the oldest (the first drawn) trendline.

Example

Assign a value, indicating the trendline ID number of the oldest trendline drawn on the chart by the current study, to Value1 variable:

```
Value1=TL_GetFirst(1);
```

TL_GetLock

Locked trendline drawings cannot be moved manually. Keyword returns a value of True for locked drawings, and a value of False for unlocked.

Usage

`TL_GetLock(`*`TL_ID`*`)`

Where: *`TL_ID`* - a numerical expression specifying the trendline ID number

Notes

A trendline-specific ID number is returned by `TL_New` when the trendline is created.

Example

Assign Lock property of the trendline drawing with an ID number of 3 to Condition1 variable:

```
Condition1=TL_GetLock(3);
```

TL_GetNext

Returns an ID number of the first existing trendline drawn subsequent to a trendline with the specified ID number, with both trendlines of a specified origin; returns a value of -2 if the specified object ID number is invalid.

Usage

`TL_GetNext` (*`TL_ID`*,*`Origin`*)

Parameters

`TL_ID` - a numerical expression specifying the trendline ID number

`Origin` - a numerical expression specifying the origin of the trendlines:

`1` - drawn by the current study
`2` - drawn by a study other then the current study, or drawn manually by the user
`3` - drawn by any study, or drawn manually by the user
`4` - drawn by the current study, or drawn manually by the user
`5` - drawn by a study other then the current study
`6` - drawn by any study
`7` - drawn manually by the user

Example

Assign a value to Value1 variable, indicating the ID number of the first existing trendline drawn subsequent to a trendline with the ID number of 3, with both trendlines drawn by the current study:

```
Value1=TL_GetNext(3,1);
```

TL_GetSize

Returns a numerical value indicating the width of a trendline with the specified ID number; returns a value of -2 if the specified trendline ID number is invalid.

Usage

`TL_GetSize(`*`TL_ID`*`)`

Where: *`TL_ID`* - a numerical expression specifying the trendline ID number

Notes

A trendline-specific ID number is returned by `TL_New` when the trendline is created.

Example

Assign a value, indicating the width of a trendline with the ID number of 3, to Value1 variable:

```
Value1=TL_GetSize(3);
```

TL_GetStyle

Returns a numerical value, indicating the style of a trendline with the specified ID number; returns a value of -2 if the specified trendline ID number is invalid.

Usage

`TL_GetStyle(`*`TL_ID`*`)`

Where: *`TL_ID`* - a numerical expression specifying the trendline ID number

Return

Trendline styles:

1	Tool Solid	________________________
2	Tool Dashed	- - - - - - - - - - - - - - - - - - - -
3	Tool Dotted	....
4	Tool Dashed2	__ _ __ _ __ _ __ _ __ _ __
5	Tool Dashed3	___ _ _ ___ _ _ ___ _ _ ___

Notes

A trendline-specific ID number is returned by `TL_New` when the trendline is created.

Example

Assign a value, indicating the style of a trendline with the ID number of 3, to Value1 variable:

```
Value1=TL_GetStyle(3);
```

TL_GetValue

Returns a price value, at the specified date and time, of a trendline with the specified ID number; returns a value of -2 if the specified trendline ID number is invalid.

If the trendline does not extend to the specified date and time, a price value along the same slope as the trendline will be returned.

Usage

`TL_GetValue` (*TL_ID*, *Date*, *Time*)

Parameters

`TL_ID` - a numerical expression specifying the trendline ID number

`Date` - a numerical expression specifying the date; the date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month

`Time` - a numerical expression specifying the time; the time is indicated in the 24-hour HHmm format, where 1300 = 1:00 PM

Notes

A trendline-specific ID number is returned by `TL_New` when the trendline is created.

Example

Assign the price value, at 10:00 AM on January 17th, of a trendline with the ID number of 3, to Value1 variable:

```
Value1=TL_GetValue(3,1080117,1000);
```

TL_GetValue_BN

Returns a price value, at the specified bar number, of a trendline with the specified ID number; returns a value of -2 if the specified trendline ID number is invalid.

If the trendline does not extend to the specified bar, a price value along the same slope as the trendline will be returned.

Usage

`TL_GetValue_BN(`*`TL_ID`*`, `*`Barnumber`*`)`

Parameters

`TL_ID` - a numerical expression specifying the trendline ID number;

`Barnumber` - a numerical expression specifying a number of a bar;

Notes

A trendline-specific ID number is returned by `TL_New` when the trendline is created.

Example

Assign the price value at bar number 100 of a trendline with the ID number of 3 to Value1 variable:

```
Value1 = TL_GetValue_BN(3, 100);
```

TL_GetValue_Dt

Returns a price value, at the specified date and time, of a trendline with the specified ID number; returns a value of -2 if the specified trendline ID number is invalid.

If the trendline does not extend to the specified date and time, a price value along the same slope as the trendline will be returned.

Usage

`TL_GetValue_Dt(`*`TL_ID`*, *DT*`)`

Parameters

`TL_ID` - a numerical expression specifying the trendline ID number.

`DT` - a numerical expression specifying the trendline starting point date and time; indicated in the DateTime format. The integer portion of the DateTime value indicates the number of days that have elapsed since January 1st, 1900, and the fractional portion of the DateTime value indicates the fraction of the day that has passed since midnight.

Notes

A trendline-specific ID number is returned by `TL_New_Dt` when the trendline is created.

Example

`TL_GetValue_Dt(value1`, 41422.752623935186); will return the price of trend line with ID = Value1 at 08:30:00.882 of 5/28/2013

TL_GetValue_s

Returns a price value, at the specified date and time, of a trendline with the specified ID number; returns a value of -2 if the specified trendline ID number is invalid.

If the trendline does not extend to the specified date and time, a price value along the same slope as the trendline will be returned.

Usage

`TL_GetValue_s` (*TL_ID*, *Date*, *Time_s*)

Parameters

`TL_ID` - a numerical expression specifying the trendline ID number

`Date` - a numerical expression specifying the date; the date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month

`Time_s` - a numerical expression specifying the time, including seconds; the time is indicated in the 24-hour HHmmss format, where 130000 = 1:00:00 PM

Notes

A trendline-specific ID number is returned by `TL_New_s` when the trendline is created.

Example

Assign the price value, at 10:00:00 AM on January 17th, of a trendline with the ID number of 3, to Value1 variable:

```
Value1=TL_GetValue_s(3,1080117,100000);
```

TL_Get_Anchor_to_Bars

Returns the value of the "anchor to bar" option of the trendline drawing with a specified ID.

Usage

`TL_Get_Anchor_to_Bars(`*`TL_ID`*`)`

Where: *`TL_ID`* is a numerical expression specifying the trendline ID number

Notes

A trendline-specific ID number is returned by `TL_New` when the trendline is created.

Example

Assign the "anchor to bars" option of the trendline drawing with an ID number of 3 to the Condition1 variable:

```
Condition1=TL_Get_Anchor_to_Bars(3);
```

TL_Lock

Locks corresponding trendline drawing so it cannot be moved manually; returns a value of 0 if the operation was performed successfully, and a value of -2 if the specified trendline ID number is invalid.

Usage

`TL_Lock(`*`TL_ID`*`,`*`LogicalExpression`*`)`

Where: *`TL_ID`* - a numerical expression specifying the trendline ID number
`LogicalExpression` - a logical value; True = Add and False = Remove

Notes

A trendline-specific ID number is returned by `TL_New` when the trendline is created.

Example

Lock the trendline with an ID number of 3:

```
Value1=TL_Lock(3,True);
```

Unlock the trendline with an ID number of 5:

```
Value1=TL_Lock(5,False);
```

TL_New

Displays a trendline, with the specified starting and ending points, on the chart that the study is based on; returns a trendline-specific ID number, required to modify the trendline.

Usage

`TL_New` (*sDate*, *sTime*, *sPriceValue*, *eDate*, *eTime*, *ePriceValue*)

Parameters

`sDate` - a numerical expression specifying the trendline starting point date; the date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month

`sTime` - a numerical expression specifying the trendline starting point time; the time is indicated in the 24-hour HHmm format, where 1300 = 1:00 PM

`sPriceValue` - a numerical expression specifying the trendline starting point price value (vertical position, corresponding to a value on the price scale of a chart)

`eDate` - a numerical expression specifying the trendline ending point date

`eTime` - a numerical expression specifying the trendline ending point time

`ePriceValue` - a numerical expression specifying the trendline ending point price value

Example

Display a trendline, that begins at 9:00 AM at a price value of 1381, and ends at 3:00 PM at a price value of 1337, on January 17th, 2008, on the chart that the study is based on:

```
Value1=TL_New(1080117,900,1381,1080117,1500,1337);
```

TL_New_BN

Displays a trendline with the specified starting and ending points on the chart that the study is based on; returns a trendline-specific ID number required to modify the trendline.

Instead of separate Date and Time values or a single DateTime value, bar number of price series is used.

Usage

`TL_New_BN` (*b_BarNumber*, *b_Price*, *e_BarNumber*, *e_Price*);

Parameters

`b_BarNumber` - a numerical expression specifying the trendline starting point bar number.

`b_Price` - a numerical expression specifying the trendline starting point price value (vertical position, corresponding to a value on the price scale of a chart).

`e_BarNumber` - a numerical expression specifying the trendline ending point bar number.

`e_Price` - a numerical expression specifying the trendline ending point price value.

Example

Display a trendline that will connect close price of the first bar of price series with the current close price.

```
once Value2=close;

Value1=TL_New_BN(1,Value2,currentbar,Close);

if Value1 <> Value1[1] then tl_delete(Value1[1]);
```

TL_New_Dt

Displays a trendline, with the specified starting and ending points, on the chart that the study is based on; returns a trendline-specific ID number, required to modify the trendline.

Instead of separate Date and Time values, a single DateTime value is used. It allows accessing millisecond time stamps of the bar.

Usage

`TL_New_Dt` (*b_DateTime, b_Price, e_DateTime, e_Price*);

Parameters

`b_DateTime` - a numerical expression specifying the trendline starting point date and time; is indicated in the DateTime format. The integer portion of the DateTime value indicates the number of days that have elapsed since January 1st, 1900, and the fractional portion of the DateTime value indicates the fraction of the day that has passed since midnight.

`b_Price` - a numerical expression specifying the trendline starting point price value (vertical position, corresponding to a value on the price scale of a chart)

`e_DateTime` - a numerical expression specifying the trendline ending point time; the date is indicated in the DateTime format.

`e_Price` - a numerical expression specifying the trendline ending point price value

Example

Display a trenline that will connect a close price 100 bars back with the current close price.

```
Value1=TL_New_Dt(DateTime[100],Close[100],DateTime,Close);

if Value1 <> Value1[1] then tl_delete(Value1[1]);
```

TL_New_s

Displays a trendline, with the specified starting and ending points, on the chart that the study is based on; returns a trendline-specific ID number, required to modify the trendline.

Usage

`TL_New_s` (*sDate, sTime_s, sPriceValue, eDate, eTime_s, ePriceValue*)

Parameters

`sDate` - a numerical expression specifying the trendline starting point date; the date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month

`sTime_s` - a numerical expression specifying the trendline starting point time, including seconds; the time is indicated in the 24-hour HHmmss format, where 130000 = 1:00:00 PM

`sPriceValue` - a numerical expression specifying the trendline starting point price value (vertical position, corresponding to a value on the price scale of a chart)

`eDate` - a numerical expression specifying the trendline ending point date

`eTime_s` - a numerical expression specifying the trendline ending point time, including seconds

`ePriceValue` - a numerical expression specifying the trendline ending point price value

Example

Display a trendline, that begins at 9:00:00 AM at a price value of 1381, and ends at 3:00:00 PM at a price value of 1337, on January 17th, 2008, on the chart that the study is based on:

```
Value1=TL_New_s(1080117,90000,1381,1080117,150000,1337);
```

TL_New_self

Displays a trendline, with the specified starting and ending points, on the SubChart containing the study; returns a trendline-specific ID number, required to modify the trendline.

Usage

`TL_New_self` (*sDate*, *sTime*, *sPriceValue*, *eDate*, *eTime*, *ePriceValue*)

Parameters

`sDate` - a numerical expression specifying the trendline starting point date; the date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month

`sTime` - a numerical expression specifying the trendline starting point time; the time is indicated in the 24-hour HHmm format, where 1300 = 1:00 PM

`sPriceValue` - a numerical expression specifying the trendline starting point price value (vertical position, corresponding to a value on the price scale of a chart)

`eDate` - a numerical expression specifying the trendline ending point date

`eTime` - a numerical expression specifying the trendline ending point time

`ePriceValue` - a numerical expression specifying the trendline ending point price value

Example

Display a trendline, that begins at 9:00 AM at a price value of 1381, and ends at 3:00 PM at a price value of 1337, on January 17th, 2008, on the SubChart containing the study:

```
Value1=TL_New_self(1080117,900,1381,1080117,1500,1337);
```

TL_New_Self_BN

The same as `TL_New_BN`. Difference: Displays a trendline on the SubChart containing the study.

TL_New_Self_Dt

Displays a trendline, with the specified starting and ending points, on the SubChart containing the study; returns a trendline-specific ID number, required to modify the trendline.

Instead of separate Date and Time values, a single DateTime value is used. It allows accessing millisecond time stamps of the bar.

Usage

`TL_New_Self_Dt` (*b_DateTime*, *b_Price*, *e_DateTime*, *e_Price*);

Parameters

`b_DateTime` - a numerical expression specifying the trendline starting point date and time; is indicated in the DateTime format. The integer portion of the DateTime value indicates the number of days that have elapsed since January 1st, 1900, and the fractional portion of the DateTime value indicates the fraction of the day that has passed since midnight.

`b_Price` - a numerical expression specifying the trendline starting point price value (vertical position, corresponding to a value on the price scale of a chart)

`e_DateTime` - a numerical expression specifying the trendline ending point time; the date is indicated in the DateTime format.

`e_Price` - a numerical expression specifying the trendline ending point price value

Example

Display a trenline that will connect a close price 100 bars back with the current close price on the SubChart containing the study.

```
Value1=TL_New_Self_Dt(DateTime[100],Close[100],DateTime,Close);

if Value1 <> Value1[1] then tl_delete(Value1[1]);
```

TL_New_Self_s

Displays a trendline, with the specified starting and ending points, on the SubChart containing the study; returns a trendline-specific ID number, required to modify the trendline.

Usage

`TL_New_s` (*sDate, sTime_s, sPriceValue, eDate, eTime_s, ePriceValue*)

Parameters

`sDate` - a numerical expression specifying the trendline starting point date; the date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month

`sTime_s` - a numerical expression specifying the trendline starting point time, including seconds; the time is indicated in the 24-hour HHmmss format, where 130000 = 1:00:00 PM

`sPriceValue` - a numerical expression specifying the trendline starting point price value (vertical position, corresponding to a value on the price scale of a chart)

`eDate` - a numerical expression specifying the trendline ending point date

`eTime_s` - a numerical expression specifying the trendline ending point time, including seconds

`ePriceValue` - a numerical expression specifying the trendline ending point price value

Example

Display a trendline, that begins at 9:00:00 AM at a price value of 1381, and ends at 3:00:00 PM at a price value of 1337, on January 17th, 2008, on the SubChart containing the study:

```
Value1=TL_New_s(1080117,90000,1381,1080117,150000,1337);
```

TL_SetAlert

Sets the alert status for a trendline with the specified ID number; returns a value of 0 if alert status was successfully modified, and a value of -2 if the specified trendline ID number is invalid.

Usage

`TL_SetAlert(`*`TL_ID`*`,`*`AlertStatus`*`)`

Parameters

`TL_ID` - a numerical expression specifying the trendline ID number

`AlertStatus` - a numerical expression specifying the alert status for the trendline:

`0` - Alert disabled

`1` - Breakout intra-bar

An alert is triggered if the High crosses over the trendline or the Low crosses under the trendline. Alert triggering conditions are evaluated intra-bar.

`2` - Breakout on close

An alert is triggered if the Close of the previous bar was below the trendline and the Close of the current bar is above the trendline, or if the Close of the previous bar was above the trendline and the Close of the current bar is below the trendline. Alert triggering conditions are evaluated at Close of a bar.

Notes

A trendline-specific ID number is returned by `TL_New` when the trendline is created.

Example

Set alert status to "Breakout on close" for a trendline with the ID number of 3:

```
Value1=TL_SetAlert(3,2);
```

TL_SetBegin

Modifies the starting point location of a trendline with the specified ID number; returns a value of 0 if the starting point location was successfully modified, and a value of -2 if the specified trendline ID number is invalid.

Of the two end points of a trendline, the point with the earlier date and time is always considered to be the starting point; if the trendline is vertical, the point with the lower price value is considered to be the starting point.

Usage

`TL_SetBegin` (*TL_ID*, *sDate*, *sTime*, *sPriceValue*)

Parameters

`TL_ID` - a numerical expression specifying the trendline ID number

`sDate` - a numerical expression specifying the new starting point date; the date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month

`sTime` - a numerical expression specifying the new starting point time; the time is indicated in the 24-hour HHmm format, where 1300 = 1:00 PM

`sPriceValue` - a numerical expression specifying the new starting point price value (vertical position, corresponding to a value on the price scale of a chart)

Notes

A trendline-specific ID number is returned by `TL_New` when the trendline is created.

Example

Move the starting point of the trendline with an ID number of 3 to 10:00 AM on January 17th at a price value of 1365:

```
Value1=TL_SetBegin(3,1080117,1000,1365);
```

TL_SetBegin_BN

Modifies the starting point location of a trendline with the specified ID number; returns a value of 0 if the starting point location was successfully modified, and a value of -2 if the specified trendline ID number is invalid.

Between the two end points of a trendline, the point with the earlier date and time is always considered to be the starting point; if the trendline is vertical, the point with the lower price value is considered to be the starting point.

Instead of separate Date and Time values or a single DateTime value, bar number is used.

Usage

`TL_SetBegin_BN` (*TL_ID*, *BarNumber*, *Price*);

Parameters

`TL_ID` - a numerical expression specifying the trendline ID number.

`BarNumber` - a numerical expression specifying the trendline starting point bar number.

`Price` - a numerical expression specifying the trendline starting point price value (vertical position, corresponding to a value on the price scale of a chart).

Notes

A trendline-specific ID number is returned by `TL_New` when the trendline is created.

Example

Set the beginning point of Trend Line with ID = 3 to close price of current bar

```
Value1=TL_SetBegin_BN(3,currentbar[100],Close[100]);
```

TL_SetBegin_DT

Modifies the starting point location of a trendline with the specified ID number; returns a value of 0 if the starting point location was successfully modified, and a value of -2 if the specified trendline ID number is invalid.

Of the two end points of a trendline, the point with the earlier date and time is always considered to be the starting point; if the trendline is vertical, the point with the lower price value is considered to be the starting point.

Instead of separate Date and Time values, a single DateTime value is used. It allows accessing millisecond time stamps of the bar.

Usage

`TL_SetBegin_DT` (*TL_ID*, *b_DateTime*, *b_Price*);

Parameters

`TL_ID` - a numerical expression specifying the trendline ID number

`b_DateTime` - a numerical expression specifying the trendline starting point date and time; is indicated in the DateTime format. The integer portion of the DateTime value indicates the number of days that have elapsed since January 1st, 1900, and the fractional portion of the DateTime value indicates the fraction of the day that has passed since midnight.

`b_Price` - a numerical expression specifying the trendline starting point price value (vertical position, corresponding to a value on the price scale of a chart)

Notes

A trendline-specific ID number is returned by `TL_New_DT` when the trendline is created.

Example

Set the beginning point of Trend Line with ID = value 1 to open price 100 bars back

```
TL_SetBegin_DT(value1,DateTime[100],Open[100]);
```

TL_SetBegin_s

Modifies the starting point location of a trendline with the specified ID number; returns a value of 0 if the starting point location was successfully modified, and a value of -2 if the specified trendline ID number is invalid.

Of the two end points of a trendline, the point with the earlier date and time is always considered to be the starting point; if the trendline is vertical, the point with the lower price value is considered to be the starting point.

Usage

`TL_SetBegin_s` (*TL_ID*, *sDate*, *sTime_s*, *sPriceValue*)

Parameters

`TL_ID` - a numerical expression specifying the trendline ID number

`sDate` - a numerical expression specifying the new starting point date; the date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month

`sTime_s` - a numerical expression specifying the new starting point time, including seconds; the time is indicated in the 24-hour HHmmss format, where 130000 = 1:00:00 PM

`sPriceValue` - a numerical expression specifying the new starting point price value (vertical position, corresponding to a value on the price scale of a chart)

Notes

A trendline-specific ID number is returned by `TL_New_s` when the trendline is created.

Example

Move the starting point of the trendline with an ID number of 3 to 10:00:00 AM on January 17th at a price value of 1365:

```
Value1=TL_SetBegin_s(3,1080117,100000,1365);
```

TL_SetColor

Assigns the specified color to a trendline with the specified ID number; returns a value of 0 if the color was successfully assigned, and a value of -2 if the specified trendline ID number is invalid.

Usage

`TL_SetColor(`*TL_ID*,*`TL_Color`*`)`

Parameters

`TL_ID` - a numerical expression specifying the trendline ID number

`TL_Color` - an expression specifying the trendline color

Trendline color can be specified by a numerical expression representing an RGB color number or a legacy color value, or by one of 17 base color words.

Notes

A trendline-specific ID number is returned by `TL_New` when the trendline is created.

Example

Assign the color blue to the trendline with an ID number of 3:

```
Value1=TL_SetColor(3,Blue);
```

Assign the RGB color 2138336 (Orange) to the trendline with an ID number of 3:

```
Value1=TL_SetColor(3,2138336);
```

Assign the legacy color 4 (Green) to the trendline with an ID number of 3:

```
[LegacyColorValue=True];
Value1=TL_SetColor(3,4);
```

TL_SetEnd

Modifies the ending point location of a trendline with the specified ID number; returns a value of 0 if the ending point location was successfully modified, and a value of -2 if the specified trendline ID number is invalid.

Of the two end points of a trendline, the point with the later date and time is always considered to be the ending point; if the trendline is vertical, the point with the higher price value is considered to be the ending point.

Usage

`TL_SetEnd` (*TL_ID*, *eDate*, *eTime*, *ePriceValue*)

Parameters

`TL_ID` - a numerical expression specifying the trendline ID number

`eDate` - a numerical expression specifying the new ending point date; the date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month

`eTime` - a numerical expression specifying the new ending point time; the time is indicated in the 24-hour HHmm format, where 1300 = 1:00 PM

`ePriceValue` - a numerical expression specifying the new ending point price value (vertical position, corresponding to a value on the price scale of a chart)

Notes

A trendline-specific ID number is returned by `TL_New` when the trendline is created.

Example

Move the ending point of the trendline with an ID number of 3 to 14:15 PM on January 17th at a price value of 1350:

```
Value1=TL_SetEnd(3,1080117,1415,1350);
```

TL_SetEnd_BN

Modifies the ending point location of a trendline with the specified ID number; returns a value of 0 if the ending point location was successfully modified, and a value of -2 if the specified trendline ID number is invalid.

Between the two end points of a trendline, the point with the later date and time is always considered to be the ending point; if the trendline is vertical, the point with the higher price value is considered to be the ending point.

Instead of separate Date and Time values or a single DateTime value, bar number is used.

Usage

`TL_SetEnd_BN` (*TL_ID*, *BarNumber*, *Price*);

Parameters

`TL_ID` - a numerical expression specifying the trendline ID number.

`BarNumber` - a numerical expression specifying the trendline ending point bar number.

`Price` - a numerical expression specifying the trendline ending point price value.

Example

Set the ending point of Trend Line with ID = 3 to close price of the current bar

```
Value1=TL_SetEnd_BN(3,currentbar,Close);
```

TL_SetEnd_Dt

Modifies the ending point location of a trendline with the specified ID number; returns a value of 0 if the ending point location was successfully modified, and a value of -2 if the specified trendline ID number is invalid.

Of the two end points of a trendline, the point with the later date and time is always considered to be the ending point; if the trendline is vertical, the point with the higher price value is considered to be the ending point.

Instead of separate Date and Time values, a single DateTime value is used. It allows accessing millisecond time stamps of the bar.

Usage

`TL_SetEnd_Dt` (*TL_ID*, *e_DateTime*, *e_Price*);

Parameters

`TL_ID` - a numerical expression specifying the trendline ID number.

`e_DateTime` - a numerical expression specifying the trendline ending point date and time; the date and time are indicated in the DateTime format.

`e_Price` - a numerical expression specifying the trendline ending point price value.

Example

Set the ending point of Trend Line with ID = value 1 to close price of the current bar

```
TL_SetEnd_Dt(value1,DateTime,Close);
```

TL_SetEnd_s

Modifies the ending point location of a trendline with the specified ID number; returns a value of 0 if the ending point location was successfully modified, and a value of -2 if the specified trendline ID number is invalid.

Of the two end points of a trendline, the point with the later date and time is always considered to be the ending point; if the trendline is vertical, the point with the higher price value is considered to be the ending point.

Usage

`TL_SetEnd_s` (*TL_ID*, *eDate*, *eTime_s*, *ePriceValue*)

Parameters

`TL_ID` - a numerical expression specifying the trendline ID number

`eDate` - a numerical expression specifying the new ending point date; the date is indicated in the YYYMMdd format, where YYY is the number of years since 1900, MM is the month, and dd is the day of the month

`eTime_s` - a numerical expression specifying the new ending point time, including seconds; the time is indicated in the 24-hour HHmmss format, where 130000 = 1:00:00 PM

`ePriceValue` - a numerical expression specifying the new ending point price value (vertical position, corresponding to a value on the price scale of a chart)

Notes

A trendline-specific ID number is returned by `TL_New_s` when the trendline is created.

Example

Move the ending point of the trendline with an ID number of 3 to 14:15:00 PM on January 17th at a price value of 1350:

```
Value1=TL_SetEnd_s(3,1080117,141500,1350);
```

TL_SetExtLeft

Adds or removes a left-side extension for a trendline with the specified ID number; returns a value of 0 if the operation was performed successfully, and a value of -2 if the specified trendline ID number is invalid.

Usage

`TL_SetExtLeft(`*`TL_ID`*`,`*`LogicalExpression`*`)`

Where: *`TL_ID`* - a numerical expression specifying the trendline ID number
`LogicalExpression` - a logical value; True = Add and False = Remove

Notes

A trendline-specific ID number is returned by `TL_New` when the trendline is created.

Example

Add a left-side extension to the trendline with an ID number of 3:

```
Value1=TL_SetExtLeft(3,True);
```

Remove a left-side extension from the trendline with an ID number of 3:

```
Value1=TL_SetExtLeft(3,False);
```

TL_SetExtRight

Adds or removes a right-side extension for a trendline with the specified ID number; returns a value of 0 if the operation was performed successfully, and a value of -2 if the specified trendline ID number is invalid.

Usage

`TL_SetExtRight(`*`TL_ID`*`,`*`LogicalExpression`*`)`

Where: *`TL_ID`* - a numerical expression specifying the trendline ID number
`LogicalExpression` - a logical value; True = Add and False = Remove

Notes

A trendline-specific ID number is returned by `TL_New` when the trendline is created.

Example

Add a right-side extension to the trendline with an ID number of 3:

```
Value1=TL_SetExtRight(3,True);
```

Remove a right-side extension from the trendline with an ID number of 3:

```
Value1=TL_SetExtRight(3,False);
```

TL_SetSize

Assigns the specified width to a trendline with the specified ID number; returns a value of 0 if the line width was successfully assigned, and a value of -2 if the specified trendline ID number is invalid.

Usage

`TL_SetSize(`*TL_ID*,*`LineWidth`*`)`

Where: *`TL_ID`* - a numerical expression specifying the trendline ID number
`LineWidth` - a numerical expression specifying the trendline width; trendline width can range from 0 to 6

Notes

A trendline-specific ID number is returned by `TL_New` when the trendline is created.

Example

Assign the width of 5 to the trendline with an ID number of 3:

```
Value1=TL_SetSize(3,5);
```

TL_SetStyle

Assigns the specified style to a trendline with the specified ID number; returns a value of 0 if the trendline style was successfully assigned, and a value of -2 if the specified trendline ID number is invalid.

Usage

`TL_SetStyle(`*`TL_ID`*`,`*`TL_Style`*`)`

Parameters

`TL_ID` - a numerical expression specifying the trendline ID number

`TL_Style` - a TL style constant or a numerical expression specifying the trendline style as follows:

`Tool_Solid`	1	________________________
`Tool_Dashed`	2	- - - - - - - - - - - - - - - - -
`Tool_Dotted`	3	....
`Tool_Dashed2`	4	__ _ __ _ __ _ __ _ __ _ .
`Tool_Dashed3`	5	___ _ _ ___ _ _ ___ _ _ _

Notes

A trendline-specific ID number is returned by `TL_New` when the trendline is created.

Example

Set the style of the trendline with an ID number of 3 to Tool Dashed:

```
Value1=TL_SetStyle(3, 2);
```

Set the style of the trendline with an ID number of 3 to Tool Dashed:

```
Value1=TL_SetStyle(3, Tool_Dashed);
```

Tool_Dashed

Constant, used in combination with TL_SetStyle to designate the Tool Dashed style; can be substituted by a numerical value of 2.

Usage

`TL_SetStyle(`*`TL_ID`*`, Tool_Dashed)`

or:

`TL_SetStyle(`*`TL_ID`*`, 2)`

Example

Set the style of the trendline with an ID number of 3 to Tool Dashed:

```
Value1=TL_SetStyle(3, 2);
```

Set the style of the trendline with an ID number of 3 to Tool Dashed:

```
Value1=TL_SetStyle(3, Tool_Dashed);
```

Tool_Dashed2

Constant, used in combination with TL_SetStyle to designate the Tool Dashed2 style; can be substituted by a numerical value of 4.

Usage

`TL_SetStyle(`*`TL_ID`*`, Tool_Dashed2)`

or:

`TL_SetStyle(`*`TL_ID`*`, 4)`

Example

Set the style of the trendline with an ID number of 3 to Tool Dashed2:

```
Value1=TL_SetStyle(3, 4);
```

Set the style of the trendline with an ID number of 3 to Tool Dashed2:

```
Value1=TL_SetStyle(3, Tool_Dashed2);
```

Tool_Dashed3

Constant, used in combination with TL_SetStyle to designate the Tool Dashed3 style; can be substituted by a numerical value of 5.

Usage

`TL_SetStyle(`*`TL_ID`*`, Tool_Dashed3)`

or:

`TL_SetStyle(`*`TL_ID`*`, 5)`

Example

Set the style of the trendline with an ID number of 3 to Tool Dashed3:

```
Value1=TL_SetStyle(3, 5);
```

Set the style of the trendline with an ID number of 3 to Tool Dashed3:

```
Value1=TL_SetStyle(3, Tool_Dashed3);
```

Tool_Dotted

Constant, used in combination with TL_SetStyle to designate the Tool Dotted style; can be substituted by a numerical value of 3.

Usage

`TL_SetStyle(`*`TL_ID`*`, Tool_Dotted)`

or:

`TL_SetStyle(`*`TL_ID`*`, 3)`

Example

Set the style of the trendline with an ID number of 3 to Tool Dotted:

```
Value1=TL_SetStyle(3, 3);
```

Set the style of the trendline with an ID number of 3 to Tool Dotted:

```
Value1=TL_SetStyle(3, Tool_Dotted);
```

Tool_Solid

Constant, used in combination with TL_SetStyle to designate the Tool Solid style; can be substituted by a numerical value of 1.

Usage

`TL_SetStyle(`*`TL_ID`*`, Tool_Solid)`

or:

`TL_SetStyle(`*`TL_ID`*`, 1)`

Example

Set the style of the trendline with an ID number of 3 to Tool Solid:

```
Value1=TL_SetStyle(3, 1);
```

Set the style of the trendline with an ID number of 3 to Tool Solid:

```
Value1=TL_SetStyle(3, Tool_Solid);
```